

# **Oracle® Application Express**

API Reference

Release 5.0

**E39149-06**

September 2016

Oracle Application Express API Reference, Release 5.0

E39149-06

Copyright © 2003, 2016, Oracle and/or its affiliates. All rights reserved.

Primary Author: Terri Jennings

Contributors: Christina Cho, Hilary Farrell, Michael Hichwa, Joel Kallman, Sharon Kennedy, Christian Neumueller, Anthony Raynor, Marc Sewtz, John Snyders, Jason Straub, Drue Swadener, Vladislav Unarov, Patrick Wolf

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

---

---

# Contents

<b>Preface</b> .....	xxi
Audience .....	xxi
Documentation Accessibility .....	xxi
Related Documents .....	xxi
Conventions .....	xxii
<b>Changes in This Release</b> .....	xxiii
Changes in Oracle Application Express Release 5.0 .....	xxiii
<b>1 APEX_APPLICATION</b>	
Global Variables .....	1-2
Referencing Arrays .....	1-3
Referencing Values Within an On Submit Process .....	1-4
Converting an Array to a Single Value .....	1-5
HELP Procedure .....	1-6
STOP_APEX_ENGINE Procedure .....	1-8
<b>2 APEX_APPLICATION_INSTALL</b>	
Package Overview .....	2-2
Attributes Manipulated by APEX_APPLICATION_INSTALL .....	2-3
Import Script Examples .....	2-4
CLEAR_ALL Procedure .....	2-7
GENERATE_APPLICATION_ID Procedure .....	2-8
GENERATE_OFFSET Procedure .....	2-9
GET_APPLICATION_ALIAS Function .....	2-10
GET_APPLICATION_ID Function .....	2-11
GET_APPLICATION_NAME Function .....	2-12
GET_AUTO_INSTALL_SUP_OBJ Function .....	2-13
GET_IMAGE_PREFIX Function .....	2-14
GET_OFFSET Function .....	2-15
GET_PROXY Function .....	2-16
GET_SCHEMA Function .....	2-17
GET_WORKSPACE_ID Function .....	2-18
SET_APPLICATION_ALIAS Procedure .....	2-19
SET_APPLICATION_ID Procedure .....	2-20

SET_APPLICATION_NAME Procedure .....	2-21
SET_AUTO_INSTALL_SUP_OBJ Procedure .....	2-22
SET_IMAGE_PREFIX Procedure .....	2-23
SET_OFFSET Procedure.....	2-24
SET_PROXY Procedure.....	2-25
SET_SCHEMA Procedure.....	2-26
SET_WORKSPACE_ID Procedure .....	2-27

### 3 APEX\_AUTHENTICATION

Constants.....	3-2
CALLBACK Procedure.....	3-3
GET_CALLBACK_URL Function .....	3-5
GET_LOGIN_USERNAME_COOKIE Function.....	3-6
IS_AUTHENTICATED Function.....	3-7
IS_PUBLIC_USER Function.....	3-8
LOGIN Procedure .....	3-9
LOGOUT Procedure .....	3-10
POST_LOGIN Procedure.....	3-11
SEND_LOGIN_USERNAME_COOKIE Procedure .....	3-12

### 4 APEX\_AUTHORIZATION

ENABLE_DYNAMIC_GROUPS Procedure.....	4-2
IS_AUTHORIZED Function .....	4-3
RESET_CACHE Procedure.....	4-4

### 5 APEX\_COLLECTION

About the APEX_COLLECTION API .....	5-3
Naming Collections .....	5-4
Creating a Collection .....	5-5
About the Parameter p_generate_md5.....	5-6
Accessing a Collection.....	5-7
Merging Collections .....	5-8
Truncating a Collection.....	5-9
Deleting a Collection .....	5-10
Deleting All Collections for the Current Application .....	5-11
Deleting All Collections in the Current Session.....	5-12
Adding Members to a Collection .....	5-13
About the Parameters p_generate_md5, p_clob001, p_blob001, and p_xmltype001 .....	5-14
Updating Collection Members .....	5-15
Deleting Collection Members.....	5-16
Obtaining a Member Count .....	5-17
Resequencing a Collection .....	5-18
Verifying Whether a Collection Exists.....	5-19
Adjusting a Member Sequence ID .....	5-20
Sorting Collection Members .....	5-21
Clearing Collection Session State .....	5-22



Determining Collection Status .....	5-23
ADD_MEMBER Procedure .....	5-24
ADD_MEMBER Function .....	5-26
ADD_MEMBERS Procedure.....	5-28
COLLECTION_EXISTS Function .....	5-30
COLLECTION_HAS_CHANGED Function.....	5-31
COLLECTION_MEMBER_COUNT Function.....	5-32
CREATE_COLLECTION Procedure.....	5-33
CREATE_OR_TRUNCATE_COLLECTION Procedure .....	5-34
CREATE_COLLECTION_FROM_QUERY Procedure.....	5-35
CREATE_COLLECTION_FROM_QUERY2 Procedure.....	5-36
CREATE_COLLECTION_FROM_QUERY_B Procedure .....	5-37
CREATE_COLLECTION_FROM_QUERYB2 Procedure .....	5-39
DELETE_ALL_COLLECTIONS Procedure.....	5-41
DELETE_ALL_COLLECTIONS_SESSION Procedure.....	5-42
DELETE_COLLECTION Procedure .....	5-43
DELETE_MEMBER Procedure .....	5-44
DELETE_MEMBERS Procedure.....	5-45
GET_MEMBER_MD5 Function .....	5-46
MERGE_MEMBERS Procedure .....	5-47
MOVE_MEMBER_DOWN Procedure.....	5-49
MOVE_MEMBER_UP Procedure.....	5-50
RESEQUENCE_COLLECTION Procedure .....	5-51
RESET_COLLECTION_CHANGED Procedure .....	5-52
RESET_COLLECTION_CHANGED_ALL Procedure .....	5-53
SORT_MEMBERS Procedure .....	5-54
TRUNCATE_COLLECTION Procedure.....	5-55
UPDATE_MEMBER Procedure.....	5-56
UPDATE_MEMBERS Procedure .....	5-58
UPDATE_MEMBER_ATTRIBUTE Procedure Signature 1 .....	5-60
UPDATE_MEMBER_ATTRIBUTE Procedure Signature 2 .....	5-62
UPDATE_MEMBER_ATTRIBUTE Procedure Signature 3 .....	5-64
UPDATE_MEMBER_ATTRIBUTE Procedure Signature 4 .....	5-66
UPDATE_MEMBER_ATTRIBUTE Procedure Signature 5 .....	5-68
UPDATE_MEMBER_ATTRIBUTE Procedure Signature 6 .....	5-70

## 6 APEX\_CSS

ADD Procedure.....	6-2
ADD_3RD_PARTY_LIBRARY_FILE Procedure.....	6-3
ADD_FILE Procedure.....	6-4

## 7 APEX\_CUSTOM\_AUTH

APPLICATION_PAGE_ITEM_EXISTS Function.....	7-2
CURRENT_PAGE_IS_PUBLIC Function .....	7-3
DEFINE_USER_SESSION Procedure .....	7-4
GET_COOKIE_PROPS Procedure.....	7-5

GET_LDAP_PROPS Procedure .....	7-6
GET_NEXT_SESSION_ID Function .....	7-8
GET_SECURITY_GROUP_ID Function.....	7-9
GET_SESSION_ID Function .....	7-10
GET_SESSION_ID_FROM_COOKIE Function .....	7-11
GET_USER Function .....	7-12
GET_USERNAME Function .....	7-13
IS_SESSION_VALID Function .....	7-14
LOGIN Procedure .....	7-15
LOGOUT Procedure [DEPRECATED].....	7-16
POST_LOGIN Procedure.....	7-17
SESSION_ID_EXISTS Function.....	7-18
SET_SESSION_ID Procedure .....	7-19
SET_SESSION_ID_TO_NEXT_VALUE Procedure.....	7-20
SET_USER Procedure .....	7-21

## 8 APEX\_DEBUG

Constants.....	8-3
DISABLE Procedure .....	8-4
ENABLE Procedure .....	8-5
ENTER Procedure.....	8-6
ERROR Procedure .....	8-8
INFO Procedure .....	8-9
LOG_DBMS_OUTPUT Procedure.....	8-10
LOG_LONG_MESSAGE Procedure.....	8-11
LOG_MESSAGE Procedure [Deprecated].....	8-12
LOG_PAGE_SESSION_STATE Procedure.....	8-13
MESSAGE Procedure .....	8-14
REMOVE_DEBUG_BY_AGE Procedure .....	8-16
REMOVE_DEBUG_BY_APP Procedure .....	8-17
REMOVE_DEBUG_BY_VIEW Procedure.....	8-18
REMOVE_SESSION_MESSAGES Procedure.....	8-19
TOCHAR Function .....	8-20
TRACE Procedure .....	8-21
WARN Procedure .....	8-22

## 9 APEX\_ERROR

Constants and Attributes used for Result Types .....	9-2
Example of an Error Handling Function.....	9-4
ADD_ERROR Procedure Signature 1 .....	9-6
ADD_ERROR Procedure Signature 2 .....	9-7
ADD_ERROR Procedure Signature 3 .....	9-8
ADD_ERROR Procedure Signature 4 .....	9-10
ADD_ERROR Procedure Signature 5 .....	9-12
AUTO_SET_ASSOCIATED_ITEM Procedure.....	9-14
EXTRACT_CONSTRAINT_NAME Function .....	9-15
GET_ARIA_ERROR_ATTRIBUTES Function.....	9-16

GET_FIRST_ORA_ERROR_TEXT Function .....	9-17
INIT_ERROR_RESULT Function .....	9-18
<b>10 APEX_ESCAPE</b>	
Constants.....	10-2
HTML Function .....	10-3
HTML_ATTRIBUTE Function .....	10-5
HTML_TRUNC Function .....	10-6
HTML_WHITELIST Function .....	10-7
JS_LITERAL Function .....	10-8
JSON Function .....	10-9
LDAP_DN Function.....	10-10
LDAP_SEARCH_FILTER Function .....	10-11
NOOP Function .....	10-12
REGEXP Function .....	10-13
SET_HTML_ESCAPING_MODE Procedure.....	10-14
<b>11 APEX_INSTANCE_ADMIN</b>	
Available Parameter Values .....	11-2
ADD_SCHEMA Procedure .....	11-8
ADD_WORKSPACE Procedure .....	11-9
CREATE_SCHEMA_EXCEPTION Procedure .....	11-10
FREE_WORKSPACE_APP_IDS Procedure .....	11-11
GET_PARAMETER Function .....	11-12
GET_SCHEMAS Function.....	11-13
GET_WORKSPACE_PARAMETER.....	11-14
REMOVE_APPLICATION Procedure .....	11-15
REMOVE_SAVED_REPORT Procedure .....	11-16
REMOVE_SAVED_REPORTS Procedure.....	11-17
REMOVE_SCHEMA Procedure .....	11-18
REMOVE_SCHEMA_EXCEPTION Procedure .....	11-19
REMOVE_SCHEMA_EXCEPTIONS Procedure.....	11-20
REMOVE_SUBSCRIPTION Procedure.....	11-21
REMOVE_WORKSPACE Procedure.....	11-22
REMOVE_WORKSPACE_EXCEPTIONS Procedure .....	11-23
RESERVE_WORKSPACE_APP_IDS Procedure.....	11-24
RESTRICT_SCHEMA Procedure.....	11-26
SET_LOG_SWITCH_INTERVAL Procedure .....	11-27
SET_WORKSPACE_PARAMETER .....	11-28
SET_PARAMETER Procedure.....	11-29
SET_WORKSPACE_CONSUMER_GROUP Procedure.....	11-30
TRUNCATE_LOG Procedure .....	11-31
UNRESTRICT_SCHEMA Procedure .....	11-32
<b>12 APEX_IR</b>	
ADD_FILTER Procedure Signature 1.....	12-2

ADD_FILTER Procedure Signature 2 .....	12-4
CHANGE_SUBSCRIPTION_EMAIL Procedure .....	12-6
CHANGE_REPORT_OWNER Procedure .....	12-7
CHANGE_SUBSCRIPTION_EMAIL Procedure .....	12-8
CHANGE_SUBSCRIPTION_LANG Procedure .....	12-9
CLEAR_REPORT Procedure Signature 1 .....	12-10
CLEAR_REPORT Procedure Signature 2 .....	12-11
DELETE_REPORT Procedure .....	12-12
DELETE_SUBSCRIPTION Procedure .....	12-13
GET_LAST_VIEWED_REPORT_ID Function .....	12-14
GET_REPORT Function.....	12-15
RESET_REPORT Procedure Signature 1 .....	12-16
RESET_REPORT Procedure Signature 2 .....	12-17

### 13 APEX\_ITEM

CHECKBOX2 Function .....	13-2
DATE_POPUP Function .....	13-4
DATE_POPUP2 Function .....	13-6
DISPLAY_AND_SAVE Function.....	13-8
HIDDEN Function .....	13-9
MD5_CHECKSUM Function .....	13-11
MD5_HIDDEN Function.....	13-12
POPUP_FROM_LOV Function.....	13-13
POPUP_FROM_QUERY Function .....	13-15
POPUPKEY_FROM_LOV Function .....	13-17
POPUPKEY_FROM_QUERY Function.....	13-19
RADIOGROUP Function .....	13-21
SELECT_LIST Function .....	13-22
SELECT_LIST_FROM_LOV Function.....	13-24
SELECT_LIST_FROM_LOV_XL Function.....	13-25
SELECT_LIST_FROM_QUERY Function .....	13-27
SELECT_LIST_FROM_QUERY_XL Function .....	13-29
TEXT Function .....	13-31
TEXTAREA Function.....	13-32
TEXT_FROM_LOV Function.....	13-33
TEXT_FROM_LOV_QUERY Function .....	13-34

### 14 APEX\_JAVASCRIPT

ADD_3RD_PARTY_LIBRARY_FILE Procedure.....	14-2
ADD_ATTRIBUTE Function Signature 1 .....	14-3
ADD_ATTRIBUTE Function Signature 2 .....	14-5
ADD_ATTRIBUTE Function Signature 3 .....	14-6
ADD_ATTRIBUTE Function Signature 4 .....	14-7
ADD_INLINE_CODE Procedure.....	14-8
ADD_LIBRARY Procedure .....	14-9
ADD_ONLOAD_CODE Procedure.....	14-10
ADD_VALUE Function Signature 1.....	14-11

ADD_VALUE Function Signature 2.....	14-12
ADD_VALUE Function Signature 3.....	14-13
ADD_VALUE Function Signature 4.....	14-14
Escape Function .....	14-15

## 15 APEX\_JSON

Package Overview and Examples.....	15-3
Constants and Data Types .....	15-4
CLOSE_ALL Procedure.....	15-5
CLOSE_ARRAY Procedure .....	15-6
CLOSE_OBJECT Procedure .....	15-7
DOES_EXIST Function .....	15-8
FIND_PATHS_LIKE Function.....	15-9
FREE_OUTPUT Procedure .....	15-10
FLUSH Procedure.....	15-11
GET_BOOLEAN Function.....	15-12
GET_CLOB_OUTPUT Function.....	15-13
GET_COUNT Function.....	15-14
GET_DATE Function.....	15-15
GET_MEMBERS Function .....	15-16
GET_NUMBER Function.....	15-17
GET_VALUE Function .....	15-18
GET_VARCHAR2 Function .....	15-19
INITIALIZE_CLOB_OUTPUT Procedure.....	15-20
INITIALIZE_OUTPUT Procedure .....	15-21
OPEN_ARRAY Procedure .....	15-22
OPEN_OBJECT Procedure .....	15-23
PARSE Procedure Signature 1 .....	15-24
PARSE Procedure Signature 2 .....	15-25
STRINGIFY Function Signature 1 .....	15-26
STRINGIFY Function Signature 2 .....	15-27
STRINGIFY Function Signature 3 .....	15-28
STRINGIFY Function Signature 4 .....	15-29
TO_XMLTYPE Function .....	15-30
WRITE Procedure Signature 1 .....	15-31
WRITE Procedure Signature 2 .....	15-32
WRITE Procedure Signature 3 .....	15-33
WRITE Procedure Signature 4 .....	15-34
WRITE Procedure Signature 5 .....	15-35
WRITE Procedure Signature 6 .....	15-36
WRITE Procedure Signature 7 .....	15-37
WRITE Procedure Signature 8 .....	15-38
WRITE Procedure Signature 9 .....	15-39
WRITE Procedure Signature 10 .....	15-40
WRITE Procedure Signature 11 .....	15-41
WRITE Procedure Signature 12 .....	15-42
WRITE Procedure Signature 13 .....	15-43

WRITE Procedure Signature 14 .....	15-44
WRITE Procedure Signature 15 .....	15-45
WRITE Procedure Signature 16 .....	15-46
<b>16 APEX_LANG</b>	
CREATE_LANGUAGE_MAPPING Procedure.....	16-2
DELETE_LANGUAGE_MAPPING Procedure .....	16-4
LANG Function .....	16-6
MESSAGE Function .....	16-7
PUBLISH_APPLICATION Procedure.....	16-9
SEED_TRANSLATIONS Procedure .....	16-10
UPDATE_LANGUAGE_MAPPING Procedure.....	16-11
UPDATE_MESSAGE Procedure .....	16-13
UPDATE_TRANSLATED_STRING Procedure.....	16-14
<b>17 APEX_LDAP</b>	
AUTHENTICATE Function .....	17-2
GET_ALL_USER_ATTRIBUTES Procedure .....	17-3
GET_USER_ATTRIBUTES Procedure.....	17-5
IS_MEMBER Function .....	17-7
MEMBER_OF Function.....	17-9
MEMBER_OF2 Function.....	17-10
APEX_LDAP.SEARCH Function.....	17-11
<b>18 APEX_MAIL</b>	
Configuring Oracle Application Express to Send Email.....	18-2
ADD_ATTACHMENT Procedure .....	18-3
GET_IMAGES_URL Function.....	18-4
GET_INSTANCE_URL Function.....	18-5
PUSH_QUEUE Procedure.....	18-6
SEND Procedure.....	18-7
<b>19 APEX_PAGE</b>	
Global Constants .....	19-2
IS_DESKTOP_UI Function .....	19-3
IS_JQM_SMARTPHONE_UI Function .....	19-4
IS_JQM_TABLET_UI Function .....	19-5
GET_UI_TYPE Function .....	19-6
IS_READ_ONLY Function .....	19-7
GET_PAGE_MODE Function.....	19-8
PURGE_CACHE Procedure.....	19-9
GET_URL Function.....	19-10
<b>20 APEX_PLUGIN</b>	
Data Types .....	20-2

GET AJAX IDENTIFIER Function.....	20-10
GET_INPUT_NAME_FOR_PAGE_ITEM Function.....	20-11

## 21 APEX\_PLUGIN\_UTIL

DEBUG_DYNAMIC_ACTION Procedure.....	21-3
DEBUG_PAGE_ITEM Procedure Signature 1.....	21-4
DEBUG_PAGE_ITEM Procedure Signature 2.....	21-5
DEBUG_PROCESS Procedure.....	21-6
DEBUG_REGION Procedure Signature 1.....	21-7
DEBUG_REGION Procedure Signature 2.....	21-8
ESCAPE Function.....	21-9
EXECUTE_PLSQL_CODE Procedure.....	21-10
GET_ATTRIBUTE_AS_NUMBER Function.....	21-11
GET_DATA Function Signature 1.....	21-12
GET_DATA Function Signature 2.....	21-14
GET_DATA2 Function Signature 1.....	21-16
GET_DATA2 Function Signature 2.....	21-18
GET_DISPLAY_DATA Function Signature 1.....	21-20
GET_DISPLAY_DATA Function Signature 2.....	21-22
GET_ELEMENT_ATTRIBUTES Function.....	21-24
GET_PLSQL_EXPRESSION_RESULT Function.....	21-26
GET_PLSQL_FUNCTION_RESULT Function.....	21-27
GET_POSITION_IN_LIST Function.....	21-28
GET_SEARCH_STRING Function.....	21-29
IS_EQUAL Function.....	21-30
PAGE_ITEM_NAMES_TO_JQUERY Function.....	21-31
PRINT_DISPLAY_ONLY Procedure.....	21-32
PRINT_ESCAPED_VALUE Procedure.....	21-33
PRINT_HIDDEN_IF_READONLY Procedure.....	21-34
PRINT_JSON_HTTP_HEADER Procedure.....	21-35
PRINT_LOV_AS_JSON Procedure.....	21-36
PRINT_OPTION Procedure.....	21-37
REPLACE_SUBSTITUTIONS Function.....	21-38

## 22 APEX\_REGION

IS_READ_ONLY Function.....	22-2
PURGE_CACHE Procedure.....	22-3

## 23 APEX\_SPATIAL

Data Types.....	23-2
CHANGE_GEOM_METADATA Procedure.....	23-3
CIRCLE_POLYGON Function.....	23-4
DELETE_GEOM_METADATA Procedure.....	23-5
INSERT_GEOM_METADATA Procedure.....	23-6
INSERT_GEOM_METADATA_LONLAT Procedure.....	23-7
POINT Function.....	23-8

RECTANGLE Function .....	23-9
--------------------------	------

## 24 APEX\_UI\_DEFAULT\_UPDATE

ADD_AD_COLUMN Procedure .....	24-3
ADD_AD_SYNONYM Procedure .....	24-5
DEL_AD_COLUMN Procedure.....	24-6
DEL_AD_SYNONYM Procedure .....	24-7
DEL_COLUMN Procedure .....	24-8
DEL_GROUP Procedure .....	24-9
DEL_TABLE Procedure.....	24-10
SYNCH_TABLE Procedure .....	24-11
UPD_AD_COLUMN Procedure .....	24-12
UPD_AD_SYNONYM Procedure .....	24-14
UPD_COLUMN Procedure .....	24-15
UPD_DISPLAY_IN_FORM Procedure .....	24-17
UPD_DISPLAY_IN_REPORT Procedure .....	24-18
UPD_FORM_REGION_TITLE Procedure .....	24-19
UPD_GROUP Procedure .....	24-20
UPD_ITEM_DISPLAY_HEIGHT Procedure .....	24-21
UPD_ITEM_DISPLAY_WIDTH Procedure .....	24-22
UPD_ITEM_FORMAT_MASK Procedure .....	24-23
UPD_ITEM_HELP Procedure .....	24-24
UPD_LABEL Procedure .....	24-25
UPD_REPORT_ALIGNMENT Procedure.....	24-26
UPD_REPORT_FORMAT_MASK Procedure .....	24-27
UPD_REPORT_REGION_TITLE Procedure .....	24-28
UPD_TABLE Procedure .....	24-29

## 25 APEX\_UTIL

CACHE_GET_DATE_OF_PAGE_CACHE Function .....	25-6
CACHE_GET_DATE_OF_REGION_CACHE Function.....	25-7
CACHE_PURGE_BY_APPLICATION Procedure .....	25-8
CACHE_PURGE_BY_PAGE Procedure.....	25-9
CACHE_PURGE_STALE Procedure .....	25-10
CHANGE_CURRENT_USER_PW Procedure .....	25-11
CHANGE_PASSWORD_ON_FIRST_USE Function.....	25-12
CLOSE_OPEN_DB_LINKS Procedure .....	25-13
CLEAR_APP_CACHE Procedure.....	25-14
CLEAR_PAGE_CACHE Procedure.....	25-15
CLEAR_USER_CACHE Procedure .....	25-16
COUNT_CLICK Procedure .....	25-17
CREATE_USER Procedure .....	25-18
CREATE_USER_GROUP Procedure.....	25-21
CURRENT_USER_IN_GROUP Function.....	25-22
CUSTOM_CALENDAR Procedure .....	25-23
DELETE_USER_GROUP Procedure Signature 1.....	25-24
DELETE_USER_GROUP Procedure Signature 2.....	25-25



DOWNLOAD_PRINT_DOCUMENT Procedure Signature 1 .....	25-26
DOWNLOAD_PRINT_DOCUMENT Procedure Signature 2 .....	25-27
DOWNLOAD_PRINT_DOCUMENT Procedure Signature 3 .....	25-29
DOWNLOAD_PRINT_DOCUMENT Procedure Signature 4 .....	25-31
EDIT_USER Procedure.....	25-32
END_USER_ACCOUNT_DAYS_LEFT Function .....	25-36
EXPIRE_END_USER_ACCOUNT Procedure.....	25-37
EXPIRE_WORKSPACE_ACCOUNT Procedure .....	25-38
EXPORT_USERS Procedure.....	25-39
FETCH_APP_ITEM Function .....	25-40
FETCH_USER Procedure Signature 1 .....	25-41
FETCH_USER Procedure Signature 2 .....	25-44
FETCH_USER Procedure Signature 3 .....	25-46
FIND_SECURITY_GROUP_ID Function.....	25-49
FIND_WORKSPACE Function.....	25-50
GET_ACCOUNT_LOCKED_STATUS Function.....	25-51
GET_ATTRIBUTE Function .....	25-52
GET_AUTHENTICATION_RESULT Function .....	25-53
GET_BLOB_FILE_SRC Function .....	25-54
GET_BUILD_OPTION_STATUS Function Signature 1 .....	25-56
GET_BUILD_OPTION_STATUS Function Signature 2 .....	25-57
GET_CURRENT_USER_ID Function .....	25-58
GET_DEFAULT_SCHEMA Function.....	25-59
GET_EDITION Function .....	25-60
GET_EMAIL Function.....	25-61
GET_FEEDBACK_FOLLOW_UP Function.....	25-62
GET_FILE Procedure .....	25-63
GET_FILE_ID Function.....	25-65
GET_FIRST_NAME Function.....	25-66
GET_GROUPS_USER_BELONGS_TO Function.....	25-67
GET_GROUP_ID Function .....	25-68
GET_GROUP_NAME Function .....	25-69
GET_HASH Function.....	25-70
GET_HIGH_CONTRAST_MODE_TOGGLE Function.....	25-71
GET_LAST_NAME Function.....	25-72
GET_NUMERIC_SESSION_STATE Function.....	25-73
GET_PREFERENCE Function.....	25-74
GET_PRINT_DOCUMENT Function Signature 1 .....	25-75
GET_PRINT_DOCUMENT Function Signature 2 .....	25-76
GET_PRINT_DOCUMENT Function Signature 3 .....	25-77
GET_PRINT_DOCUMENT Function Signature 4 .....	25-78
GET_SCREEN_READER_MODE_TOGGLE Function .....	25-80
GET_SESSION_LANG Function .....	25-81
GET_SESSION_STATE Function.....	25-82
GET_SESSION_TERRITORY Function .....	25-83
GET_SESSION_TIME_ZONE Function.....	25-84
GET_SINCE Function.....	25-85

GET_USER_ID Function .....	25-86
GET_USER_ROLES Function.....	25-87
GET_USERNAME Function .....	25-88
HOST_URL Function .....	25-89
HTML_PCT_GRAPH_MASK Function .....	25-90
INCREMENT_CALENDAR Procedure .....	25-91
IR_CLEAR Procedure [DEPRECATED].....	25-92
IR_DELETE_REPORT Procedure [DEPRECATED].....	25-93
IR_DELETE_SUBSCRIPTION Procedure [DEPRECATED] .....	25-94
IR_FILTER Procedure [DEPRECATED] .....	25-95
IR_RESET Procedure [DEPRECATED] .....	25-97
IS_HIGH_CONTRAST_SESSION Function.....	25-98
IS_HIGH_CONTRAST_SESSION_YN Function.....	25-99
IS_LOGIN_PASSWORD_VALID Function .....	25-100
IS_SCREEN_READER_SESSION Function .....	25-101
IS_SCREEN_READER_SESSION_YN Function .....	25-102
IS_USERNAME_UNIQUE Function .....	25-103
KEYVAL_NUM Function .....	25-104
KEYVAL_VC2 Function.....	25-105
LOCK_ACCOUNT Procedure .....	25-106
PASSWORD_FIRST_USE_OCCURRED Function .....	25-107
PREPARE_URL Function.....	25-108
PUBLIC_CHECK_AUTHORIZATION Function [DEPRECATED].....	25-110
PURGE_REGIONS_BY_APP Procedure.....	25-111
PURGE_REGIONS_BY_NAME Procedure.....	25-112
PURGE_REGIONS_BY_PAGE Procedure .....	25-113
REDIRECT_URL Procedure .....	25-114
REMOVE_PREFERENCE Procedure.....	25-115
REMOVE_SORT_PREFERENCES Procedure .....	25-116
REMOVE_USER Procedure .....	25-117
RESET_AUTHORIZATIONS Procedure [DEPRECATED].....	25-118
RESET_PASSWORD Procedure.....	25-119
RESET_PW Procedure .....	25-120
SAVEKEY_NUM Function.....	25-121
SAVEKEY_VC2 Function .....	25-122
SET_ATTRIBUTE Procedure .....	25-123
SET_AUTHENTICATION_RESULT Procedure.....	25-124
SET_BUILD_OPTION_STATUS Procedure.....	25-125
SET_CURRENT_THEME_STYLE Procedure .....	25-126
SET_CUSTOM_AUTH_STATUS Procedure.....	25-128
SET_EDITION Procedure.....	25-129
SET_EMAIL Procedure .....	25-130
SET_FIRST_NAME Procedure .....	25-131
SET_GROUP_GROUP_GRANTS Procedure .....	25-132
SET_GROUP_USER_GRANTS Procedure .....	25-133
SET_LAST_NAME Procedure .....	25-134
SET_PREFERENCE Procedure .....	25-135

SET_SECURITY_GROUP_ID Procedure .....	25-136
SET_SESSION_HIGH_CONTRAST_OFF Procedure .....	25-138
SET_SESSION_HIGH_CONTRAST_ON Procedure .....	25-139
SET_SESSION_LANG Procedure.....	25-140
SET_SESSION_LIFETIME_SECONDS Procedure .....	25-141
SET_SESSION_MAX_IDLE_SECONDS Procedure .....	25-142
SET_SESSION_SCREEN_READER_OFF Procedure.....	25-143
SET_SESSION_SCREEN_READER_ON Procedure.....	25-144
SET_SESSION_STATE Procedure.....	25-145
SET_SESSION_TERRITORY Procedure.....	25-146
SET_SESSION_TIME_ZONE Procedure .....	25-147
SET_USERNAME Procedure .....	25-148
SHOW_HIGH_CONTRAST_MODE_TOGGLE Procedure.....	25-149
SHOW_SCREEN_READER_MODE_TOGGLE Procedure .....	25-150
STRING_TO_TABLE Function .....	25-151
STRONG_PASSWORD_CHECK Procedure.....	25-152
STRONG_PASSWORD_VALIDATION Function .....	25-156
SUBMIT_FEEDBACK Procedure.....	25-157
SUBMIT_FEEDBACK_FOLLOWUP Procedure .....	25-159
TABLE_TO_STRING Function .....	25-160
UNEXPIRE_END_USER_ACCOUNT Procedure .....	25-161
UNEXPIRE_WORKSPACE_ACCOUNT Procedure.....	25-162
UNLOCK_ACCOUNT Procedure.....	25-163
URL_ENCODE Function .....	25-164
WORKSPACE_ACCOUNT_DAYS_LEFT Function .....	25-166

## 26 APEX\_WEB\_SERVICE

About the APEX_WEB_SERVICE API.....	26-2
Invoking a SOAP Style Web Service.....	26-3
Invoking a RESTful Style Web Service .....	26-5
Retrieving Cookies and HTTP Headers.....	26-6
Setting Cookies and HTTP Headers.....	26-7
BLOB2CLOBBASE64 Function.....	26-8
CLOBBASE642BLOB Function.....	26-9
MAKE_REQUEST Procedure .....	26-10
MAKE_REQUEST Function.....	26-12
MAKE_REST_REQUEST Function .....	26-14
MAKE_REST_REQUEST_B Function.....	26-16
PARSE_RESPONSE Function.....	26-18
PARSE_RESPONSE_CLOB Function .....	26-19
PARSE_XML Function .....	26-20
PARSE_XML_CLOB Function.....	26-22

## 27 APEX\_ZIP

Data Types .....	27-2
ADD_FILE Procedure .....	27-3

<b>FINISH Procedure</b> .....	27-4
<b>GET_FILE_CONTENT Function</b> .....	27-5
<b>GET_FILES Function</b> .....	27-6

## 28 JavaScript APIs

<b>Using the apex namespace</b> .....	28-2
Global Variables .....	28-3
About apex.confirm Function .....	28-4
apex.confirm(pMessage, pRequest) .....	28-5
apex.confirm(pMessage, pOptions) .....	28-6
About apex.submit Function .....	28-7
apex.submit(pOptions) .....	28-8
apex.submit(pRequest) .....	28-9
<b>apex.da namespace</b> .....	28-10
apex.da.resume (pCallback, pErrorOccurred) .....	28-11
<b>apex.debug namespace</b> .....	28-12
Log Level Constants .....	28-13
apex.debug.error(...*) .....	28-14
apex.debug.getLevel() .....	28-15
apex.debug.info(...*) .....	28-16
apex.debug.log(...*) .....	28-17
apex.debug.message(pLevel,...*) .....	28-18
apex.debug.setLevel(pLevel) .....	28-19
apex.debug.trace(...*) .....	28-20
apex.debug.warn(...*) .....	28-21
<b>apex.event namespace</b> .....	28-22
apex.event.trigger(pSelector,pEvent,pData) .....	28-23
<b>apex.item</b> .....	28-24
apex.item( pNd ) .....	28-25
apex.item( pNd ).addValue( pValue ) .....	28-26
apex.item( pNd ).disable() .....	28-27
apex.item( pNd ).enable() .....	28-28
apex.item( pNd ).getValue() .....	28-29
apex.item( pNd ).hide( pHideRow) .....	28-30
apex.item( pNd ).isEmpty() .....	28-31
apex.item( pNd ).setFocus() .....	28-32
apex.item( pNd ).setStyle( pPropertyName, pPropertyValue ) .....	28-33
apex.item( pNd ).setValue(pValue, pDisplayValue, pSuppressChangeEvent) .....	28-34
apex.item( pNd ).show( pShowRow ) .....	28-36
<b>apex.lang namespace</b> .....	28-37
apex.lang.addMessages ( pMessages ) .....	28-38
apex.lang.clearMessages ( pMessages ) .....	28-39
apex.lang.format ( pPattern, ...* ) .....	28-40
apex.lang.formatMessage ( pKey, ...* ) .....	28-41
apex.lang.formatMessageNoEscape ( pKey, ...* ) .....	28-42
apex.lang.formatNoEscape ( pPattern, ...* ) .....	28-43
apex.lang.getMessage ( pKey ) .....	28-44

<b>apex.navigation namespace</b> .....	28-45
apex.navigation.dialog(pUrl,pOptions,pCssClasses,pTriggeringElement) .....	28-46
apex.navigation.dialog.cancel(plsModal).....	28-47
apex.navigation.dialog.close(plsModal,pAction).....	28-48
apex.navigation.dialog.fireCloseHandler(pHandler\$,pAction).....	28-49
apex.navigation.dialog.registerCloseHandler(pOptions) .....	28-50
apex.navigation.openInNewWindow(pUrl,pWindowName,pOptions).....	28-51
apex.navigation.popup(pOptions) .....	28-52
apex.navigation.popup.close(pItem,pValue).....	28-53
apex.navigation.redirect(pUrl).....	28-54
<b>apex.server namespace</b> .....	28-55
apex.server.plugin(pAjaxIdentifier,pData,pOptions).....	28-56
apex.server.pluginUrl( pAjaxIdentifier, pData ) .....	28-62
apex.server.process( pName, pData, pOptions ).....	28-63
apex.server.url( pData ) (pPage) .....	28-66
<b>apex.storage namespace</b> .....	28-67
apex.storage.getCookie(pName).....	28-68
apex.storage.setCookie(pName,pValue) .....	28-69
<b>apex.util namespace</b> .....	28-70
apex.util.showSpinner.....	28-71
apex.util.delayLinger .....	28-72
apex.util.delayLinger.start.....	28-72
apex.util.delayLinger.finish.....	28-72
<b>apex.widget namespace</b> .....	28-74
apex.widget.initPageItem( pName, pOptions) .....	28-75
<b>Non-namespace Javascript APIs</b> .....	28-82
\$(pNd) .....	28-84
\$(pNd) .....	28-85
\$(pNd) .....	28-86
\$(pNd, pValue, pDisplayValue, pSuppressChangeEvent) .....	28-87
\$_Narray(pNd) .....	28-88
\$_Carray(pNd) .....	28-89
\$nv1(pTest, pDefault) .....	28-90
\$_Style(pNd, pStyle, pString).....	28-91
\$_Hide(pNd).....	28-92
\$_Show(pNd).....	28-93
\$_Toggle(pNd).....	28-94
\$_Remove(pNd) .....	28-95
\$_Value(pNd,pValue).....	28-96
\$_UpTill(pNd, pToTag).....	28-97
\$_ItemRow(pNd,pFunc) .....	28-98
\$_HideItemRow(pNd).....	28-99
\$_ShowItemRow(pNd).....	28-100
\$_ToggleItemRow(pNd) .....	28-101
\$_HideAllExcept(pNd,pNdArray) .....	28-102
\$_HideSiblings(pNd).....	28-103
\$_ShowSiblings(pNd).....	28-104

\$x_Class(pNd,pClass).....	28-105
\$x_SetSiblingsClass(pNd, pClass, pNdClass).....	28-106
\$x_ByClass(pClass, pNd, pTag).....	28-107
\$x_ShowAllByClass(pNd, pClass, pTag) .....	28-108
\$x_ShowChildren(pNd) .....	28-109
\$x_HideChildren(pNd) .....	28-110
\$x_disableItem(pNd, pTest) .....	28-111
\$f_get_emptyys(pNd, pClassFail, pClass).....	28-112
\$v_Array(pNd).....	28-113
\$f_ReturnChecked(pNd).....	28-114
\$d_ClearAndHide(pNd) .....	28-115
\$f_SelectedOptions(pNd).....	28-116
\$f_SelectValue(pNd).....	28-117
\$u_ArrayToString(pArray, pDelim).....	28-118
\$x_CheckImageSrc(pId,pSearch) .....	28-119
\$v_CheckValueAgainst(pThis, pValue).....	28-120
\$f_Hide_On_Value_Item(pThis, pThat, pValue).....	28-121
\$f_Show_On_Value_Item(pThis, pThat, pValue) .....	28-122
\$f_Hide_On_Value_Item_Row(pThis, pThat, pValue) .....	28-123
\$f_Show_On_Value_Item_Row(pThis, pThat, pValue) .....	28-124
\$f_DisableOnValue(pThis, pValue, pThat) .....	28-125
\$x_ClassByClass(pNd, pClass, pTag, pClass2).....	28-126
\$f_ValuesToArray(pThis, pClass, pTag).....	28-127
\$x_FormItems(pNd, pType) .....	28-128
\$f_CheckAll(pThis, pCheck, pArray).....	28-129
\$f_CheckFirstColumn(pNd).....	28-130
\$x_ToggleWithImage(pThis,pNd).....	28-131
\$x_SwitchImageSrc(pNd, pSearch, pReplace).....	28-132
\$x_CheckImageSrc(pNd, pSearch) .....	28-133
\$u_SubString(pText,pMatch) .....	28-134
html_RemoveAllChildren(pNd).....	28-135
html_SetSelectValue(pId,pValue).....	28-136
addLoadEvent(pFunction).....	28-137
\$f_Swap(pThis,pThat) .....	28-138
\$f_SetValueSequence(pArray,pMultiple).....	28-139
\$dom_AddTag(pThis, pTag, pText).....	28-140
\$tr_AddTD(pThis,pText) .....	28-141
\$tr_AddTH(pThis,pText) .....	28-142
\$dom_AddInput(pThis,pType,pId,pName,pValue) .....	28-143
\$dom_MakeParent(p_Node,p_Parent) .....	28-144
\$x_RowHighlight(pThis, pColor) .....	28-145
\$x_RowHighlightOff(pThis).....	28-146
\$v_Upper(pNd) .....	28-147
\$d_Find(pThis,pString,pTags,pClass).....	28-148
\$f_First_field(pNd).....	28-149
<b>Legacy JavaScript APIs</b> .....	28-150
\$v_PopupReturn(pValue, pThat) [Deprecated] .....	28-151

\$v\_IsEmpty(pThis) [Deprecated]..... 28-152  
submitEnter(pNd,e) [Deprecated] ..... 28-153  
setReturn(p\_R,p\_D) [Deprecated] ..... 28-154  
GetCookie (pName) [Deprecated] ..... 28-155  
SetCookie (pName,pValue) [Deprecated] ..... 28-156

**Index**





---

---

# Preface

*Oracle Application Express API Reference* describes the Application Programming Interfaces, referred to as APIs, available when programming in the Oracle Application Express environment.

- [Audience](#)
- [Documentation Accessibility](#)
- [Related Documents](#)
- [Conventions](#)

## Audience

*Oracle Application Express API Reference* is intended for application developers who are building database-centric web applications using Oracle Application Express. The guide describes the APIs available when programming in the Oracle Application Express environment.

To use this guide, you need to have a general understanding of relational database concepts and an understanding of the operating system environment under which you are running Oracle Application Express.

**See Also:** *Oracle Application Express Application Builder User's Guide*

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

### Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

## Related Documents

For more information, see these Oracle resources:

- *Oracle Application Express Release Notes*

- *Oracle Application Express Installation Guide*
- *Oracle Application Express Application Builder User's Guide*
- *Oracle Application Express Administration Guide*
- *Oracle Application Express Migration Guide*
- *Oracle Application Express SQL Workshop Guide*
- *Oracle Application Express End User's Guide*
- *Oracle Database Concepts*
- *Oracle Database Advanced Application Developer's Guide*
- *Oracle Database Administrator's Guide*
- *Oracle Database SQL Language Reference*
- *SQL\*Plus User's Guide and Reference*
- *Oracle Database PL/SQL Language Reference*

## Conventions

For a description of PL/SQL subprogram conventions, refer to the *Oracle Database PL/SQL Language Reference*. This document contains the following information:

- Specifying subprogram parameter modes
- Specifying default values for subprogram parameters
- Overloading PL/SQL subprogram Names

The following text conventions are used in this document:

<b>Convention</b>	<b>Meaning</b>
<b>boldface</b>	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

---

---

# Changes in This Release

This preface contains:

- [Changes in Oracle Application Express Release 5.0](#)

## Changes in Oracle Application Express Release 5.0

The following are changes in *Oracle Application Express API Reference* for Oracle Application Express release 5.0.

### New Features

The following features are new in this release:

- **APEX\_AUTHORIZATION (New)**  
This is a new package containing public utility functions used for controlling and querying access rights to the application.
- **APEX\_ERROR (Updates)**  
New `error_statement` attribute added to the `T_ERROR` record type.
- **APEX\_ESCAPE (Updates)**  
This function escapes characters that can change the context in a regular expression. It should be used to secure user input.
- **APEX\_JSON (New)**  
This is a new package that includes utilities to parse and generate JSON.
- **APEX\_INSTANCE\_ADMIN (Updates)**
  - Available parameters updated to include new `HTTP_RESPONSE_HEADERS` and `HTTP_STS_MAX_AGE`.
  - `FREE_WORKSPACE_APP_IDS` Procedure (New) - This procedure removes the reservation of application IDs for a given workspace ID.
  - `RESERVE_WORKSPACE_APP_IDS` Procedure (New) - This procedure permanently reserves the IDs of websheet and database applications in a given workspace.
  - Available parameters updated to include new `PASSWORD_HASH_FUNCTION` and `CHECKSUM_HASH_FUNCTION`.
  - New instance preference `APEX_REST_PATH_PREFIX`. added to controls the URI path prefix used to access built-in RESTful Services exposed by Application Express.

- New instance administrator APIs added for schema restriction: `CREATE_SCHEMA_EXCEPTION`, `REMOVE_SCHEMA_EXCEPTION`, `REMOVE_SCHEMA_EXCEPTIONS`, `REMOVE_WORKSPACE_EXCEPTIONS`, `RESTRICT_SCHEMA`, `UNRESTRICT_SCHEMA`.
- APEX\_IR (Updates)
 

New `CHANGE_REPORT_OWNER` procedure added to change the owner of a saved interactive report using a report ID.
- APEX\_PLUGIN (Updates)
 

New `t_authorization` and `t_authorization_exec_result` added.
- APEX\_SPATIAL (New)
 

This is a new package enabling you to use oracle locator and the spatial option within Oracle Application Express.
- APEX\_ZIP (New)
 

This is a new package to manage the zipping and unzipping of files.
- APEX\_UTIL (Updated)
  - `SET_GROUP_GROUP_GRANTS` Procedure (NEW)
 

This procedure modifies the group grants for a given group.
  - `SET_GROUP_USER_GRANTS` Procedure (NEW)
 

This procedure modifies the group grants for a given user.
  - `RESET_AUTHORIZATIONS` Procedure (DEPRECATED)
 

Use this procedure to undo caching, requiring each authorization scheme be revalidated when it is next encountered during page show or accept processing.

Use the "[RESET\\_CACHE Procedure](#)" on page 4-4 instead of this deprecated procedure.
  - `PUBLIC_CHECK_AUTHORIZATIONS` Function (DEPRECATED)
 

Given the name of a security scheme, this function determines if the current user passes the security check.

Use the "[IS\\_AUTHORIZED Function](#)" on page 4-3 instead of this deprecated function.
  - `RESET_PASSWORD` Procedure (New)
 

Use this procedure to change the password of a given user name for the current workspace.
- APEX\_PLUGIN\_UTIL (Updates)
  - `GET_ATTRIBUTE_AS_NUMBER` Function (New)
 

This function returns the value of a plug-in attribute as a number, taking into account NLS decimal separator effective for the current database session. Use this function in plug-in PL/SQL source for custom attributes of type `NUMBER` instead of the built-in `to_number` function.
- JavaScript APIs (Updates)
  - `apex.debug` namespace (New)
 

This namespace is used to store all debug functions of Oracle Application Express.

- apex.lang namespace (New)

This namespace is used for localization related functions of Oracle Application Express.

- apex.server namespace (Updated)

The `apex.server.url` function added to return the URL to issue a GET request to the current page.

- apex.navigation namespace (New)

The `apex.navigation` namespace contains popup and redirect related functions of Oracle Application Express, and also contains utility functions for handling dialog pages in an application.

### **Deprecated and Desupported Features**

See "Deprecated Features" and "Desupported Features" in *Oracle Application Express Release Notes*.



---

---

# APEX\_APPLICATION

The `APEX_APPLICATION` package is a PL/SQL package that implements the Oracle Application Express rendering engine. You can use this package to take advantage of many global variables. [Table 1-1](#) describes the global variables available in the `APEX_APPLICATION` package.

- [Global Variables](#)
- [Referencing Arrays](#)
- [Referencing Values Within an On Submit Process](#)
- [Converting an Array to a Single Value](#)
- [HELP Procedure](#)
- [STOP\\_APEX\\_ENGINE Procedure](#)

---

## Global Variables

**Table 1–1 Global Variables Available in APEX\_APPLICATION**

Global Variable	Description
G_USER	Specifies the currently logged in user.
G_FLOW_ID	Specifies the ID of the currently running application.
G_FLOW_STEP_ID	Specifies the ID of the currently running page.
G_FLOW_OWNER	Specifies the schema to parse for the currently running application.
G_REQUEST	Specifies the value of the request variable most recently passed to or set within the show or accept modules.
G_BROWSER_LANGUAGE	Refers to the web browser's current language preference.
G_DEBUG	Refers to whether debugging is currently switched on or off. Valid values for the DEBUG flag are 'Yes' or 'No'. Turning debug on shows details about application processing.
G_HOME_LINK	Refers to the home page of an application. The Application Express engine redirects to this location if no page is given and if no alternative page is dictated by the authentication scheme's logic.
G_LOGIN_URL	Used to display a link to a login page for users that are not currently logged in.
G_IMAGE_PREFIX	Refers to the virtual path the web server uses to point to the images directory distributed with Oracle Application Express.
G_FLOW_SCHEMA_OWNER	Refers to the owner of the Application Express schema.
G_PRINTER_FRIENDLY	Refers to whether the Application Express engine is running in print view mode. This setting can be referenced in conditions to eliminate elements not desired in a printed document from a page.
G_PROXY_SERVER	Refers to the application attribute 'Proxy Server'.
G_SYSDATE	Refers to the current date on the database server. this uses the DATE DATATYPE.
G_PUBLIC_USER	Refers to the Oracle schema used to connect to the database through the database access descriptor (DAD).
G_GLOBAL_NOTIFICATION	Specifies the application's global notification attribute.
G_X01, ... G_X10	Specifies the values of the X01, ... X10 variables most recently passed to or set within the show or accept modules. You typically use these variables in On Demand AJAX processes.  <b>See Also:</b> APP_AJAX_X01, ... APP_AJAX_X10 in <i>Oracle Application Express Application Builder User's Guide</i>



## Referencing Arrays

Items are typically HTML form elements such as text fields, select lists, and check boxes. When you create a new form item using a wizard, the wizard uses a standard naming format. The naming format provides a handle so you can retrieve the value of the item later on.

To create your own items, you can access them after a page is submitted by referencing APEX\_APPLICATION.G\_F01 to APEX\_APPLICATION.G\_F50 arrays. You can create your own HTML form fields by providing the input parameters using the format F01, F02, F03 and so on. You can create up to 50 input parameters ranging from F01 to F50, for example:

```
<INPUT TYPE="text" NAME="F01" SIZE="32" MAXLENGTH="32" VALUE="some value">

<TEXTAREA NAME="F02" ROWS=4 COLS=90 WRAP="VIRTUAL">this is the example of a text
area.</TEXTAREA>

<SELECT NAME="F03" SIZE="1">
<OPTION VALUE="abc">abc
<OPTION VALUE="123">123
</SELECT>
```

Because the F01 to F50 input items are declared as PL/SQL arrays, you can have multiple items named the same value. For example:

```
<INPUT TYPE="text" NAME="F01" SIZE="32" MAXLENGTH="32" VALUE="array element 1">
<INPUT TYPE="text" NAME="F01" SIZE="32" MAXLENGTH="32" VALUE="array element 2">
<INPUT TYPE="text" NAME="F01" SIZE="32" MAXLENGTH="32" VALUE="array element 3">
```

Note that following PL/SQL code produces the same HTML as show in the previous example.

```
FOR i IN 1..3 LOOP
APEX_ITEM.TEXT(P_IDX      => 1,
  p_value      =>'array element '||i ,
  p_size       =>32,
  p_maxlength  =>32);
END LOOP;
```

---

## Referencing Values Within an On Submit Process

You can reference the values posted by an HTML form using the PL/SQL variable `APEX_APPLICATION.G_F01` to `APEX_APPLICATION.G_F50`. Because this element is an array, you can reference values directly, for example:

```
FOR i IN 1..APEX_APPLICATION.G_F01.COUNT LOOP
    http.p('element '||I||' has a value of '||APEX_APPLICATION.G_F01(i));
END LOOP;
```

Note that check boxes displayed using `APEX_ITEM.CHECKBOX` only contain values in the `APEX_APPLICATION` arrays for those rows which are checked. Unlike other items (`TEXT`, `TEXTAREA`, and `DATE_POPUP`) which can contain an entry in the corresponding `APEX_APPLICATION` array for every row submitted, a check box only has an entry in the `APEX_APPLICATION` array if it is selected.

## Converting an Array to a Single Value

You can also use Oracle Application Express public utility functions to convert an array into a single value. The resulting string value is a colon-separated list of the array element values. For example:

```
http.p(APEX_UTIL.TABLE_TO_STRING(APEX_APPLICATION.G_F01));
```

This function enables you to reference G\_F01 to G\_F50 values in an application process that performs actions on data. The following sample process demonstrates how values are inserted into a table:

```
INSERT INTO my_table (my_column) VALUES  
APEX_UTIL.TABLE_TO_STRING(APEX_APPLICATION.G_F01)
```

## HELP Procedure

This function outputs page and item level help text as formatted HTML. You can also use it to customize how help information is displayed in your application.

### Syntax

```
APEX_APPLICATION.HELP (
  p_request          IN VARCHAR2 DEFAULT NULL,
  p_flow_id          IN VARCHAR2 DEFAULT NULL,
  p_flow_step_id    IN VARCHAR2 DEFAULT NULL,
  p_show_item_help  IN VARCHAR2 DEFAULT 'YES',
  p_show_regions    IN VARCHAR2 DEFAULT 'YES',
  p_before_page_html IN VARCHAR2 DEFAULT '<p>',
  p_after_page_html IN VARCHAR2 DEFAULT NULL,
  p_before_region_html IN VARCHAR2 DEFAULT NULL,
  p_after_region_html IN VARCHAR2 DEFAULT '</td></tr></table></p>',
  p_before_prompt_html IN VARCHAR2 DEFAULT '<p><b>',
  p_after_prompt_html IN VARCHAR2 DEFAULT '</b></p>:&nbspnbsp; ',
  p_before_item_html IN VARCHAR2 DEFAULT NULL,
  p_after_item_html IN VARCHAR2 DEFAULT NULL);
```

### Parameters

Table 1–2 describes the parameters available in the HELP procedure.

**Table 1–2 HELP Parameters**

Parameter	Description
p_request	Not used.
p_flow_id	The application ID that contains the page or item level help you want to output.
p_flow_step_id	The page ID that contains the page or item level help you want to display.
p_show_item_help	Flag to determine if item level help is output. If this parameter is supplied, the value must be either 'YES' or 'NO', if not the default value is 'YES'.
p_show_regions	Flag to determine if region headers are output (for regions containing page items). If this parameter is supplied, the value must be either 'YES' or 'NO', if not the default value is 'YES'.
p_before_page_html	Use this parameter to include HTML between the page level help text and item level help text.
p_after_page_html	Use this parameter to include HTML at the bottom of the output, after all other help.
p_before_region_html	Use this parameter to include HTML before every region section. Note this parameter is ignored if p_show_regions is set to 'NO'.
p_after_region_html	Use this parameter to include HTML after every region section. Note this parameter is ignored if p_show_regions is set to 'NO'.
p_before_prompt_html	Use this parameter to include HTML before every item label for item level help. Note this parameter is ignored if p_show_item_help is set to 'NO'.

**Table 1–2 (Cont.) HELP Parameters**

Parameter	Description
p_after_prompt_html	Use this parameter to include HTML after every item label for item level help. Note this parameter is ignored if p_show_item_help is set to 'NO'.
p_before_item_html	Use this parameter to include HTML before every item help text for item level help. Note this parameter is ignored if p_show_item_help is set to 'NO'.
p_after_item_html	Use this parameter to include HTML after every item help text for item level help. Note this parameter is ignored if p_show_item_help is set to 'NO'.

**Example**

The following example shows how to use the `APEX_APPLICATION.HELP` procedure to customize how help information is displayed.

In this example, the `p_flow_step_id` parameter is set to `:REQUEST`, which means that a page ID specified in the `REQUEST` section of the URL controls which page's help information to display (see note after example for full details on how this can be achieved).

Also, the help display has been customized so that the region sub-header now has a different color (through the `p_before_region_html` parameter) and also the `!` has been removed that appeared by default after every item prompt (through the `p_after_prompt_html` parameter).

```
APEX_APPLICATION.HELP(
  p_flow_id => :APP_ID,
  p_flow_step_id => :REQUEST,
  p_before_region_html => '<p><br/><table bgcolor="#A3BED8"
width="100%"><tr><td><b>',
  p_after_prompt_html => '</b></p>&nbsp;&nbsp;&nbsp;');
```

To implement this type of call in your application, you can do the following:

1. Create a page that will be your application help page.
2. Create a region of type 'PL/SQL Dynamic Content' and add the `APEX_APPLICATION.HELP` call as PL/SQL Source.
3. Then you can add a 'Navigation Bar' link to this page, ensuring that the `REQUEST` value set in the link is `&APP_PAGE_ID`.

## STOP\_APEX\_ENGINE Procedure

This procedure signals the Application Express engine to stop further processing and immediately exit to avoid adding additional HTML code to the HTTP buffer.

---

---

**Note:** This procedure raises the exception `apex_application.e_stop_apex_engine` internally. You must raise that exception again, if you use a `WHEN OTHERS` exception handler.

---

---

### Syntax

```
APEX_APPLICATION.STOP_APEX_ENGINE
```

### Parameters

None

### Example 1

This example tells the browser to redirect to `http://apex.oracle.com/` and immediately stops further processing.

```
owa_util.redirect_url('http://apex.oracle.com');
apex_application.stop_apex_engine;
```

### Example 2

This example also tells the browser to redirect to `http://apex.oracle.com/` and immediately stops further processing. But, this time the code also contains a `WHEN OTHERS` exception handler which deals with the `apex_application.e_stop_apex_engine` used by `apex_application.stop_apex_engine`.

```
begin
    ... code which can raise an exception ...
    owa_util.redirect_url('http://apex.oracle.com');
    apex_application.stop_apex_engine;
exception
    when apex_application.e_stop_apex_engine then
        raise; -- raise again the stop Application Express engine exception
    when others then
        ...; -- code to handle the exception
end;
```

---

---

## APEX\_APPLICATION\_INSTALL

The APEX\_APPLICATION\_INSTALL package provides many methods to modify application attributes during the Application Express application installation process.

- [Package Overview](#)
- [Attributes Manipulated by APEX\\_APPLICATION\\_INSTALL](#)
- [Import Script Examples](#)
- [CLEAR\\_ALL Procedure](#)
- [GENERATE\\_APPLICATION\\_ID Procedure](#)
- [GENERATE\\_OFFSET Procedure](#)
- [GET\\_APPLICATION\\_ALIAS Function](#)
- [GET\\_APPLICATION\\_ID Function](#)
- [GET\\_APPLICATION\\_NAME Function](#)
- [GET\\_AUTO\\_INSTALL\\_SUP\\_OBJ Function](#)
- [GET\\_IMAGE\\_PREFIX Function](#)
- [GET\\_OFFSET Function](#)
- [GET\\_PROXY Function](#)
- [GET\\_SCHEMA Function](#)
- [GET\\_WORKSPACE\\_ID Function](#)
- [SET\\_APPLICATION\\_ALIAS Procedure](#)
- [SET\\_APPLICATION\\_ID Procedure](#)
- [SET\\_APPLICATION\\_NAME Procedure](#)
- [SET\\_AUTO\\_INSTALL\\_SUP\\_OBJ Procedure](#)
- [SET\\_IMAGE\\_PREFIX Procedure](#)
- [SET\\_OFFSET Procedure](#)
- [SET\\_PROXY Procedure](#)
- [SET\\_SCHEMA Procedure](#)
- [SET\\_WORKSPACE\\_ID Procedure](#)

## Package Overview

Oracle Application Express provides two ways to import an application into an Application Express instance:

1. Upload and installation of an application export file by using the web interface of Application Express.
2. Execution of the application export file as a SQL script, typically in the command-line utility SQL\*Plus.

Using the file upload capability of the web interface of Application Express, developers can import an application with a different application ID, different workspace ID and different parsing schema. But when importing an application by using a command-line tool like SQL\*Plus, none of these attributes (application ID, workspace ID, parsing schema) can be changed without directly modifying the application export file.

To view the install log, enter the following from the command-line tool, so the server outputs are displayed:

```
set serveroutput on unlimited
```

As more and more Application Express customers create applications which are meant to be deployed by using command-line utilities or by using a non-web-based installer, they are faced with this challenge of how to import their application into an arbitrary workspace on any Application Express instance.

Another common scenario is in a training class when installing an application into 50 different workspaces that all use the same application export file. Today, customers work around this by adding their own global variables to an application export file and then varying the values of these globals at installation time. However, this manual modification of the application export file (usually done with a post-export sed or awk script) should not be necessary.

Application Express 4.0 and higher includes the APEX\_APPLICATION\_INSTALL API. This PL/SQL API provides many methods to set application attributes during the Application Express application installation process. All export files in Application Express 4.0 and higher contain references to the values set by the APEX\_APPLICATION\_INSTALL API. However, the methods in this API is only used to override the default application installation behavior.



## Attributes Manipulated by APEX\_APPLICATION\_INSTALL

The table below lists the attributes that can be set by functions in this API.

**Table 2–1** *Attributes Manipulated by the APEX\_APPLICATION\_INSTALL API*

Attribute	Description
Workspace ID	Workspace ID of the imported application. See <a href="#">GET_WORKSPACE_ID Function</a> , <a href="#">SET_WORKSPACE_ID Procedure</a> .
Application ID	Application ID of the imported application. See <a href="#">GENERATE_APPLICATION_ID Procedure</a> , <a href="#">GET_APPLICATION_ID Function</a> , <a href="#">SET_APPLICATION_ID Procedure</a> .
Offset	Offset value used during application import. See <a href="#">GENERATE_OFFSET Procedure</a> , <a href="#">GET_OFFSET Function</a> , <a href="#">SET_OFFSET Procedure</a> .
Schema	The parsing schema ("owner") of the imported application. See <a href="#">GET_SCHEMA Function</a> , <a href="#">SET_SCHEMA Procedure</a> .
Name	Application name of the imported application. See <a href="#">GET_APPLICATION_NAME Function</a> , <a href="#">SET_APPLICATION_NAME Procedure</a> .
Alias	Application alias of the imported application. See <a href="#">GET_APPLICATION_ALIAS Function</a> , <a href="#">SET_APPLICATION_ALIAS Procedure</a> .
Image Prefix	The image prefix of the imported application. See <a href="#">GET_IMAGE_PREFIX Function</a> , <a href="#">SET_IMAGE_PREFIX Procedure</a> .
Proxy	The proxy server attributes of the imported application. See <a href="#">GET_PROXY Function</a> , <a href="#">SET_PROXY Procedure</a> .

## Import Script Examples

Using the workspace FRED\_DEV on the development instance, you generate an application export of application 645 and save it as file f645.sql. All examples in this section assume you are connected to SQL\*Plus.

### Import Application without Modification

To import this application back into the FRED\_DEV workspace on the same development instance using the same application ID:

```
@f645.sql
```

### Import Application with Specified Application ID

To import this application back into the FRED\_DEV workspace on the same development instance, but using application ID 702:

```
begin
  apex_application_install.set_application_id( 702);
  apex_application_install.generate_offset;
  apex_application_install.set_application_alias( 'F' || apex_application.get_
application_id );
end;
/
```

```
@645.sql
```

### Import Application with Generated Application ID

To import this application back into the FRED\_DEV workspace on the same development instance, but using an available application ID generated by Application Express:

```
begin
  apex_application_install.generate_application_id;
  apex_application_install.generate_offset;
  apex_application_install.set_application_alias( 'F' || apex_application.get_
application_id );
end;
/
```

```
@f645.sql
```

### Import Application into Different Workspace using Different Schema

To import this application into the FRED\_PROD workspace on the production instance, using schema FREDDY, and the workspace ID of FRED\_DEV and FRED\_PROD are different:

```
declare
  l_workspace_id number;
begin
```

```

select workspace_id into l_workspace_id
  from apex_workspaces
  where workspace = 'FRED_PROD';
--
apex_application_install.set_workspace_id( l_workspace_id );
apex_application_install.generate_offset;
apex_application_install.set_schema( 'FREDDY' );
apex_application_install.set_application_alias( 'FREDPROD_APP' );
end;
/

@f645.sql

```

### Import into Training Instance for Three Different Workspaces

To import this application into the Training instance for 3 different workspaces:

```

declare
  l_workspace_id number;
begin
  select workspace_id into l_workspace_id
    from apex_workspaces
    where workspace = 'TRAINING1';
  --
  apex_application_install.set_workspace_id( l_workspace_id );
  apex_application_install.generate_application_id;
  apex_application_install.generate_offset;
  apex_application_install.set_schema( 'STUDENT1' );
  apex_application_install.set_application_alias( 'F' || apex_application.get_
application_id );
end;
/

@f645.sql

```

```

declare
  l_workspace_id number;
begin
  select workspace_id into l_workspace_id
    from apex_workspaces
    where workspace = 'TRAINING2';
  --
  apex_application_install.set_workspace_id( l_workspace_id );
  apex_application_install.generate_application_id;
  apex_application_install.generate_offset;
  apex_application_install.set_schema( 'STUDENT2' );
  apex_application_install.set_application_alias( 'F' || apex_application.get_
application_id );
end;
/

@f645.sql

```

```

declare
  l_workspace_id number;
begin
  select workspace_id into l_workspace_id
    from apex_workspaces
    where workspace = 'TRAINING3';
  --

```

```
apex_application_install.set_workspace_id( l_workspace_id );
apex_application_install.generate_application_id;
apex_application_install.generate_offset;
apex_application_install.set_schema( 'STUDENT3' );
apex_application_install.set_application_alias( 'F' || apex_application.get_
application_id );
end;
/

@f645.sql
```

## CLEAR\_ALL Procedure

This procedure clears all values currently maintained in the APEX\_APPLICATION\_INSTALL package.

### Syntax

```
APEX_APPLICATION_INSTALL.CLEAR_ALL;
```

### Parameters

None.

### Example

The following example clears all values currently set by the APEX\_APPLICATION\_INSTALL package.

```
begin
    apex_application_install.clear_all;
end;
```

## GENERATE\_APPLICATION\_ID Procedure

This procedure generates an available application ID on the instance and sets the application ID in APEX\_APPLICATION\_INSTALL.

### Syntax

```
APEX_APPLICATION_INSTALL.GENERATE_APPLICATION_ID;
```

### Parameters

None.

### Example

For an example of this procedure call, see ["Import Application with Generated Application ID"](#) on page 2-4 and [Import into Training Instance for Three Different Workspaces](#) on page 2-5.

**See Also:** ["SET\\_APPLICATION\\_ID Procedure"](#) on page 2-20, ["GET\\_APPLICATION\\_ID Function"](#) on page 2-11

---

## GENERATE\_OFFSET Procedure

This procedure generates the offset value used during application import. Use the offset value to ensure that the metadata for the Application Express application definition does not collide with other metadata on the instance. For a new application installation, it is usually sufficient to call this procedure to have Application Express generate this offset value for you.

### Syntax

```
APEX_APPLICATION_INSTALL.GENERATE_OFFSET;
```

### Parameters

None.

### Example

For examples of this procedure call, see ["Import Application with Specified Application ID"](#) on page 2-4, ["Import Application with Generated Application ID"](#) on page 2-4, and ["Import into Training Instance for Three Different Workspaces"](#) on page 2-5.

**See Also:** ["GET\\_OFFSET Function"](#) on page 2-15, ["SET\\_OFFSET Procedure"](#) on page 2-24

## GET\_APPLICATION\_ALIAS Function

This function gets the application alias for the application to be imported. This is only used if the application to be imported has an alias specified. An application alias must be unique within a workspace and it is recommended to be unique within an instance.

### Syntax

```
APEX_APPLICATION_INSTALL.GET_APPLICATION_ALIAS  
RETURN VARCHAR2;
```

### Parameters

None.

### Example

The following example returns the value of the application alias value in the APEX\_APPLICATION\_INSTALL package. The application alias cannot be more than 255 characters.

```
declare  
    l_alias varchar2(255);  
begin  
    l_alias := apex_application_install.get_application_alias;  
end;
```

**See Also:** ["SET\\_APPLICATION\\_ALIAS Procedure"](#) on page 2-19



## GET\_APPLICATION\_ID Function

Use this function to get the application ID of the application to be imported. The application ID should either not exist in the instance or, if it does exist, must be in the workspace where the application is being imported to.

### Syntax

```
APEX_APPLICATION_INSTALL.GET_APPLICATION_ID  
RETURN NUMBER;
```

### Parameters

None.

### Example

The following example returns the value of the application ID value in the APEX\_APPLICATION\_INSTALL package.

```
declare  
    l_id number;  
begin  
    l_id := apex_application_install.get_application_id;  
end;
```

**See Also:** ["SET\\_APPLICATION\\_ID Procedure"](#) on page 2-20,  
["GENERATE\\_APPLICATION\\_ID Procedure"](#) on page 2-8

## GET\_APPLICATION\_NAME Function

This function gets the application name of the import application.

### Syntax

```
APEX_APPLICATION_INSTALL.GET_APPLICATION_NAME  
RETURN VARCHAR2;
```

### Parameters

None.

### Example

The following example returns the value of the application name value in the APEX\_APPLICATION\_INSTALL package.

```
declare  
    l_application_name varchar2(255);  
begin  
    l_application_name := apex_application_install.get_application_name;  
end;
```

**See Also:** ["SET\\_APPLICATION\\_NAME Procedure"](#) on page 2-21

---

## GET\_AUTO\_INSTALL\_SUP\_OBJ Function

Use this function to get the automatic install of supporting objects setting used during the import of an application. This setting is valid only for command line installs. If the setting is set to TRUE and the application export contains supporting objects, it automatically installs or upgrades the supporting objects when an application imports from the command line.

### Syntax

```
APEX_APPLICATION_INSTALL.GET_AUTO_INSTALL_SUP_OBJ  
RETURN BOOLEAN;
```

### Parameters

None.

### Example

The following example returns the value of automatic install of supporting objects setting in the APEX\_APPLICATION\_INSTALL package.

```
declare  
    l_auto_install_sup_obj boolean;  
begin  
    l_auto_install_sup_obj := apex_application_install.get_auto_install_sup_obj;  
end;
```

## GET\_IMAGE\_PREFIX Function

This function gets the image prefix of the import application. Most Application Express instances use the default image prefix of /i/.

### Syntax

```
APEX_APPLICATION_INSTALL.GET_IMAGE_PREFIX  
RETURN VARCHAR2;
```

### Parameters

None.

### Example

The following example returns the value of the application image prefix in the APEX\_APPLICATION\_INSTALL package. The application image prefix cannot be more than 255 characters.

```
declare  
    l_image_prefix varchar2(255);  
begin  
    l_image_prefix := apex_application_install.get_image_prefix;  
end;
```

**See Also:** ["SET\\_IMAGE\\_PREFIX Procedure"](#) on page 2-23

## GET\_OFFSET Function

Use function to get the offset value used during the import of an application.

### Syntax

```
APEX_APPLICATION_INSTALL.GET_OFFSET  
RETURN NUMBER;
```

### Parameters

None.

### Example

The following example returns the value of the application offset value in the APEX\_APPLICATION\_INSTALL package.

```
declare  
    l_offset number;  
begin  
    l_offset := apex_application_install.get_offset;  
end;
```

**See Also:** ["SET\\_OFFSET Procedure"](#) on page 2-24, ["GENERATE\\_OFFSET Procedure"](#) on page 2-9

## GET\_PROXY Function

Use this function to get the proxy server attribute of an application to be imported.

### Syntax

```
APEX_APPLICATION_INSTALL.GET_PROXY  
RETURN VARCHAR2;
```

### Parameters

None.

### Example

The following example returns the value of the proxy server attribute in the APEX\_APPLICATION\_INSTALL package. The proxy server attribute cannot be more than 255 characters.

```
declare  
    l_proxy varchar2(255);  
begin  
    l_proxy := apex_application_install.get_proxy;  
end;
```

**See Also:** ["SET\\_PROXY Procedure"](#) on page 2-25

## GET\_SCHEMA Function

Use this function to get the parsing schema ("owner") of the Application Express application.

### Syntax

```
APEX_APPLICATION_INSTALL.GET_SCHEMA  
RETURN VARCHAR2;
```

### Parameters

None.

### Example

The following example returns the value of the application schema in the APEX\_APPLICATION\_INSTALL package.

```
declare  
    l_schema varchar2(30);  
begin  
    l_schema := apex_application_install.get_schema;  
end;
```

**See Also:** ["SET\\_SCHEMA Procedure"](#) on page 2-26

## GET\_WORKSPACE\_ID Function

Use this function to get the workspace ID for the application to be imported.

### Syntax

```
APEX_APPLICATION_INSTALL.GET_WORKSPACE_ID  
RETURN NUMBER;
```

### Parameters

None.

### Example

The following example returns the value of the workspace ID value in the APEX\_APPLICATION\_INSTALL package.

```
declare  
    l_workspace_id number;  
begin  
    l_workspace_id := apex_application_install.get_workspace_id;  
end;
```

**See Also:** ["SET\\_WORKSPACE\\_ID Procedure"](#) on page 2-27



---

## SET\_APPLICATION\_ALIAS Procedure

This procedure sets the application alias for the application to be imported. This is only used if the application to be imported has an alias specified. An application alias must be unique within a workspace and it is recommended to be unique within an instance.

### Syntax

```
APEX_APPLICATION_INSTALL.SET_APPLICATION_ALIAS (
    p_application_alias IN VARCHAR2);
```

### Parameters

Table 2–2 describes the parameters available in SET\_APPLICATION\_ALIAS procedure.

**Table 2–2 SET\_APPLICATION\_ALIAS Parameters**

Parameter	Description
p_application_alias	The application alias. The application alias is an alphanumeric identifier. It cannot exceed 255 characters, must be unique within a workspace and, ideally, is unique within an entire instance.

### Example

For examples of this procedure call, see ["Import Application with Specified Application ID"](#) on page 2-4, ["Import Application with Generated Application ID"](#) on page 2-4, ["Import Application into Different Workspace using Different Schema"](#) on page 2-4 and ["Import into Training Instance for Three Different Workspaces"](#) on page 2-5.

**See Also:** ["GET\\_APPLICATION\\_ALIAS Function"](#) on page 2-10

---

## SET\_APPLICATION\_ID Procedure

Use this procedure to set the application ID of the application to be imported. The application ID should either not exist in the instance or, if it does exist, must be in the workspace where the application is being imported to. This number must be a positive integer and must not be from the reserved range of Application Express application IDs.

### Syntax

```
APEX_APPLICATION_INSTALL.SET_APPLICATION_ID (  
    p_application_id IN NUMBER);
```

### Parameters

[Table 2-3](#) describes the parameters available in SET\_APPLICATION\_ID procedure.

**Table 2-3** SET\_APPLICATION\_ID Parameters

Parameter	Description
p_application_id	This is the application ID. The application ID must be a positive integer, and cannot be in the reserved range of application IDs (3000 - 8999). It must be less than 3000 or greater than or equal to 9000.

### Example

For an example of this procedure call, see ["Import Application with Specified Application ID"](#) on page 2-4.

**See Also:** ["SET\\_APPLICATION\\_ID Procedure"](#) on page 2-20,  
["GENERATE\\_APPLICATION\\_ID Procedure"](#) on page 2-8

---

## SET\_APPLICATION\_NAME Procedure

This procedure sets the application name of the import application.

### Syntax

```
APEX_APPLICATION_INSTALL.SET_APPLICATION_NAME;(
    p_application_name IN VARCHAR2);
```

### Parameters

[Table 2-4](#) describes the parameters available in SET\_APPLICATION\_NAME procedure.

**Table 2-4** SET\_APPLICATION\_NAME Parameters

Parameter	Description
p_application_name	This is the application name. The application name cannot be null and cannot be longer than 255 characters.

### Example

The following example sets the application name in APEX\_APPLICATION\_INSTALL to "Executive Dashboard".

```
declare
    l_name varchar2(255) := 'Executive Dashboard';
begin
    apex_application_install.set_application_name( p_application_name => l_name );
end;
```

**See Also:** ["GET\\_APPLICATION\\_NAME Function"](#) on page 2-21

---

## SET\_AUTO\_INSTALL\_SUP\_OBJ Procedure

This procedure sets the automatic install of supporting objects value used during application import. This setting is valid only for command line installs. If the value is set to TRUE and the application export contains supporting objects, it automatically installs or upgrades the supporting objects when an application imports from the command line.

### Syntax

```
APEX_APPLICATION_INSTALL.SET_AUTO_INSTALL_SUP_OBJ(  
    p_auto_install_sup_obj IN BOOLEAN);
```

### Parameters

[Table 2–5](#) describes the parameters available in SET\_AUTO\_INSTALL\_SUP\_OBJ procedure.

**Table 2–5 SET\_AUTO\_INSTALL\_SUP\_OBJ Parameters**

Parameter	Description
p_auto_install_sup_obj	The automatic install of supporting objects Boolean value.

### Example

The following example gets the automatic install of supporting objects setting. If it is not set to install automatically, it sets to true to override export file settings of automatic install of supporting objects.

```
begin  
    apex_application_install.set_auto_install_sup_obj( p_auto_install_sup_obj =>  
true );  
end;
```

---

## SET\_IMAGE\_PREFIX Procedure

This procedure sets the image prefix of the import application. Most Application Express instances use the default image prefix of /i/.

### Syntax

```
APEX_APPLICATION_INSTALL.SET_IMAGE_PREFIX(
    p_image_prefix IN VARCHAR2);
```

### Parameters

[Table 2-4](#) describes the parameters available in SET\_APPLICATION\_NAME procedure.

**Table 2-6** SET\_AUTO\_INSTALL\_SUP\_OBJ Parameters

Parameter	Description
p_auto_install_sup_obj	The automatic install of supporting objects Boolean value.

### Example

The following example sets the value of the image prefix variable in APEX\_APPLICATION\_INSTALL.

```
declare
    l_prefix varchar2(255) := '/i/';
begin
    apex_application_install.set_image_prefix( p_image_prefix => l_prefix );
end;
```

**See Also:** ["GET\\_IMAGE\\_PREFIX Function"](#) on page 2-14

## SET\_OFFSET Procedure

This procedure sets the offset value used during application import. Use the offset value to ensure that the metadata for the Application Express application definition does not collide with other metadata on the instance. For a new application installation, it is usually sufficient to call the `generate_offset` procedure to have Application Express generate this offset value for you.

### Syntax

```
APEX_APPLICATION_INSTALL.SET_OFFSET(
    p_offset IN NUMBER);
```

### Parameters

[Table 2-7](#) describes the parameters available in `SET_OFFSET` procedure.

**Table 2-7 SET\_OFFSET Parameters**

Parameter	Description
<code>p_offset</code>	The offset value. The offset must be a positive integer. In most cases you do not need to specify the offset, and instead, call <code>APEX_APPLICATION_INSTALL.GENERATE_OFFSET</code> , which generates a large random value and then set it in the <code>APEX_APPLICATION_INSTALL</code> package.

### Example

The following example generates a random number from the database and uses this as the offset value in `APEX_APPLICATION_INSTALL`.

```
declare
    l_offset number;
begin
    l_offset := dbms_random.value(100000000000, 999999999999);
    apex_application_install.set_offset( p_offset => l_offset );
end/
```

**See Also:** ["GET\\_OFFSET Function"](#) on page 2-15, ["GENERATE\\_OFFSET Procedure"](#) on page 2-9

---

## SET\_PROXY Procedure

Use this procedure to set the proxy server attributes of an application to be imported.

### Syntax

```
APEX_APPLICATION_INSTALL.SET_PROXY (
    p_proxy IN VARCHAR2);
```

### Parameters

[Table 2–8](#) describes the parameters available in SET\_PROXY procedure.

**Table 2–8 SET\_PROXY Parameters**

Parameter	Description
p_proxy	The proxy server. There is no default value. The proxy server cannot be more than 255 characters and should not include any protocol prefix such as http://. A sample value might be: www-proxy.company.com

### Example

The following example sets the value of the proxy variable in APEX\_APPLICATION\_INSTALL.

```
declare
    l_proxy varchar2(255) := 'www-proxy.company.com'
begin
    apex_application_install.set_proxy( p_proxy => l_proxy );
end;
```

**See Also:** ["SET\\_PROXY Procedure"](#) on page 2-16

## SET\_SCHEMA Procedure

Use this function to set the parsing schema ("owner") of the Application Express application. The database user of this schema must already exist, and this schema name must already be mapped to the workspace used to import the application.

### Syntax

```
APEX_APPLICATION_INSTALL.SET_SCHEMA (  
    p_schema IN VARCHAR2);
```

### Parameters

[Table 2-9](#) describes the parameters available in the SET\_SCHEMA procedure.

**Table 2-9 SET\_SCHEMA Parameters**

Parameter	Description
p_schema	The schema name.

### Example

For examples of this procedure call, see ["Import Application into Different Workspace using Different Schema"](#) on page 2-4 and ["Import into Training Instance for Three Different Workspaces"](#) on page 2-5.

**See Also:** ["GET\\_SCHEMA Function"](#) on page 2-17



---

## SET\_WORKSPACE\_ID Procedure

Use this function to set the workspace ID for the application to be imported.

### Syntax

```
APEX_APPLICATION_INSTALL.SET_WORKSPACE_ID (  
    p_workspace_id IN NUMBER);
```

### Parameters

[Table 2–10](#) describes the parameters available in the SET\_WORKSPACE\_ID procedure.

**Table 2–10** SET\_WORKSPACE\_ID Parameters

Parameter	Description
p_workspace_id	The workspace ID.

### Example

For examples of this procedure call, see ["Import Application into Different Workspace using Different Schema"](#) on page 2-4 and ["Import into Training Instance for Three Different Workspaces"](#) on page 2-5.

**See Also:** ["SET\\_WORKSPACE\\_ID Procedure"](#) on page 2-27



---

---

## APEX\_AUTHENTICATION

The APEX\_AUTHENTICATION package provides a public API for authentication plugins.

- [Constants](#)
- [CALLBACK Procedure](#)
- [GET\\_CALLBACK\\_URL Function](#)
- [GET\\_LOGIN\\_USERNAME\\_COOKIE Function](#)
- [IS\\_AUTHENTICATED Function](#)
- [IS\\_PUBLIC\\_USER Function](#)
- [LOGIN Procedure](#)
- [LOGOUT Procedure](#)
- [POST\\_LOGIN Procedure](#)
- [SEND\\_LOGIN\\_USERNAME\\_COOKIE Procedure](#)

## Constants

The following constant is used by this package.

```
c_default_username_cookie constant varchar2(30) := 'LOGIN_USERNAME_COOKIE';
```

## CALLBACK Procedure

This procedure is the landing resource for external login pages. Call this procedure directly from the browser.

### Syntax

```
APEX_AUTHENTICATION.CALLBACK (
    p_session_id IN NUMBER,
    p_app_id IN NUMBER,
    p_page_id IN NUMBER DEFAULT NULL,
    p_ajax_identifrier IN VARCHAR2,
    p_x01 IN VARCHAR2 DEFAULT NULL,
    p_x02 IN VARCHAR2 DEFAULT NULL,
    p_x03 IN VARCHAR2 DEFAULT NULL,
    p_x04 IN VARCHAR2 DEFAULT NULL,
    p_x05 IN VARCHAR2 DEFAULT NULL,
    p_x06 IN VARCHAR2 DEFAULT NULL,
    p_x07 IN VARCHAR2 DEFAULT NULL,
    p_x08 IN VARCHAR2 DEFAULT NULL,
    p_x09 IN VARCHAR2 DEFAULT NULL,
    p_x10 IN VARCHAR2 DEFAULT NULL );
```

### Parameters

[Table 3–1](#) describes the parameters available in `CALLBACK` procedure.

**Table 3–1** *APEX\_AUTHENTICATION.CALLBACK Procedure Parameters*

Parameters	Description
<code>p_session_id</code>	The Application Express session identifier.
<code>p_app_id</code>	The database application identifier.
<code>p_page_id</code>	Optional page identifier.
<code>p_ajax_identifrierp</code>	The system generated Ajax identifier. See " <a href="#">GET AJAX IDENTIFIER Function</a> " on page 20-10.
<code>p_x01</code> through <code>p_x10</code>	Optional parameters that the external login passes to the authentication plugin.

### Example 1

In this example, a redirect is performed to an external login page and the callback is passed into Application Express, which the external login redirects to after successful authentication.

```
declare
    l_callback varchar2(4000) := apex_application.get_callback_url;
begin
    sys.owa_util.redirect_url(
        'https://single-signon.example.com/my_custom_sso.login?p_on_success=' ||
        sys.utl_url.escape (
            url => l_callback,
            escape_reserved_chars => true );
    apex_application.stop_apex_engine;
end;
```

**Example 2**

In this example, an external login page saves user data in a shared table and performs a call back with a handle to the data. In Application Express, the callback activates the authentication plugin's ajax code. It can take the value of x01 and fetch the actual user data from the shared table.

```
---- create or replace package body my_custom_sso as
procedure login (
    p_on_success in varchar2 )
is
    l_login_id varchar2(32);
begin
    l_login_id := rawtohex(sys.dbms_crypto.random(32));
    insert into login_data(id, username) values (l_login_id, 'JOE USER');
    sys.owa_util.redirect_url (
        p_on_success||'&p_x01='||l_login_id );
end;
---- end my_custom_sso;
```

---

---

**See Also:** ["GET\\_CALLBACK\\_URL Function"](#) on page 3-5

---

---

---

## GET\_CALLBACK\_URL Function

This function is a plugin helper function to return a URL that is used as a landing request for external login pages. When the browser sends the request, it triggers the authentication plugin ajax callback, which can be used to log the user in.

### Syntax

```
APEX_AUTHENTICATION.GET_CALLBACK_URL (
  p_x01 IN VARCHAR2 DEFAULT NULL,
  p_x02 IN VARCHAR2 DEFAULT NULL,
  p_x03 IN VARCHAR2 DEFAULT NULL,
  p_x04 IN VARCHAR2 DEFAULT NULL,
  p_x05 IN VARCHAR2 DEFAULT NULL,
  p_x06 IN VARCHAR2 DEFAULT NULL,
  p_x07 IN VARCHAR2 DEFAULT NULL,
  p_x08 IN VARCHAR2 DEFAULT NULL,
  p_x09 IN VARCHAR2 DEFAULT NULL,
  p_x10 IN VARCHAR2 DEFAULT NULL )
return VARCHAR2;
```

### Parameters

[Table 3–2](#) describes the parameters available in GET\_CALLBACK\_URL function.

**Table 3–2 APEX\_AUTHENTICATION.GET\_CALLBACK\_URL Function Parameters**

Parameters	Description
p_x01 through p_x10	Optional parameters that the external login passes to the authentication plugin.

### Example

See example in "[CALLBACK Procedure](#)" on page 3-3.

---

**See Also:** "[CALLBACK Procedure](#)" on page 3-3

---

## GET\_LOGIN\_USERNAME\_COOKIE Function

This function reads the cookie with the username from the default login page.

### Syntax

```
APEX_AUTHENTICATION.GET_LOGIN_USERNAME_COOKIE (  
    p_cookie_name IN VARCHAR2 DEFAULT c_default_username_cookie )  
    return varchar2;
```

### Parameters

[Table 3–3](#) describes the parameters available in GET\_LOGIN\_USERNAME\_COOKIE function.

**Table 3–3** APEX\_AUTHENTICATION.GET\_LOGIN\_USERNAME\_COOKIE Function Parameters

Parameters	Description
p_cookie_name	The cookie name that stores the username in the browser.

### Example

In this example, GET\_LOGIN\_USERNAME\_COOKIE saves the username cookie value into the page item P101\_USERNAME.

```
:P101_USERNAME := apex_authentication.get_login_username_cookie;
```



## IS\_AUTHENTICATED Function

This function checks if the user is authenticated in the session and returns TRUE if the user is already logged in or FALSE if the user of the current session is not yet authenticated.

### Syntax

```
APEX_AUTHENTICATION.IS_AUTHENTICATED  
    RETURN BOOLEAN;
```

### Parameters

None.

### Example

In this example, IS\_AUTHENTICATED is used to emit the username if the user has already logged in or a notification if the user has not.

```
if apex_authentication.is_authenticated then  
    sys.htp.p(apex_escape.html(:APP_USER) || ', you are known to the system');  
else  
    sys.htp.p('Please sign in');  
end if;
```

---

---

**See Also:** ["IS\\_PUBLIC\\_USER Function"](#) on page 3-8

---

---

## IS\_PUBLIC\_USER Function

This function checks if the user is not authenticated in the session. A FALSE is returned if the user is already logged on or TRUE if the user of the current session is not yet authenticated.

### Syntax

```
APEX_AUTHENTICATION.IS_PUBLIC_USER  
    return BOOLEAN;
```

### Parameters

None.

### Example

In this example, IS\_PUBLIC\_USER is used to show a notification if the user has not already logged in or the username if the user has not.

```
if apex_authentication.is_public_user then  
    sys.htp.p('Please sign in');  
else  
    sys.htp.p(apex_escape.html(:APP_USER)||', you are known to the system');  
end if;
```

---

## LOGIN Procedure

This procedure authenticates the user in the current session.

Login processing has the following steps:

1. Run authentication scheme's pre-authentication procedure.
2. Run authentication scheme's authentication function to check the user credentials (p\_username, p\_password), returning TRUE on success.
3. If result=true: run post-authentication procedure.
4. If result=true: save username in session table.
5. If result=true: set redirect url to deep link.
6. If result=false: set redirect url to current page, with an error message in the notification\_msg parameter.
7. Log authentication result.
8. Redirect.

### Syntax

```
APEX_AUTHENTICATION.LOGIN (
    p_username IN VARCHAR2,
    p_password IN VARCHAR2,
    p_uppercase_username IN BOOLEAN DEFAULT TRUE );
```

### Parameters

[Table 3–4](#) describes the parameters available in LOGIN Procedure.

**Table 3–4** *APEX\_AUTHENTICATION.LOGIN Procedure Parameters*

Parameters	Description
p_username	The user's name.
p_password	The user's password.
p_uppercase_username	If TRUE then p_username is converted to uppercase.

### Example

This example passes user credentials, username and password, to the authentication scheme.

```
apex_authentication.login('JOE USER', 'mysecret');
```

---

**See Also:** ["POST\\_LOGIN Procedure"](#) on page 3-11

---

## LOGOUT Procedure

This procedure closes the session and redirects to the application's home page. Call this procedure directly from the browser.

### Syntax

```
APEX_AUTHENTICATION.LOGOUT (  
    p_session_id in number,  
    p_app_id in number,  
    p_ws_app_id in number default null );
```

### Parameters

Table 3–5 describes the parameters available in LOGOUT Procedure.

**Table 3–5** *APEX\_AUTHENTICATION.LOGOUT Procedure Parameters*

Parameters	Description
p_session_id	The Application Express session identifier of the session to close.
p_app_id	The database application identifier.
p_ws_app_id	The worksheet application identifier.

### Example

This example logs the session out.

```
apex_authentication.logout (:SESSION, :APP_ID);
```

---

## POST\_LOGIN Procedure

This procedure authenticates the user in the current session. It runs a subset of `login()`, without steps 1 and 2. For steps, see "[LOGIN Procedure](#)" on page 3-9. It is primarily useful in authentication schemes where user credentials checking is done externally to Application Express.

### Syntax

```
APEX_AUTHENTICATION.POST_LOGIN (  
    p_username IN VARCHAR2,  
    p_password IN VARCHAR2,  
    p_uppercase_username IN BOOLEAN DEFAULT TRUE );
```

### Parameters

[Table 3-6](#) describes the parameters available in POST\_LOGIN Procedure.

**Table 3-6** *APEX\_AUTHENTICATION.POST\_LOGIN Procedure Parameters*

Parameters	Description
<code>p_username</code>	The user's name.
<code>p_password</code>	The user's password.
<code>p_uppercase_username</code>	If TRUE then <code>p_username</code> is converted to uppercase.

### Example

This procedure call passes user credentials, username and password, to the authentication scheme to finalize the user's authentication.

```
apex_authentication.post_login('JOE USER', 'mysecret');
```

---

**See Also:** "[LOGIN Procedure](#)" on page 3-9

---

---

## SEND\_LOGIN\_USERNAME\_COOKIE Procedure

This procedure sends a cookie with the username.

### Syntax

```
APEX_AUTHENTICATION.SEND_LOGIN_USERNAME_COOKIE (  
    p_username IN VARCHAR2,  
    p_cookie_name IN VARCHAR2 DEFAULT c_default_username_cookie );
```

### Parameters

Table 3–7 describes the parameters available in SEND\_LOGIN\_USERNAME\_COOKIE Procedure.

**Table 3–7 APEX\_AUTHENTICATION.SEND\_LOGIN\_USERNAME\_COOKIE Procedure Parameters**

Parameters	Description
p_username	The user's name.
p_cookie_name	The cookie name which stores p_username in the browser.

### Example

This example shows how to call SEND\_LOGIN\_USERNAME\_COOKIE, to save the value of item P101\_USERNAME in the login cookie.

```
apex_authentication.send_login_username_cookie (  
    p_username => :P101_USERNAME );
```

---

---

## APEX\_AUTHORIZATION

The APEX\_AUTHORIZATION package contains public utility functions used for controlling and querying access rights to the application.

- [ENABLE\\_DYNAMIC\\_GROUPS Procedure](#)
- [IS\\_AUTHORIZED Function](#)
- [RESET\\_CACHE Procedure](#)

---

## ENABLE\_DYNAMIC\_GROUPS Procedure

This procedure enables groups in the current session. These groups do not have to be created in the Application Express workspace repository, but can, for example, be loaded from a LDAP repository. Enabling a group that exists in the workspace repository and has other groups granted to it, also enables the granted groups.

If Real Application Security, available with Oracle Database Release 12g, is enabled for the authentication scheme, all dynamic groups are enabled as RAS dynamic or external groups (depending whether the group exists in `dba_xs_dynamic_roles`).

This procedure must be called during or right after authentication, for example, in a post-authentication procedure.

### Syntax

```
APEX_AUTHORIZATION.ENABLE_DYNAMIC_GROUPS (  
    p_group_names IN apex_t_varchar2 );
```

### Parameters

**Table 4–1** *ENABLE\_DYNAMIC\_GROUPS Procedure Parameter*

Parameter	Description
<code>p_group_names</code>	Table of group names.

### Example

This example enables the dynamic groups SALES and HR, for example, from within a post authentication procedure.

```
begin  
    apex_authorization.enable_dynamic_groups (  
        p_group_names => apex_t_varchar2('SALES', 'HR') );  
end;
```

**See Also:** View `APEX_WORKSPACE_SESSION_GROUPS` and View `APEX_WORKSPACE_GROUP_GROUPS`



---

## IS\_AUTHORIZED Function

Determine if the current user passes the authorization with name `p_authorization_name`. For performance reasons, authorization results are cached. Because of this, the function may not always evaluate the authorization when called, but take the result out of the cache.

**See Also:** "Changing the Evaluation Point Attribute" in *Oracle Application Express Application Builder User's Guide*

### Syntax

```
APEX_AUTHORIZATION.IS_AUTHORIZED (
  p_authorization_name IN VARCHAR2 )
RETURN BOOLEAN;
```

### Parameters

**Table 4–2 IS\_AUTHORIZED Function Parameters**

Parameter	Description
<code>p_authorization_name</code>	The name of an authorization scheme in the application.

### Returns

**Table 4–3 IS\_AUTHORIZED Function Returns**

Parameter	Description
TRUE	If the authorization is successful.
FALSE	If the authorization is not successful.

### Example

This example prints the result of the authorization "User Is Admin".

```
begin
  sys.http.p('User Is Admin: ' ||
    case apex_authorization.is_authorized (
      p_authorization_name => 'User Is Admin' )
    when true then 'YES'
    when false then 'NO'
    else 'null'
    end);
end;
```

---

## RESET\_CACHE Procedure

This procedure resets the authorization caches for the session and forces a re-evaluation when an authorization is checked next.

### Syntax

```
APEX_AUTHORIZATION.RESET_CACHE;
```

### Parameters

None.

### Example

This examples resets the authorization cache.

```
apex_authorization.reset_cache;
```

---

---

## APEX\_COLLECTION

Collections enable you to temporarily capture one or more nonscalar values. You can use collections to store rows and columns currently in session state so they can be accessed, manipulated, or processed during a user's specific session. You can think of a collection as a bucket in which you temporarily store and name rows of information.

- [About the APEX\\_COLLECTION API](#)
- [Naming Collections](#)
- [Creating a Collection](#)
- [About the Parameter p\\_generate\\_md5](#)
- [Accessing a Collection](#)
- [Merging Collections](#)
- [Truncating a Collection](#)
- [Deleting a Collection](#)
- [Deleting All Collections for the Current Application](#)
- [Deleting All Collections in the Current Session](#)
- [Adding Members to a Collection](#)
- [About the Parameters p\\_generate\\_md5, p\\_clob001, p\\_blob001, and p\\_xmltype001](#)
- [Updating Collection Members](#)
- [Deleting Collection Members](#)
- [Obtaining a Member Count](#)
- [Resequencing a Collection](#)
- [Verifying Whether a Collection Exists](#)
- [Adjusting a Member Sequence ID](#)
- [Sorting Collection Members](#)
- [Clearing Collection Session State](#)
- [Determining Collection Status](#)
- [ADD\\_MEMBER Procedure](#)
- [ADD\\_MEMBER Function](#)
- [ADD\\_MEMBERS Procedure](#)
- [COLLECTION\\_EXISTS Function](#)

- 
- [COLLECTION\\_HAS\\_CHANGED Function](#)
  - [COLLECTION\\_MEMBER\\_COUNT Function](#)
  - [CREATE\\_COLLECTION Procedure](#)
  - [CREATE\\_OR\\_TRUNCATE\\_COLLECTION Procedure](#)
  - [CREATE\\_COLLECTION\\_FROM\\_QUERY Procedure](#)
  - [CREATE\\_COLLECTION\\_FROM\\_QUERY2 Procedure](#)
  - [CREATE\\_COLLECTION\\_FROM\\_QUERY\\_B Procedure](#)
  - [CREATE\\_COLLECTION\\_FROM\\_QUERYB2 Procedure](#)
  - [DELETE\\_ALL\\_COLLECTIONS Procedure](#)
  - [DELETE\\_ALL\\_COLLECTIONS\\_SESSION Procedure](#)
  - [DELETE\\_COLLECTION Procedure](#)
  - [DELETE\\_MEMBER Procedure](#)
  - [DELETE\\_MEMBERS Procedure](#)
  - [GET\\_MEMBER\\_MD5 Function](#)
  - [MERGE\\_MEMBERS Procedure](#)
  - [MOVE\\_MEMBER\\_DOWN Procedure](#)
  - [MOVE\\_MEMBER\\_UP Procedure](#)
  - [RESEQUENCE\\_COLLECTION Procedure](#)
  - [RESET\\_COLLECTION\\_CHANGED Procedure](#)
  - [RESET\\_COLLECTION\\_CHANGED\\_ALL Procedure](#)
  - [SORT\\_MEMBERS Procedure](#)
  - [TRUNCATE\\_COLLECTION Procedure](#)
  - [UPDATE\\_MEMBER Procedure](#)
  - [UPDATE\\_MEMBERS Procedure](#)
  - [UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 1](#)
  - [UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 2](#)
  - [UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 3](#)
  - [UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 4](#)
  - [UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 5](#)
  - [UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 6](#)

---

## About the APEX\_COLLECTION API

Every collection contains a named list of data elements (or members) which can have up to 50 character attributes (`VARCHAR2(4000)`), five number attributes, five date attributes, one XML Type attribute, one large binary attribute (`BLOB`), and one large character attribute (`CLOB`). You insert, update, and delete collection information using the PL/SQL API `APEX_COLLECTION`.

The following are examples of when you might use collections:

- When you are creating a data-entry wizard in which multiple rows of information first need to be collected within a logical transaction. You can use collections to temporarily store the contents of the multiple rows of information, before performing the final step in the wizard when both the physical and logical transactions are completed.
- When your application includes an update page on which a user updates multiple detail rows on one page. The user can make many updates, apply these updates to a collection and then call a final process to apply the changes to the database.
- When you are building a wizard where you are collecting an arbitrary number of attributes. At the end of the wizard, the user then performs a task that takes the information temporarily stored in the collection and applies it to the database.

Beginning in Oracle Database 12c, database columns of data type `VARCHAR2` can be defined up to 32,767 bytes. This requires that the database initialization parameter `MAX_STRING_SIZE` has a value of `EXTENDED`. If Application Express was installed in Oracle Database 12c and with `MAX_STRING_SIZE = EXTENDED`, then the tables for the Application Express collections will be defined to support up to 32,767 bytes for the character attributes of a collection. For the methods in the `APEX_COLLECTION` API, all references to character attributes (c001 through c050) can support up to 32,767 bytes.

## Naming Collections

When you create a collection, you must give it a name that cannot exceed 255 characters. Note that collection names are not case-sensitive and are converted to uppercase.

Once the collection is named, you can access the values in the collection by running a SQL query against the view `APEX_COLLECTIONS`.

**See Also:** ["Accessing a Collection"](#) on page 5-7, ["CREATE\\_COLLECTION Procedure Parameters"](#) on page 5-33, ["CREATE\\_OR\\_TRUNCATE\\_COLLECTION Procedure Parameters"](#) on page 5-34

---

## Creating a Collection

Every collection contains a named list of data elements (or members) which can have up to 50 character attributes (`VARCHAR2(4000)`), five number attributes, one XML Type attribute, one large binary attribute (`BLOB`), and one large character attribute (`CLOB`). You use the following methods to create a collection:

- `CREATE_COLLECTION`

This method creates an empty collection with the provided name. An exception is raised if the named collection exists.

- `CREATE_OR_TRUNCATE_COLLECTION`

If the provided named collection does not exist, this method creates an empty collection with the given name. If the named collection exists, this method truncates it. Truncating a collection empties it, but leaves it in place.

- `CREATE_COLLECTION_FROM_QUERY`

This method creates a collection and then populates it with the results of a specified query. An exception is raised if the named collection exists. This method can be used with a query with up to 50 columns in the `SELECT` clause. These columns in the `SELECT` clause populate the 50 character attributes of the collection (`C001` through `C050`).

- `CREATE_COLLECTION_FOM_QUERY2`

This method creates a collection and then populates it with the results of a specified query. An exception is raised if the named collection exists. It is identical to the `CREATE_COLLECTION_FROM_QUERY`, however, the first 5 columns of the `SELECT` clause must be numeric. After the numeric columns, there can be up to 50 character columns in the `SELECT` clause.

- `CREATE_COLLECTION_FROM_QUERY_B`

This method offers significantly faster performance than the `CREATE_COLLECTION_FROM_QUERY` method by performing bulk SQL operations, but has the following limitations:

- No column value in the select list of the query can be more than 2,000 bytes. If a row is encountered that has a column value of more than 2,000 bytes, an error is raised during execution.
- The MD5 checksum is not computed for any members in the collection.

- `CREATE_COLLECTION_FROM_QUERYB2`

This method also creates a collection and then populates it with the results of a specified query. An exception is raised if the named collection exists. It is identical to the `CREATE_COLLECTION_FROM_QUERY_B`, however, the first five columns of the `SELECT` clause must be numeric. After the numeric columns, there can be up to 50 character columns in the `SELECT` clause.

**See Also:** ["CREATE\\_COLLECTION Procedure"](#) on page 5-33,  
["CREATE\\_OR\\_TRUNCATE\\_COLLECTION Procedure"](#) on page 5-34,  
["CREATE\\_COLLECTION\\_FROM\\_QUERY Procedure"](#) on page 5-35,  
["CREATE\\_COLLECTION\\_FROM\\_QUERY2 Procedure"](#) on page 5-36,  
["CREATE\\_COLLECTION\\_FROM\\_QUERY\\_B Procedure"](#) on page 5-37,  
["CREATE\\_COLLECTION\\_FROM\\_QUERYB2 Procedure"](#) on page 5-39

## About the Parameter p\_generate\_md5

Use the p\_generate\_md5 flag to specify if the message digest of the data of the collection member should be computed. By default, this flag is set to NO. Use this parameter to check the MD5 of the collection member (that is, compare it with another member or see if a member has changed).

**See Also:** ["Determining Collection Status"](#) on page 5-23 for information about using the GET\_MEMBER\_MD5 function, ["GET\\_MEMBER\\_MD5 Function"](#) on page 5-46



---

## Accessing a Collection

You can access the members of a collection by querying the database view `APEX_COLLECTIONS`. Collection names are always converted to uppercase. When querying the `APEX_COLLECTIONS` view, always specify the collection name in all uppercase. The `APEX_COLLECTIONS` view has the following definition:

```
COLLECTION_NAME  NOT NULL VARCHAR2(255)
SEQ_ID           NOT NULL NUMBER
C001             VARCHAR2(4000)
C002             VARCHAR2(4000)
C003             VARCHAR2(4000)
C004             VARCHAR2(4000)
C005             VARCHAR2(4000)
...
C050             VARCHAR2(4000)
N001             NUMBER
N002             NUMBER
N003             NUMBER
N004             NUMBER
N005             NUMBER
CLOB001          CLOB
BLOB001          BLOB
XMLTYPE001       XMLTYPE
MD5_ORIGINAL     VARCHAR2(4000)
```

Use the `APEX_COLLECTIONS` view in an application just as you would use any other table or view in an application, for example:

```
SELECT c001, c002, c003, n001, clob001
       FROM APEX_collections
       WHERE collection_name = 'DEPARTMENTS'
```

## Merging Collections

You can merge members of a collection with values passed in a set of arrays. By using the `p_init_query` argument, you can create a collection from the supplied query.

**See Also:** ["MERGE\\_MEMBERS Procedure"](#) on page 5-47

## Truncating a Collection

If you truncate a collection, you remove all members from the specified collection, but the named collection remains in place.

**See Also:** ["TRUNCATE\\_COLLECTION Procedure"](#) on page 5-55

---

## Deleting a Collection

If you delete a collection, you delete the collection and all of its members. Be aware that if you do not delete a collection, it is eventually deleted when the session is purged.

**See Also:** ["DELETE\\_COLLECTION Procedure"](#) on page 5-43

## Deleting All Collections for the Current Application

Use the `DELETE_ALL_COLLECTIONS` method to delete all collections defined in the current application.

**See Also:** ["DELETE\\_ALL\\_COLLECTIONS Procedure"](#) on page 5-41

---

## Deleting All Collections in the Current Session

Use the `DELETE_ALL_COLLECTIONS_SESSION` method to delete all collections defined in the current session.

**See Also:** ["DELETE\\_ALL\\_COLLECTIONS\\_SESSION Procedure"](#) on page 5-42

## Adding Members to a Collection

When data elements (or members) are added to a collection, they are assigned a unique sequence ID. As you add members to a collection, the sequence ID is change in increments of 1, with the newest members having the largest ID.

You add new members to a collection using the `ADD_MEMBER` function. Calling this function returns the sequence ID of the newly added member.

You can also add new members (or an array of members) to a collection using the `ADD_MEMBERS` procedure. The number of members added is based on the number of elements in the first array.

**See Also:** ["ADD\\_MEMBER Procedure"](#) on page 5-24, ["ADD\\_MEMBER Function"](#) on page 5-26, ["ADD\\_MEMBERS Procedure"](#) on page 5-28

## About the Parameters p\_generate\_md5, p\_clob001, p\_blob001, and p\_xmltype001

Use the p\_generate\_md5 flag to specify if the message digest of the data of the collection member should be computed. By default, this flag is set to NO. Use this parameter to check the MD5 of the collection member (that is, compare it with another member or see if a member has changed).

Use p\_clob001 for collection member attributes which exceed 4,000 characters. Use p\_blob001 for binary collection member attributes. Use p\_xmltype001 to store well-formed XML.

**See Also:** ["Determining Collection Status"](#) on page 5-23 for information about using the function GET\_MEMBER\_MD5



## Updating Collection Members

You can update collection members by calling the `UPDATE_MEMBER` procedure and referencing the desired collection member by its sequence ID. The `UPDATE_MEMBER` procedure replaces an entire collection member, not individual member attributes.

Use the `p_clob001` parameter for collection member attributes which exceed 4,000 characters.

To update a single attribute of a collection member, use the `UPDATE_MEMBER_ATTRIBUTE` procedure.

**See Also:** ["UPDATE\\_MEMBER Procedure" on page 5-56](#), ["UPDATE\\_MEMBERS Procedure" on page 5-58](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 1" on page 5-60](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 2" on page 5-62](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 3" on page 5-64](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 4" on page 5-66](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 5" on page 5-68](#)

## Deleting Collection Members

You can delete a collection member by calling the `DELETE_MEMBER` procedure and referencing the desired collection member by its sequence ID. Note that this procedure leaves a gap in the sequence IDs in the specified collection.

You can also delete all members from a collection by when an attribute matches a specific value. Note that the `DELETE_MEMBERS` procedure also leaves a gap in the sequence IDs in the specified collection. If the supplied attribute value is null, then all members of the named collection are deleted where the attribute (specified by `p_attr_number`) is null.

**See Also:** ["DELETE\\_MEMBER Procedure"](#) on page 5-44, ["DELETE\\_MEMBERS Procedure"](#) on page 5-45

## Obtaining a Member Count

Use `COLLECTION_MEMBER_COUNT` to return the total count of all members in a collection. Note that this count does not indicate the highest sequence in the collection.

**See Also:** ["COLLECTION\\_MEMBER\\_COUNT Function"](#) on page 5-32

## Resequencing a Collection

Use `RESEQUENCE_COLLECTION` to resequence a collection to remove any gaps in sequence IDs while maintaining the same element order.

**See Also:** ["RESEQUENCE\\_COLLECTION Procedure"](#) on page 5-51

## Verifying Whether a Collection Exists

Use `COLLECTION_EXISTS` to determine if a collection exists.

**See Also:** ["COLLECTION\\_EXISTS Function"](#) on page 5-30

## Adjusting a Member Sequence ID

You can adjust the sequence ID of a specific member within a collection by moving the ID up or down. When you adjust a sequence ID, the specified ID is exchanged with another ID. For example, if you were to move the ID 2 up, 2 becomes 3, and 3 would become 2.

Use `MOVE_MEMBER_UP` to adjust a member sequence ID up by one. Alternately, use `MOVE_MEMBER_DOWN` to adjust a member sequence ID down by one.

**See Also:** ["MOVE\\_MEMBER\\_DOWN Procedure"](#) on page 5-49,  
["MOVE\\_MEMBER\\_UP Procedure"](#) on page 5-50

## Sorting Collection Members

Use the `SORT_MEMBERS` method to reorder members of a collection by the column number. This method sorts the collection by a particular column number and also reassigns the sequence IDs for each member to remove gaps.

**See Also:** ["SORT\\_MEMBERS Procedure"](#) on page 5-54

## Clearing Collection Session State

Clearing the session state of a collection removes the collection members. A shopping cart is a good example of when you might need to clear collection session state. When a user requests to empty the shopping cart and start again, you must clear the session state for a collection. You can remove session state of a collection by calling the `TRUNCATE_COLLECTION` method or by using `f?p` syntax.

Calling the `TRUNCATE_COLLECTION` method deletes the existing collection and then recreates it, for example:

```
APEX_COLLECTION.TRUNCATE_COLLECTION(  
    p_collection_name => collection name);
```

You can also use the sixth `f?p` syntax argument to clear session state, for example:

```
f?p=App:Page:Session::NO:collection name
```

**See Also:** ["TRUNCATE\\_COLLECTION Procedure"](#) on page 5-55



---

## Determining Collection Status

The `p_generate_md5` parameter determines if the MD5 message digests are computed for each member of a collection. The collection status flag is set to `FALSE` immediately after you create a collection. If any operations are performed on the collection (such as add, update, truncate, and so on), this flag is set to `TRUE`.

You can reset this flag manually by calling `RESET_COLLECTION_CHANGED`.

Once this flag has been reset, you can determine if a collection has changed by calling `COLLECTION_HAS_CHANGED`.

When you add a new member to a collection, an MD5 message digest is computed against all 50 attributes and the CLOB attribute if the `p_generated_md5` parameter is set to `YES`. You can access this value from the `MD5_ORIGINAL` column of the view `APEX_COLLECTION`. You can access the MD5 message digest for the current value of a specified collection member by using the function `GET_MEMBER_MD5`.

**See Also:** ["RESET\\_COLLECTION\\_CHANGED Procedure"](#) on page 5-52, ["COLLECTION\\_HAS\\_CHANGED Function"](#) on page 5-31, ["GET\\_MEMBER\\_MD5 Function"](#) on page 5-46

## ADD\_MEMBER Procedure

Use this procedure to add a new member to an existing collection. An error is raised if the specified collection does not exist for the current user in the same session for the current Application ID. Gaps are not used when adding a new member, so an existing collection with members of sequence IDs (1,2,5,8) adds the new member with a sequence ID of 9.

### Syntax

```
APEX_COLLECTION.ADD_MEMBER (
    p_collection_name IN VARCHAR2,
    p_c001 IN VARCHAR2 default null,
    ...
    p_c050 IN VARCHAR2 default null,
    p_n001 IN NUMBER default null,
    p_n002 IN NUMBER default null,
    p_n003 IN NUMBER default null,
    p_n004 IN NUMBER default null,
    p_n005 IN NUMBER default null,
    p_d001 IN DATE default null,
    p_d002 IN DATE default null,
    p_d003 IN DATE default null,
    p_d004 IN DATE default null,
    p_d005 IN DATE default null,
    p_clob001 IN CLOB default empty_clob(),
    p_blob001 IN BLOB default empty_blob(),
    p_xmltype001 IN XMLTYPE default null,
    p_generate_md5 IN VARCHAR2 default 'NO');
```

### Parameters

[Table 5–1](#) describes the parameters available in the `ADD_MEMBER` procedure.

---



---

**Note:** Any character attribute exceeding 4,000 characters is truncated to 4,000 characters.

---



---

**Table 5–1** *ADD\_MEMBER Procedure Parameters*

Parameter	Description
<code>p_collection_name</code>	The name of an existing collection. Maximum length is 255 bytes. Collection names are not case sensitive and are converted to upper case.
<code>p_c001</code> through <code>p_c050</code>	Attribute value of the member to be added. Maximum length is 4,000 bytes. Any character attribute exceeding 4,000 characters is truncated to 4,000 characters.
<code>p_n001</code> through <code>p_n005</code>	Attribute value of the numeric attributes to be added.
<code>p_d001</code> through <code>p_d005</code>	Attribute value of the date attribute.
<code>p_clob001</code>	Use <code>p_clob001</code> for collection member attributes that exceed 4,000 characters.
<code>p_blob001</code>	Use <code>p_blob001</code> for binary collection member attributes.
<code>p_xmltype001</code>	Use <code>p_xmltype001</code> to store well-formed XML.

**Table 5–1 (Cont.) ADD\_MEMBER Procedure Parameters**

Parameter	Description
p_generate_md5	Valid values include YES and NO. YES to specify if the message digest of the data of the collection member should be computed. Use this parameter to compare the MD5 of the collection member with another member or to see if that member has changed.

**Example**

The following is an example of the ADD\_MEMBER procedure.

```
APEX_COLLECTION.ADD_MEMBER(  
    p_collection_name => 'GROCERIES'  
    p_c001             => 'Grapes',  
    p_c002             => 'Imported',  
    p_n001             => 125,  
    p_d001             => sysdate );  
END;
```

**See Also:** ["GET\\_MEMBER\\_MD5 Function"](#) on page 5-46, ["ADD\\_MEMBER Function"](#) on page 5-26, ["ADD\\_MEMBERS Procedure"](#) on page 5-28

## ADD\_MEMBER Function

Use this function to add a new member to an existing collection. Calling this function returns the sequence ID of the newly added member. An error is raised if the specified collection does not exist for the current user in the same session for the current Application ID. Gaps are not used when adding a new member, so an existing collection with members of sequence IDs (1,2,5,8) adds the new member with a sequence ID of 9.

### Syntax

```
APEX_COLLECTION.ADD_MEMBER (
    p_collection_name IN VARCHAR2,
    p_c001 IN VARCHAR2 default null,
    ...
    p_c050 IN VARCHAR2 default null,
    p_n001 IN NUMBER default null,
    p_n002 IN NUMBER default null,
    p_n003 IN NUMBER default null,
    p_n004 IN NUMBER default null,
    p_n005 IN NUMBER default null,
    p_d001 IN DATE default null,
    p_d002 IN DATE default null,
    p_d003 IN DATE default null,
    p_d004 IN DATE default null,
    p_d005 IN DATE default null,
    p_clob001 IN CLOB default empty_clob(),
    p_blob001 IN BLOB default empty_blob(),
    p_xmltype001 IN XMLTYPE default null,
    p_generate_md5 IN VARCHAR2 default 'NO')
RETURN NUMBER;
```

### Parameters

[Table 5–2](#) describes the parameters available in the `ADD_MEMBER` function.

---

**Note:** Any character attribute exceeding 4,000 characters is truncated to 4,000 characters.

---

**Table 5–2** *ADD\_MEMBER Function Parameters*

Parameter	Description
<code>p_collection_name</code>	The name of an existing collection. Maximum length is 255 bytes. Collection names are not case sensitive and are converted to upper case.
<code>p_c001</code> through <code>p_c050</code>	Attribute value of the member to be added. Maximum length is 4,000 bytes. Any character attribute exceeding 4,000 characters is truncated to 4,000 characters.
<code>p_n001</code> through <code>p_n005</code>	Attribute value of the numeric attributes to be added.
<code>p_d001</code> through <code>p_d005</code>	Attribute value of the date attribute to be added.
<code>p_clob001</code>	Use <code>p_clob001</code> for collection member attributes that exceed 4,000 characters.
<code>p_blob001</code>	Use <code>p_blob001</code> for binary collection member attributes.

**Table 5–2 (Cont.) ADD\_MEMBER Function Parameters**

Parameter	Description
p_xmltype001	Use p_xmltype001 to store well-formed XML.
p_generate_md5	Valid values include YES and NO. YES to specify if the message digest of the data of the collection member should be computed. Use this parameter to compare the MD5 of the collection member with another member or to see if that member has changed.

**Example**

```
DECLARE
    l_seq number;
BEGIN
    l_seq := APEX_COLLECTION.ADD_MEMBER(
        p_collection_name => 'GROCERIES'
        p_c001             => 'Grapes',
        p_c002             => 'Imported',
        p_n001             => 125,
        p_d001             => sysdate );
END;
```

**See Also:** ["GET\\_MEMBER\\_MD5 Function"](#) on page 5-46, ["ADD\\_MEMBER Procedure"](#) on page 5-24, ["ADD\\_MEMBERS Procedure"](#) on page 5-28

## ADD\_MEMBERS Procedure

Use this procedure to add an array of members to a collection. An error is raised if the specified collection does not exist for the current user in the same session for the current Application ID. Gaps are not used when adding a new member, so an existing collection with members of sequence IDs (1,2,5,8) adds the new member with a sequence ID of 9. The count of elements in the p\_c001 PL/SQL table is used as the total number of items across all PL/SQL tables. For example, if p\_c001.count is 2 and p\_c002.count is 10, only 2 members are added. If p\_c001 is null an application error is raised.

### Syntax

```
APEX_COLLECTION.ADD_MEMBERS (
    p_collection_name IN VARCHAR2,
    p_c001 IN APEX_APPLICATION_GLOBAL.VC_ARR2 default empty_vc_arr,
    p_c002 IN APEX_APPLICATION_GLOBAL.VC_ARR2 default empty_vc_arr,
    p_c003 IN APEX_APPLICATION_GLOBAL.VC_ARR2 default empty_vc_arr,
    ...
    p_c050 IN APEX_APPLICATION_GLOBAL.VC_ARR2 default empty_vc_arr,
    p_n001 IN APEX_APPLICATION_GLOBAL.N_ARR default empty_n_arr,
    p_n002 IN APEX_APPLICATION_GLOBAL.N_ARR default empty_n_arr,
    p_n003 IN APEX_APPLICATION_GLOBAL.N_ARR default empty_n_arr,
    p_n004 IN APEX_APPLICATION_GLOBAL.N_ARR default empty_n_arr,
    p_n005 IN APEX_APPLICATION_GLOBAL.N_ARR default empty_n_arr,
    p_d001 IN APEX_APPLICATION_GLOBAL.D_ARR default empty_d_arr,
    p_d002 IN APEX_APPLICATION_GLOBAL.D_ARR default empty_d_arr,
    p_d003 IN APEX_APPLICATION_GLOBAL.D_ARR default empty_d_arr,
    p_d004 IN APEX_APPLICATION_GLOBAL.D_ARR default empty_d_arr,
    p_d005 IN APEX_APPLICATION_GLOBAL.D_ARR default empty_d_arr,
    p_generate_md5 IN VARCHAR2 default 'NO');
```

### Parameters

Table 5–3 describes the parameters available in the ADD\_MEMBERS procedure.

---



---

**Note:** Any character attribute exceeding 4,000 characters is truncated to 4,000 characters. Also, the number of members added is based on the number of elements in the first array.

---



---

**Table 5–3** ADD\_MEMBERS Procedure Parameters

Parameter	Description
p_collection_name	The name of an existing collection. Maximum length is 255 bytes. Collection names are not case sensitive and are converted to upper case.
p_c001 through p_c050	Array of character attribute values to be added.
p_n001 through p_n005	Array of numeric attribute values to be added.
p_d001 through p_d005	Array of date attribute values to be added.

**Table 5-3 (Cont.) ADD\_MEMBERS Procedure Parameters**

Parameter	Description
p_generate_md5	Valid values include YES and NO. YES to specify if the message digest of the data of the collection member should be computed. Use this parameter to compare the MD5 of the collection member with another member or to see if that member has changed.

**Example**

The following example shows how to add two new members to the EMPLOYEE table.

```
Begin
  APEX_COLLECTION.ADD_MEMBERS(
    p_collection_name => 'EMPLOYEE',
    p_c001 => l_arr1,
    p_c002 => l_arr2);
End;
```

**See Also:** ["GET\\_MEMBER\\_MD5 Function"](#) on page 5-46, ["ADD\\_MEMBER Procedure"](#) on page 5-24, ["ADD\\_MEMBER Function"](#) on page 5-26

## COLLECTION\_EXISTS Function

Use this function to determine if a collection exists. A `TRUE` is returned if the specified collection exists for the current user in the current session for the current Application ID, otherwise `FALSE` is returned.

### Syntax

```
APEX_COLLECTION.COLLECTION_EXISTS (  
    p_collection_name IN VARCHAR2)  
RETURN BOOLEAN;
```

### Parameters

[Table 5–4](#) describes the parameters available in the `COLLECTION_EXISTS` function.

**Table 5–4** *COLLECTION\_EXISTS Function Parameters*

Parameter	Description
<code>p_collection_name</code>	The name of the collection. Maximum length is 255 bytes. The collection name is not case sensitive and is converted to upper case.

### Example

The following example shows how to use the `COLLECTION_EXISTS` function to determine if the collection named `EMPLOYEES` exists.

```
Begin  
    l_exists := APEX_COLLECTION.COLLECTION_EXISTS (  
        p_collection_name => 'EMPLOYEES');  
End;
```



---

## COLLECTION\_HAS\_CHANGED Function

Use this function to determine if a collection has changed since it was created or the collection changed flag was reset.

### Syntax

```
APEX_COLLECTION.COLLECTION_HAS_CHANGED (
    p_collection_name IN VARCHAR2)
RETURN BOOLEAN;
```

### Parameters

[Table 5-5](#) describes the parameters available in the `COLLECTION_HAS_CHANGED` function.

**Table 5-5** *COLLECTION\_HAS\_CHANGED Function Parameters*

Parameter	Description
<code>p_collection_name</code>	The name of the collection. An error is returned if this collection does not exist with the specified name of the current user and in the same session.

### Example

The following example shows how to use the `COLLECTION_HAS_CHANGED` function to determine if the `EMPLOYEES` collection has changed since it was created or last reset.

```
Begin
    l_exists := APEX_COLLECTION.COLLECTION_HAS_CHANGED (
        p_collection_name => 'EMPLOYEES');
End;
```

## COLLECTION\_MEMBER\_COUNT Function

Use this function to get the total number of members for the named collection. If gaps exist, the total member count returned is not equal to the highest sequence ID in the collection. If the named collection does not exist for the current user in the current session, an error is raised.

### Syntax

```
APEX_COLLECTION.COLLECTION_MEMBER_COUNT (  
    p_collection_name IN VARCHAR2)  
RETURN NUMBER;
```

### Parameters

[Table 5–6](#) describes the parameters available in the `COLLECTION_MEMBER_COUNT` function.

**Table 5–6** *COLLECTION\_MEMBER\_COUNT Function Parameters*

Parameter	Description
<code>p_collection_name</code>	The name of the collection.

### Example

This example shows how to use the `COLLECTION_MEMBER_COUNT` function to get the total number of members in the `DEPARTMENTS` collection.

```
Begin  
    l_count := APEX_COLLECTION.COLLECTION_MEMBER_COUNT( p_collection_name =>  
        'DEPARTMENTS');  
End;
```

---

## CREATE\_COLLECTION Procedure

Use this procedure to create an empty collection that does not already exist. If a collection exists with the same name for the current user in the same session for the current Application ID, an application error is raised.

### Syntax

```
APEX_COLLECTION.CREATE_COLLECTION(
    p_collection_name IN VARCHAR2);
```

### Parameters

[Table 5-7](#) describes the parameters available in the CREATE\_COLLECTION procedure.

**Table 5-7 CREATE\_COLLECTION Procedure Parameters**

Parameter	Description
p_collection_name	The name of the collection. The maximum length is 255 characters. An error is returned if this collection exists with the specified name of the current user and in the same session.

### Example

This example shows how to use the CREATE\_COLLECTION procedure to create an empty collection named EMPLOYEES.

```
Begin
    APEX_COLLECTION.CREATE_COLLECTION(
        p_collection_name => 'EMPLOYEES');
End;
```

**See Also:** ["CREATE\\_OR\\_TRUNCATE\\_COLLECTION Procedure"](#) on page 5-34, ["CREATE\\_COLLECTION\\_FROM\\_QUERY Procedure"](#) on page 5-35, ["CREATE\\_COLLECTION\\_FROM\\_QUERY2 Procedure"](#) on page 5-36, ["CREATE\\_COLLECTION\\_FROM\\_QUERY\\_B Procedure"](#) on page 5-37, ["CREATE\\_COLLECTION\\_FROM\\_QUERYB2 Procedure"](#) on page 5-39

---

## CREATE\_OR\_TRUNCATE\_COLLECTION Procedure

Use this procedure to create a collection. If a collection exists with the same name for the current user in the same session for the current Application ID, all members of the collection are removed. In other words, the named collection is truncated.

### Syntax

```
APEX_COLLECTION.CREATE_OR_TRUNCATE_COLLECTION(  
    p_collection_name IN VARCHAR2);
```

### Parameters

[Table 5–8](#) describes the parameters available in the CREATE\_OR\_TRUNCATE\_COLLECTION procedure.

**Table 5–8 CREATE\_OR\_TRUNCATE\_COLLECTION Procedure Parameters**

Parameter	Description
p_collection_name	The name of the collection. The maximum length is 255 characters. All members of the named collection are removed if the named collection exists for the current user in the current session.

### Example

This example shows how to use the CREATE\_OR\_TRUNCATE\_COLLECTION procedure to remove all members in an existing collection named EMPLOYEES.

```
Begin  
    APEX_COLLECTION.CREATE_OR_TRUNCATE_COLLECTION(  
        p_collection_name => 'EMPLOYEES');  
End;
```

**See Also:** ["CREATE\\_COLLECTION Procedure"](#) on page 5-33,,  
["CREATE\\_COLLECTION\\_FROM\\_QUERY Procedure"](#) on page 5-35,  
["CREATE\\_COLLECTION\\_FROM\\_QUERY2 Procedure"](#) on page 5-36,  
["CREATE\\_COLLECTION\\_FROM\\_QUERY\\_B Procedure"](#) on page 5-37,  
["CREATE\\_COLLECTION\\_FROM\\_QUERYB2 Procedure"](#) on page 5-39

## CREATE\_COLLECTION\_FROM\_QUERY Procedure

Use this procedure to create a collection from a supplied query. The query is parsed as the application owner. This method can be used with a query with up to 50 columns in the `SELECT` clause. These columns in the `SELECT` clause populates the 50 character attributes of the collection (C001 through C050). If a collection exists with the same name for the current user in the same session for the current Application ID, an application error is raised.

### Syntax

```
APEX_COLLECTION.CREATE_COLLECTION_FROM_QUERY (
    p_collection_name IN VARCHAR2,
    p_query IN VARCHAR2,
    p_generate_md5 IN VARCHAR2 default 'NO');
```

### Parameters

[Table 5–9](#) describes the parameters available in the `CREATE_COLLECTION_FROM_QUERY` procedure.

**Table 5–9 CREATE\_COLLECTION\_FROM\_QUERY Procedure Parameters**

Parameter	Description
<code>p_collection_name</code>	The name of the collection. The maximum length is 255 characters. An error is returned if this collection exists with the specified name of the current user and in the same session.
<code>p_query</code>	Query to execute to populate the members of the collection. If <code>p_query</code> is numeric, it is assumed to be a <code>DBMS_SQL</code> cursor.
<code>p_generate_md5</code>	Valid values include <code>YES</code> and <code>NO</code> . <code>YES</code> to specify if the message digest of the data of the collection member should be computed. Use this parameter to compare the MD5 of the collection member with another member or to see if that member has changed.

### Example

The following example shows how to use the `CREATE_COLLECTION_FROM_QUERY` procedure to create a collection named `AUTO` and populate it with data from the `AUTOS` table. Because `p_generate_md5` is `'YES'`, the MD5 checksum is computed to allow comparisons to determine change status.

```
Begin
    l_query := 'select make, model, year from AUTOS';
    APEX_COLLECTION.CREATE_COLLECTION_FROM_QUERY (
        p_collection_name => 'AUTO',
        p_query => l_query,
        p_generate_md5 => 'YES');
End;
```

**See Also:** ["GET\\_MEMBER\\_MD5 Function"](#) on page 5-46, ["CREATE\\_COLLECTION Procedure"](#) on page 5-33, ["CREATE\\_OR\\_TRUNCATE\\_COLLECTION Procedure"](#) on page 5-34, ["CREATE\\_COLLECTION\\_FROM\\_QUERY2 Procedure"](#) on page 5-36, ["CREATE\\_COLLECTION\\_FROM\\_QUERY\\_B Procedure"](#) on page 5-37, ["CREATE\\_COLLECTION\\_FROM\\_QUERYB2 Procedure"](#) on page 5-39

## CREATE\_COLLECTION\_FROM\_QUERY2 Procedure

Use this procedure to create a collection from a supplied query. This method is identical to `CREATE_COLLECTION_FROM_QUERY`, however, the first 5 columns of the `SELECT` clause must be numeric and the next 5 must be date. After the numeric and date columns, there can be up to 50 character columns in the `SELECT` clause. The query is parsed as the application owner. If a collection exists with the same name for the current user in the same session for the current Application ID, an application error is raised.

### Syntax

```
APEX_COLLECTION.CREATE_COLLECTION_FROM_QUERY2 (
    p_collection_name IN VARCHAR2,
    p_query IN VARCHAR2,
    p_generate_md5 IN VARCHAR2 default 'NO');
```

### Parameters

[Table 5–10](#) describes the parameters available in the `CREATE_COLLECTION_FROM_QUERY2` procedure.

**Table 5–10 CREATE\_COLLECTION\_FROM\_QUERY2 Procedure Parameters**

Parameter	Description
<code>p_collection_name</code>	The name of the collection. The maximum length is 255 characters. An error is returned if this collection exists with the specified name of the current user and in the same session.
<code>p_query</code>	Query to execute to populate the members of the collection. If <code>p_query</code> is numeric, it is assumed to be a <code>DBMS_SQL</code> cursor.
<code>p_generate_md5</code>	Valid values include <code>YES</code> and <code>NO</code> . <code>YES</code> to specify if the message digest of the data of the collection member should be computed. Use this parameter to compare the MD5 of the collection member with another member or to see if that member has changed.

### Example

The following example shows how to use the `CREATE_COLLECTION_FROM_QUERY2` procedure to create a collection named `EMPLOYEE` and populate it with data from the `EMP` table. The first five columns (`mgr`, `sal`, `comm`, `deptno`, and `null`) are all numeric. Because `p_generate_md5` is `'NO'`, the MD5 checksum is not computed.

```
begin;
    APEX_COLLECTION.CREATE_COLLECTION_FROM_QUERY2 (
        p_collection_name => 'EMPLOYEE',
        p_query => 'select empno, sal, comm, deptno, null, hiredate, null, null,
null, null, ename, job, mgr from emp',
        p_generate_md5 => 'NO');
end;
```

**See Also:** ["GET\\_MEMBER\\_MD5 Function"](#) on page 5-46, ["CREATE\\_COLLECTION Procedure"](#) on page 5-33, ["CREATE\\_OR\\_TRUNCATE\\_COLLECTION Procedure"](#) on page 5-34, ["CREATE\\_COLLECTION\\_FROM\\_QUERY Procedure"](#) on page 5-35, ["CREATE\\_COLLECTION\\_FROM\\_QUERY\\_B Procedure"](#) on page 5-37, ["CREATE\\_COLLECTION\\_FROM\\_QUERYB2 Procedure"](#) on page 5-39

## CREATE\_COLLECTION\_FROM\_QUERY\_B Procedure

Use this procedure to create a collection from a supplied query using bulk operations. This method offers significantly faster performance than the `CREATE_COLLECTION_FROM_QUERY` method. The query is parsed as the application owner. If a collection exists with the same name for the current user in the same session for the current Application ID, an application error is raised.

This procedure uses bulk dynamic SQL to perform the fetch and insert operations into the named collection. Two limitations are imposed by this procedure:

1. The MD5 checksum for the member data is not computed.
2. No column value in query `p_query` can exceed 2,000 bytes. If a row is encountered that has a column value of more than 2,000 bytes, an error is raised during execution. In Oracle Database 11g Release 2 (11.2.0.1) or later, this column limit is 4,000 bytes.

### Syntax

```
APEX_COLLECTION.CREATE_COLLECTION_FROM_QUERY_B (
    p_collection_name IN VARCHAR2,
    p_query IN VARCHAR2,
    p_names IN apex_application_global.vc_arr2 DEFAULT,
    p_values IN apex_applicationN_globa.vc_arr2 DEFAULT,
    p_max_row_count IN NUMBER DEFAULT);
```

### Parameters

[Table 5–11](#) describes the parameters available in the `CREATE_COLLECTION_FROM_QUERY_B` procedure.

**Table 5–11 CREATE\_COLLECTION\_FROM\_QUERY\_B Procedure Parameters**

Parameter	Description
<code>p_collection_name</code>	The name of the collection. The maximum length is 255 characters. An error is returned if this collection exists with the specified name of the current user and in the same session.
<code>p_query</code>	Query to execute to populate the members of the collection. If <code>p_query</code> is numeric, it is assumed to be a <code>DBMS_SQL</code> cursor.
<code>p_names</code>	Array of bind variable names used in the query statement.
<code>p_values</code>	Array of bind variable values used in the bind variables in the query statement.
<code>p_max_row_count</code>	Maximum number of rows returned from the query in <code>p_query</code> which should be added to the collection.

### Example

The following examples shows how to use the `CREATE_COLLECTION_FROM_QUERY_B` procedure to create a collection named `AUTO` and populate it with data from the `AUTOS` table.

```
Begin
    l_query := 'select make, model, year from AUTOS';
    APEX_COLLECTION.CREATE_COLLECTION_FROM_QUERY_B (
        p_collection_name => 'AUTO',
        p_query => l_query);
End;
```

**See Also:** ["GET\\_MEMBER\\_MD5 Function"](#) on page 5-46, ["CREATE\\_COLLECTION Procedure"](#) on page 5-33, ["CREATE\\_OR\\_TRUNCATE\\_COLLECTION Procedure"](#) on page 5-34, ["CREATE\\_COLLECTION\\_FROM\\_QUERY Procedure"](#) on page 5-35, ["CREATE\\_COLLECTION\\_FROM\\_QUERY2 Procedure"](#) on page 5-36, ["CREATE\\_COLLECTION\\_FROM\\_QUERYB2 Procedure"](#) on page 5-39



## CREATE\_COLLECTION\_FROM\_QUERYB2 Procedure

Use this procedure to create a collection from a supplied query using bulk operations. This method offers significantly faster performance than the `CREATE_COLLECTION_FROM_QUERY_2` method. The query is parsed as the application owner. If a collection exists with the same name for the current user in the same session for the current Application ID, an application error is raised. It is identical to the `CREATE_COLLECTION_FROM_QUERY_B`, however, the first five columns of the `SELECT` clause must be numeric and the next five columns must be date. After the date columns, there can be up to 50 character columns in the `SELECT` clause

This procedure uses bulk dynamic SQL to perform the fetch and insert operations into the named collection. Two limitations are imposed by this procedure:

1. The MD5 checksum for the member data is not computed.
2. No column value in query `p_query` can exceed 2,000 bytes. If a row is encountered that has a column value of more than 2,000 bytes, an error is raised during execution. In Oracle Database 11g Release 2 (11.2.0.1) or later, this column limit is 4,000 bytes.

### Syntax

```
APEX_COLLECTION.CREATE_COLLECTION_FROM_QUERYB2 (
    p_collection_name IN VARCHAR2,
    p_query IN VARCHAR2,
    p_names IN apex_application_global.vc_arr2 DEFAULT,
    p_values IN apex_applicationN_globa.vc_arr2 DEFAULT,
    p_max_row_count IN NUMBER DEFAULT);
```

### Parameters

[Table 5–12](#) describes the parameters available in the `CREATE_COLLECTION_FROM_QUERYB2` procedure.

**Table 5–12 CREATE\_COLLECTION\_FROM\_QUERYB2 Procedure Parameters**

Parameter	Description
<code>p_collection_name</code>	The name of the collection. The maximum length is 255 characters. An error is returned if this collection exists with the specified name of the current user and in the same session.
<code>p_query</code>	Query to execute to populate the members of the collection. If <code>p_query</code> is numeric, it is assumed to be a <code>DBMS_SQL</code> cursor.
<code>p_names</code>	Array of bind variable names used in the query statement.
<code>p_values</code>	Array of bind variable values used in the bind variables in the query statement.
<code>p_max_row_count</code>	Maximum number of rows returned from the query in <code>p_query</code> which should be added to the collection.

### Example

The following example shows how to use the `CREATE_COLLECTION_FROM_QUERYB2` procedure to create a collection named `EMPLOYEES` and populate it with data from the `EMP` table. The first five columns (`mgr`, `sal`, `comm`, `deptno`, and `null`) are all numeric and the next five are all date. Because `p_generate_md5` is `'NO'`, the MD5 checksum is not computed.

```
Begin
  l_query := 'select empno, sal, comm, deptno, null, hiredate, null, null, null,
null, ename, job, mgr from emp';
  APEX_COLLECTION.CREATE_COLLECTION_FROM_QUERYB2 (
    p_collection_name => 'EMPLOYEES',
    p_query => l_query,
    p_generate_md5 => 'NO');
End;
```

**See Also:** ["GET\\_MEMBER\\_MD5 Function"](#) on page 5-46, ["CREATE\\_COLLECTION Procedure"](#) on page 5-33, ["CREATE\\_OR\\_TRUNCATE\\_COLLECTION Procedure"](#) on page 5-34, ["CREATE\\_COLLECTION\\_FROM\\_QUERY Procedure"](#) on page 5-35, ["CREATE\\_COLLECTION\\_FROM\\_QUERY2 Procedure"](#) on page 5-36, ["CREATE\\_COLLECTION\\_FROM\\_QUERY\\_B Procedure"](#) on page 5-37

---

## DELETE\_ALL\_COLLECTIONS Procedure

Use this procedure to delete all collections that belong to the current user in the current Application Express session for the current Application ID.

### Syntax

```
APEX_COLLECTION.DELETE_ALL_COLLECTIONS;
```

### Parameters

None.

### Example

This example shows how to use the DELETE\_ALL\_COLLECTIONS procedure to remove all collections that belong to the current user in the current session and Application ID.

```
Begin  
    APEX_COLLECTION.DELETE_ALL_COLLECTIONS;  
End;
```

**See Also:** ["DELETE\\_ALL\\_COLLECTIONS Procedure"](#) on page 5-41, ["DELETE\\_COLLECTION Procedure"](#) on page 5-43, ["DELETE\\_MEMBER Procedure"](#) on page 5-44, ["DELETE\\_MEMBERS Procedure"](#) on page 5-45

## DELETE\_ALL\_COLLECTIONS\_SESSION Procedure

Use this procedure to delete all collections that belong to the current user in the current Application Express session regardless of the Application ID.

### Syntax

```
APEX_COLLECTION.DELETE_ALL_COLLECTIONS_SESSION;
```

### Parameters

None.

### Example

This example shows how to use the DELETE\_ALL\_COLLECTIONS\_SESSION procedure to remove all collections that belong to the current user in the current session regardless of Application ID.

```
Begin  
    APEX_COLLECTION.DELETE_ALL_COLLECTIONS_SESSION;  
End;
```

**See Also:** ["DELETE\\_ALL\\_COLLECTIONS Procedure"](#) on page 5-41, ["DELETE\\_COLLECTION Procedure"](#) on page 5-43, ["DELETE\\_MEMBER Procedure"](#) on page 5-44, ["DELETE\\_MEMBERS Procedure"](#) on page 5-45

---

## DELETE\_COLLECTION Procedure

Use this procedure to delete a named collection. All members that belong to the collection are removed and the named collection is dropped. If the named collection does not exist for the same user in the current session for the current Application ID, an application error is raised.

### Syntax

```
APEX_COLLECTION.DELETE_COLLECTION (
    p_collection_name IN VARCHAR2);
```

### Parameters

[Table 5-13](#) describes the parameters available in the DELETE\_COLLECTION procedure.

**Table 5-13** DELETE\_COLLECTION Procedure Parameters

Parameter	Description
p_collection_name	The name of the collection to remove all members from and drop. An error is returned if this collection does not exist with the specified name of the current user and in the same session.

### Example

This example shows how to use the DELETE\_COLLECTION procedure to remove the 'EMPLOYEE' collection.

```
Begin
    APEX_COLLECTION.DELETE_COLLECTION(
        p_collection_name => 'EMPLOYEE');
End;
```

**See Also:** ["DELETE\\_ALL\\_COLLECTIONS\\_SESSION Procedure"](#) on page 5-42, ["DELETE\\_ALL\\_COLLECTIONS Procedure"](#) on page 5-41, ["DELETE\\_MEMBER Procedure"](#) on page 5-44, ["DELETE\\_MEMBERS Procedure"](#) on page 5-45

---

## DELETE\_MEMBER Procedure

Use this procedure to delete a specified member from a given named collection. If the named collection does not exist for the same user in the current session for the current Application ID, an application error is raised.

### Syntax

```
APEX_COLLECTION.DELETE_MEMBER (  
    p_collection_name IN VARCHAR2,  
    p_seq IN VARCHAR2);
```

### Parameters

[Table 5–14](#) describes the parameters available in the `DELETE_MEMBER` procedure.

**Table 5–14** *DELETE\_MEMBER Parameters*

Parameter	Description
<code>p_collection_name</code>	The name of the collection to delete the specified member from. The maximum length is 255 characters. Collection names are not case sensitive and are converted to upper case. An error is returned if this collection does not exist for the current user in the same session.
<code>p_seq</code>	This is the sequence ID of the collection member to be deleted.

### Example

This example shows how to use the `DELETE_MEMBER` procedure to remove the member with a sequence ID of '2' from the collection named `EMPLOYEES`.

```
Begin  
    APEX_COLLECTION.DELETE_MEMBER(  
        p_collection_name => 'EMPLOYEES',  
        p_seq => '2');  
End;
```

**See Also:** ["DELETE\\_ALL\\_COLLECTIONS\\_SESSION Procedure"](#) on page 5-42, ["DELETE\\_ALL\\_COLLECTIONS Procedure"](#) on page 5-41, ["DELETE\\_COLLECTION Procedure"](#) on page 5-43, ["DELETE\\_MEMBERS Procedure"](#) on page 5-45

## DELETE\_MEMBERS Procedure

Use this procedure to delete all members from a given named collection where the attribute specified by the attribute number equals the supplied value. If the named collection does not exist for the same user in the current session for the current Application ID, an application error is raised. If the attribute number specified is invalid or outside the range of 1 to 50, an error is raised.

If the supplied attribute value is null, then all members of the named collection are deleted where the attribute, specified by `p_attr_number`, is null.

### Syntax

```
APEX_COLLECTION.DELETE_MEMBERS (
    p_collection_name IN VARCHAR2,
    p_attr_number IN VARCHAR2,
    p_attr_value IN VARCHAR2);
```

### Parameters

[Table 5–14](#) describes the parameters available in the `DELETE_MEMBERS` procedure.

**Table 5–15** *DELETE\_MEMBERS Parameters*

Parameter	Description
<code>p_collection_name</code>	The name of the collection to delete the specified members from. The maximum length is 255 characters. Collection names are not case sensitive and are converted to upper case. An error is returned if this collection does not exist for the current user in the same session.
<code>p_attr_number</code>	Attribute number of the member attribute used to match for the specified attribute value for deletion. Valid values are 1 through 50 and null.
<code>p_attr_value</code>	Attribute value of the member attribute used to match for deletion. Maximum length can be 4,000 bytes. The attribute value is truncated to 4,000 bytes if greater than this amount.

### Example

The following example deletes all members of the collection named 'GROCERIES' where the 5th character attribute is equal to 'APPLE'.

```
Begin
    apex_collection.delete_members(
        p_collection_name => 'GROCERIES'
        p_attr_number     => 5,
        p_attr_value     => 'APPLE' );
    Commit;
End;
```

**See Also:** ["DELETE\\_ALL\\_COLLECTIONS\\_SESSION Procedure"](#) on page 5-42, ["DELETE\\_ALL\\_COLLECTIONS Procedure"](#) on page 5-41, ["DELETE\\_COLLECTION Procedure"](#) on page 5-43, ["DELETE\\_MEMBER Procedure"](#) on page 5-44

## GET\_MEMBER\_MD5 Function

Use this function to compute and return the message digest of the attributes for the member specified by the sequence ID. This computation of message digest is equal to the computation performed natively by collections. Thus, the result of this function could be compared to the MD5\_ORIGINAL column of the view `wwv_flow_collections`.

If a collection does not exist with the specified name for the current user in the same session and for the current Application ID, an application error is raised. If the member specified by sequence ID `p_seq` does not exist, an application error is raised.

### Syntax

```
APEX_COLLECTION.GET_MEMBER_MD5 (
    p_collection_name IN VARCHAR2,
    p_seq IN NUMBER)
RETURN VARCHAR2;
```

### Parameters

[Table 5–16](#) describes the parameters available in the `GET_MEMBER_MD5` function.

**Table 5–16** *GET\_MEMBER\_MD5 Parameters*

Parameter	Description
<code>p_collection_name</code>	The name of the collection to add this array of members to. An error is returned if this collection does not exist with the specified name of the current user and in the same session.
<code>p_seq</code>	Sequence ID of the collection member.

### Example

The following example computes the MD5 for the 5th member of the `GROCERIES` collection.

```
declare
    l_md5 varchar2(4000);
begin
    l_md5 := apex_collection.get_member_md5(
        p_collection_name => 'GROCERIES'
        p_seq              => 10 );
end;
```

**See Also:** ["COLLECTION\\_HAS\\_CHANGED Function"](#) on page 5-31, ["RESET\\_COLLECTION\\_CHANGED Procedure"](#) on page 5-52, ["RESET\\_COLLECTION\\_CHANGED\\_ALL Procedure"](#) on page 5-53



## MERGE\_MEMBERS Procedure

Use this procedure to merge members of the given named collection with the values passed in the arrays. If the named collection does not exist one is created. If a `p_init_query` is provided, the collection is created from the supplied SQL query. If the named collection exists, the following occurs:

1. Rows in the collection and not in the arrays are deleted.
2. Rows in the collections and in the arrays are updated.
3. Rows in the arrays and not in the collection are inserted.

The count of elements in the `p_c001` PL/SQL table is used as the total number of items across all PL/SQL tables. For example, if `p_c001.count` is 2 and `p_c002.count` is 10, only 2 members are merged. If `p_c001` is null an application error is raised.

### Syntax

```
APEX_COLLECTION.MERGE_MEMBERS (
    p_collection_name IN VARCHAR2,
    p_seq      IN APEX_APPLICATION_GLOBAL.VC_ARR2 DEFAULT empty_vc_arr,
    p_c001    IN APEX_APPLICATION_GLOBAL.VC_ARR2 DEFAULT empty_vc_arr,
    p_c002    IN APEX_APPLICATION_GLOBAL.VC_ARR2 DEFAULT empty_vc_arr,
    p_c003    IN APEX_APPLICATION_GLOBAL.VC_ARR2 DEFAULT empty_vc_arr,
    ...
    p_c050    IN APEX_APPLICATION_GLOBAL.VC_ARR2 DEFAULT empty_vc_arr,
    p_null_index IN NUMBER DEFAULT 1,
    p_null_value IN VARCHAR2 DEFAULT null,
    p_init_query IN VARCHAR2 DEFAULT null);
```

### Parameters

Table 5–17 describes the parameters available in the `MERGE_MEMBERS` procedure.

---



---

**Note:** Any character attribute exceeding 4,000 characters is truncated to 4,000 characters. Also, the number of members added is based on the number of elements in the first array.

---



---

**Table 5–17 MERGE\_MEMBERS Procedure Parameters**

Parameter	Description
<code>p_collection_name</code>	The name of the collection. Maximum length is 255 bytes. Collection names are not case sensitive and are converted to upper case.
<code>p_c001</code> through <code>p_c050</code>	Array of attribute values to be merged. Maximum length is 4,000 bytes. Any character attribute exceeding 4,000 characters is truncated to 4,000 characters. The count of the <code>p_c001</code> array is used across all arrays. If no values are provided then no actions are performed.
<code>p_c0xx</code>	Attribute of NN attributes values to be merged. Maximum length can be 4,000 bytes. The attribute value is truncated to 4,000 bytes if greater than this amount.
<code>p_seq</code>	Identifies the sequence number of the collection to be merged.

**Table 5–17 (Cont.) MERGE\_MEMBERS Procedure Parameters**

Parameter	Description
p_null_index	That is if the element identified by this value is null, then treat this row as a null row. For example, if p_null_index is 3, then p_c003 is treated as a null row. In other words, tell the merge function to ignore this row. This results in the null rows being removed from the collection. The null index works with the null value. If the value of the p_cXXX argument is equal to the p_null_value then the row is treated as null.
p_null_value	Used with the p_null_index argument. Identifies the null value. If used, this value must not be null. A typical value for this argument is "0"
p_init_query	If the collection does not exist, the collection is created using this query.

**Example**

The following example creates a collection on the table of employees, and then merges the contents of the local arrays with the collection, updating the job of two employees.

```

DECLARE
    l_seq    APEX_APPLICATION_GLOBAL.VC_ARR2;
    l_c001  APEX_APPLICATION_GLOBAL.VC_ARR2;
    l_c002  APEX_APPLICATION_GLOBAL.VC_ARR2;
    l_c003  APEX_APPLICATION_GLOBAL.VC_ARR2;
BEGIN
    l_seq(1) := 1;
    l_c001(1) := 7369;
    l_c002(1) := 'SMITH';
    l_c003(1) := 'MANAGER';
    l_seq(2) := 2;
    l_c001(2) := 7499;
    l_c002(2) := 'ALLEN';
    l_c003(2) := 'CLERK';

    APEX_COLLECTION.MERGE_MEMBERS(
        p_collection_name => 'EMPLOYEES',
        p_seq => l_seq,
        p_c001 => l_c001,
        p_c002 => l_c002,
        p_c003 => l_c003,
        p_init_query => 'select empno, ename, job from emp order by empno');
END;
```

## MOVE\_MEMBER\_DOWN Procedure

Use this procedure to adjust the sequence ID of specified member in the given named collection down by one (subtract one), swapping sequence ID with the one it is replacing. For example, 3 becomes 2 and 2 becomes 3. If a collection does not exist with the specified name for the current user in the same session and for the current Application ID, an application error is raised. If the member specified by sequence ID p\_seq does not exist, an application error is raised. If the member specified by sequence ID p\_seq is the lowest sequence in the collection, an application error is NOT returned.

### Syntax

```
APEX_COLLECTION.MOVE_MEMBER_DOWN (
    p_collection_name IN VARCHAR2,
    p_seq IN NUMBER);
```

### Parameters

[Table 5–19](#) describes the parameters available in the MOVE\_MEMBER\_DOWN procedure.

**Table 5–18 MOVE\_MEMBER\_DOWN Parameters**

Parameter	Description
p_collection_name	The name of the collection. Maximum length is 255 bytes. Collection names are not case sensitive and are converted to upper case. An error is returned if this collection does not exist with the specified name of the current user in the same session.
p_seq	Identifies the sequence number of the collection member to be moved down by one.

### Example

This example shows how to a member of the EMPLOYEES collection down one position. After executing this example, sequence ID '5' becomes sequence ID '4' and sequence ID '4' becomes sequence ID '5'.

```
BEGIN;
    APEX_COLLECTION.MOVE_MEMBER_DOWN(
        p_collection_name => 'EMPLOYEES',
        p_seq => '5' );
END;
```

**See Also:** ["MOVE\\_MEMBER\\_UP Procedure"](#) on page 5-50

## MOVE\_MEMBER\_UP Procedure

Use this procedure to adjust the sequence ID of specified member in the given named collection up by one (add one), swapping sequence ID with the one it is replacing. For example, 2 becomes 3 and 3 becomes 2. If a collection does not exist with the specified name for the current user in the same session and for the current Application ID, an application error is raised. If the member specified by sequence ID `p_seq` does not exist, an application error is raised. If the member specified by sequence ID `p_seq` is the highest sequence in the collection, an application error is not returned.

### Syntax

```
APEX_COLLECTION.MOVE_MEMBER_UP (
    p_collection_name IN VARCHAR2,
    p_seq IN NUMBER);
```

### Parameters

[Table 5–19](#) describes the parameters available in the `MOVE_MEMBER_UP` procedure.

**Table 5–19** *MOVE\_MEMBER\_UP Parameters*

Parameter	Description
<code>p_collection_name</code>	The name of the collection. Maximum length is 255 bytes. Collection names are not case sensitive and are converted to upper case. An error is returned if this collection does not exist with the specified name of the current user in the same session.
<code>p_seq</code>	Identifies the sequence number of the collection member to be moved up by one.

### Example

This example shows how to a member of the `EMPLOYEES` collection down one position. After executing this example, sequence ID '5' becomes sequence ID '6' and sequence ID '6' becomes sequence ID '5'.

```
BEGIN;
    APEX_COLLECTION.MOVE_MEMBER_UP(
        p_collection_name => 'EMPLOYEES',
        p_seq => '5' );
END;
```

**See Also:** ["MOVE\\_MEMBER\\_DOWN Procedure"](#) on page 5-49

---

## RESEQUENCE\_COLLECTION Procedure

For a named collection, use this procedure to update the `seq_id` value of each member so that no gaps exist in the sequencing. For example, a collection with the following set of sequence IDs (1,2,3,5,8,9) becomes (1,2,3,4,5,6). If a collection does not exist with the specified name for the current user in the same session and for the current Application ID, an application error is raised.

### Syntax

```
APEX_COLLECTION.RESEQUENCE_COLLECTION (
    p_collection_name IN VARCHAR2);
```

### Parameters

[Table 5–20](#) describes the parameters available in the `RESEQUENCE_COLLECTION` procedure.

**Table 5–20** *RESEQUENCE\_COLLECTION Parameters*

Parameter	Description
<code>p_collection_name</code>	The name of the collection to resequence. An error is returned if this collection does not exist with the specified name of the current user and in the same session.

### Example

This example shows how to resequence the `DEPARTMENTS` collection to remove gaps in the sequence IDs.

```
BEGIN;
    APEX_COLLECTION.RESEQUENCE_COLLECTION (
        p_collection_name => 'DEPARTMENTS');
END;
```

**See Also:** "[MOVE\\_MEMBER\\_DOWN Procedure](#)" on page 5-49, "[MOVE\\_MEMBER\\_UP Procedure](#)" on page 5-50

---

## RESET\_COLLECTION\_CHANGED Procedure

Use this procedure to reset the collection changed flag (mark as not changed) for a given collection. If a collection does not exist with the specified name for the current user in the same session and for the current Application ID, an application error is raised.

### Syntax

```
APEX_COLLECTION.RESET_COLLECTION_CHANGED (  
    p_collection_name IN VARCHAR2);
```

### Parameters

[Table 5–21](#) describes the parameters available in the RESET\_COLLECTION\_CHANGED procedure.

**Table 5–21** RESET\_COLLECTION\_CHANGED Parameters

Parameter	Description
p_collection_name	The name of the collection to reset the collection changed flag. An error is returned if this collection does not exist with the specified name of the current user and in the same session.

### Example

This example shows how to reset the changed flag for the DEPARTMENTS collection.

```
BEGIN;  
    APEX_COLLECTION.RESET_COLLECTION_CHANGED (  
        p_collection_name => 'DEPARTMENTS');  
END;
```

**See Also:** ["RESET\\_COLLECTION\\_CHANGED\\_ALL Procedure"](#) on page 5-53

## RESET\_COLLECTION\_CHANGED\_ALL Procedure

Use this procedure to reset the collection changed flag (mark as not changed) for all collections in the user's current session.

### Syntax

```
APEX_COLLECTION.RESET_COLLECTION_CHANGED_ALL; (
```

### Parameters

None.

### Example

This example shows how to reset the changed flag for all collections in the user's current session.

```
BEGIN;  
    APEX_COLLECTION.RESET_COLLECTION_CHANGED_ALL;  
END;
```

**See Also:** ["RESET\\_COLLECTION\\_CHANGED Procedure"](#) on page 5-52

---

## SORT\_MEMBERS Procedure

Use this procedure to reorder the members of a given collection by the column number specified by `p_sort_on_column_number`. This sorts the collection by a particular column/attribute in the collection and reassigns the sequence IDs of each number such that no gaps exist. If a collection does not exist with the specified name for the current user in the same session and for the current Application ID, an application error is raised.

### Syntax

```
APEX_COLLECTION.SORT_MEMBERS (  
    p_collection_name IN VARCHAR2,  
    p_sort_on_column_number IN NUMBER);
```

### Parameters

[Table 5–22](#) describes the parameters available in the `SORT_MEMBERS` procedure.

**Table 5–22** SORT\_MEMBERS Parameters

Parameter	Description
<code>p_collection_name</code>	The name of the collection to sort. An error is returned if this collection does not exist with the specified name of the current user and in the same session.
<code>p_sort_on_column_number</code>	The column number used to sort the collection. The domain of possible values is 1 to 50.

### Example

In this example, column 2 of the `DEPARTMENTS` collection is the department location. The collection is reorder according to the department location.

```
BEGIN;  
    APEX_COLLECTION.SORT_MEMBERS (  
        p_collection_name => 'DEPARTMENTS',  
        p_sort_on_column_number => '2';  
END;
```



---

## TRUNCATE\_COLLECTION Procedure

Use this procedure to remove all members from a named collection. If a collection does not exist with the specified name for the current user in the same session and for the current Application ID, an application error is raised.

### Syntax

```
APEX_COLLECTION.TRUNCATE_COLLECTION (  
    p_collection_name IN VARCHAR2);
```

### Parameters

[Table 5-23](#) describes the parameters available in the TRUNCATE\_COLLECTION procedure.

**Table 5-23** TRUNCATE\_COLLECTION Parameters

Parameter	Description
p_collection_name	The name of the collection to truncate. An error is returned if this collection does not exist with the specified name of the current user and in the same session.

### Example

This example shows how to remove all members from the DEPARTMENTS collection.

```
BEGIN;  
    APEX_COLLECTION.TRUNCATE_COLLECTION(  
        p_collection_name => 'DEPARTMENTS');  
END;
```

**See Also:** ["CREATE\\_OR\\_TRUNCATE\\_COLLECTION Procedure"](#) on page 5-34

## UPDATE\_MEMBER Procedure

Use this procedure to update the specified member in the given named collection. If a collection does not exist with the specified name for the current user in the same session and for the current Application ID, an application error is raised. If the member specified by sequence ID p\_seq does not exist, an application error is raised.

---

**Note:** Using this procedure sets the columns identified and nullifies any columns not identified. To update specific columns, without affecting the values of other columns, use "[UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 1](#)" on page 5-60.

---

### Syntax

```
APEX_COLLECTION.UPDATE_MEMBER (
    p_collection_name IN VARCHAR2,
    p_seq IN VARCHAR2 DEFAULT NULL,
    p_c001 IN VARCHAR2 DEFAULT NULL,
    p_c002 IN VARCHAR2 DEFAULT NULL,
    p_c003 IN VARCHAR2 DEFAULT NULL,
    ...
    p_c050 IN VARCHAR DEFAULT NULL,
    p_n001 IN NUMBER DEFAULT NULL,
    p_n002 IN NUMBER DEFAULT NULL,
    p_n003 IN NUMBER DEFAULT NULL,
    p_n004 IN NUMBER DEFAULT NULL,
    p_n005 IN NUMBER DEFAULT NULL,
    p_d001 IN DATE DEFAULT NULL,
    p_d002 IN DATE DEFAULT NULL,
    p_d003 IN DATE DEFAULT NULL,
    p_d004 IN DATE DEFAULT NULL,
    p_d005 IN DATE DEFAULT NULL,
    p_clob001 IN CLOB DEFAULT empty_clob(),
    p_blob001 IN BLOB DEFAULT empty_blob(),
    p_xmltype001 IN XMLTYPE DEFAULT NULL);
```

### Parameters

[Table 5–24](#) describes the parameters available in the UPDATE\_MEMBER procedure.

---

**Note:** Any character attribute exceeding 4,000 characters is truncated to 4,000 characters. Also, the number of members added is based on the number of elements in the first array.

---

**Table 5–24** UPDATE\_MEMBER Parameters

Parameter	Description
p_collection_name	The name of the collection to update. Maximum length is 255 bytes. Collection names are not case sensitive and are converted to upper case.
p_c001 through p_c050	Attribute value of the member to be added. Maximum length is 4,000 bytes. Any character attribute exceeding 4,000 characters is truncated to 4,000 characters.

**Table 5–24 (Cont.) UPDATE\_MEMBER Parameters**

Parameter	Description
p_n001 through p_n005	Attribute value of the numeric attributes to be added or updated.
p_d001 through p_d005	Attribute value of the date attributes to be added or updated.
p_clob001	Use p_clob001 for collection member attributes that exceed 4,000 characters.
p_blob001	Use p_blob001 for binary collection member attributes.
p_xmltype001	Use p_xmltype001 to store well-formed XML.

**Example**

Update the second member of the collection named 'Departments', updating the first member attribute to 'Engineering' and the second member attribute to 'Sales'.

```
BEGIN;
  APEX_COLLECTION.UPDATE_MEMBER (
    p_collection_name => 'Departments',
    p_seq => '2',
    p_c001 => 'Engineering',
    p_c002 => 'Sales');
```

**See Also:** ["UPDATE\\_MEMBERS Procedure"](#) on page 5-58

## UPDATE\_MEMBERS Procedure

Use this procedure to update the array of members for the given named collection. If a collection does not exist with the specified name for the current user in the same session and for the current Application ID, an application error is raised. The count of elements in the p\_seq PL/SQL table is used as the total number of items across all PL/SQL tables. That is, if p\_seq.count = 2 and p\_c001.count = 10, only 2 members are updated. If p\_seq is null, an application error is raised. If the member specified by sequence ID p\_seq does not exist, an application error is raised.

### Syntax

```
APEX_COLLECTION.UPDATE_MEMBERS (
    p_collection_name IN VARCHAR2,
    p_seq IN apex_application_global.VC_ARR2 DEFAULT empty_vc_arr,
    p_c001 IN apex_application_global.VC_ARR2 DEFAULT empty_vc_arr,
    p_c002 IN apex_application_global.VC_ARR2 DEFAULT empty_vc_arr,
    p_c003 IN apex_application_global.VC_ARR2 DEFAULT empty_vc_arr,
    ...
    p_c050 IN apex_application_global.VC_ARR2 DEFAULT empty_vc_arr,
    p_n001 IN apex_application_global.N_ARR DEFAULT empty_n_arr,
    p_n002 IN apex_application_global.N_ARR DEFAULT empty_n_arr,
    p_n003 IN apex_application_global.N_ARR DEFAULT empty_n_arr,
    p_n004 IN apex_application_global.N_ARR DEFAULT empty_n_arr,
    p_n005 IN apex_application_global.N_ARR DEFAULT empty_n_arr,
    p_d001 IN apex_application_global.D_ARR DEFAULT empty_d_arr,
    p_d002 IN apex_application_global.D_ARR DEFAULT empty_d_arr,
    p_d003 IN apex_application_global.D_ARR DEFAULT empty_d_arr,
    p_d004 IN apex_application_global.D_ARR DEFAULT empty_d_arr,
    p_d005 IN apex_application_global.D_ARR DEFAULT empty_d_arr)
```

### Parameters

[Table 5–25](#) describes the parameters available in the UPDATE\_MEMBERS procedure.

---



---

**Note:** Any character attribute exceeding 4,000 characters is truncated to 4,000 characters. Also, the number of members added is based on the number of elements in the first array.

---



---

**Table 5–25** UPDATE\_MEMBERS Parameters

Parameter	Description
p_collection_name	The name of the collection to update. Maximum length is 255 bytes. Collection names are not case sensitive and are converted to upper case.
p_seq	Array of member sequence IDs to be updated. The count of the p_seq array is used across all arrays.
p_c001 through p_c050	Array of attribute values to be updated.
p_n001 through p_n005	Attribute value of numeric
p_d001 through p_d005	Array of date attribute values to be updated.

### Example

```
DECLARE
    l_seq apex_application_global.vc_arr2;
```

```
l_carr apex_application_global.vc_arr2;
l_narr apex_application_global.n_arr;
l_darr apex_application_global.d_arr;
BEGIN
  l_seq(1) := 10;
  l_seq(2) := 15;
  l_carr(1) := 'Apples';
  l_carr(2) := 'Grapes';
  l_narr(1) := 100;
  l_narr(2) := 150;
  l_darr(1) := sysdate;
  l_darr(2) := sysdate;

  APEX_COLLECTION.UPDATE_MEMBERS (
    p_collection_name => 'Groceries',
    p_seq => l_seq,
    p_c001 => l_carr,
    p_n001 => l_narr,
    p_d001 => l_darr);
END;
```

**See Also:** ["UPDATE\\_MEMBER Procedure"](#) on page 5-56

## UPDATE\_MEMBER\_ATTRIBUTE Procedure Signature 1

Update the specified member attribute in the given named collection. If a collection does not exist with the specified name for the current user in the same session for the current Application ID, an application error is raised. If the member specified by sequence ID `p_seq` does not exist, an application error is raised. If the attribute number specified is invalid or outside the range 1-50, an error is raised. Any attribute value exceeding 4,000 bytes are truncated to 4,000 bytes.

### Syntax

```
APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (
  p_collection_name IN VARCHAR2,
  p_seq IN VARCHAR2,
  p_attr_number IN VARCHAR2,
  p_attr_value IN VARCHAR2);
```

### Parameters

Table 5–26 describes the parameters available in the UPDATE\_MEMBER\_ATTRIBUTE procedure signature 1.

---



---

**Note:** Any character attribute exceeding 4,000 characters is truncated to 4,000 characters. Also, the number of members added is based on the number of elements in the first array.

---



---

**Table 5–26** UPDATE\_MEMBER\_ATTRIBUTE Signature 1 Parameters

Parameter	Description
<code>p_collection_name</code>	The name of the collection. Maximum length can be 255 bytes. Collection names are case-insensitive, as the collection name is converted to upper case. An error is returned if this collection does not exist with the specified name of the current user and in the same session.
<code>p_seq</code>	Sequence ID of the collection member to be updated.
<code>p_attr_number</code>	Attribute number of the member attribute to be updated. Valid values are 1 through 50. Any number outside of this range is ignored.
<code>p_attr_value</code>	Attribute value of the member attribute to be updated.

### Example

Update the second member of the collection named 'Departments', updating the first member attribute to 'Engineering' and the second member attribute to 'Sales'.

```
BEGIN;
  APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (
    p_collection_name => 'Departments',
    p_seq => '2',
    p_attr_number => '1',
    p_attr_value => 'Engineering');
END;
```

**See Also:** ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 2"](#) on page 5-62, ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 3"](#) on page 5-64, ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 4"](#) on page 5-66, ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 5"](#) on page 5-68, ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 6"](#) on page 5-70

## UPDATE\_MEMBER\_ATTRIBUTE Procedure Signature 2

Update the specified CLOB member attribute in the given named collection. If a collection does not exist with the specified name for the current user in the same session for the current Application ID, an application error is raised. If the member specified by sequence ID `p_seq` does not exist, an application error is raised. If the attribute number specified is invalid or outside the valid range (currently only 1 for CLOB), an error is raised.

### Syntax

```
APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (
  p_collection_name IN VARCHAR2,
  p_seq IN VARCHAR2,
  p_clob_number IN NUMBER,
  p_clob_value IN CLOB);
```

### Parameters

Table 5–27 describes the parameters available in the UPDATE\_MEMBER\_ATTRIBUTE procedure signature 2.

---



---

**Note:** Any character attribute exceeding 4,000 characters is truncated to 4,000 characters. Also, the number of members added is based on the number of elements in the first array.

---



---

**Table 5–27** UPDATE\_MEMBER\_ATTRIBUTE Signature 2 Parameters

Parameter	Description
<code>p_collection_name</code>	The name of the collection. Maximum length can be 255 bytes. Collection names are case-insensitive, as the collection name is converted to upper case. An error is returned if this collection does not exist with the specified name of the current user and in the same session.
<code>p_seq</code>	Sequence ID of the collection member to be updated.
<code>p_clob_number</code>	Attribute number of the CLOB member attribute to be updated. Valid value is 1. Any number outside of this range is ignored.
<code>p_clob_value</code>	Attribute value of the CLOB member attribute to be updated.

### Example

The following example sets the first and only CLOB attribute of collection sequence number 2 in the collection named 'Departments' to a value of 'Engineering'.

```
BEGIN;
  APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (
    p_collection_name => 'Departments',
    p_seq => '2',
    p_clob_number => '1',
    p_clob_value => 'Engineering');
END;
```



**See Also:** ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 1" on page 5-60](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 3" on page 5-64](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 4" on page 5-66](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 5" on page 5-68](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 6" on page 5-70](#)

## UPDATE\_MEMBER\_ATTRIBUTE Procedure Signature 3

Update the specified BLOB member attribute in the given named collection. If a collection does not exist with the specified name for the current user in the same session for the current Application ID, an application error is raised. If the member specified by sequence ID `p_seq` does not exist, an application error is raised. If the attribute number specified is invalid or outside the valid range (currently only 1 for BLOB), an error is raised.

### Syntax

```
APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (
  p_collection_name IN VARCHAR2,
  p_seq IN VARCHAR2,
  p_blob_number IN NUMBER,
  p_blob_value IN BLOB);
```

### Parameters

Table 5–28 describes the parameters available in the `UPDATE_MEMBER_ATTRIBUTE` procedure signature 3.

---



---

**Note:** Any character attribute exceeding 4,000 characters is truncated to 4,000 characters. Also, the number of members added is based on the number of elements in the first array.

---



---

**Table 5–28** *UPDATE\_MEMBER\_ATTRIBUTE Signature 3 Parameters*

Parameter	Description
<code>p_collection_name</code>	The name of the collection. Maximum length can be 255 bytes. Collection names are case-insensitive, as the collection name is converted to upper case. An error is returned if this collection does not exist with the specified name of the current user and in the same session.
<code>p_seq</code>	Sequence ID of the collection member to be updated.
<code>p_blob_number</code>	Attribute number of the BLOB member attribute to be updated. Valid value is 1. Any number outside of this range is ignored.
<code>p_blob_value</code>	Attribute value of the BLOB member attribute to be updated.

### Example

The following example sets the first and only BLOB attribute of collection sequence number 2 in the collection named 'Departments' to a value of the BLOB variable `l_blob_content`.

```
BEGIN;
  APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (
    p_collection_name => 'Departments',
    p_seq => '2',
    p_blob_number => '1',
    p_blob_value => l_blob_content);
END;
```

**See Also:** ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 1"](#) on page 5-60, ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 2"](#) on page 5-62, ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 4"](#) on page 5-66, ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 5"](#) on page 5-68, ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 6"](#) on page 5-70

## UPDATE\_MEMBER\_ATTRIBUTE Procedure Signature 4

Update the specified XMLTYPE member attribute in the given named collection. If a collection does not exist with the specified name for the current user in the same session for the current Application ID, an application error is raised. If the member specified by sequence ID `p_seq` does not exist, an application error is raised. If the attribute number specified is invalid or outside the valid range (currently only 1 for XMLTYPE), an error is raised.

### Syntax

```
APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (
    p_collection_name IN VARCHAR2,
    p_seq IN VARCHAR2,
    p_xmltype_number IN NUMBER,
    p_xmltype_value IN BLOB);
```

### Parameters

Table 5–29 describes the parameters available in the UPDATE\_MEMBER\_ATTRIBUTE procedure signature 4.

---



---

**Note:** Any character attribute exceeding 4,000 characters is truncated to 4,000 characters. Also, the number of members added is based on the number of elements in the first array.

---



---

**Table 5–29** UPDATE\_MEMBER\_ATTRIBUTE Signature 4 Parameters

Parameter	Description
<code>p_collection_name</code>	The name of the collection. Maximum length can be 255 bytes. Collection names are case-insensitive, as the collection name is converted to upper case. An error is returned if this collection does not exist with the specified name of the current user and in the same session.
<code>p_seq</code>	Sequence ID of the collection member to be updated.
<code>p_xmltype_number</code>	Attribute number of the XMLTYPE member attribute to be updated. Valid value is 1. Any number outside of this range is ignored.
<code>p_xmltype_value</code>	Attribute value of the XMLTYPE member attribute to be updated.

### Example

The following example sets the first and only XML attribute of collection sequence number 2 in the collection named 'Departments' to a value of the XMLType variable `l_xmltype_content`.

```
BEGIN;
    APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (
        p_collection_name => 'Departments',
        p_seq => '2',
        p_xmltype_number => '1',
        p_xmltype_value => l_xmltype_content);
END;
```

**See Also:** ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 1"](#) on page 5-60, ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 2"](#) on page 5-62, ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 3"](#) on page 5-64, ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 5"](#) on page 5-68, ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 6"](#) on page 5-70

## UPDATE\_MEMBER\_ATTRIBUTE Procedure Signature 5

Update the specified NUMBER member attribute in the given named collection. If a collection does not exist with the specified name for the current user in the same session for the current Application ID, an application error is raised. If the member specified by sequence ID `p_seq` does not exist, an application error is raised. If the attribute number specified is invalid or outside the valid range (currently only 1 through 5 for NUMBER), an error is raised.

### Syntax

```
APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (
  p_collection_name IN VARCHAR2,
  p_seq IN VARCHAR2,
  p_attr_number IN NUMBER,
  p_number_value IN NUMBER);
```

### Parameters

Table 5–30 describes the parameters available in the UPDATE\_MEMBER\_ATTRIBUTE procedure signature 5.

---



---

**Note:** Any character attribute exceeding 4,000 characters is truncated to 4,000 characters. Also, the number of members added is based on the number of elements in the first array.

---



---

**Table 5–30** UPDATE\_MEMBER\_ATTRIBUTE Signature 5 Parameters

Parameter	Description
<code>p_collection_name</code>	The name of the collection. Maximum length can be 255 bytes. Collection names are case-insensitive, as the collection name is converted to upper case. An error is returned if this collection does not exist with the specified name of the current user and in the same session.
<code>p_seq</code>	Sequence ID of the collection member to be updated.
<code>p_attr_number</code>	Attribute number of the NUMBER member attribute to be updated. Valid value is 1 through 5. Any number outside of this range is ignored.
<code>p_number_value</code>	Attribute value of the NUMBER member attribute to be updated.

### Example

The following example sets the first numeric attribute of collection sequence number 2 in the collection named 'Departments' to a value of 3000.

```
BEGIN;
  APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (
    p_collection_name => 'Departments',
    p_seq => '2',
    p_attr_number => '1',
    p_number_value => 3000);
END;
```

**See Also:** ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 1" on page 5-60](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 2" on page 5-62](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 3" on page 5-64](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 4" on page 5-66](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 6" on page 5-70](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 6" on page 5-70](#)

## UPDATE\_MEMBER\_ATTRIBUTE Procedure Signature 6

Update the specified DATE member attribute in the given named collection. If a collection does not exist with the specified name for the current user in the same session for the current Application ID, an application error is raised. If the member specified by sequence ID `p_seq` does not exist, an application error is raised. If the attribute number specified is invalid or outside the valid range (currently only 1 through 5 for DATE), an error is raised.

### Syntax

```
APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (
    p_collection_name IN VARCHAR2,
    p_seq IN VARCHAR2,
    p_attr_number IN NUMBER,
    p_date_value IN DATE);
```

### Parameters

Table 5–30 describes the parameters available in the UPDATE\_MEMBER\_ATTRIBUTE procedure signature 6.

---



---

**Note:** Any character attribute exceeding 4,000 characters is truncated to 4,000 characters. Also, the number of members added is based on the number of elements in the first array.

---



---

**Table 5–31** UPDATE\_MEMBER\_ATTRIBUTE Signature 6 Parameters

Parameter	Description
<code>p_collection_name</code>	The name of the collection. Maximum length can be 255 bytes. Collection names are case-insensitive, as the collection name is converted to upper case. An error is returned if this collection does not exist with the specified name of the current user and in the same session.
<code>p_seq</code>	Sequence ID of the collection member to be updated.
<code>p_attr_number</code>	Attribute number of the DATE member attribute to be updated. Valid value is 1 through 5. Any number outside of this range is ignored.
<code>p_date_value</code>	Attribute value of the DATE member attribute to be updated.

### Example

Update the first attribute of the second collection member in collection named 'Departments', and set it to a value of 100.

```
BEGIN;
    APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (
        p_collection_name => 'Departments',
        p_seq => '2',
        p_attr_number => '1',
        p_date_value => 100 );
END;
```



**See Also:** ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 1" on page 5-60](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 2" on page 5-62](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 3" on page 5-64](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 4" on page 5-66](#), ["UPDATE\\_MEMBER\\_ATTRIBUTE Procedure Signature 5" on page 5-68](#)



The `APEX_CSS` package provides utility functions for adding CSS styles to HTTP output. This package is usually used for plug-in development.

- [ADD Procedure](#)
- [ADD\\_3RD\\_PARTY\\_LIBRARY\\_FILE Procedure](#)
- [ADD\\_FILE Procedure](#)

## ADD Procedure

This procedure adds a CSS style snippet that is included inline in the HTML output. Use this procedure to add new CSS style declarations.

### Syntax

```
APEX_CSS.ADD (  
    p_css          IN    VARCHAR2,  
    p_key          IN    VARCHAR2 DEFAULT NULL);
```

### Parameters

[Table 6–1](#) describes the parameters available in the ADD procedure.

**Table 6–1** ADD Parameters

Parameter	Description
p_css	The CSS style snippet. For example, #test {color:#fff}
p_key	Identifier for the style snippet. If specified and a style snippet with the same name has already been added the new style snippet will be ignored.

### Example

Adds an inline CSS definition for the class autocomplete into the HTML page. The key autocomplete\_widget prevents the definition from being included another time if the apex\_css.add is called another time.

```
apex_css.add (  
    p_css => '.autocomplete { color:#ffffff }',  
    p_key => 'autocomplete_widget' );
```

## ADD\_3RD\_PARTY\_LIBRARY\_FILE Procedure

This procedure adds the link tag to load a 3rd party css file and also takes into account the specified Content Delivery Network for the application. Supported libraries include: jQuery, jQueryUI, jQueryMobile.

If a library has already been added, it is not added a second time.

### Syntax

```
add_3rd_party_library_file (
  p_library in varchar2,
  p_file_name in varchar2,
  p_directory in varchar2 default null,
  p_version in varchar2 default null,
  p_media_query in varchar2 default null );
```

### Parameters

[Table 6–2](#) describes the parameters available in the ADD\_3RD\_PARTY\_LIBRARY\_FILE procedure.

**Table 6–2 ADD\_3RD\_PARTY\_LIBRARY\_FILE Parameters**

Parameters	Description
p_library	Use one of the c_library_* constants
p_file_name	Specifies the file name without version, .min and .css
p_directory	Directory where the file p_file_name is located (optional)
p_version	If no value is provided then the same version Application Express ships is used (optional)
p_media_query	Value that is set as media query (optional)

### Example

The following example loads the Cascading Style Sheet file of the Accordion component of the jQuery UI.

```
apex_css.add_3rd_party_library_file (
  p_library => apex_css.c_library_jquery_ui,
  p_file_name => 'jquery.ui.accordion' )
```

## ADD\_FILE Procedure

This procedure adds the link tag to load a CSS library. If a library has already been added, it will not be added a second time.

### Syntax

```
APEX_CSS.ADD_FILE (
    p_name          IN    VARCHAR2,
    p_directory     IN    VARCHAR2 DEFAULT WWV_FLOW.G_IMAGE_PREFIX || 'css/',
    p_version       IN    VARCHAR2 DEFAULT NULL,
    p_skip_extension IN    BOOLEAN DEFAULT FALSE
    p_media_query   IN    VARCHAR2 DEFAULT NULL,
    p_ie_condition  IN    VARCHAR2 DEFAULT NULL);
```

### Parameters

Table 6–3 describes the parameters available in the ADD\_FILE procedure.

**Table 6–3 ADD\_FILE Parameters**

Parameter	Description
p_name	Name of the CSS file.
p_directory	Begin of the URL where the CSS file should be read from. If you use this function for a plug-in you should set this parameter to p_plugin.file_prefix.
p_version	Identifier of the version of the CSS file. The version will be added to the CSS filename. In most cases you should use the default of NULL as the value.
p_skip_extension	The function automatically adds ".css" to the CSS filename. If this parameter is set to TRUE this will not be done.
p_media_query	Value set as media query.
p_ie_condition	Condition used as Internet Explorer condition.

### Example

Adds the CSS file `jquery.autocomplete.css` in the directory specified by `p_plugin.image_prefix` to the HTML output of the page and makes sure that it will only be included once if `apex_css.add_file` is called multiple times with that name.

```
apex_css.add_file (
    p_name => 'jquery.autocomplete',
    p_directory => p_plugin.image_prefix );
```

---

---

## APEX\_CUSTOM\_AUTH

You can use the APEX\_CUSTOM\_AUTH package to perform various operations related to authentication and session management.

- APPLICATION\_PAGE\_ITEM\_EXISTS Function
- CURRENT\_PAGE\_IS\_PUBLIC Function
- DEFINE\_USER\_SESSION Procedure
- GET\_COOKIE\_PROPS Procedure
- GET\_LDAP\_PROPS Procedure
- GET\_NEXT\_SESSION\_ID Function
- GET\_SECURITY\_GROUP\_ID Function
- GET\_SESSION\_ID Function
- GET\_SESSION\_ID\_FROM\_COOKIE Function
- GET\_USER Function
- GET\_USERNAME Function
- IS\_SESSION\_VALID Function
- LOGIN Procedure
- LOGOUT Procedure [DEPRECATED]
- POST\_LOGIN Procedure
- SESSION\_ID\_EXISTS Function
- SET\_SESSION\_ID Procedure
- SET\_SESSION\_ID\_TO\_NEXT\_VALUE Procedure
- SET\_USER Procedure

---

## APPLICATION\_PAGE\_ITEM\_EXISTS Function

This function checks for the existence of page-level item within the current page of an application. This function requires the parameter `p_item_name`. This function returns a Boolean value (TRUE or FALSE).

### Syntax

```
APEX_CUSTOM_AUTH.APPLICATION_PAGE_ITEM_EXISTS(  
    p_item_name IN VARCHAR2)  
RETURN BOOLEAN;
```

### Parameters

[Table 7-1](#) describes the parameters available in the APPLICATION\_PAGE\_ITEM\_EXISTS function.

**Table 7-1 APPLICATION\_PAGE\_ITEM\_EXISTS Parameters**

Parameter	Description
<code>p_item_name</code>	The name of the page-level item.

### Example

The following example checks for the existence of a page-level item, `ITEM_NAME`, within the current page of the application.

```
DECLARE  
    L_VAL BOOLEAN;  
BEGIN  
    VAL := APEX_CUSTOM_AUTH.APPLICATION_PAGE_ITEM_EXISTS(:ITEM_NAME);  
    IF L_VAL THEN  
        http.p('Item Exists');  
    ELSE  
        http.p('Does not Exist');  
    END IF;  
END;
```



---

## CURRENT\_PAGE\_IS\_PUBLIC Function

This function checks whether the current page's authentication attribute is set to **Page Is Public** and returns a Boolean value (TRUE or FALSE)

**See Also:** "Editing Page Attributes" in *Oracle Application Express Application Builder User's Guide*.

### Syntax

```
APEX_CUSTOM_AUTH.CURRENT_PAGE_IS_PUBLIC  
RETURN BOOLEAN;
```

### Example

The following example checks whether the current page in an application is public.

```
DECLARE  
    L_VAL BOOLEAN;  
BEGIN  
    L_VAL := APEX_CUSTOM_AUTH.CURRENT_PAGE_IS_PUBLIC;  
    IF L_VAL THEN  
        http.p('Page is Public');  
    ELSE  
        http.p('Page is not Public');  
    END IF;  
END;
```

---

## DEFINE\_USER\_SESSION Procedure

This procedure combines the SET\_USER and SET\_SESSION\_ID procedures to create one call.

### Syntax

```
APEX_CUSTOM_AUTH.DEFINE_USER_SESSION(  
    p_user          IN    VARCHAR2,  
    p_session_id    IN    NUMBER);
```

### Parameters

Table 7-2 describes the parameters available in the DEFINE\_USER\_SESSION procedure.

**Table 7-2** DEFINE\_USER\_SESSION Parameters

Parameter	Description
p_user	Login name of the user.
p_session_id	The session ID.

### Example

In the following example, a new session ID is generated and registered along with the current application user.

```
APEX_CUSTOM_AUTH.DEFINE_USER_SESSION (  
    :APP_USER,  
    APEX_CUSTOM_AUTH.GET_NEXT_SESSION_ID);
```

**See Also:** "SET\_USER Procedure" on page 7-21 and "SET\_SESSION\_ID Procedure" on page 7-19.

## GET\_COOKIE\_PROPS Procedure

This procedure obtains the properties of the session cookie used in the current authentication scheme for the specified application. These properties can be viewed directly in the Application Builder by viewing the authentication scheme cookie attributes.

### Syntax

```
APEX_CUSTOM_AUTH.GET_COOKIE_PROPS (
    p_app_id           IN NUMBER,
    p_cookie_name      OUT VARCHAR2,
    p_cookie_path      OUT VARCHAR2,
    p_cookie_domain    OUT VARCHAR2
    p_secure           OUT BOOLEAN);
```

### Parameters

Table 7–3 describes the parameters available in the GET\_COOKIE\_PROPS procedure.

**Table 7–3** GET\_COOKIE\_PROPS Parameters

Parameter	Description
p_app_id	An application ID in the current workspace.
p_cookie_name	The cookie name.
p_cookie_path	The cookie path.
p_cookie_domain	The cookie domain.
p_secure	Flag to set secure property of cookie.

### Example

The following example retrieves the session cookie values used by the authentication scheme of the current application.

```
DECLARE
    l_cookie_name  varchar2(256);
    l_cookie_path  varchar2(256);
    l_cookie_domain varchar2(256);
    l_secure       boolean;
BEGIN
    APEX_CUSTOM_AUTH.GET_COOKIE_PROPS (
        p_app_id => 2918,
        p_cookie_name => l_cookie_name,
        p_cookie_path => l_cookie_path,
        p_cookie_domain => l_cookie_domain,
        p_secure => l_secure);
END;
```

## GET\_LDAP\_PROPS Procedure

This procedure obtains the LDAP attributes of the current authentication scheme for the current application. These properties can be viewed directly in Application Builder by viewing the authentication scheme attributes.

### Syntax

```
APEX_CUSTOM_AUTH.GET_LDAP_PROPS (
    p_ldap_host          OUT VARCHAR2,
    p_ldap_port         OUT INTEGER,
    p_use_ssl           OUT VARCHAR2,
    p_use_exact_dn      OUT VARCHAR2,
    p_search_filter      OUT VARCHAR2,
    p_ldap_dn           OUT VARCHAR2,
    p_ldap_edit_function OUT VARCHAR2);
```

### Parameters

Table 7–4 describes the parameters available in the GET\_LDAP\_PROPS procedure.

**Table 7–4 GET\_LDAP\_PROPS Parameters**

Parameter	Description
p_ldap_host	LDAP host name.
p_ldap_port	LDAP port number.
p_use_ssl	Whether SSL is used.
p_use_exact_dn	Whether exact distinguished names are used.
p_search_filter	The search filter used if exact DN is not used.
p_ldap_dn	LDAP DN string.
p_ldap_edit_function	LDAP edit function name.

### Example

The following example retrieves the LDAP attributes associated with the current application.

```
DECLARE
    l_ldap_host          VARCHAR2(256);
    l_ldap_port         INTEGER;
    l_use_ssl           VARCHAR2(1);
    l_use_exact_dn      VARCHAR2(1);
    l_search_filter      VARCHAR2(256);
    l_ldap_dn           VARCHAR2(256);
    l_ldap_edit_function VARCHAR2(256);
BEGIN
    APEX_CUSTOM_AUTH.GET_LDAP_PROPS (
        p_ldap_host      => l_ldap_host,
        p_ldap_port      => l_ldap_port,
        p_use_ssl        => l_use_ssl,
        p_use_exact_dn   => l_use_exact_dn,
        p_search_filter   => l_search_filter,
        p_ldap_dn        => l_ldap_dn,
        p_ldap_edit_function => l_ldap_edit_function);
```

END;

---

## GET\_NEXT\_SESSION\_ID Function

This function generates the next session ID from the Oracle Application Express sequence generator. This function returns a number.

### Syntax

```
APEX_CUSTOM_AUTH.GET_NEXT_SESSION_ID  
RETURN NUMBER;
```

### Example

The following example generates the next session ID and stores it into a variable.

```
DECLARE  
    VAL NUMBER;  
BEGIN  
    VAL := APEX_CUSTOM_AUTH.GET_NEXT_SESSION_ID;  
END;
```

## GET\_SECURITY\_GROUP\_ID Function

This function returns a number with the value of the security group ID that identifies the workspace of the current user.

### Syntax

```
APEX_CUSTOM_AUTH.GET_SECURITY_GROUP_ID  
RETURN NUMBER;
```

### Example

The following example retrieves the Security Group ID for the current user.

```
DECLARE  
    VAL NUMBER;  
BEGIN  
    VAL := APEX_CUSTOM_AUTH.GET_SECURITY_GROUP_ID;  
END;
```

## GET\_SESSION\_ID Function

This function returns APEX\_APPLICATION.G\_INSTANCE global variable. GET\_SESSION\_ID returns a number.

### Syntax

```
APEX_CUSTOM_AUTH.GET_SESSION_ID  
RETURN NUMBER;
```

### Example

The following example retrieves the session ID for the current user.

```
DECLARE  
    VAL NUMBER;  
BEGIN  
    VAL := APEX_CUSTOM_AUTH.GET_SESSION_ID;  
END;
```



## GET\_SESSION\_ID\_FROM\_COOKIE Function

This function returns the Oracle Application Express session ID located by the session cookie in a page request in the current browser session.

### Syntax

```
APEX_CUSTOM_AUTH.GET_SESSION_ID_FROM_COOKIE  
RETURN NUMBER;
```

### Example

The following example retrieves the session ID from the current session cookie.

```
DECLARE  
    VAL NUMBER;  
BEGIN  
    VAL := APEX_CUSTOM_AUTH.GET_SESSION_ID_FROM_COOKIE;  
END;
```

## GET\_USER Function

This function returns the `APEX_APPLICATION.G_USER` global variable (VARCHAR2).

### Syntax

```
APEX_CUSTOM_AUTH.GET_USER  
RETURN VARCHAR2;
```

### Examples

The following example retrieves the username associated with the current session.

```
DECLARE  
    VAL VARCHAR2(256);  
BEGIN  
    VAL := APEX_CUSTOM_AUTH.GET_USER;  
END;
```

## GET\_USERNAME Function

This function returns user name registered with the current Oracle Application Express session in the internal sessions table. This user name is usually the same as the authenticated user running the current page.

### Syntax

```
APEX_CUSTOM_AUTH.GET_USERNAME  
RETURN VARCHAR2;
```

### Example

The following example retrieves the username registered with the current application session.

```
DECLARE  
    VAL VARCHAR2(256);  
BEGIN  
    VAL := APEX_CUSTOM_AUTH.GET_USERNAME;  
END;
```

## IS\_SESSION\_VALID Function

This function is a Boolean result obtained from executing the current application's authentication scheme to determine if a valid session exists. This function returns the Boolean result of the authentication scheme's page sentry.

### Syntax

```
APEX_CUSTOM_AUTH.IS_SESSION_VALID  
RETURN BOOLEAN;
```

### Example

The following example verifies whether the current session is valid.

```
DECLARE  
    L_VAL BOOLEAN;  
BEGIN  
    L_VAL := APEX_CUSTOM_AUTH.IS_SESSION_VALID;  
    IF L_VAL THEN  
        http.p('Valid');  
    ELSE  
        http.p('Invalid');  
    END IF;  
END;
```

---

## LOGIN Procedure

Also referred to as the "Login API," this procedure performs authentication and session registration.

### Syntax

```
APEX_CUSTOM_AUTH.LOGIN (
    p_username          IN  VARCHAR2  DEFAULT NULL,
    p_password          IN  VARCHAR2  DEFAULT NULL,
    p_session_id       IN  VARCHAR2  DEFAULT NULL,
    p_app_page         IN  VARCHAR2  DEFAULT NULL,
    p_entry_point      IN  VARCHAR2  DEFAULT NULL,
    p_preserve_case    IN  BOOLEAN   DEFAULT FALSE);
```

### Parameter

Table 7–5 describes the parameters available in the LOGIN procedure.

**Table 7–5** LOGIN Parameters

Parameter	Description
p_username	Login name of the user.
p_password	Clear text user password.
p_session_id	Current Oracle Application Express session ID.
p_app_page	Current application ID. After login page separated by a colon (:).
p_entry_point	Internal use only.
p_preserve_case	If TRUE, do not upper p_username during session registration

### Example

The following example performs the user authentication and session registration.

```
BEGIN
    APEX_CUSTOM_AUTH.LOGIN (
        p_username => 'FRANK',
        p_password => 'secret99',
        p_session_id => V('APP_SESSION'),
        p_app_page => :APP_ID||':1');
END;
```

---

**Note:** Do not use bind variable notations for p\_session\_id argument.

---

---

## LOGOUT Procedure [DEPRECATED]

---



---

**Note:** This procedure is deprecated. Use `APEX_AUTHENTICATION.LOGOUT` instead.

---

This procedure causes a logout from the current session by unsetting the session cookie and redirecting to a new location.

### Syntax

```
APEX_CUSTOM_AUTH.LOGOUT (
    p_this_app           IN VARCHAR2  DEFAULT NULL,
    p_next_app_page_sess IN VARCHAR2  DEFAULT NULL,
    p_next_url           IN VARCHAR2  DEFAULT NULL);
```

### Parameter

[Table 7–6](#) describes the parameters available in the `LOGOUT` procedure.

**Table 7–6 LOGOUT Parameters**

Parameter	Description
<code>p_this_app</code>	Current application ID.
<code>p_next_app_page_sess</code>	Application and page number to redirect to. Separate multiple pages using a colon (:), and optionally followed by a colon (:), and the session ID (if control over the session ID is desired).
<code>p_next_url</code>	URL to redirect to (use this instead of <code>p_next_app_page_sess</code> ).

### Example

The following example causes a logout from the current session and redirects to page 99 of application 1000.

```
BEGIN
    APEX_CUSTOM_AUTH.LOGOUT (
        p_this_app           => '1000',
        p_next_app_page_sess => '1000:99');
END;
```

---

## POST\_LOGIN Procedure

This procedure performs session registration, assuming the authentication step has been completed. It can be called only from within an Oracle Application Express application page context.

### Syntax

```
APEX_CUSTOM_AUTH.POST_LOGIN(
    p_username          IN VARCHAR2 DEFAULT NULL,
    p_session_id       IN VARCHAR2 DEFAULT NULL,
    p_app_page         IN VARCHAR2 DEFAULT NULL,
    p_preserve_case    IN BOOLEAN  DEFAULT FALSE);
```

### Parameter

[Table 7-7](#) describes the parameters available in the POST\_LOGIN procedure.

**Table 7-7 POST\_LOGIN Parameters**

Parameter	Description
p_username	Login name of user.
p_session_id	Current Oracle Application Express session ID.
p_app_page	Current application ID and after login page separated by a colon (:).
p_preserve_case	If TRUE, do not include p_username in uppercase during session registration.

### Example

The following example performs the session registration following a successful authentication.

```
BEGIN
    APEX_CUSTOM_AUTH.POST_LOGIN (
        p_username => 'FRANK',
        p_session_id => V('APP_SESSION'),
        p_app_page => :APP_ID||':1');
END;
```

## SESSION\_ID\_EXISTS Function

This function returns a Boolean result based on the global package variable containing the current Oracle Application Express session ID. Returns TRUE if the result is a positive number and returns FALSE if the result is a negative number.

### Syntax

```
APEX_CUSTOM_AUTH.SESSION_ID_EXISTS  
RETURN BOOLEAN;
```

### Example

The following example checks whether the current session ID is valid and exists.

```
DECLARE  
    L_VAL BOOLEAN;  
BEGIN  
    L_VAL := APEX_CUSTOM_AUTH.SESSION_ID_EXISTS;  
    IF VAL THEN  
        http.p('Exists');  
    ELSE  
        http.p('Does not exist');  
    END IF;  
END;
```



---

## SET\_SESSION\_ID Procedure

This procedure sets APEX\_APPLICATION.G\_INSTANCE global variable. This procedure requires the parameter P\_SESSION\_ID (NUMBER) which specifies a session ID.

### Syntax

```
APEX_CUSTOM_AUTH.SET_SESSION_ID(  
    p_session_id    IN    NUMBER);
```

### Parameters

[Table 7–8](#) describes the parameters available in the SET\_SESSION\_ID procedure.

**Table 7–8 SET\_SESSION\_ID Parameters**

Parameter	Description
p_session_id	The session ID to be registered.

### Example

In the following example, the session ID value registered is retrieved from the browser cookie.

```
APEX_CUSTOM_AUTH.SET_SESSION_ID(APEX_CUSTOM_AUTH.GET_SESSION_ID_FROM_COOKIE);
```

---

## SET\_SESSION\_ID\_TO\_NEXT\_VALUE Procedure

This procedure combines the operation of `GET_NEXT_SESSION_ID` and `SET_SESSION_ID` in one call.

### Syntax

```
APEX_CUSTOM_AUTH.SET_SESSION_ID_TO_NEXT_VALUE;
```

### Example

In the following example, if the current session is not valid, a new session ID is generated and registered.

```
IF NOT APEX_CUSTOM_AUTH.SESSION_ID_EXISTS THEN  
    APEX_CUSTOM_AUTH.SET_SESSION_ID_TO_NEXT_VALUE;  
END IF;
```

---

## SET\_USER Procedure

This procedure sets the APEX\_APPLICATION.G\_USER global variable. SET\_USER requires the parameter P\_USER (VARCHAR2) which defines a user ID.

### Syntax

```
APEX_CUSTOM_AUTH.SET_USER(  
    p_user IN VARCHAR2);
```

### Parameters

Table 7–9 describes the parameters available in the SET\_USER procedure.

**Table 7–9 SET\_USER Parameters**

Parameter	Description
p_user	The user ID to be registered.

### Example

In the following example, if the current application user is **NOBODY**, then **JOHN.DOE** is registered as the application user.

```
IF V('APP_USER') = 'NOBODY' THEN  
    APEX_CUSTOM_AUTH.SET_USER('JOHN.DOE');  
END IF;
```



---

---

## APEX\_DEBUG

The `APEX_DEBUG` package provides utility functions for managing the debug message log. Specifically, this package provides the necessary APIs to instrument and debug PL/SQL code contained within your Application Express application as well as PL/SQL code in database stored procedures and functions. Instrumenting your PL/SQL code makes it much easier to track down bugs and isolate unexpected behavior more quickly.

The package also provides the means to enable and disable debugging at different debug levels and utility procedures to clean up the message log.

You can view the message log either as described in the "Accessing Debugging Mode" section of the *Oracle Application Express Application Builder User's Guide* or by querying the `APEX_DEBUG_MESSAGES` view.

For further information, see the individual API descriptions.

---

---

**Note:** In Oracle Application Express 4.2, the `APEX_DEBUG_MESSAGE` package was renamed to `APEX_DEBUG`. The `APEX_DEBUG_MESSAGE` package name is still supported to provide backward compatibility. As a best practice, however, use the new `APEX_DEBUG` package for new applications unless you plan to run them in an earlier version of Oracle Application Express.

---

---

- [Constants](#)
- [DISABLE Procedure](#)
- [ENABLE Procedure](#)
- [ENTER Procedure](#)
- [ERROR Procedure](#)
- [INFO Procedure](#)
- [LOG\\_DBMS\\_OUTPUT Procedure](#)
- [LOG\\_LONG\\_MESSAGE Procedure](#)
- [LOG\\_MESSAGE Procedure \[Deprecated\]](#)
- [LOG\\_PAGE\\_SESSION\\_STATE Procedure](#)
- [MESSAGE Procedure](#)
- [REMOVE\\_DEBUG\\_BY\\_AGE Procedure](#)
- [REMOVE\\_DEBUG\\_BY\\_APP Procedure](#)

- 
- [REMOVE\\_DEBUG\\_BY\\_VIEW Procedure](#)
  - [REMOVE\\_SESSION\\_MESSAGES Procedure](#)
  - [TOCHAR Function](#)
  - [TRACE Procedure](#)
  - [WARN Procedure](#)

---

## Constants

The following constants are used by this package.

```
subtype t_log_level is pls_integer;
c_log_level_error constant t_log_level := 1; -- critical error
c_log_level_warn constant t_log_level := 2; -- less critical error
c_log_level_info constant t_log_level := 4; -- default level if debugging is
enabled (for example, used by apex_application.debug)
c_log_level_app_enter constant t_log_level := 5; -- application: messages when
procedures/functions are entered
c_log_level_app_trace constant t_log_level := 6; -- application: other messages
within procedures/functions
c_log_level_engine_enter constant t_log_level := 8; -- Application Express engine:
messages when procedures/functions are entered
c_log_level_engine_trace constant t_log_level := 9; -- Application Express engine:
other messages within procedures/functions
```

## DISABLE Procedure

This procedure turns off debug messaging.

### Syntax

```
APEX_DEBUG.DISABLE;
```

### Parameters

None.

### Example

This example shows how you can turn off debug messaging.

```
BEGIN  
    APEX_DEBUG.DISABLE();  
END;
```

**See Also:** ["ENABLE Procedure"](#) on page 8-5



---

## ENABLE Procedure

This procedure turns on debug messaging. You can specify, by level of importance, the types of debug messages that are monitored.

---

**Note:** You only need to call `ENABLE` procedure once per page view or page accept.

---

### Syntax

```
APEX_DEBUG.ENABLE (
    p_level      IN  T_LOG_LEVEL DEFAULT C_LOG_LEVEL_INFO );
```

### Parameters

[Table 8–1](#) describes the parameters available in the `APEX_DEBUG.ENABLE` procedure.

**Table 8–1** *APEX\_DEBUG.ENABLE Procedure Parameters*

Parameter	Description
<code>p_level</code>	Level or levels of messages to log. Must be an integer from 1 to 9, where level 1 is the most important messages and level 9 (the default) is the least important.  Setting to a specific level logs messages both at that level and below that level. For example, setting <code>p_level</code> to 2 logs any message at level 1 and 2.

### Example

This examples shows how to enable logging of messages for levels 1, 2 and 4. Messages at higher levels are not logged.

```
BEGIN
    APEX_DEBUG.ENABLE (
        apex_debug.c_log_level_info);
END;
```

## ENTER Procedure

This procedure logs messages at level `c_log_level_app_enter`. Use `APEX_DEBUG. ENTER()` to log the routine name and its arguments at the beginning of a procedure or function.

### Syntax

```
APEX_DEBUG. ENTER (
  p_routine_name IN VARCHAR2,
  p_name01 IN VARCHAR2 DEFAULT NULL,
  p_value01 IN VARCHAR2 DEFAULT NULL,
  p_name02 IN VARCHAR2 DEFAULT NULL,
  p_value02 IN VARCHAR2 DEFAULT NULL,
  p_name03 IN VARCHAR2 DEFAULT NULL,
  p_value03 IN VARCHAR2 DEFAULT NULL,
  p_name04 IN VARCHAR2 DEFAULT NULL,
  p_value04 IN VARCHAR2 DEFAULT NULL,
  p_name05 IN VARCHAR2 DEFAULT NULL,
  p_value05 IN VARCHAR2 DEFAULT NULL,
  p_name06 IN VARCHAR2 DEFAULT NULL,
  p_value06 IN VARCHAR2 DEFAULT NULL,
  p_name07 IN VARCHAR2 DEFAULT NULL,
  p_value07 IN VARCHAR2 DEFAULT NULL,
  p_name08 IN VARCHAR2 DEFAULT NULL,
  p_value08 IN VARCHAR2 DEFAULT NULL,
  p_name09 IN VARCHAR2 DEFAULT NULL,
  p_value09 IN VARCHAR2 DEFAULT NULL,
  p_name10 IN VARCHAR2 DEFAULT NULL,
  p_value10 IN VARCHAR2 DEFAULT NULL,
  p_value_max_length IN PLS_INTEGER DEFAULT 1000 );
```

### Parameters

[Table 8–2](#) describes the parameters available for the `APEX_DEBUG. ENTER` procedure.

**Table 8–2 APEX\_DEBUG. ENTER Procedure Parameters**

Parameter	Description
<code>p_routine_name</code>	The name of the procedure or function.
<code>p_namexx/p_valuexx</code>	The procedure or function parameter name and value.
<code>p_value_max_length</code>	The <code>p_valuexx</code> values is truncated to this length.

### Example

This example shows how to use `APEX_ENTER` to add a debug message at the beginning of a procedure.

```
procedure foo (
  p_widget_id in number,
  p_additional_data in varchar2,
  p_emp_rec in emp%rowtype )
is
begin
  apex_debug.enter('foo',
    'p_widget_id' , p_widget_id,
    'p_additional_data', p_additional_data,
    'p_emp_rec.id' , p_emp_rec.id );
```

```
....do something....  
end foo;
```

**See Also:** ["MESSAGE Procedure"](#) on page 8-14, ["ERROR Procedure"](#) on page 8-8, ["WARN Procedure"](#) on page 8-22, ["TRACE Procedure"](#) on page 8-21, ["INFO Procedure"](#) on page 8-9

## ERROR Procedure

This procedure logs messages at level `c_log_level_error`. This procedure always logs, even if debug mode is turned off.

### Syntax

```
APEX_DEBUG.ERROR (
    p_message IN VARCHAR2,
    p0 IN VARCHAR2 DEFAULT NULL,
    p1 IN VARCHAR2 DEFAULT NULL,
    p2 IN VARCHAR2 DEFAULT NULL,
    p3 IN VARCHAR2 DEFAULT NULL,
    p4 IN VARCHAR2 DEFAULT NULL,
    p5 IN VARCHAR2 DEFAULT NULL,
    p6 IN VARCHAR2 DEFAULT NULL,
    p7 IN VARCHAR2 DEFAULT NULL,
    p8 IN VARCHAR2 DEFAULT NULL,
    p9 IN VARCHAR2 DEFAULT NULL,
    p_max_length IN PLS_INTEGER DEFAULT 1000 );
```

### Parameters

[Table 8–3](#) describes parameters available for the `ERROR` procedure.

**Table 8–3** *APEX\_DEBUG.ERROR Procedure Parameters*

Parameter	Description
<code>p_message</code>	The debug message. Occurrences of <code>'%s'</code> are replaced by <code>p0</code> to <code>p19</code> , as in <code>utl_lms.format_message</code> and C's <code>sprintf</code> . Occurrences of <code>'%%'</code> represent the special character <code>'%'</code> . Occurrences of <code>'%&lt;n&gt;'</code> are replaced by <code>p&lt;n&gt;</code> .
<code>p0</code> through <code>p9</code>	Substitution strings for <code>'%s'</code> placeholders.
<code>p_max_length</code>	The <code>p&lt;n&gt;</code> values are truncated to this length.

### Example

This example shows how to use `APEX_ERROR` to log a critical error in the debug log.

```
apex_debug.error('Critical error %s', sqlerrm);
```

**See Also:** ["MESSAGE Procedure"](#) on page 8-14, ["ERROR Procedure"](#) on page 8-8, ["WARN Procedure"](#) on page 8-22, ["TRACE Procedure"](#) on page 8-21, ["INFO Procedure"](#) on page 8-9

## INFO Procedure

This procedure logs messages at level `c_log_level_info`.

### Syntax

```
APEX_DEBUG.INFO (
  p_message IN VARCHAR2,
  p0 IN VARCHAR2 DEFAULT NULL,
  p1 IN VARCHAR2 DEFAULT NULL,
  p2 IN VARCHAR2 DEFAULT NULL,
  p3 IN VARCHAR2 DEFAULT NULL,
  p4 IN VARCHAR2 DEFAULT NULL,
  p5 IN VARCHAR2 DEFAULT NULL,
  p6 IN VARCHAR2 DEFAULT NULL,
  p7 IN VARCHAR2 DEFAULT NULL,
  p8 IN VARCHAR2 DEFAULT NULL,
  p9 IN VARCHAR2 DEFAULT NULL,
  p_max_length IN PLS_INTEGER DEFAULT 1000 );
```

### Parameters

[Table 8–4](#) describes parameters available for the `APEX_DEBUG.INFO` procedure.

**Table 8–4** *APEX\_DEBUG.INFO Procedure Parameters*

Parameter	Description
<code>p_message</code>	The debug message. Occurrences of <code>'%s'</code> are replaced by <code>p0</code> to <code>p19</code> , as in <code>utl_lms.format_message</code> and C's <code>sprintf</code> . Occurrences of <code>'%%'</code> represent the special character <code>'%'</code> . Occurrences of <code>'%&lt;n&gt;'</code> are replaced by <code>p&lt;n&gt;</code> .
<code>p0</code> through <code>p9</code>	Substitution strings for <code>'%s'</code> placeholders.
<code>p_max_length</code>	The <code>p&lt;n&gt;</code> values are truncated to this length.

### Example

This example shows how to use `APEX_DEBUG.INFO` to log information in the debug log.

```
apex_debug.info('Important: %s', 'fnord');
```

**See Also:** ["MESSAGE Procedure"](#) on page 8-14, ["ERROR Procedure"](#) on page 8-8, ["WARN Procedure"](#) on page 8-22, ["TRACE Procedure"](#) on page 8-21, ["ENTER Procedure"](#) on page 8-6

## LOG\_DBMS\_OUTPUT Procedure

This procedure writes the contents of `dbms_output.get_lines` to the debug log. Messages of legacy applications which use `dbms_output` are copied into the debug log. In order to write to the debug log, `dbms_output.enable` must be performed

### Syntax

```
APEX_DEBUG.LOG_DBMS_OUTPUT;
```

### Parameters

None.

### Example

This example shows how to log the contents of the `DBMS_OUTPUT` buffer in the debug log.

```
sys.dbms_output.enable;  
sys.dbms_output.put_line('some data');  
sys.dbms_output.put_line('other data');  
apex_debug.log_dbms_output;
```

**See Also:** ["MESSAGE Procedure"](#) on page 8-14, ["ERROR Procedure"](#) on page 8-8, ["WARN Procedure"](#) on page 8-22, ["TRACE Procedure"](#) on page 8-21, ["INFO Procedure"](#) on page 8-9

---

## LOG\_LONG\_MESSAGE Procedure

Use this procedure to emit debug messages from PLSQL components of Application Express, or PLSQL procedures and functions. This procedure is the same as LOG\_MESSAGE, except it allows logging of much longer messages, which are subsequently split into 4,000 character chunks in the debugging output (because a single debug message is constrained to 4,000 characters).

---

**Note:** Instead of this procedure, use ["ERROR Procedure"](#) on page 8-8, ["WARN Procedure"](#) on page 8-22, ["MESSAGE Procedure"](#) on page 8-14, ["INFO Procedure"](#) on page 8-9, ["ENTER Procedure"](#) on page 8-6, or ["TRACE Procedure"](#) on page 8-21.

---

### Syntax

```
APEX_DEBUG.LOG_LONG_MESSAGE (
    p_message      IN VARCHAR2  DEFAULT NULL,
    p_enabled      IN BOOLEAN   DEFAULT FALSE,
    p_level        IN T_LOG_LEVEL DEFAULT C_LOG_LEVEL_APP_TRACE);
```

### Parameters

[Table 8-5](#) describes parameters available for the APEX\_DEBUG.LOG\_LONG\_MESSAGE procedure.

**Table 8-5 APEX\_DEBUG.LOG\_LONG\_MESSAGE Procedure Parameters**

Parameter	Description
p_message	Log long message with maximum size of 32767 bytes.
p_enabled	Set to TRUE to always log messages, irrespective of whether debugging is enabled. Set to FALSE to only log messages if debugging is enabled.
p_level	Identifies the level of the long log message. See <a href="#">"Constants"</a> on page 8-3.

### Example

This example shows how to enable debug message logging for 1 and 2 level messages and display a level 1 message that could contain anything up to 32767 characters. Note, the p\_enabled parameter need not be specified, as debugging has been explicitly enabled and the default of FALSE for this parameter respects this enabling.

```
DECLARE
    l_msg VARCHAR2(32767) := 'Debug outputs anything up to varchar2 limit';
BEGIN
    APEX_DEBUG.ENABLE (p_level => 2);

    APEX_DEBUG.LOG_LONG_MESSAGE(
        p_message => l_msg,
        p_level => 1 );
END;
```

**See Also:** ["MESSAGE Procedure"](#) on page 8-14, ["ERROR Procedure"](#) on page 8-8, ["WARN Procedure"](#) on page 8-22, ["TRACE Procedure"](#) on page 8-21, ["INFO Procedure"](#) on page 8-9

## LOG\_MESSAGE Procedure [Deprecated]

This procedure logs a debug message.

---

**Note:** Instead of this procedure, use ["ERROR Procedure"](#) on page 8-8, ["WARN Procedure"](#) on page 8-22, ["MESSAGE Procedure"](#) on page 8-14, ["INFO Procedure"](#) on page 8-9, ["ENTER Procedure"](#) on page 8-6, or ["TRACE Procedure"](#) on page 8-21.

---

### Syntax

```
APEX_DEBUG.LOG_MESSAGE (
    p_message IN VARCHAR2 DEFAULT NULL,
    p_enabled IN BOOLEAN DEFAULT FALSE,
    p_level IN T_LOG_LEVEL DEFAULT C_LOG_LEVEL_APP_TRACE );
```

### Parameters

[Table 8–6](#) describes parameters available for the APEX\_DEBUG.LOG\_MESSAGE procedure.

**Table 8–6** APEX\_DEBUG.LOG\_MESSAGE Procedure Parameters

Parameter	Description
p_message	The debug message with a maximum length of 1000 bytes.
p_enabled	Messages are logged when logging is enabled, setting a value of TRUE enables logging.
p_level	Identifies the level of the log message where 1 is most important and 9 is least important. This is an integer value.

### Example

This example shows how to enable debug message logging for 1 and 2 level messages and display a level 1 message showing a variable value. Note, the p\_enabled parameter need not be specified, as debugging has been explicitly enabled and the default of FALSE for this parameter respects this enabling.

```
DECLARE
    l_value varchar2(100) := 'test value';
BEGIN
    APEX_DEBUG.ENABLE (p_level => 2);

    APEX_DEBUG.LOG_MESSAGE(
        p_message => 'l_value = ' || l_value,
        p_level => 1 );

END;
```

**See Also:** ["MESSAGE Procedure"](#) on page 8-14, ["ERROR Procedure"](#) on page 8-8, ["WARN Procedure"](#) on page 8-22, ["TRACE Procedure"](#) on page 8-21, ["INFO Procedure"](#) on page 8-9



## LOG\_PAGE\_SESSION\_STATE Procedure

This procedure logs the session's item values.

### Syntax

```
APEX_DEBUG.LOG_PAGE_SESSION_STATE (
    p_page_id IN NUMBER DEFAULT NULL,
    p_enabled IN BOOLEAN DEFAULT FALSE,
    p_level IN T_LOG_LEVEL DEFAULT C_LOG_LEVEL_APP_TRACE );
```

### Parameters

[Table 8-7](#) describes parameters available for the `APEX_DEBUG.LOG_SESSION_STATE` procedure.

**Table 8-7 APEX\_DEBUG.LOG\_SESSION\_STATE Procedure Parameters**

Parameter	Description
<code>p_page_id</code>	Identifies a page within the current application and workspace context.
<code>p_enabled</code>	Messages are logged when logging is enabled, setting a value of <code>TRUE</code> enables logging.
<code>p_level</code>	Identifies the level of the log message where 1 is most important, 9 is least important. Must be an integer value.

### Example

This example shows how to enable debug message logging for 1 and 2 level messages and display a level 1 message containing all the session state for the application's current page. Note, the `p_enabled` parameter need not be specified, as debugging has been explicitly enabled and the default of `FALSE` for this parameter respects this enabling. Also note the `p_page_id` has not been specified, as this example just shows session state information for the application's current page.

```
BEGIN
    APEX_DEBUG.ENABLE (p_level => 2);

    APEX_DEBUG.LOG_PAGE_SESSION_STATE (p_level => 1);

END;
```

## MESSAGE Procedure

This procedure logs a formatted debug message, general version.

### Syntax

```
APEX_DEBUG.MESSAGE (
    p_message IN VARCHAR2,
    p0 IN VARCHAR2 DEFAULT NULL,
    p1 IN VARCHAR2 DEFAULT NULL,
    p2 IN VARCHAR2 DEFAULT NULL,
    p3 IN VARCHAR2 DEFAULT NULL,
    p4 IN VARCHAR2 DEFAULT NULL,
    p5 IN VARCHAR2 DEFAULT NULL,
    p6 IN VARCHAR2 DEFAULT NULL,
    p7 IN VARCHAR2 DEFAULT NULL,
    p8 IN VARCHAR2 DEFAULT NULL,
    p9 IN VARCHAR2 DEFAULT NULL,
    p10 IN VARCHAR2 DEFAULT NULL,
    p11 IN VARCHAR2 DEFAULT NULL,
    p12 IN VARCHAR2 DEFAULT NULL,
    p13 IN VARCHAR2 DEFAULT NULL,
    p14 IN VARCHAR2 DEFAULT NULL,
    p15 IN VARCHAR2 DEFAULT NULL,
    p16 IN VARCHAR2 DEFAULT NULL,
    p17 IN VARCHAR2 DEFAULT NULL,
    p18 IN VARCHAR2 DEFAULT NULL,
    p19 IN VARCHAR2 DEFAULT NULL,
    p_max_length IN PLS_INTEGER DEFAULT 1000,
    p_level IN T_LOG_LEVEL DEFAULT C_LOG_LEVEL_INFO,
    p_force IN BOOLEAN DEFAULT FALSE );
```

### Parameters

[Table 8–8](#) describes parameters available for the `APEX_DEBUG.MESSAGE` procedure.

**Table 8–8** *APEX\_DEBUG.MESSAGE Procedure Parameters*

Parameter	Description
<code>p_message</code>	The debug message. Occurrences of '%s' is replaced by <code>p0</code> to <code>p19</code> , as in <code>utl_lms.format_message</code> and C's <code>sprintf</code> . Occurrences of '%%' represent the special character '%'. Occurrences of '%<n>' are replaced by <code>p&lt;n&gt;</code> .
<code>p0</code> through <code>p19</code>	Substitution strings for '%s' placeholders.
<code>p_max_length</code>	The <code>p&lt;n&gt;</code> values is truncated to this length.
<code>p_level</code>	The log level for the message, default is <code>c_log_level_info</code> . See <a href="#">"Constants"</a> on page 8-3.
<code>p_force</code>	If TRUE, this generates a debug message even if the page is not rendered in debug mode or <code>p_level</code> is greater than the configured debug messaging (using the URL or using the enable procedure).

### Example

This example shows how to use the `APEX_DEBUG.MESSAGE` procedure to add text to the debug log.

```
apex_debug.message('the value of %s + %s equals %s', 3, 5, 'eight');
```

**See Also:** ["ERROR Procedure"](#) on page 8-8, ["WARN Procedure"](#) on page 8-22, ["TRACE Procedure"](#) on page 8-21, ["INFO Procedure"](#) on page 8-9, ["ENTER Procedure"](#) on page 8-6

## REMOVE\_DEBUG\_BY\_AGE Procedure

Use this procedure to delete from the debug message log all data older than the specified number of days.

### Syntax

```
APEX_DEBUG.REMOVE_DEBUG_BY_AGE (  
    p_application_id IN NUMBER,  
    p_older_than_days IN NUMBER);
```

### Parameters

[Table 8–9](#) describes parameters available for the `APEX_DEBUG.REMOVE_DEBUG_BY_AGE` procedure.

**Table 8–9** *APEX\_DEBUG.REMOVE\_DEBUG\_BY\_AGE Procedure Parameters*

Parameter	Description
<code>p_application_id</code>	The application ID of the application.
<code>p_older_than_days</code>	The number of days data can exist in the debug message log before it is deleted.

### Example

This example demonstrates removing debug messages relating to the current application, that are older than 3 days old.

```
BEGIN  
    APEX_DEBUG.REMOVE_DEBUG_BY_AGE (  
        p_application_id => TO_NUMBER(:APP_ID),  
        p_older_than_days => 3 );  
END;
```

---

## REMOVE\_DEBUG\_BY\_APP Procedure

Use this procedure to delete from the debug message log all data belonging to a specified application.

### Syntax

```
APEX_DEBUG.REMOVE_DEBUG_BY_APP (  
    p_application_id IN NUMBER);
```

### Parameters

Table 8–10 describes parameters available for the APEX\_DEBUG.REMOVE\_DEBUG\_BY\_APP procedure.

**Table 8–10** APEX\_DEBUG.REMOVE\_DEBUG\_BY\_APP Procedure Parameters

Parameter	Description
p_application_id	The application ID of the application.

### Example

This example demonstrates removing all debug messages logged for the current application.

```
BEGIN  
    APEX_DEBUG.REMOVE_DEBUG_BY_APP(  
        p_application_id => TO_NUMBER(:APP_ID) );  
END;
```

---

## REMOVE\_DEBUG\_BY\_VIEW Procedure

Use this procedure to delete all data for a specified view from the message log.

### Syntax

```
APEX_DEBUG.REMOVE_DEBUG_BY_VIEW (  
    p_application_id IN NUMBER,  
    p_view_id IN NUMBER);
```

### Parameters

[Table 8–11](#) describes parameters available for the `APEX_DEBUG.REMOVE_DEBUG_BY_VIEW` procedure.

**Table 8–11** *APEX\_DEBUG.REMOVE\_DEBUG\_BY\_VIEW Procedure Parameters*

Parameter	Description
<code>p_application_id</code>	The application ID of the application.
<code>p_view_id</code>	The view ID of the view.

### Example

This example demonstrates the removal of debug messages within the 'View Identifier' of 12345, belonging to the current application.

```
BEGIN  
    APEX_DEBUG.REMOVE_DEBUG_BY_VIEW (  
        p_application_id => TO_NUMBER(:APP_ID),  
        p_view_id        => 12345 );  
END;
```

---

## REMOVE\_SESSION\_MESSAGES Procedure

This procedure deletes from the debug message log all data for a given session in your workspace defaults to your current session.

### Syntax

```
APEX_DEBUG.REMOVE_SESSION_MESSAGES (  
    p_session    IN NUMBER    DEFAULT NULL);
```

### Parameters

Table 8–12 describes parameters available for the APEX\_DEBUG.REMOVE\_SESSION\_MESSAGES procedure.

**Table 8–12** APEX\_DEBUG.REMOVE\_SESSION\_MESSAGES Procedure Parameters

Parameter	Description
p_session	The session ID. Defaults to your current session.

### Example

This example demonstrates the removal of all debug messages logged within the current session. Note: As no value is passed for the p\_session parameter, the procedure defaults to the current session.

```
BEGIN  
    APEX_DEBUG.REMOVE_SESSION_MESSAGES();  
END;
```

---

## TOCHAR Function

This procedure converts a BOOLEAN to a VARCHAR2.

### Syntax

```
APEX_DEBUG.TOCHAR (  
    p_value IN BOOLEAN )  
    return VARCHAR2;
```

### Parameters

[Table 8–13](#) describes parameters available for the APEX\_DEBUG.TOCHAR function.

**Table 8–13** APEX\_DEBUG.TOCHAR Procedure Parameters

Parameter	Description
p_value	A BOOLEAN 0 or 1 that is converted to FALSE or TRUE respectively.

### Example

This example shows how to use the APEX\_DEBUG.TOCHAR function to convert boolean values to varchar2, so they can be passed to the other debug procedures.

```
declare  
    l_state boolean;  
begin  
    ....  
    apex_debug.info('Value of l_state is %s', apex_debug.tochar(l_state));  
    ....  
end;
```



## TRACE Procedure

This procedure logs messages at level `c_log_level_app_trace`.

### Syntax

```
APEX_DEBUG.TRACE (
  p_message IN VARCHAR2,
  p0 IN VARCHAR2 DEFAULT NULL,
  p1 IN VARCHAR2 DEFAULT NULL,
  p2 IN VARCHAR2 DEFAULT NULL,
  p3 IN VARCHAR2 DEFAULT NULL,
  p4 IN VARCHAR2 DEFAULT NULL,
  p5 IN VARCHAR2 DEFAULT NULL,
  p6 IN VARCHAR2 DEFAULT NULL,
  p7 IN VARCHAR2 DEFAULT NULL,
  p8 IN VARCHAR2 DEFAULT NULL,
  p9 IN VARCHAR2 DEFAULT NULL,
  p_max_length IN PLS_INTEGER DEFAULT 1000 );
```

### Parameters

[Table 8–14](#) describes parameters available for the `APEX_DEBUG.TRACE` procedure.

**Table 8–14** *APEX\_DEBUG.TRACE Procedure Parameters*

Parameter	Description
<code>p_message</code>	The debug message. Occurrences of <code>'%s'</code> are replaced by <code>p0</code> to <code>p19</code> , as in <code>utl_lms.format_message</code> and C's <code>sprintf</code> . Occurrences of <code>'%%'</code> represent the special character <code>'%'</code> . Occurrences of <code>'%&lt;n&gt;'</code> are replaced by <code>p&lt;n&gt;</code> .
<code>p0</code> through <code>p9</code>	Substitution strings for <code>'%s'</code> placeholders.
<code>p_max_length</code>	The <code>p&lt;n&gt;</code> values are truncated to this length.

### Example

This example shows how to use `APEX_DEBUG.TRACE` to log low-level debug information in the debug log.

```
apex_debug.trace('Low-level information: %s+%s=%s', 1, 2, 3);
```

**See Also:** ["MESSAGE Procedure"](#) on page 8-14, ["ERROR Procedure"](#) on page 8-8, ["WARN Procedure"](#) on page 8-22, ["ENTER Procedure"](#) on page 8-6, ["INFO Procedure"](#) on page 8-9

## WARN Procedure

This procedure logs messages at level `c_log_level_warn`.

### Syntax

```
APEX_DEBUG.WARN (
  p_message IN VARCHAR2,
  p0 IN VARCHAR2 DEFAULT NULL,
  p1 IN VARCHAR2 DEFAULT NULL,
  p2 IN VARCHAR2 DEFAULT NULL,
  p3 IN VARCHAR2 DEFAULT NULL,
  p4 IN VARCHAR2 DEFAULT NULL,
  p5 IN VARCHAR2 DEFAULT NULL,
  p6 IN VARCHAR2 DEFAULT NULL,
  p7 IN VARCHAR2 DEFAULT NULL,
  p8 IN VARCHAR2 DEFAULT NULL,
  p9 IN VARCHAR2 DEFAULT NULL,
  p_max_length IN PLS_INTEGER DEFAULT 1000 );
```

### Parameters

[Table 8–15](#) describes parameters available for the `APEX_DEBUG.WARN` procedure.

**Table 8–15** *APEX\_DEBUG.WARN Procedure Parameters*

Parameter	Description
<code>p_message</code>	The debug message. Occurrences of <code>'%s'</code> are replaced by <code>p0</code> to <code>p19</code> , as in <code>utl_lms.format_message</code> and C's <code>sprintf</code> . Occurrences of <code>'%%'</code> represent the special character <code>'%'</code> . Occurrences of <code>'%&lt;n&gt;'</code> are replaced by <code>p&lt;n&gt;</code> .
<code>p0</code> through <code>p9</code>	Substitution strings for <code>'%s'</code> placeholders.
<code>p_max_length</code>	The <code>p&lt;n&gt;</code> values are truncated to this length.

### Example

This example shows how to use `APEX_DEBUG.WARN` to log highly important data in the debug log.

```
apex_debug.warn('Soft constraint %s violated: %s', 4711, sqlerrm);
```

**See Also:** ["MESSAGE Procedure"](#) on page 8-14, ["ERROR Procedure"](#) on page 8-8, ["ENTER Procedure"](#) on page 8-6, ["TRACE Procedure"](#) on page 8-21, ["INFO Procedure"](#) on page 8-9

The APEX\_ERROR package provides the interface declarations and some utility functions for an error handling function and includes procedures and functions to raise errors in an Application Express application.

- [Constants and Attributes used for Result Types](#)
- [Example of an Error Handling Function](#)
- [ADD\\_ERROR Procedure Signature 1](#)
- [ADD\\_ERROR Procedure Signature 2](#)
- [ADD\\_ERROR Procedure Signature 3](#)
- [ADD\\_ERROR Procedure Signature 4](#)
- [ADD\\_ERROR Procedure Signature 5](#)
- [AUTO\\_SET\\_ASSOCIATED\\_ITEM Procedure](#)
- [EXTRACT\\_CONSTRAINT\\_NAME Function](#)
- [GET\\_ARIA\\_ERROR\\_ATTRIBUTES Function](#)
- [GET\\_FIRST\\_ORA\\_ERROR\\_TEXT Function](#)
- [INIT\\_ERROR\\_RESULT Function](#)

## Constants and Attributes used for Result Types

The following constants are used for the API parameter `p_display_location` and the attribute `display_location` in the `t_error` and `t_error_result` types.

```
c_inline_with_field          constant varchar2(40):='INLINE_WITH_FIELD';
c_inline_with_field_and_notif constant varchar2(40):='INLINE_WITH_FIELD_AND_
NOTIFICATION';
c_inline_in_notification     constant varchar2(40):='INLINE_IN_NOTIFICATION';
c_on_error_page              constant varchar2(40):='ON_ERROR_PAGE';
```

The following constants are used for the API parameter `associated_type` in the `t_error` type.

```
c_ass_type_page_item        constant varchar2(30):='PAGE_ITEM';
c_ass_type_region           constant varchar2(30):='REGION';
c_ass_type_region_column    constant varchar2(30):='REGION_COLUMN';
```

The following record structure is passed into an error handling callout function and contains all the relevant information about the error.

```
type t_error is record (
    message varchar2(32767),          /* Error message which will be displayed
*/
    additional_info varchar2(32767),  /* Only used for display_location ON_
ERROR_PAGE to display additional error information */
    display_location varchar2(40),    /* Use constants "used for display_
location" below */
    association_type varchar2(40),    /* Use constants "used for association_
type" below */
    page_item_name varchar2(255),     /* Associated page item name */
    region_id number,                /* Associated tabular form region id of
the primary application */
    column_alias varchar2(255),       /* Associated tabular form column alias
*/
    row_num pls_integer,             /* Associated tabular form row */
    is_internal_error boolean,        /* Set to TRUE if it's a critical error
raised by the APEX engine, like an invalid SQL/PLSQL statements, ... Internal
Errors are always displayed on the Error Page */
    apex_error_code varchar2(255),    /* Contains the system message code if
it's an error raised by APEX */
    ora_sqlcode number,              /* SQLCODE on exception stack which
triggered the error, NULL if the error was not raised by an ORA error */
    ora_sqlerrm varchar2(32767),     /* SQLERRM which triggered the error,
NULL if the error was not raised by an ORA error */
    error_backtrace varchar2(32767), /* Output of sys.dbms_utility.format_
error_backtrace or sys.dbms_utility.format_call_stack */
    error_statement varchar2(32767), /* Statement that was parsed when the
error occurred - only suitable when parsing caused the error */
    component wwv_flow.t_component /* Component which has been processed
when the error occurred */
);
```

The following record structure must be returned by an error handling callout function.

```
type t_error_result is record (
    message varchar2(32767), /* Error message which will be displayed */
    additional_info varchar2(32767), /* Only used for display_location ON_ERROR_
```

```
PAGE to display additional error information */
  display_location varchar2(40),    /* Use constants "used for display_location"
below */
  page_item_name   varchar2(255), /* Associated page item name */
  column_alias     varchar2(255)  /* Associated tabular form column alias */
);
```

## Example of an Error Handling Function

The following is an example of an error handling function.

```

create or replace function apex_error_handling_example (
    p_error in apex_error.t_error )
    return apex_error.t_error_result
is
    l_result          apex_error.t_error_result;
    l_reference_id    number;
    l_constraint_name varchar2(255);
begin
    l_result := apex_error.init_error_result (
        p_error => p_error );

    -- If it's an internal error raised by APEX, like an invalid statement or
    -- code which cannot be executed, the error text might contain security
    sensitive
    -- information. To avoid this security problem rewrite the error to
    -- a generic error message and log the original error message for further
    -- investigation by the help desk.

    if p_error.is_internal_error then
        -- mask all errors that are not common runtime errors (Access Denied
        -- errors raised by application / page authorization and all errors
        -- regarding session and session state)
        if not p_error.is_common_runtime_error then
            -- log error for example with an autonomous transaction and return
            -- l_reference_id as reference#
            -- l_reference_id := log_error (
            --                     p_error => p_error );
            --
            -- Change the message to the generic error message which doesn't
            expose
            -- any sensitive information.
            l_result.message      := 'An unexpected internal application error has
            occurred. '||
            provide '||
            '999G999G999G990')||
            'Please get in contact with XXX and
            'reference# '||to_char(l_reference_id,
            ' for further investigation.';
            l_result.additional_info := null;
            end if;
        else
            -- Always show the error as inline error
            -- Note: If you have created manual tabular forms (using the package
            -- apex_item/htmldb_item in the SQL statement) you should still
            -- use "On error page" on that pages to avoid loosing entered data
            l_result.display_location := case
                when l_result.display_location = apex_
            error.c_on_error_page then apex_error.c_inline_in_notification
                else l_result.display_location
            end;

            -- If it's a constraint violation like
            --
            -- -) ORA-00001: unique constraint violated

```

```

-- -) ORA-02091: transaction rolled back (-> can hide a deferred
constraint)
-- -) ORA-02290: check constraint violated
-- -) ORA-02291: integrity constraint violated - parent key not found
-- -) ORA-02292: integrity constraint violated - child record found
--
-- try to get a friendly error message from our constraint lookup
configuration.
-- If the constraint in our lookup table is not found, fallback to
-- the original ORA error message.
if p_error.ora_sqlcode in (-1, -2091, -2290, -2291, -2292) then
    l_constraint_name := apex_error.extract_constraint_name (
        p_error => p_error );

    begin
        select message
            into l_result.message
            from constraint_lookup
            where constraint_name = l_constraint_name;
        exception when no_data_found then null; -- not every constraint has to
be in our lookup table
    end;
end if;

-- If an ORA error has been raised, for example a raise_application_
error(-20xxx, '...')
-- in a table trigger or in a PL/SQL package called by a process and the
-- error has not been found in the lookup table, then display
-- the actual error text and not the full error stack with all the ORA
error numbers.
if p_error.ora_sqlcode is not null and l_result.message = p_error.message
then
    l_result.message := apex_error.get_first_ora_error_text (
        p_error => p_error );
end if;

-- If no associated page item/tabular form column has been set, use
-- apex_error.auto_set_associated_item to automatically guess the affected
-- error field by examine the ORA error for constraint names or column
names.
if l_result.page_item_name is null and l_result.column_alias is null then
    apex_error.auto_set_associated_item (
        p_error => p_error,
        p_error_result => l_result );
end if;
end if;

return l_result;
end apex_error_handling_example;

```

## ADD\_ERROR Procedure Signature 1

This procedure adds an error message to the error stack that is used to display an error on an error page or inline in a notification. It can be called in a validation or process to add one or more errors to the error stack.

---

---

**Note:** This procedure must be called before the Application Express application has performed the last validation or process. Otherwise, the error is ignored if it does not have a display location of `apex_error.c_on_error_page`.

---

---

### Syntax

```
APEX_ERROR.ADD_ERROR (  
    p_message          in varchar2,  
    p_additional_info  in varchar2 default null,  
    p_display_location in varchar2 );
```

### Parameters

Table 9–1 describes the parameters available in the ADD\_ERROR Procedure Signature 1.

**Table 9–1 ADD\_ERROR Procedure Signature 1 Parameters**

Parameters	Description
<code>p_message</code>	Displayed error message.
<code>p_additional_info</code>	Additional error information needed if the error is displayed on the error page.
<code>p_display_location</code>	Specifies where the error message is displayed. Use the constant <code>apex_error.inline_notification</code> or <code>apex_error.c_error_page</code> . See <a href="#">"Constants and Attributes used for Result Types"</a> on page 9-2.

### Example

This example illustrates how to add a custom error message to the error stack. The error message is displayed inline in a notification. This example can be used in a validation or process.

```
apex_error.add_error (  
    p_message          => 'This custom account is not active!',  
    p_display_location => apex_error.c_inline_in_notification );
```



## ADD\_ERROR Procedure Signature 2

This procedure adds an error message to the error stack that is used to display an error for a page item inline in a notification. It can be called in a validation or process to add one or more errors to the error stack.

---

**Note:** This procedure must be called before the Application Express application has performed the last validation or process. Otherwise, the error is ignored if it does not have a display location of `apex_error.c_on_error_page`.

---

### Syntax

```
APEX_ERROR.ADD_ERROR (
    p_message          in varchar2,
    p_additional_info  in varchar2 default null,
    p_display_location in varchar2,
    p_page_item_name   in varchar2);
```

### Parameters

Table 9-2 describes the parameters available in the ADD\_ERROR Procedure Signature 2.

**Table 9-2 ADD\_ERROR Procedure Signature 2 Parameters**

Parameters	Description
p_message	Displayed error message.
p_additional_info	Additional error information needed if the error is displayed on the error page.
p_display_location	Specifies where the error message is displayed. Use the constant <code>apex_error.c_inline_with_field</code> or <code>apex_error.c_inline_with_field_and_notif</code> . See " <a href="#">Constants and Attributes used for Result Types</a> " on page 9-2.
p_page_item_name	Name of the page item on the current page that is highlighted if <code>apex_error.c_inline_with_field</code> or <code>apex_error.c_inline_with_field_and_notif</code> are used as the display location.

### Example

This example illustrates how to add a custom error message to the error stack. The P5\_CUSTOMER\_ID item is highlighted on the page. The error message is displayed inline in a notification. This example can be used in a validation or process.

```
apex_error.add_error (
    p_message          => 'Invalid Customer ID!',
    p_display_location => apex_error.c_inline_with_field_and_notif,
    p_page_item_name   => 'P5_CUSTOMER_ID');
```

## ADD\_ERROR Procedure Signature 3

This procedure adds an error message to the error stack that is used to display text as defined by a shared component. This error message can be displayed to all display locations. It can be called in a validation or process to add one or more errors to the error stack.

---

**Note:** This procedure must be called before the Application Express application has performed the last validation or process. Otherwise, the error is ignored if it does not have a display location of `apex_error.c_on_error_page`.

---

### Syntax

```
APEX_ERROR.ADD_ERROR (
    p_error_code          in varchar2,
    p0                    in varchar2 default null,
    p1                    in varchar2 default null,
    p2                    in varchar2 default null,
    p3                    in varchar2 default null,
    p4                    in varchar2 default null,
    p5                    in varchar2 default null,
    p6                    in varchar2 default null,
    p7                    in varchar2 default null,
    p8                    in varchar2 default null,
    p9                    in varchar2 default null,
    p_escape_placeholders in boolean default true,
    p_additional_info     in varchar2 default null,
    p_display_location    in varchar2,
    p_page_item_name      in varchar2 );
```

### Parameters

Table 9-3 describes the parameters available in the ADD\_ERROR Procedure Signature 3.

**Table 9-3 ADD\_ERROR Procedure Signature 3 Parameters**

Parameters	Description
<code>p_error_code</code>	Name of shared component text message.
<code>p_additional_info</code>	Additional error information needed if the error is displayed on the error page.
<code>p0</code> through <code>p9</code>	Values for %0 through %9 placeholders defined in the text message.
<code>p_escape_placeholders</code>	If set to TRUE, the values provided in <code>p0</code> through <code>p9</code> are escaped with <code>sys.htf.escape_sc</code> before replacing the placeholder in the text message. If set to FALSE, values are not escaped.
<code>p_display_location</code>	Specifies where the error message is displayed. Use the constants defined for <code>p_display_location</code> . See <a href="#">"Constants and Attributes used for Result Types"</a> on page 9-2.
<code>p_page_item_name</code>	Name of the page item on the current page that is highlighted if <code>apex_error.c_inline_with_field</code> or <code>apex_error.c_inline_with_field_and_notif</code> are used as the display location.

**Example**

This example illustrates how to add a custom error message, where the text is stored in a text message, to the error stack. The P5\_CUSTOMER\_ID item is highlighted on the page. The error message is displayed inline in a notification. This example can be used in a validation or process.

```
apex_error.add_error (
    p_error_code      => 'INVALID_CUSTOMER_ID',
    p0                => l_customer_id,
    p_display_location => apex_error.c_inline_with_field_and_notif,
    p_page_item_name  => 'P5_CUSTOMER_ID' );
```

## ADD\_ERROR Procedure Signature 4

This procedure adds an error message to the error stack that is used to display an error for a tabular form inline in a notification. It can be called in a validation or process to add one or more errors to the error stack.

---

**Note:** This procedure must be called before the Application Express application has performed the last validation or process. Otherwise, the error is ignored if it does not have a display location of `apex_error.c_on_error_page`.

---

### Syntax

```
APEX_ERROR.ADD_ERROR (
    p_message          in varchar2,
    p_additional_info  in varchar2 default null,
    p_display_location in varchar2,
    p_region_id        in number,
    p_column_alias     in varchar2 default null,
    p_row_num          in number );
```

### Parameters

Table 9–4 describes the parameters available in the ADD\_ERROR Procedure Signature 4.

**Table 9–4 ADD\_ERROR Procedure Signature 4 Parameters**

Parameters	Description
<code>p_message</code>	Displayed error message.
<code>p_additional_info</code>	Additional error information needed if the error is displayed on the error page.
<code>p_display_location</code>	Specifies where the error message is displayed. Use the constant <code>apex_error.c_inline_with_field</code> or <code>apex_error.c_inline_with_field_and_notif</code> . See "Constants and Attributes used for Result Types" on page 9-2.
<code>p_region_id</code>	The ID of a tabular form region on the current page. The ID can be read from the view <code>APEX_APPLICATION_PAGE_REGIONS</code> .
<code>p_column_alias</code>	Name of a tabular form column alias defined for <code>p_region_id</code> that is highlighted if <code>apex_error.c_inline_with_field</code> or <code>apex_error.c_inline_with_field_and_notif</code> are used as a display location.
<code>p_row_num</code>	Number of the tabular form row where the error occurred.

### Example

This example illustrates how to add a custom error message for a tabular form, where the column `CUSTOMER_ID` is highlighted, to the error stack. The error message is displayed inline in a notification. This example can be used in a validation or process.

```
apex_error.add_error (
    p_message          => 'Invalid Customer ID!',
    p_display_location => apex_error.c_inline_with_field_and_notif,
    p_region_id        => l_region_id,
    p_column_alias     => 'CUSTOMER_ID',
```

```
p_row_num      => l_row_num );
```

## ADD\_ERROR Procedure Signature 5

This procedure adds an error message to the error stack of a tabular form that is used to display text as defined by a shared component. This error message can be displayed to all display locations. It can be called in a validation or process to add one or more errors to the error stack.

---



---

**Note:** This procedure must be called before the Application Express application has performed the last validation or process. Otherwise, the error is ignored if it does not have a display location of `apex_error.c_on_error_page`.

---



---

### Syntax

```
APEX_ERROR.ADD_ERROR (
    p_error_code          in varchar2,
    p0                    in varchar2 default null,
    p1                    in varchar2 default null,
    p2                    in varchar2 default null,
    p3                    in varchar2 default null,
    p4                    in varchar2 default null,
    p5                    in varchar2 default null,
    p6                    in varchar2 default null,
    p7                    in varchar2 default null,
    p8                    in varchar2 default null,
    p9                    in varchar2 default null,
    p_escape_placeholders in boolean default true,
    p_additional_info     in varchar2 default null,
    p_display_location    in varchar2,
    p_region_id           in number,
    p_column_alias        in varchar2 default null,
    p_row_num             in number );
```

### Parameters

Table 9–5 describes the parameters available in the ADD\_ERROR Procedure Signature 5.

**Table 9–5 ADD\_ERROR Procedure Signature 5 Parameters**

Parameters	Description
p_error_code	Name of shared component text message.
p0 through p9	Values for %0 through %9 placeholders defined in the text message.
p_escape_placeholders	If set to TRUE, the values provided in p0 through p9 are escaped with <code>sys.htf.escape_sc</code> before replacing the placeholder in the text message. If set to FALSE, values are not escaped.
p_additional_info	Additional error information needed if the error is displayed on the error page.
p_display_location	Specifies where the error message is displayed. Use the constants defined for <code>p_display_location</code> . See <a href="#">"Constants and Attributes used for Result Types"</a> on page 9-2.
p_region_id	The ID of the tabular form region on the current page. The ID can be read from the view <code>APEX_APPLICATION_PAGE_REGIONS</code> .

**Table 9–5 (Cont.) ADD\_ERROR Procedure Signature 5 Parameters**

Parameters	Description
p_column_alias	The name of the tabular form column alias defined for p_region_id that is highlighted if apex_error.c_inline_with_field or apex_error.c_inline_with_field_and_notif are used as a display location.
p_row_num	Number of the tabular form row where the error occurred.

**Example**

This example illustrates how to add a custom error message, where the text is stored in a text message, to the error stack. The CUSTOMER\_ID column on the tabular form is highlighted. The error message is displayed inline in a notification. This example can be used in a validation or process.

```
apex_error.add_error (
  p_error_code      => 'INVALID_CUSTOMER_ID',
  p0                => l_customer_id,
  p_display_location => apex_error.c_inline_with_field_and_notif,
  p_region_id      => l_region_id,
  p_column_alias   => 'CUSTOMER_ID',
  p_row_num        => l_row_num );
```

## AUTO\_SET\_ASSOCIATED\_ITEM Procedure

This procedure automatically sets the associated page item or tabular form column based on a constraint contained in `p_error.ora_sqlerrm`.

This procedure performs the following:

- Identifies the constraint by searching for the `schema.constraint` pattern.
- Only supports constraints of type P, U, R and C.
- For constraints of type C (check constraints), the procedure parses the expression to identify those columns that are used in the constraints expression.
- Using those columns, the procedure gets the first visible page item or tabular form column that is based on that column and set it as associated `p_error_result.page_item_name` or `p_error_result.column_alias`.
- If a page item or tabular form column was found, `p_error_result.display_location` is set to `apex_error.c_inline_with_field_and_notif`.

### Syntax

```
APEX_ERROR.AUTO_SET_ASSOCIATED_ITEM (
    p_error_result in out nocopy t_error_result,
    p_error        in          t_error );
```

### Parameters

[Table 9–10](#) describes the parameters available in the `AUTO_SET_ASSOCIATED_ITEM` procedure.

**Table 9–6 AUTO\_SET\_ASSOCIATED\_ITEM Procedure Parameters**

Parameters	Description
<code>p_error_result</code>	The result variable of your error handling function.
<code>p_error</code>	The <code>p_error</code> parameter of your error handling function.

### Example

See an example of how to use this procedure in ["Example of an Error Handling Function"](#) on page 9-4.



---

## EXTRACT\_CONSTRAINT\_NAME Function

This function extracts a constraint name contained in `p_error.ora_sqlerrm`. The constraint must match the pattern `schema.constraint`.

### Syntax

```
APEX_ERROR.EXTRACT_CONSTRAINT_NAME (  
    p_error          in t_error,  
    p_include_schema in boolean default false )  
return varchar2;
```

### Parameters

[Table 9-7](#) describes the parameters available in the `EXTRACT_CONSTRAINT_NAME` function.

**Table 9-7** *EXTRACT\_CONSTRAINT\_NAME Function Parameters*

Parameters	Description
<code>p_error</code>	The <code>p_error</code> parameter of your error handling function.
<code>p_include_schema</code>	If set to <code>TRUE</code> , the result is prefixed with the schema name. For example, <code>HR.DEMO_PRODUCT_INFO_PK</code> . If set to <code>FALSE</code> , only the constraint name is returned.

### Example

See an example of how to use this procedure in "[Example of an Error Handling Function](#)" on page 9-4.

## GET\_ARIA\_ERROR\_ATTRIBUTES Function

This function is useful for item plug-in developers, to enhance screen reader usability of your item, specifically when that item is associated with an error on a page. This function is called as part of rendering of the item, where the main form element(s) are output. The returned WAI-ARIA attributes include:

- `aria-invalid="true"` - Indicates the page item's current value is invalid. When the user is focused on the page item, the screen reader announces 'Invalid Entry'.
- `aria-describedby=" [page_item_name]_error"` - This attribute value matches up with the ID of a `<div>` tag containing the item's associated error message, enabling a screen reader to announce the actual error, when the user is focused on the page item.

---

**Note:** Because these attributes only enhance screen reader usability, attributes are returned only if the current session is running in Screen Reader mode.

---

### Syntax

```
function get_aria_error_attributes (
    p_item_name    in varchar2 )
    return varchar2;
```

### Parameters

Table 9–9 describes the parameters available in the GET\_ARIA\_ERROR\_ATTRIBUTES function.

**Table 9–8** GET\_ARIA\_ERROR\_ATTRIBUTES Function Parameters

Parameter	Description
<code>p_item_name</code>	The page item name. This value is available by using the name attribute of the <code>apex_plugin.t_page_item</code> record type, which is passed in as the 1st parameter to all item plug-in's Render Function Callback.

### Example

This example shows how this function can be used, in rendering a SELECT element, during processing of the Render Function callback for an item plug-in. This function returns additional attributes, if the page item has errors associated with it and if the user is running in Screen Reader mode.

```
...
    l_name := apex_plugin.get_input_name_for_page_item(false);
    sys.htp.prn('<select name="' || l_name || '" id="' || p_item.name || '" ' ||
               apex_error.get_aria_error_attributes(p_item.name) || '>');
...

```

---

## GET\_FIRST\_ORA\_ERROR\_TEXT Function

This function returns the first ORA error message text stored in `p_error.ora_sqlerrm`. If `p_error_ora_sqlerrm` does not contain a value, `NULL` is returned.

### Syntax

```
APEX_ERROR.GET_FIRST_ORA_ERROR_TEXT (  
    p_error          in t_error,  
    p_include_error_no in boolean default false )  
    return varchar2;
```

### Parameters

[Table 9-9](#) describes the parameters available in the `GET_FIRST_ORA_TEXT` function.

**Table 9-9** *GET\_FIRST\_ORA\_TEXT Function Parameters*

Parameters	Description
<code>p_error</code>	The <code>p_error</code> parameter of your error handling function.
<code>p_include_error_no</code>	If set to <code>TRUE</code> , <code>ORA-xxxx</code> is included in the returned error message. If set to <code>FALSE</code> , only the error message text is returned.

### Example

See an example of how to use this procedure in ["Example of an Error Handling Function"](#) on page 9-4.

## INIT\_ERROR\_RESULT Function

This function returns the `t_error_result` type initialized with the values stored in `p_error`.

---

---

**Note:** This function must be used to ensure initialization is compatible with future changes to `t_error_result`.

---

---

### Syntax

```
APEX_ERROR.INIT_ERROR_RESULT (  
    p_error in t_error)  
return t_error_result;
```

### Parameters

[Table 9–10](#) describes the parameters available in the `INIT_ERROR_RESULT` function.

**Table 9–10** *INT\_ERROR\_RESULT Function Parameters*

Parameters	Description
<code>p_error</code>	The <code>p_error</code> parameter of your error handling function.

### Example

See an example of how to use this function in "[Example of an Error Handling Function](#)" on page 9-4.

The APEX\_ESCAPE package provides functions for escaping special characters in strings to ensure that the data is suitable for further processing.

- [Constants](#)
- [HTML Function](#)
- [HTML\\_ATTRIBUTE Function](#)
- [HTML\\_TRUNC Function](#)
- [HTML\\_WHITELIST Function](#)
- [JS\\_LITERAL Function](#)
- [JSON Function](#)
- [LDAP\\_DN Function](#)
- [LDAP\\_SEARCH\\_FILTER Function](#)
- [NOOP Function](#)
- [REGEXP Function](#)
- [SET\\_HTML\\_ESCAPING\\_MODE Procedure](#)

## Constants

The APEX\_ESCAPE package uses the following constants.

```
c_ldap_dn_reserved_chars constant varchar2(8) := '+,;<=>\';  
c_ldap_search_reserved_chars constant varchar2(5) := '*()\|/';  
c_html_whitelist_tags constant varchar2(255) :=  
'<h1>,</h1>,<h2>,</h2>,<h3>,</h3>,<h4>,</h4>,<p>,</p>,<b>,</b>,<strong>,</strong>,<br />,</br />,<i>,</i>,<ul>,</ul>,<ol>,</ol>,<li>,</li>,<hr />';
```

## HTML Function

This function escapes characters which can change the context in an html environment. It is an extended version of the well-known `sys.htf.escape_sc`.

The function's result depends on the escaping mode that is defined by using `apex_escape.set_html_escaping_mode`. By default, the escaping mode is "Extended", but it can be overridden by manually calling `set_html_escaping_mode` or by setting the application security attribute "HTML Escaping Mode" to "Basic". If the mode is "Basic", the function behaves like `sys.htf.escape_sc`. Otherwise, the rules below apply.

The following table, [Table 10-1](#), depicts ascii characters that the function transforms and their escaped values:

**Table 10-1 Escaped Values for Transformed ASCII Characters**

Raw ASCII Characters	Returned Escaped Characters
&	&amp;
"	&quot;
<	&lt;
>	&gt;
'	&#x27;
/	&#x2F;

### Syntax

```
APEX_ESCAPE.HTML (
    p_string IN VARCHAR2 )
    return VARCHAR2;
```

### Parameters

[Table 10-2](#) describes the parameters available in the HTML function.

**Table 10-2 HTML Function Parameters**

Parameter	Description
<code>p_string</code>	The string text that is escaped

### Example

This example tests escaping in basic ('B') and extended ('E') mode.

```
declare
procedure eq(p_str1 in varchar2,p_str2 in varchar2)
    is
    begin
        if p_str1||'.' <> p_str2||'.' then
            raise_application_error(-20001,p_str1||' <> '||p_str2);
        end if;
    end eq;
begin
    apex_escape.set_html_escaping_mode('B');
    eq(apex_escape.html('hello &"<>'/'/'), 'hello &amp;&quot;&lt;&gt;'/'/');
    apex_escape.set_html_escaping_mode('E');
    eq(apex_escape.html('hello &"<>'/'/'), 'hello
```

```
&amp;&quot;&lt;&gt;&#x27;&#x2F;');  
end;
```

---

---

**See Also:** ["HTML\\_TRUNC Function"](#) on page 10-6, ["HTML\\_WHITELIST Function"](#) on page 10-7, ["HTML\\_ATTRIBUTE Function"](#) on page 10-5, ["SET\\_HTML\\_ESCAPING\\_MODE Procedure"](#) on page 10-14

---

---



---

## HTML\_ATTRIBUTE Function

Use this function to escape the values of html entity attributes. It hex escapes everything that is not alphanumeric or in one of the following characters ' ; ' ' ' \_ ' .

### Syntax

```
APEX_ESCAPE.HTML_ATTRIBUTE (  
    p_string IN VARCHAR2 )  
    return VARCHAR2;
```

### Parameters

[Table 10-3](#) describes the parameters available in the HTML\_ATTRIBUTE function.

**Table 10-3 HTML\_ATTRIBUTE Function Parameters**

Parameter	Description
p_string	The text string that is escaped.

### Example

See "[HTML\\_TRUNC Function](#)" on page 10-6.

---

**See Also:** "[HTML\\_TRUNC Function](#)" on page 10-6, "[HTML Function](#)" on page 10-3, "[HTML\\_WHITELIST Function](#)" on page 10-7, "[SET\\_HTML\\_ESCAPING\\_MODE Procedure](#)" on page 10-14

---

## HTML\_TRUNC Function

The `HTML_TRUNC` function escapes html and limits the returned string to `p_length` characters. This function returns the first `p_length` characters of an input clob and escapes them. You can use this function if the input clob might be too large to fit in a `varchar2` variable and it is sufficient to only display the first part of it.

### Syntax

```
APEX_ESCAPE.HTML_TRUNC (
    p_string IN CLOB,
    p_length IN NUMBER DEFAULT 4000 )
return VARCHAR2;
```

### Parameters

[Table 10-4](#) describes the parameters available in the `HTML_TRUNC` function.

**Table 10-4 HTML\_TRUNC Function Parameters**

Parameter	Description
<code>p_string</code>	The text string that is escaped.
<code>p_length</code>	The number of characters from <code>p_string</code> that are escaped.

### Example

This example generates a html list of of titles and text bodies. Html entity attributes are escaped with `HTML_ATTRIBUTE`, whereas normal text is escaped with `HTML` and `HTML_TRUNC`.

```
begin
    http.p('<ul>');
    for l_data in ( select title, cls, body
                   from my_topics )
    loop
        sys.http.p('<li><span class="' ||
                  apex_escape.html_attribute(l_data.cls) || '>' ||
                  apex_escape.html(l_data.title) || '</span>');
        sys.http.p(apex_escape.html_trunc(l_data.body));
        sys.http.p('</li>');
    end loop;
    http.p('</ul>');
end;
```

---

**See Also:** ["HTML\\_ATTRIBUTE Function"](#) on page 10-5, ["HTML Function"](#) on page 10-3, ["HTML\\_WHITELIST Function"](#) on page 10-7, ["SET\\_HTML\\_ESCAPING\\_MODE Procedure"](#) on page 10-14

---

---

## HTML\_WHITELIST Function

The `HTML_WHITELIST` function performs HTML escape on all characters in the input text except the specified whitelist tags. This function can be useful if the input text contains simple html markup but a developer wants to ensure that an attacker cannot use malicious tags for cross-site scripting.

### Syntax

```
APEX_ESCAPE.HTML_WHITELIST (
    p_html IN VARCHAR2,
    p_whitelist_tags IN VARCHAR2 DEFAULT c_html_whitelist_tags )
return VARCHAR2;
```

### Parameters

[Table 10-5](#) describes the parameters available in the `HTML_WHITELIST` function.

**Table 10-5 HTML\_WHITELIST Function Parameters**

Parameter	Description
<code>p_html</code>	The text string that is filtered.
<code>p_whitelist_tags</code>	The comma separated list of tags that stays in <code>p_html</code> .

### Example

This example shows how to use `HTML_WHITELIST` to remove unwanted html markup from a string, while preserving whitelisted tags.

```
begin
    sys.http.p(apex_escape.html_whitelist(
        '<h1>Hello<script>alert("XSS");</script></h1>');
end;
```

---

**See Also:** ["HTML\\_ATTRIBUTE Function"](#) on page 10-5, ["HTML Function"](#) on page 10-3, ["HTML\\_TRUNC Function"](#) on page 10-6, ["SET\\_HTML\\_ESCAPING\\_MODE Procedure"](#) on page 10-14

---

## JS\_LITERAL Function

The `JS_LITERAL` function escapes and optionally enquotes a javascript string. This function replaces non-immune characters with `\xHH` or `\uHHHH` equivalents. The result can be injected into javascript code, within `<script>` tags or inline ("javascript:xxx"). Immune characters include a through z, A through Z, 0 through 9, commas ",", periods "." and underscores "\_".

### Syntax

```
APEX_ESCAPE.JS_LITERAL (  
    p_string IN VARCHAR2,  
    p_quote  IN VARCHAR2 DEFAULT "" )  
return VARCHAR2;
```

### Parameters

[Table 10–6](#) describes the parameters available in the `JS_LITERAL` function.

**Table 10–6 JS\_LITERAL Function Parameters**

Parameter	Description
<code>p_string</code>	The text string that is escaped.
<code>p_quote</code>	If not null, this string is placed on the left and right of the result. The quotation character must be a single or a double quotation mark.

### Example

It describes how to use `JS_LITERAL` to escape special characters in the `l_string` variable.

```
declare  
    l_string varchar2(4000) := 'O''Brien';  
begin  
    sys.htp.p('<script>' ||  
        'alert(' || apex_escape.js_literal(l_string) || ');' || '</script>');  
end;
```

---

## JSON Function

This function returns `p_string` with all special characters escaped.

### Syntax

```
APEX_ESCAPE.JSON (  
    p_string IN VARCHAR2 )  
RETURN VARCHAR2;
```

### Parameters

**Table 10–7** JSON Function Parameters

Parameter	Description
<code>p_string</code>	The string to be escaped.

### Returns/Raised Errors

**Table 10–8** JSON Function Returns

Return	Description
VARCHAR2	The escaped string.

### Example

The following example prints this: `{ "name": "O\u0027Brien" }`

```
declare  
    l_string varchar2(4000) := 'O'Brien';  
begin  
    sys.htp.p('{ "name": "' || apex_escape.json(l_string) || '"}');  
end;
```

## LDAP\_DN Function

The `LDAP_DN` function escapes reserved characters in an LDAP distinguished name, according to RFC 4514. The RFC describes "+,;<=>\ " as reserved characters (see `p_reserved_chars`). These are escaped by a backslash, for example, " becomes \". Non-printable characters, ascii 0 - 31, and ones with a code > 127 (see `p_escape_non_ascii`) are escaped as `\xx`, where `xx` is the hexadecimal character code. The space character at the beginning or end of the string and a # at the beginning is also escaped with a backslash.

### Syntax

```
APEX_ESCAPE.LDAP_DN (
    p_string IN VARCHAR2,
    p_reserved_chars IN VARCHAR2 DEFAULT c_ldap_dn_reserved_chars,
    p_escaped_non_ascii IN BOOLEAN DEFAULT TRUE )
return VARCHAR2;
```

### Parameters

[Table 10-9](#) describes the parameters available in the `LDAP_DN` function.

**Table 10-9 LDAP\_DN Function Parameters**

Parameter	Description
<code>p_string</code>	The text string that is escaped.
<code>p_reserved_chars</code>	A list of characters that when found in <code>p_string</code> is escaped with a backslash.
<code>p_escaped_non_ascii</code>	If TRUE, characters above ascii 127 in <code>p_string</code> are escaped with a backslash. This is supported by RFCs 4514 and 2253, but may cause errors with older LDAP servers and Microsoft AD.

### Example

This example escapes characters in `l_name` and places the result in `l_escaped`.

```
declare
    l_name varchar2(4000) := 'Joe+User';
    l_escaped varchar2(4000);
begin
    l_escaped := apex_escape.ldap_dn(l_name);
    htp.p(l_name||'|' becomes '|'||l_escaped);
end;
```

---

**See Also:** ["LDAP\\_SEARCH\\_FILTER Function"](#) on page 10-11

---

## LDAP\_SEARCH\_FILTER Function

The `LDAP_SEARCH_FILTER` function escapes reserved characters in an LDAP search filter, according to RFC 4515. The RFC describes `*()\ /` as reserved characters (see `p_reserved_chars`). These, non-printable characters (ascii 0 - 31) and ones with a code > 127 (see `p_escape_non_ascii`) are escaped as `\xx`, where `xx` is the hexadecimal character code.

### Syntax

```
APEX_ESCAPE.LDAP_SEARCH_FILTER (
    p_string          IN VARCHAR2,
    p_reserved_chars  IN VARCHAR2 DEFAULT c_ldap_search_reserved_chars,
    p_escape_non_ascii IN BOOLEAN DEFAULT TRUE )
return VARCHAR2;
```

### Parameters

[Table 10-10](#) describes the parameters available in the `LDAP_SEARCH_FILTER` function.

**Table 10-10** LDAP\_SEARCH\_FILTER Function Parameters

Parameter	Description
<code>p_string</code>	The text string that is escaped.
<code>p_reserved_chars</code>	A list of characters that when found in <code>p_string</code> is escaped with <code>\xx</code> where <code>xx</code> is the character's ASCII hexadecimal code.
<code>p_escape_non_ascii</code>	If <code>TRUE</code> , characters above ascii 127 in <code>p_string</code> are escaped with <code>\xx</code> where <code>xx</code> is the character's ASCII hexadecimal code. This is supported by RFCs 4514, but may cause errors with older LDAP servers and Microsoft AD.

### Example

This example escapes the text in `l_name` and places the result in `l_escaped`.

```
declare
l_name varchar2(4000) := 'Joe*User';
l_escaped varchar2(4000);
begin
    l_escaped := apex_escape.ldap_search_filter(l_name);
    htp.p(l_name||' becomes '||l_escaped);
end;
```

---

**See Also:** ["LDAP\\_DN Function"](#) on page 10-10

---

## NOOP Function

Return `p_string` unchanged. Use this function to silence automatic injection detection tests, similar to `dbms_assert.noop` for SQL injection.

### Syntax

```
APEX_ESCAPE.NOOP (  
    p_string IN VARCHAR2)  
    return VARCHAR2 deterministic;
```

### Parameters

[Table 10–11](#) describes the parameters available in the NOOP function.

**Table 10–11** *APEX\_ESCAPE.NOOP Function Parameters*

Parameter	Description
<code>p_string</code>	The input text string.

### Example

This example shows how to use NOOP to show the developer's intention to explicitly not escape text.

```
begin  
    sys.http.p(apex_escape.noop('Cats & Dogs'));  
end;
```



## REGEXP Function

This function escapes characters that can change the context in a regular expression. It should be used to secure user input. The following list depicts ascii characters that the function escapes with a backslash (\):

```
\.^\$*+-?()[]{|
```

### Syntax

```
APEX_ESCAPE.REGEXP (
    p_string IN VARCHAR2);
```

### Parameters

[Table 10-13](#) describes the parameters available in the APEX\_ESCAPE\_REGEX function.

**Table 10-12** APEX\_ESCAPE.REGEXP Function Parameters

Parameter	Description
p_string	Text to escape.

### Example

The following example ensures the special character "-" in Mary-Ann will be escaped and ignored by the regular expression engine.

```
declare
    l_subscribers varchar2(4000) := 'Christina,Hilary,Mary-Ann,Joel';
    l_name varchar2(4000) := 'Mary-Ann';
begin
    if regexp_instr(l_subscribers, '^|,') || apex_escape.regexp(l_name) || '($|,)' > 0
    then
        sys.http.p('found');
    else
        sys.http.p('not found')
    endif;
end
```

---

## SET\_HTML\_ESCAPING\_MODE Procedure

The SET\_HTML\_ESCAPING\_MODE procedure configures HTML escaping mode for wv\_flow\_escape.html.

### Syntax

```
APEX_ESCAPE.SET_HTML_ESCAPING_MODE (  
    p_mode IN VARCHAR2);
```

### Parameters

Table 10-13 describes the parameters available in the SET\_HTML\_ESCAPING\_MODE procedure.

**Table 10-13** APEX\_ESCAPE.SET\_HTML\_ESCAPING\_MODE Procedure Parameters

Parameter	Description
p_mode	If equal to B, then do basic escaping, like sys.htf.escape_sc. If equal to E, then do extended escaping.

### Example

For an example, see "HTML Function" on page 10-3.

---

**See Also:** "HTML\_WHITELIST Function" on page 10-7, "HTML Function" on page 10-3, "HTML\_TRUNC Function" on page 10-6, "HTML\_ATTRIBUTE Function" on page 10-5

---

---

---

## APEX\_INSTANCE\_ADMIN

The `APEX_INSTANCE_ADMIN` package provides utilities for managing an Oracle Application Express runtime environment. You use the `APEX_INSTANCE_ADMIN` package to get and set email settings, Oracle Wallet settings, report printing settings and to manage schema to workspace mappings. `APEX_INSTANCE_ADMIN` can be executed by the `SYS`, `SYSTEM`, and `APEX_050000` database users and any database user granted the role `APEX_ADMINISTRATOR_ROLE`.

- [Available Parameter Values](#)
- [ADD\\_SCHEMA Procedure](#)
- [ADD\\_WORKSPACE Procedure](#)
- [CREATE\\_SCHEMA\\_EXCEPTION Procedure](#)
- [FREE\\_WORKSPACE\\_APP\\_IDS Procedure](#)
- [GET\\_PARAMETER Function](#)
- [GET\\_SCHEMAS Function](#)
- [GET\\_WORKSPACE\\_PARAMETER](#)
- [REMOVE\\_APPLICATION Procedure](#)
- [REMOVE\\_SAVED\\_REPORTS Procedure](#)
- [REMOVE\\_SCHEMA Procedure](#)
- [REMOVE\\_SCHEMA\\_EXCEPTION Procedure](#)
- [REMOVE\\_SCHEMA\\_EXCEPTIONS Procedure](#)
- [REMOVE\\_SUBSCRIPTION Procedure](#)
- [REMOVE\\_WORKSPACE Procedure](#)
- [REMOVE\\_WORKSPACE\\_EXCEPTIONS Procedure](#)
- [RESERVE\\_WORKSPACE\\_APP\\_IDS Procedure](#)
- [RESTRICT\\_SCHEMA Procedure](#)
- [SET\\_LOG\\_SWITCH\\_INTERVAL Procedure](#)
- [SET\\_WORKSPACE\\_PARAMETER](#)
- [SET\\_PARAMETER Procedure](#)
- [SET\\_WORKSPACE\\_CONSUMER\\_GROUP Procedure](#)
- [TRUNCATE\\_LOG Procedure](#)
- [UNRESTRICT\\_SCHEMA Procedure](#)

## Available Parameter Values

**Table 11–1** lists all the available parameter values you can set within the `APEX_INSTANCE_ADMIN` package, including parameters for email, wallet, and reporting printing.

**Table 11–1 Available Parameters**

Parameter Name	Description
<code>ACCOUNT_LIFETIME_DAYS</code>	The maximum number of days an end-user account password may be used before the account is expired.
<code>ALLOW_DB_MONITOR</code>	If set to <code>Y</code> , the default, database monitoring is enabled. If set to <code>N</code> , it is disabled.
<code>ALLOW_HOSTNAMES</code>	If set, users can only navigate to an application if the URL's hostname part contains this value. Instance administrators can configure more specific values at workspace level.
<code>ALLOW_PUBLIC_FILE_UPLOAD</code>	If set to <code>Y</code> , file uploads are allowed without user authentication. If set to <code>N</code> , the default, they are not allowed.
<code>ALLOW_RAS</code>	This parameter is only supported if running Oracle Database 12c. If set to <code>Y</code> , enable Real Application Security support for applications. If set to <code>N</code> (the default), Real Application Security cannot be used.
<code>ALLOW_REST</code>	If set to <code>Y</code> , the default, developers are allowed to expose report regions as RESTful services. If set to <code>N</code> , the are not allowed.
<code>APEX_BUILDER_AUTHENTICATION</code>	Controls the authentication scheme for the internal builder applications. Valid parameter values include: <ul style="list-style-type: none"> <li>■ <code>APEX</code> - Application Express workspace accounts authentication (default)</li> <li>■ <code>DB</code> - Database accounts authentication</li> <li>■ <code>HEADER</code> - HTTP header variable based authentication</li> <li>■ <code>SSO</code> - Oracle Single Sign-On authentication</li> <li>■ <code>LDAP</code> - LDAP authentication</li> </ul>
<code>APEX_REST_PATH_PREFIX</code>	Controls the URI path prefix used to access built-in RESTful Services exposed by Application Express. For example, built-in RESTful Service for referencing static application files using <code>#APP_IMAGES#</code> token. If the default prefix ( <code>r</code> ) conflicts with RESTful Services defined by users, adjust this preference to avoid the conflict.
<code>APPLICATION_ACTIVITY_LOGGING</code>	Controls instance wide setting of application activity log ( <code>[A]</code> lways, <code>[N]</code> ever, <code>[U]</code> se application settings)
<code>APPLICATION_ID_MAX</code>	The largest possible ID for a worksheet or database application.
<code>APPLICATION_ID_MIN</code>	The smallest possible ID for a worksheet or database application.
<code>AUTOEXTEND_TABLESPACES</code>	If set to <code>Y</code> , the default, provisioned tablespaces is autoextended up to a maximum size. If set to <code>N</code> tablespaces are not autoextended.

**Table 11–1 (Cont.) Available Parameters**

Parameter Name	Description
BIGFILE_TABLESPACES_ENABLED	If set to Y, the tablespaces provisioned through Oracle Application Express are created as bigfile tablespaces. If set to N, the tablespaces are created as smallfile tablespaces.
CHECKSUM_HASH_FUNCTION	Defines the algorithm that is used to create one way hashes for URL checksums. Valid values are MD5 (deprecated), SH1 (SHA-1), SH256 (SHA-2, 256 bit), SH384 (SHA-2, 384 bit), SH512 (SHA-2, 512 bit) and null. The SHA-2 algorithms are only available on Oracle Database Release 12g and later. A null value evaluates to the most secure algorithm available and is the default.
CHECK_FOR_UPDATES	If set to N, the check for Oracle Application Express and Oracle REST Data Services product updates is disabled for the entire instance, regardless of preferences specified by individual developers. The default is Y.
DELETE_UPLOADED_FILES_AFTER_DAYS	Uploaded files like application export files, websheet export files, spreadsheet data load files are automatically deleted after this number of days. Default is 14.
DISABLE_ADMIN_LOGIN	If set to Y, administration services are disabled. If set to N, the default, they are not disabled.
DISABLE_WORKSPACE_LOGIN	If set to Y, the workspace login is disabled. If set to N, the default, the login is not disabled.
DISABLE_WS_PROV	If set to Y, the workspace creation is disabled for requests sent out by using e-mail notification. If set to N, the default, they are not disabled.
EMAIL_IMAGES_URL	Specifies the full URL to the images directory of Application Express instance, including the trailing slash after the images directory. For example: <code>http://your_server/i/</code>  This setting is used for Oracle Application Express system-generated emails.
EMAIL_INSTANCE_URL	Specifies the URL to Oracle Application Express instance, including the trailing slash after the Database Access Descriptor. For example: <code>http://your_server/pls/apex/</code>  This setting used for Oracle Application Express system-generated emails.
ENABLE_TRANSACTIONAL_SQL	If set to Y, the default, transactional SQL commands are enabled on this instance. If set to N, they are not enabled.
ENCRYPTED_TABLESPACES_ENABLED	If set to Y, the tablespaces provisioned through Oracle Application Express are created as encrypted tablespaces. If set to N, the tablespaces are not encrypted.
EXPIRE_FIND_USER_ACCOUNTS	If set to Y, expiration of Application Express accounts is enabled. If set to N, they are not enabled.
HTTP_ERROR_STATUS_ON_ERROR_PAGE_ENABLED	Used in conjunction with the <code>APEX_INSTANCE_ADMIN.SET_PARAMETER</code> procedure.  If set to N, the default, Oracle Application Express presents an error page to the end user for all unhandled errors. If set to Y, returns an HTTP 400 status to the end user's client browser when the Application Express engine encounters an unhandled error.

**Table 11–1 (Cont.) Available Parameters**

Parameter Name	Description
HTTP_RESPONSE_HEADERS	List of http response headers, separated by newline (chr(10)). Application Express writes these headers on each request, before rendering the page. The substitution string #CDN# within the headers is replaced with the content delivery networks that are known to Application Express.
HTTP_STS_MAX_AGE	REQUIRE_HTTPS must be set to A for this parameter to be relevant. Application Express emits a Strict-Transport-Security header, with max-age=<value>, on HTTPS requests if HTTP_STS_MAX_AGE has a value greater than 0. If the request protocol is HTTP, instead of processing the request, Application Express redirects to a HTTPS URL.
INBOUND_PROXIES	Comma-separated list of IP addresses for proxy servers through which requests come in.
LOGIN_THROTTLE_DELAY	The flag which determines the time increase in seconds after failed logins.
LOGIN_THROTTLE_METHODS	The methods to count failed logins. Colon-separated list of USERNAME_IP, USERNAME, IP.
MAX_SESSION_IDLE_SEC	The number of seconds an internal application may be idle.
MAX_SESSION_IDLE_SEC	The number of seconds an internal application may be idle.
MAX_SESSION_LENGTH_SEC	The number of seconds an internal application session may exist.
MAX_SESSION_LENGTH_SEC	The number of seconds an internal application session may exist.
PASSWORD_ALPHA_CHARACTERS	The alphabetic characters used for password complexity rules. Default list of alphabetic characters include the following: abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
PASSWORD_HASH_FUNCTION	Defines the algorithm that is used to create one way hashes for workspace user passwords. Valid values are MD5 (deprecated), SH1 (SHA-1), SH256 (SHA-2, 256 bit), SH384 (SHA-2, 384 bit), SH512 (SHA-2, 512 bit) and null. The SHA-2 algorithms are only available on Oracle Database Release 12g and later. A null value evaluates to the most secure algorithm available and is the default.
PASSWORD_HASH_ITERATIONS	Defines the number of iterations for the PASSWORD_HASH_FUNCTION (default 10000).
PASSWORD_HISTORY_DAYS	Defines the maximum number of days a developer or administrator account password may be used before the account expires. The default value is 45 days.
PASSWORD_PUNCTUATION_CHARACTERS	The punctuation characters used for password complexity rules. Default list of punctuation characters include the following: !"#%&()``*+,-/;:<=>?_
PLSQL_EDITING	If set to Y, the default, the SQL Workshop Object Browser is enabled to allow users to edit and compile PL/SQL. If set to N, users are not allowed.

**Table 11–1 (Cont.) Available Parameters**

Parameter Name	Description
PRINT_BIB_LICENSED	Specify either standard support or advanced support. Advanced support requires an Oracle BI Publisher license. Valid values include: <ul style="list-style-type: none"> <li>■ APEX_LISTENER - Requires Oracle Rest Data Services</li> <li>■ ADVANCED - Requires Oracle BI Publisher</li> <li>■ STANDARD</li> </ul>
PRINT_SVR_HOST	Specifies the host address of the print server converting engine, for example, localhost. Enter the appropriate host address if the print server is installed at another location.
PRINT_SVR_PORT	Defines the port of the print server engine, for example 8888. Value must be a positive integer.
PRINT_SVR_PROTOCOL	Valid values include: <ul style="list-style-type: none"> <li>■ http</li> <li>■ https</li> </ul>
PRINT_SVR_SCRIPT	Defines the script that is the print server engine, for example: <pre>/xmlpserver/convert</pre>
QOS_MAX_SESSION_KILL_TIMEOUT	Number of seconds that an active old session can live, when QOS_MAX_SESSION_REQUESTS has been reached. The oldest database session with LAST_CALL_ET greater than QOS_MAX_SESSION_KILL_TIMEOUT is killed.
QOS_MAX_SESSION_REQUESTS	Number of allowed concurrent requests to one session associated with this workspace.
QOS_MAX_WORKSPACE_REQUESTS	Number of allowed concurrent requests to sessions in this workspace.
REQ_NEW_SCHEMA	If set to Y, the option for new schema for new workspace requests is enabled. If set to N, the default, the option is disabled.
REQUIRE_HTTPS	Set to A, to enforce HTTPS for the entire Application Express instance. Set to I, to enforce HTTPS within the Application Express development and administration applications. Set to N, to allow all applications to be used when the protocol is either HTTP or HTTPS. Please note developers can also enforce HTTPS at the application level, by setting the <b>Secure</b> attribute of an application scheme's cookie.
REQUIRE_HTTPS	Set to Y to allow authentication pages within the Application Express development and administration applications to be used only when the protocol is HTTPS. Select N to allow these application pages to be used when the protocol is either HTTP or HTTPS.
REQUIRE_VERIFICATION_CODE	If set to Y, the Verification Code is displayed and is required for someone to request a new workspace. If set to N, the default, the Verification Code is not required.
RESTFUL_SERVICES_ENABLED	If set to Y, the default, RESTful services development is enabled. If set to N, RESTful services are not enabled.

**Table 11–1 (Cont.) Available Parameters**

Parameter Name	Description
RM_CONSUMER_GROUP	If set, this is the resource manager consumer group to be used for all page events. A more specific group can be configured at workspace level.
SERVICE_REQUEST_FLOW	Determines default provisioning mode. Default is MANUAL.
SERVICE_REQUESTS_ENABLED	If set to Y, the default, workspace service requests for schemas, storage, and termination is enabled. If set to N, these requests are disabled.
SMTP_FROM	Defines the "from" address for administrative tasks that generate email, such as approving a provision request or resetting a password.  Enter a valid email address, for example:  someone@somewhere.com
SMTP_HOST_ADDRESS	Defines the server address of the SMTP server. If you are using another server as an SMTP relay, change this parameter to that server's address.  Default setting:  localhost
SMTP_HOST_PORT	Defines the port the SMTP server listens to for mail requests.  Default setting:  25
SMTP_PASSWORD	Defines the password Application Express takes to authenticate itself against the SMTP server, with the parameter SMTP_USERNAME.
SMTP_TLS_MODE	Defines whether Application Express opens an encrypted connection to the SMTP server. Encryption is only supported on database versions 11.2.0.2 and later. On earlier database versions, the connection is not encrypted.  If set to N, the connection is unencrypted (default).  If set to Y, the connection is encrypted before data is sent.  If STARTTLS, Application Express sends the SMTP commands EHLO <SMTP_HOST_ADDRESS> and STARTTLS before encrypting the connection.
SMTP_USERNAME	Defines the username Application Express takes to authenticate itself against the SMTP server (default is null). Starting with database version 11.2.0.2, Application Express uses UTL_MAIL's AUTH procedure for authentication. This procedure negotiates an authentication mode with the SMTP server. With earlier database versions, the authentication mode is always AUTH LOGIN. If SMTP_USERNAME is null, no authentication is used.
SQL_SCRIPT_MAX_OUTPUT_SIZE	The maximum allowable size for an individual script result. Default is 200000.



**Table 11–1 (Cont.) Available Parameters**

Parameter Name	Description
SSO_LOGOUT_URL	<p>Defines the URL Application Express redirects to in order to trigger a logout from the Single Sign-On server. Application Express automatically appends "?p_done_url=...login url...".</p> <p>Example:  <a href="https://login.mycompany.com/pls/orasso/orasso.wvss_o_app_admin.ls_logout">https://login.mycompany.com/pls/orasso/orasso.wvss_o_app_admin.ls_logout</a></p>
STRONG_SITE_ADMIN_PASSWORD	<p>If set to Y, the default, the apex_admin password must conform to the default set of strong complexity rules. If set to N, the password is not required to follow the strong complexity rules.</p>
SYSTEM_HELP_URL	<p>Location of the help and documentation accessed from the Help link within the development environment. Default is <a href="http://apex.oracle.com/doc41">http://apex.oracle.com/doc41</a>.</p>
TRACING_ENABLED	<p>If set to Y (the default), an application with Debug enabled can also generate server side db trace files using &amp;p_trace=YES on the URL.</p> <p>If set to N, the request to create a trace file is ignored.</p>
USERNAME_VALIDATION	<p>The regular expression used to validate a username if the Builder authentication scheme is not APEX. Default is as follows:</p> <p><code>^[[:alnum:]]_%-]+@[[:alnum:]]-\.[[:alpha:]]{2,4}\$</code></p>
WALLET_PATH	<p>The path to the wallet on the file system, for example:</p> <p><code>file:/home/&lt;username&gt;/wallets</code></p>
WALLET_PWD	<p>The password associated with the wallet.</p>
WEBSHEET_SQL_ACCESS	<p>If set to Y, the default, SQL tags and SQL reports are possible in Websheet applications. If set to N, they are not possible.</p>
WORKSPACE_EMAIL_MAXIMUM	<p>Maximum number of emails allowed to be sent by using APEX_MAIL per workspace in a 24 hour period. Default is 1000.</p>
WORKSPACE_MAX_FILE_BYTES	<p>The maximum number of bytes for uploaded files for a workspace. A setting at the workspace-level overrides the instance-level setting.</p>
WORKSPACE_MAX_OUTPUT_SIZE	<p>The maximum space allocated for script results. Default is 2000000.</p>
WORKSPACE_PROVISION_DEMO_OBJECTS	<p>If set to Y, the default, demonstration applications and database objects are created in new workspaces. If set to N, they are not created in the current workspace.</p>
WORKSPACE_WEBSHEET_OBJECTS	<p>If set to Y, the default, Application Express Websheet database objects are created in new workspaces. If set to N, they are not created in the current workspace.</p>

**See Also:** "Configuring Email in a Runtime Environment", "Configuring Wallet Information", "Configuring Report Printing Settings in a Runtime Environment" in *Oracle Application Express Administration Guide*.

## ADD\_SCHEMA Procedure

The ADD\_SCHEMA procedure adds a schema to a workspace to schema mapping.

### Syntax

```
APEX_INSTANCE_ADMIN.ADD_SCHEMA(  
    p_workspace    IN VARCHAR2,  
    p_schema       IN VARCHAR2);
```

### Parameters

[Table 11-2](#) describes the parameters available in the ADD\_SCHEMA procedure.

**Table 11-2 ADD\_SCHEMA Parameters**

Parameter	Description
p_workspace	The name of the workspace to which the schema mapping is added.
p_schema	The schema to add to the schema to workspace mapping.

### Example

The following example demonstrates how to use the ADD\_SCHEMA procedure to map a schema mapped to a workspace.

```
BEGIN  
    APEX_INSTANCE_ADMIN.ADD_SCHEMA('MY_WORKSPACE', 'FRANK');  
END;
```

## ADD\_WORKSPACE Procedure

The ADD\_WORKSPACE procedure adds a workspace to an Application Express Instance.

### Syntax

```
APEX_INSTANCE_ADMIN.ADD_WORKSPACE (
  p_workspace_id      IN NUMBER DEFAULT NULL,
  p_workspace         IN VARCHAR2,
  p_source_identifier IN VARCHAR2 DEFAULT NULL,
  p_primary_schema   IN VARCHAR2,
  p_additional_schemas IN VARCHAR2,
  p_rm_consumer_group IN VARCHAR2 DEFAULT NULL );
```

### Parameters

Table 11–3 describes the parameters available in the ADD\_WORKSPACE procedure.

**Table 11–3 ADD\_WORKSPACE Parameters**

Parameter	Description
p_workspace_id	The ID to uniquely identify the workspace in an Application Express instance. This may be left null and a new unique ID is assigned.
p_workspace	The name of the workspace to be added.
p_source_identifier	A short identifier for the workspace used when synchronizing feedback between different instances.
p_primary_schema	The primary database schema to associate with the new workspace.
p_additional_schemas	A colon delimited list of additional schemas to associate with this workspace.
p_rm_consumer_group	Resource Manager consumer group which is used when executing applications of this workspace.

### Example

The following example demonstrates how to use the ADD\_WORKSPACE procedure to add a new workspace named MY\_WORKSPACE using the primary schema, SCOTT, along with additional schema mappings for HR and OE.

```
BEGIN
  APEX_INSTANCE_ADMIN.ADD_WORKSPACE (
    p_workspace_id      => 8675309,
    p_workspace         => 'MY_WORKSPACE',
    p_primary_schema   => 'SCOTT',
    p_additional_schemas => 'HR:OE' );
END;
```

## CREATE\_SCHEMA\_EXCEPTION Procedure

This procedure creates an exception which allows assignment of a restricted schema to a specific workspace.

### Syntax

```
APEX_INSTANCE_ADMIN.CREATE_SCHEMA_EXCEPTION (
    p_schema    in varchar2,
    p_workspace in varchar2 );
```

### Parameter

[Table 11-4](#) describes the parameters available for CREATE\_SCHEMA\_EXCEPTION procedure.

**Table 11-4** CREATE\_SCHEMA\_EXCEPTION Parameters

Parameter	Description
p_schema	The schema.
p_workspace	The workspace.

### Example

This example allows the assignment of restricted schema HR to workspace HR\_WORKSPACE.

```
begin
    apex_instance_admin.create_schema_exception (
        p_schema    => 'HR',
        p_workspace => 'HR_WORKSPACE' );
    commit;
end;
```

**See also:** ["RESTRICT\\_SCHEMA Procedure"](#) on page 11-26, ["UNRESTRICT\\_SCHEMA Procedure"](#) on page 11-32, ["REMOVE\\_SCHEMA\\_EXCEPTION Procedure"](#) on page 11-19, ["REMOVE\\_SCHEMA\\_EXCEPTIONS Procedure"](#) on page 11-20, ["REMOVE\\_WORKSPACE\\_EXCEPTIONS Procedure"](#) on page 11-23

---

## FREE\_WORKSPACE\_APP\_IDS Procedure

This procedure removes the reservation of application IDs for a given workspace ID. Use this procedure to undo a reservation, when the reservation is not necessary anymore because it happened by mistake or the workspace no longer exists. To reserve application IDs for a given workspace, see "[RESERVE\\_WORKSPACE\\_APP\\_IDS Procedure](#)" on page 11-24.

### Syntax

```
APEX_INSTANCE_ADMIN.FREE_WORKSPACE_APP_IDS (  
    p_workspace_id IN NUMBER );
```

### Parameters

[Table 11-5](#) describes the parameters available in the FREE\_WORKSPACE\_APP\_IDS procedure.

**Table 11-5** FREE\_WORKSPACE\_APP\_IDS Parameters

Parameter	Description
p_workspace_id	The unique ID of the workspace.

### Example

This example illustrates how to undo the reservation of application IDs that belong to a workspace with an ID of 1234567890.

```
begin  
    apex_instance_admin.free_workspace_app_ids(1234567890);  
end;
```

## GET\_PARAMETER Function

The GET\_PARAMETER function retrieves the value of a parameter used in administering a runtime environment.

### Syntax

```
APEX_INSTANCE_ADMIN.GET_PARAMETER(  
    p_parameter    IN VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

Table 11-6 describes the parameters available in the GET\_PARAMETER function.

**Table 11-6** GET\_PARAMETER Parameters

Parameter	Description
p_parameter	The instance parameter to be retrieved. See " <a href="#">Available Parameter Values</a> " on page 11-2.

### Example

The following example demonstrates how to use the GET\_PARAMETER function to retrieve the SMTP\_HOST\_ADDRESS parameter currently defined for an Oracle Application Express instance.

```
DECLARE  
    L_VAL VARCHAR2(4000);  
BEGIN  
    L_VAL :=APEX_INSTANCE_ADMIN.GET_PARAMETER('SMTP_HOST_ADDRESS');  
    DBMS_OUTPUT.PUT_LINE('The SMTP Host Setting Is: '||L_VAL);  
END;
```

---

## GET\_SCHEMAS Function

The GET\_SCHEMAS function retrieves a comma-delimited list of schemas that are mapped to a given workspace.

### Syntax

```
APEX_INSTANCE_ADMIN.GET_SCHEMAS(  
    p_workspace    IN VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

Table 11-7 describes the parameters available in the GET\_SCHEMAS function.

**Table 11-7** GET\_SCHEMAS Parameters

Parameter	Description
p_workspace	The name of the workspace from which to retrieve the schema list.

### Example

The following example demonstrates how to use the GET\_SCHEMA function to retrieve the underlying schemas mapped to a workspace.

```
DECLARE  
    L_VAL VARCHAR2(4000);  
BEGIN  
    L_VAL :=APEX_INSTANCE_ADMIN.GET_SCHEMAS('MY_WORKSPACE');  
    DBMS_OUTPUT.PUT_LINE('The schemas for my workspace: '||L_VAL);  
END;
```

## GET\_WORKSPACE\_PARAMETER

The GET\_WORKSPACE\_PARAMETER procedure gets the workspace parameter.

### Syntax

```
get_workspace_parameter(  
    p_workspace      IN VARCHAR2,  
    p_parameter      IN VARCHAR2,
```

### Parameters

[Table 11-8](#) describes the parameters available in GET\_WORKSPACE\_PARAMETER procedure.

**Table 11-8** GET\_WORKSPACE\_PARAMETER Parameters

Parameter	Description
p_workspace	The name of the workspace to which you are getting the workspace parameter.
p_parameter	The parameter name that overrides the instance parameter value of the same name for this workspace. Parameter names include: <ul style="list-style-type: none"><li>▪ ALLOW_HOSTNAMES</li><li>▪ QOS_MAX_WORKSPACE_REQUESTS</li><li>▪ QOS_MAX_SESSION_REQUESTS</li><li>▪ QOS_MAX_SESSION_KILL_TIMEOUT</li><li>▪ RM_CONSUMER_GROUP</li><li>▪ WORKSPACE_EMAIL_MAXIMUM</li><li>▪ WORKSPACE_MAX_FILE_BYTES</li></ul>

### Example

The following example prints the value of ALLOW\_HOSTNAMES for the HR workspace.

```
BEGIN  
    DBMS_OUTPUT.PUT_LINE (  
APEX_INSTANCE_ADMIN.GET_WORKSPACE_PARAMETER (  
    p_workspace => 'HR',  
    p_parameter => 'ALLOW_HOSTNAMES' ));  
END;
```



---

## REMOVE\_APPLICATION Procedure

The REMOVE\_APPLICATION procedure removes the application specified from the Application Express instance.

### Syntax

```
APEX_INSTANCE_ADMIN.REMOVE_APPLICATION (  
    p_application_id IN NUMBER);
```

### Parameters

[Table 11–9](#) describes the REMOVE\_APPLICATION procedure parameters.

**Table 11–9 REMOVE\_APPLICATION Parameters**

Parameter	Description
p_application_id	The ID of the application to remove.

### Example

The following example demonstrates how to use the REMOVE\_APPLICATION procedure to remove an application with an ID of 100 from an Application Express instance.

```
BEGIN  
    APEX_INSTANCE_ADMIN.REMOVE_APPLICATION(100);  
END;
```

---

## REMOVE\_SAVED\_REPORT Procedure

The REMOVE\_SAVED\_REPORT procedure removes a specific user's saved interactive report settings for a particular application.

### Syntax

```
APEX_INSTANCE_ADMIN.REMOVE_SAVED_REPORT(  
    p_application_id    IN NUMBER,  
    p_report_id        IN NUMBER);
```

### Parameters

Table 11-10 describes the parameters available in the REMOVE\_SAVED\_REPORT procedure.

**Table 11-10 REMOVE\_SAVED\_REPORT Parameters**

Parameter	Description
p_application_id	The ID of the application for which to remove user saved interactive report information.
p_report_id	The ID of the saved user interactive report to be removed.

### Example

The following example demonstrates how to use the REMOVE\_SAVED\_REPORT procedure to remove user saved interactive report with the ID 123 for the application with an ID of 100.

```
BEGIN  
    APEX_INSTANCE_ADMIN.REMOVE_SAVED_REPORT(100,123);  
END;
```

---

## REMOVE\_SAVED\_REPORTS Procedure

The REMOVE\_SAVED\_REPORTS procedure removes all user saved interactive report settings for a particular application or for the entire instance.

### Syntax

```
APEX_INSTANCE_ADMIN.REMOVE_SAVED_REPORTS(  
    p_application_id    IN NUMBER DEFAULT NULL);
```

### Parameters

Table 11–11 describes the parameters available in the REMOVE\_SAVED\_REPORTS procedure.

**Table 11–11 REMOVE\_SAVED\_REPORTS Parameters**

Parameter	Description
p_application_id	The ID of the application for which to remove user saved interactive report information. If this parameter is left null, all user saved interactive reports for the entire instance is removed.

### Example

The following example demonstrates how to use the REMOVE\_SAVED\_REPORTS procedure to remove user saved interactive report information for the application with an ID of 100.

```
BEGIN  
    APEX_INSTANCE_ADMIN.REMOVE_SAVED_REPORTS(100);  
END;
```

## REMOVE\_SCHEMA Procedure

This REMOVE\_SCHEMA procedure removes a workspace to schema mapping.

### Syntax

```
APEX_INSTANCE_ADMIN.REMOVE_SCHEMA (  
    p_workspace    IN VARCHAR2,  
    p_schema       IN VARCHAR2);
```

### Parameters

[Table 11-12](#) describes the parameters available in the REMOVE\_SCHEMA procedure.

**Table 11-12 REMOVE\_SCHEMA Parameters**

Parameter	Description
p_workspace	The name of the workspace from which the schema mapping is removed.
p_schema	The schema to remove from the schema to workspace mapping.

### Example

The following example demonstrates how to use the REMOVE\_SCHEMA procedure to remove the schema named Frank from the MY\_WORKSPACE workspace to schema mapping.

```
BEGIN  
    APEX_INSTANCE_ADMIN.REMOVE_SCHEMA ('MY_WORKSPACE', 'FRANK');  
END;
```

---

## REMOVE\_SCHEMA\_EXCEPTION Procedure

This procedure removes an exception that allows the assignment of a restricted schema to a given workspace.

### Syntax

```
APEX_INSTANCE_ADMIN.REMOVE_SCHEMA_EXCEPTION (
    p_schema    in varchar2,
    p_workspace in varchar2 );
```

### Parameter

[Table 11-13](#) describes the parameters available for REMOVE\_SCHEMA\_EXCEPTION procedure.

**Table 11-13 REMOVE\_SCHEMA\_EXCEPTION Parameters**

Parameter	Description
p_schema	The schema.
p_workspace	The workspace.

### Example

This example removes the exception that allows the assignment of schema HR to workspace HR\_WORKSPACE.

```
begin
    apex_instance_admin.remove_schema_exception (
        p_schema    => 'HR',
        p_workspace => 'HR_WORKSPACE' );
    commit;
end;
```

**See also:** ["CREATE\\_SCHEMA\\_EXCEPTION Procedure"](#) on page 11-10, ["RESTRICT\\_SCHEMA Procedure"](#) on page 11-26, ["UNRESTRICT\\_SCHEMA Procedure"](#) on page 11-32, ["REMOVE\\_SCHEMA\\_EXCEPTIONS Procedure"](#) on page 11-20, ["REMOVE\\_WORKSPACE\\_EXCEPTIONS Procedure"](#) on page 11-23

---

## REMOVE\_SCHEMA\_EXCEPTIONS Procedure

This procedure removes all exceptions that allow the assignment of a given schema to workspaces.

### Syntax

```
APEX_INSTANCE_ADMIN.REMOVE_SCHEMA_EXCEPTIONS (  
    p_schema in varchar2 );
```

### Parameter

[Table 11-14](#) describes the parameters available for REMOVE\_SCHEMA\_EXCEPTIONS procedure.

**Table 11-14 REMOVE\_SCHEMA\_EXCEPTIONS Parameter**

Parameter	Description
p_schema	The schema.

### Example

This example removes all exceptions that allow the assignment of the HR schema to workspaces.

```
begin  
    apex_instance_admin.remove_schema_exceptions (  
        p_schema => 'HR' );  
    commit;  
end;
```

**See also:** ["CREATE\\_SCHEMA\\_EXCEPTION Procedure"](#) on page 11-10, ["RESTRICT\\_SCHEMA Procedure"](#) on page 11-26, ["UNRESTRICT\\_SCHEMA Procedure"](#) on page 11-32, ["REMOVE\\_SCHEMA\\_EXCEPTION Procedure"](#) on page 11-19, ["REMOVE\\_WORKSPACE\\_EXCEPTIONS Procedure"](#) on page 11-23

---

## REMOVE\_SUBSCRIPTION Procedure

The REMOVE\_SUBSCRIPTION procedure removes a specific interactive report subscription.

### Syntax

```
APEX_INSTANCE_ADMIN.REMOVE_SUBSCRIPTION(  
    p_subscription_id    IN NUMBER);
```

### Parameters

Table 11–15 describes the parameters available for the REMOVE\_SUBSCRIPTION procedure.

**Table 11–15 REMOVE\_SUBSCRIPTION Procedure Parameters**

Parameter	Description
p_subscription_id	The ID of the interactive report subscription to be removed.

### Example

The following example demonstrates how to use the REMOVE\_SUBSCRIPTION procedure to remove interactive report subscription with the ID 12345. Use of APEX\_APPLICATION\_PAGE\_IR\_SUB view can help identifying the subscription ID to remove.

```
BEGIN  
    APEX_INSTANCE_ADMIN.REMOVE_SUBSCRIPTION (  
        p_subscription_id => 12345);  
END;
```

---

## REMOVE\_WORKSPACE Procedure

The REMOVE\_WORKSPACE procedure removes a workspace from an Application Express instance.

### Syntax

```
APEX_INSTANCE_ADMIN.REMOVE_WORKSPACE(  
    p_workspace          IN VARCHAR2,  
    p_drop_users         IN VARCHAR2 DEFAULT 'N',  
    p_drop_tablespaces  IN VARCHAR2 DEFAULT 'N' );
```

### Parameters

Table 11–16 describes the parameters available in the REMOVE\_WORKSPACE procedure.

**Table 11–16 REMOVE\_WORKSPACE Parameters**

Parameter	Description
p_workspace	The name of the workspace to be removed.
p_drop_users	'Y' to drop the database user associated with the workspace. The default is 'N'.
p_drop_tablespaces	'Y' to drop the tablespace associated with the database user associated with the workspace. The default is 'N'.

### Example

The following example demonstrates how to use the REMOVE\_WORKSPACE procedure to remove an existing workspace named MY\_WORKSPACE, along with the associated database users and tablespace.

```
BEGIN  
    APEX_INSTANCE_ADMIN.REMOVE_WORKSPACE('MY_WORKSPACE', 'Y', 'Y');  
END;
```



---

## REMOVE\_WORKSPACE\_EXCEPTIONS Procedure

This procedure removes all exceptions that allow the assignment of restricted schemas to given workspace.

### Syntax

```
APEX_INSTANCE_ADMIN.REMOVE_WORKSPACE_EXCEPTIONS (  
    p_workspace in varchar2 );
```

### Parameter

[Table 11-14](#) describes the parameters available for REMOVE\_WORKSPACE\_EXCEPTIONS procedure.

**Table 11-17 REMOVE\_WORKSPACE\_EXCEPTIONS Parameter**

Parameter	Description
p_workspace	The workspace.

### Example

This example removes all exceptions that allow the assignment of restricted schemas to HR\_WORKSPACE.

```
begin  
    apex_instance_admin.remove_schema_exceptions (  
        p_workspace => 'HR_WORKSPACE' );  
    commit;  
end;
```

**See also:** ["CREATE\\_SCHEMA\\_EXCEPTION Procedure"](#) on page 11-10, ["RESTRICT\\_SCHEMA Procedure"](#) on page 11-26, ["UNRESTRICT\\_SCHEMA Procedure"](#) on page 11-32, ["REMOVE\\_SCHEMA\\_EXCEPTION Procedure"](#) on page 11-19, ["REMOVE\\_SCHEMA\\_EXCEPTIONS Procedure"](#) on page 11-20,

## RESERVE\_WORKSPACE\_APP\_IDS Procedure

This procedure permanently reserves the IDs of websheet and database applications in a given workspace. Even if the workspace and its applications get removed, developers can not create other applications with one of these IDs. To undo a reservation, see "[FREE\\_WORKSPACE\\_APP\\_IDS Procedure](#)" on page 11-11.

### Syntax

```
APEX_INSTANCE_ADMIN.RESERVE_WORKSPACE_APP_IDS (
    p_workspace_id IN NUMBER );
```

### Parameters

[Table 11-18](#) describes the parameters available in the RESERVE\_WORKSPACE\_APP\_IDS procedure.

**Table 11-18 RESERVE\_WORKSPACE\_APP\_IDS Parameters**

Parameter	Description
p_workspace_id	The unique ID of the workspace.

### Example

This example demonstrates setting up two separate Application Express instances where the application IDs are limited to within a specific range. At a later point, a workspace and all of its applications are moved from instance 1 to instance 2. For the workspace that is moved, the developer reserves all of its application IDs to ensure that no applications with the same IDs are created on instance 1.

1. After setting up Application Express instance 1, ensure that application IDs are between 100000 and 199999.

```
begin
    apex_instance_admin.set_parameter('APPLICATION_ID_MIN', 100000);
    apex_instance_admin.set_parameter('APPLICATION_ID_MAX', 199999);
end;
```

2. After setting up Application Express instance 2, ensure that application IDs are between 200000 and 299999.

```
begin
    apex_instance_admin.set_parameter('APPLICATION_ID_MIN', 200000);
    apex_instance_admin.set_parameter('APPLICATION_ID_MAX', 299999);
end;
```

3. Later, the operations team decides that workspace MY\_WORKSPACE with ID 1234567890 should be moved from instance 1 to instance 2. The required steps are:

- a. Export the workspace, applications and data on instance 1 (not shown here).
- b. Ensure that no other application on instance 1 can reuse application IDs of this workspace.

```
begin
    apex_instance_admin.reserve_workspace_app_ids(1234567890);
end;
```

- c. Drop workspace, accompanying data and users on instance 1.

```
begin
```

```
apex_instance_admin.remove_workspace('MY_WORKSPACE');  
end;
```

- d.** Import the workspace, applications and data on instance 2 (not shown here).

---

## RESTRICT\_SCHEMA Procedure

This procedure revokes the privilege to assign a schema to workspaces.

### Syntax

```
APEX_INSTANCE_ADMIN.RESTRICT_SCHEMA (  
    p_schema in varchar2 );
```

### Parameter

[Table 11–19](#) describes the parameters available for RESTRICT\_SCHEMA procedure.

**Table 11–19** RESTRICT\_SCHEMA Parameters

Parameter	Description
p_schema	The schema.

### Example

This example revokes the privilege to assign schema HR to workspaces.

```
begin  
    apex_instance_admin.restrict_schema(p_schema => 'HR');  
    commit;  
end;
```

**See also:** ["CREATE\\_SCHEMA\\_EXCEPTION Procedure"](#) on page 11-10, ["UNRESTRICT\\_SCHEMA Procedure"](#) on page 11-32, ["REMOVE\\_SCHEMA\\_EXCEPTION Procedure"](#) on page 11-19, ["REMOVE\\_SCHEMA\\_EXCEPTIONS Procedure"](#) on page 11-20, ["REMOVE\\_WORKSPACE\\_EXCEPTIONS Procedure"](#) on page 11-23

---

## SET\_LOG\_SWITCH\_INTERVAL Procedure

Set the log switch interval for each of the logs maintained by Application Express.

### Syntax

```
APEX_INSTANCE_ADMIN.SET_LOG_SWITCH_INTERVAL(  
    p_log_name           IN VARCHAR2,  
    p_log_switch_after_days IN NUMBER );
```

### Parameters

[Table 11–20](#) describes the parameters available in the SET\_LOG\_SWITCH\_INTERVAL procedure.

**Table 11–20** SET\_LOG\_SWITCH\_INTERVAL Parameters

Parameters	Description
p_log_name	Specifies the name of the log. Valid values include ACCESS, ACTIVITY, CLICKTHRU, and DEBUG.
p_log_switch_after_days	This interval must be a positive integer between 1 and 180.

### Example

This example sets the log switch interval for the ACTIVITY log to 30 days.

```
begin  
    apex_instance_admin.set_log_switch_interval( p_log_name in 'ACTIVITY', p_log_  
switch_after_days => 30 );  
    commit;  
end;
```

## SET\_WORKSPACE\_PARAMETER

The SET\_WORKSPACE\_PARAMETER procedure sets the designated workspace parameter.

### Syntax

```
SET_WORKSPACE_PARAMETER(  
    p_workspace      IN VARCHAR2,  
    p_parameter      IN VARCHAR2,  
    p_value          IN VARCHAR2 );
```

### Parameters

[Table 11–21](#) describes the parameters available in SET\_WORKSPACE\_PARAMETER procedure.

**Table 11–21** SET\_WORKSPACE\_PARAMETER Parameters

Parameter	Description
p_workspace	The name of the workspace to which you are setting the workspace parameter.
p_parameter	The parameter name which overrides the instance parameter value of the same for this workspace. Parameter names include: <ul style="list-style-type: none"><li>■ ALLOW_HOSTNAMES</li><li>■ QOS_MAX_WORKSPACE_REQUESTS</li><li>■ QOS_MAX_SESSION_REQUESTS</li><li>■ QOS_MAX_SESSION_KILL_TIMEOUT</li><li>■ RM_CONSUMER_GROUP</li><li>■ WORKSPACE_EMAIL_MAXIMUM</li><li>■ WORKSPACE_MAX_FILE_BYTES</li></ul>
p_value	The parameter value.

### Example

The following example demonstrates how to use the set\_workspace\_parameter procedure to restrict URLs for accessing applications in the HR workspace that have hr.example.com in the hostname or domain name.

```
BEGIN  
apex_instance_admin.set_workspace_parameter (  
    p_workspace => 'HR',  
    p_parameter => 'ALLOW_HOSTNAMES' );  
    p_value     => 'hr.example.com' );  
END;
```

---

## SET\_PARAMETER Procedure

The SET\_PARAMETER procedure sets a parameter used in administering a runtime environment. You must issue a commit for the parameter change to take affect.

### Syntax

```
APEX_INSTANCE_ADMIN.SET_PARAMETER (  
    p_parameter    IN VARCHAR2,  
    p_value        IN VARCHAR2 DEFAULT 'N');
```

### Parameters

[Table 11-22](#) describes the parameters available in the SET\_PARAMETER procedure.

**Table 11-22 SET\_PARAMETER Parameters**

Parameter	Description
p_parameter	The instance parameter to be set.
p_value	The value of the parameter. See " <a href="#">Available Parameter Values</a> " on page 11-2.

### Example

The following example demonstrates how to use the SET\_PARAMETER procedure to set the SMTP\_HOST\_ADDRESS parameter for an Oracle Application Express instance.

```
BEGIN  
    APEX_INSTANCE_ADMIN.SET_PARAMETER('SMTP_HOST_ADDRESS', 'mail.example.com');  
    COMMIT;  
END;
```

---

## SET\_WORKSPACE\_CONSUMER\_GROUP Procedure

The SET\_WORKSPACE\_CONSUMER\_GROUP procedure sets a Resource Manager Consumer Group to a workspace.

### Syntax

```
set_workspace_consumer_group(
    p_workspace in varchar2,
    p_rm_consumer_group in varchar2 );
```

### Parameters

Table 11-23 describes the parameters available for the SET\_WORKSPACE\_CONSUMER\_GROUP procedure.

**Table 11-23 SET\_WORKSPACE\_CONSUMER\_GROUP Parameters**

Parameters	Description
p_workspace	This is the name of the workspace for which the resource consumer group is to be set.
p_rm_consumer_group	The parameter P_RM_CONSUMER_GROUP is the Oracle Database Resource Manager Consumer Group name. The consumer group does not have to exist at the time this procedure is invoked. But if the Resource Manager Consumer Group is set for a workspace and the consumer group does not exist, then an error will be raised when anyone attempts to login to this workspace or execute any application in the workspace.  If the value of P_RM_CONSUMER_GROUP is null, then the Resource Manager consumer group associated with the specified workspace is cleared.

### Example

The following example sets the workspace to the Resource Manager consumer group "CUSTOM\_GROUP1":

```
begin
    apex_instance_admin.set_workspace_consumer_group(
        p_workspace => 'MY_WORKSPACE',
        p_rm_consumer_group => 'CUSTOM_GROUP1' );
    commit;
end;
/
```



---

## TRUNCATE\_LOG Procedure

The TRUNCATE\_LOG procedure truncates the log entries specified by the input parameter.

### Syntax

```
APEX_INSTANCE_ADMIN.TRUNCATE_LOG (
    p_log      IN VARCHAR2);
```

### Parameters

[Table 11-24](#) describes the parameters available in the TRUNCATE\_LOG procedure.

**Table 11-24 TRUNCATE\_LOG Parameters**

Parameter	Description
p_log	<p>This parameter can have one of the following values:</p> <p>ACTIVITY - removes all entries that record page access.</p> <p>USER_ACCESS - removes all entries that record user login.</p> <p>MAIL - removes all entries that record mail sent.</p> <p>CLICKS - removes all entries that record clicks tracked to external sites.</p> <p>LOCK_INSTALL_SCRIPT - removes all entries that record developer locking of supporting objects script.</p> <p>LOCK_PAGE - removes all entries that record developer locking of pages.</p> <p>WORKSPACE_HIST - removes all entries that record daily workspace summary.</p> <p>PURGE - removes all entries that record automatic workspace purge activity.</p> <p>FILE - removes all entries that record automatic file purge activity.</p> <p>SCRIPT - removes all entries that record results of SQL scripts executed in SQL Workshop.</p> <p>SQL - removes all entries that record the history of commands executed in SQL Workshop SQL Commands</p>

### Example

The following example demonstrates how to use the TRUNCATE\_LOG procedure to remove all log entries that record access to Application Express application pages.

```
BEGIN
    APEX_INSTANCE_ADMIN.TRUNCATE_LOG ('ACTIVITY');
END;
```

## UNRESTRICT\_SCHEMA Procedure

This procedure re-grants the privilege to assign a schema to workspaces, if it has been revoked before.

### Syntax

```
APEX_INSTANCE_ADMIN.UNRESTRICT_SCHEMA (  
    p_schema in varchar2 );
```

### Parameter

[Table 11-25](#) describes the parameters available for UNRESTRICT\_SCHEMA procedure.

**Table 11-25** *RESTRICT\_SCHEMA Parameters*

Parameter	Description
p_schema	The schema.

### Example

This example re-grants the privilege to assign schema HR to workspaces.

```
begin  
    apex_instance_admin.unrestrict_schema(p_schema => 'HR');  
    commit;  
end;
```

**See also:** ["CREATE\\_SCHEMA\\_EXCEPTION Procedure"](#) on page 11-10, ["RESTRICT\\_SCHEMA Procedure"](#) on page 11-26, ["REMOVE\\_SCHEMA\\_EXCEPTION Procedure"](#) on page 11-19, ["REMOVE\\_SCHEMA\\_EXCEPTIONS Procedure"](#) on page 11-20, ["REMOVE\\_WORKSPACE\\_EXCEPTIONS Procedure"](#) on page 11-23

The `APEX_IR` package provides utilities you can use when programming in the Oracle Application Express environment related to interactive reports. You can use the `APEX_IR` package to get an interactive report runtime query, add filters, reset or clear report settings, delete saved reports and manage subscriptions.

- [ADD\\_FILTER Procedure Signature 1](#)
- [ADD\\_FILTER Procedure Signature 2](#)
- [CHANGE\\_SUBSCRIPTION\\_EMAIL Procedure](#)
- [CHANGE\\_REPORT\\_OWNER Procedure](#)
- [CHANGE\\_SUBSCRIPTION\\_EMAIL Procedure](#)
- [CHANGE\\_SUBSCRIPTION\\_LANG Procedure](#)
- [CLEAR\\_REPORT Procedure Signature 1](#)
- [CLEAR\\_REPORT Procedure Signature 2](#)
- [DELETE\\_REPORT Procedure](#)
- [DELETE\\_SUBSCRIPTION Procedure](#)
- [GET\\_LAST\\_VIEWED\\_REPORT\\_ID Function](#)
- [GET\\_REPORT Function](#)
- [RESET\\_REPORT Procedure Signature 1](#)
- [RESET\\_REPORT Procedure Signature 2](#)

## ADD\_FILTER Procedure Signature 1

This procedure creates a filter on an interactive report using a report ID.

---

---

**Note:** The use of this procedure in a page rendering process causes report download issues (CSV, HTML, Email, and so on). When a user downloads the report, the interactive report reloads the page with download format in the REQUEST value. Any interactive report settings changes (such as add filter or reset report) are done in partial page refresh. Thus, the download data may not match the report data user is seeing. For this reason, Oracle recommends only using this procedure in a page submit process.

---

---

### Syntax

```
APEX_IR.ADD_FILTER(  
    p_page_id      IN NUMBER,  
    p_region_id    IN NUMBER,  
    p_report_column IN VARCHAR2,  
    p_filter_value  IN VARCHAR2,  
    p_operator_abbr IN VARCHAR2 DEFAULT NULL,  
    p_report_id    IN NUMBER DEFAULT NULL);
```

### Parameters

[Table 12–1](#) describes the available parameters for the ADD\_FILTER procedure signature 1.

**Table 12–1 ADD\_FILTER Procedure Signature 1 Parameters**

Parameter	Description
p_page_id	Page of the current Application Express application that contains an interactive report.
p_region_id	The interactive report region (ID).
p_report_column	Name of the report SQL column, or column alias, to be filtered.
p_filter_value	The filter value. This value is not used for N and NN.

**Table 12–1 (Cont.) ADD\_FILTER Procedure Signature 1 Parameters**

Parameter	Description
p_operator_abbr	Filter type. Valid values are as follows: EQ = Equals NEQ = Not Equals LT = Less than LTE = Less then or equal to GT = Greater Than GTE = Greater than or equal to LIKE = SQL Like operator NLIKE = Not Like N = Null NN = Not Null C = Contains NC = Not Contains IN = SQL In Operator NIN = SQL Not In Operator
p_report_id	The saved report ID within the current application page. If p_report_id is null, it adds the filter to the last viewed report settings.

**Example**

The following example shows how to use the ADD\_FILTER procedure to filter the interactive report with report ID of 880629800374638220 in page 1, region 2505704029884282 of the current application with DEPTNO equals 30.

```
BEGIN
  APEX_IR.ADD_FILTER(
    p_page_id      => 1,
    p_region_id    => 2505704029884282,
    p_report_column => 'DEPTNO',
    p_filter_value => '30',
    p_operator_abbr => 'EQ',
    p_report_id    => 880629800374638220);
END;
```

## ADD\_FILTER Procedure Signature 2

This procedure creates a filter on an interactive report using a report alias.

---

**Note:** The use of this procedure in a page rendering process causes report download issues (CSV, HTML, Email, and so on). When a user downloads the report, the interactive report reloads the page with download format in the REQUEST value. Any interactive report settings changes (such as add filter or reset report) are done in partial page refresh. Thus, the download data may not match the report data user is seeing. For this reason, Oracle recommends only using this procedure in a page submit process.

---

### Syntax

```
APEX_IR.ADD_FILTER(  
    p_page_id      IN NUMBER,  
    p_region_id    IN NUMBER,  
    p_report_column IN VARCHAR2,  
    p_filter_value  IN VARCHAR2,  
    p_operator_abbr IN VARCHAR2 DEFAULT NULL,  
    p_report_alias  IN VARCHAR2 DEFAULT NULL);
```

### Parameters

[Table 12–2](#) describes the available parameters for the ADD\_FILTER Procedure Signature 2.

**Table 12–2** ADD\_FILTER Procedure Signature 2

Parameter	Description
p_page_id	Page of the current Application Express application that contains an interactive report.
p_region_id	The interactive report region (ID).
p_report_column	Name of the report SQL column, or column alias, to be filtered.
p_filter_value	This is the filter value. This value is not used for N and NN.

**Table 12–2 (Cont.) ADD\_FILTER Procedure Signature 2**

Parameter	Description
p_operator_abbr	Filter type. Valid values are as follows: EQ = Equals NEQ = Not Equals LT = Less than LTE = Less then or equal to GT = Greater Than GTE = Greater than or equal to LIKE = SQL Like operator NLIKE = Not Like N = Null NN = Not Null C = Contains NC = Not Contains IN = SQL In Operator NIN = SQL Not In Operator
p_report_alias	The saved report alias within the current application page. If p_report_alias is null, it adds filter to the last viewed report settings.

**Example**

The following example shows how to use the ADD\_FILTER procedure to filter an interactive report with a report alias of CATEGORY\_REPORT in page 1, region 2505704029884282 of the current application with DEPTNO equals 30.

```
BEGIN
  APEX_IR.ADD_FILTER(
    p_page_id      => 1,
    p_region_id    => 2505704029884282,
    p_report_column => 'DEPTNO',
    p_filter_value => '30',
    p_operator_abbr => 'EQ',
    p_report_alias => 'CATEGORY_REPORT');
END;
```

---

## CHANGE\_SUBSCRIPTION\_EMAIL Procedure

This procedure changes interactive report subscriptions email address. When an email is sent out, the subscription sends message to the defined email address.

### Syntax

```
APEX_IR.CHANGE_SUBSCRIPTION_EMAIL (  
    p_subscription_id    IN NUMBER,  
    p_email_address      IN VARCHAR2);
```

### Parameters

Table 12-3 describes the parameters available in CHANGE\_SUBSCRIPTION\_EMAIL procedure.

**Table 12-3** CHANGE\_SUBSCRIPTION\_EMAIL Parameters

Parameter	Description
p_subscription_id	Subscription ID to change the email address within the current workspace.
p_email_address	The new email address to change to. The email address needs to be a valid email syntax and cannot be set to null.

### Example

The following example shows how to use CHANGE\_SUBSCRIPTION\_EMAIL procedure to change the email address to some.user@somecompany.com for the interactive report subscription 956136850459718525.

```
BEGIN  
    APEX_IR.CHANGE_SUBSCRIPTION_EMAIL (  
        p_subscription_id => 956136850459718525,  
        p_email_address => 'some.user@somecompany.com');  
END;
```



## CHANGE\_REPORT\_OWNER Procedure

This procedure changes the owner of a saved interactive report using a report ID. This procedure cannot change the owner of default interactive reports.

### Syntax

```
APEX_IR.CHANGE_REPORT_OWNER (
    p_report_id    in number,
    p_old_owner    in varchar2,
    p_new_owner    in varchar2);
```

### Parameters

Table 12–4 describes the available parameters for the CHANGE\_REPORT\_OWNER procedure.

**Table 12–4** CHANGE\_REPORT\_OWNER Procedure

Parameters	Description
p_report_id	The saved report ID within the current application page.
p_old_owner	The previous owner name to change from (case sensitive). The owner needs to a valid login user accessing the report.
p_new_owner	The new owner name to change to (case sensitive). The owner must be a valid login user accessing the report.

### Example

This example shows how to use CHANGE\_REPORT\_OWNER procedure to change the old owner name of *JOHN* to the new owner name of *JOHN.DOE* for a saved report. The saved report has a report ID of 1235704029884282.

```
BEGIN
    APEX_IR.CHANGE_REPORT_OWNER (
        p_report_id    => 1235704029884282,
        p_old_owner    => 'JOHN',
        p_new_owner    => 'JOHN.DOE');
END;
```

---

## CHANGE\_SUBSCRIPTION\_EMAIL Procedure

This procedure changes interactive report subscriptions email address. When an email is sent out, the subscription sends message to the defined email address.

### Syntax

```
APEX_IR.CHANGE_SUBSCRIPTION_EMAIL (  
    p_subscription_id    IN NUMBER,  
    p_email_address      IN VARCHAR2);
```

### Parameters

[Table 12-5](#) describes the parameters available in CHANGE\_SUBSCRIPTION\_EMAIL procedure.

**Table 12-5** CHANGE\_SUBSCRIPTION\_EMAIL Parameters

Parameter	Description
p_subscription_id	Subscription ID to change the email address within the current workspace.
p_email_address	The new email address to change to. The email address needs to be a valid email syntax and cannot be set to null.

### Example

The following example shows how to use CHANGE\_SUBSCRIPTION\_EMAIL procedure to change the email address to `some.user@somecompany.com` for the interactive report subscription 956136850459718525.

```
BEGIN  
    APEX_IR.CHANGE_SUBSCRIPTION_EMAIL (  
        p_subscription_id => 956136850459718525,  
        p_email_address => 'some.user@somecompany.com');  
END;
```

---

## CHANGE\_SUBSCRIPTION\_LANG Procedure

This procedure changes the interactive report subscription language.

### Syntax

```
APEX_IR.CHANGE_SUBSCRIPTION_LANG(  
    p_subscription_id IN NUMBER,  
    p_language        IN VARCHAR2);
```

### Parameters

[Table 12–6](#) describes the available parameters for the

**Table 12–6** *CHANGE\_SUBSCRIPTION\_LANG Procedure Parameters*

Parameter	Description
p_subscription_id	Subscription ID to change the language within the current workspace.
p_language	This is an IANA language code. Some examples include: en, de, de-at, zh-cn, and pt-br.

### Example

The following example shows how to use the CHANGE\_SUBSCRIPTION\_LANG procedure to change the subscription with the ID of 567890123 to German in the current workspace.

```
BEGIN  
    APEX_IR.CHANGE_SUBSCRIPTION_LANG(  
        p_subscription_id => 567890123,  
        p_language        => 'de');  
END;
```

---

## CLEAR\_REPORT Procedure Signature 1

This procedure clears report settings using the report ID.

---

**Note:** The use of this procedure in a page rendering process causes report download issues (CSV, HTML, Email, and so on). When a user downloads the report, the interactive report reloads the page with download format in the REQUEST value. Any interactive report settings changes (such as add filter or reset report) are done in partial page refresh. Thus, the download data may not match the report data user is seeing. For this reason, Oracle recommends only using this procedure in a page submit process.

---

### Syntax

```
APEX_IR.CLEAR_REPORT(
    p_page_id    IN NUMBER,
    p_region_id  IN NUMBER,
    p_report_id  IN NUMBER DEFAULT NULL);
```

### Parameters

Table 12-7 describes the available parameters for the CLEAR\_REPORT procedure signature 1.

**Table 12-7** CLEAR\_REPORT Procedure Signature 1 Parameters

Parameter	Description
p_page_id	Page of the current Application Express application that contains an interactive report.
p_region_id	The interactive report region (ID).
p_report_id	The saved report ID within the current application page. If p_report_id is null, it clears the last viewed report settings.

### Example

The following example shows how to use the CLEAR\_REPORT procedure to clear interactive report settings with a report ID of 880629800374638220 in page 1, region 2505704029884282 of the current application.

```
BEGIN
    APEX_IR.CLEAR_REPORT(
        p_page_id    => 1,
        p_region_id  => 2505704029884282,
        p_report_id  => 880629800374638220);
END;
```

## CLEAR\_REPORT Procedure Signature 2

This procedure clears report settings using report alias.

---

**Note:** The use of this procedure in a page rendering process causes report download issues (CSV, HTML, Email, and so on). When a user downloads the report, the interactive report reloads the page with download format in the REQUEST value. Any interactive report settings changes (such as add filter or reset report) are done in partial page refresh. Thus, the download data may not match the report data user is seeing. For this reason, Oracle recommends only using this procedure in a page submit process.

---

### Syntax

```
APEX_IR.CLEAR_REPORT(
    p_page_id      IN NUMBER,
    p_region_id    IN NUMBER,
    p_report_alias IN VARCHAR2 DEFAULT NULL);
```

### Parameters

Table 12–8 describes the available parameters for the CLEAR\_REPORT procedure signature 2.

**Table 12–8** CLEAR\_REPORT Procedure Signature 2 Parameters

Parameter	Description
p_page_id	Page of the current Application Express application that contains an interactive report.
p_region_id	The interactive report region (ID).
p_report_alias	The saved report alias within the current application page. If p_report_alias is null, it clears the last viewed report settings.

### Example

The following example shows how to use the CLEAR\_REPORT procedure to clear interactive report settings with report alias of CATEGORY\_REPORT in page 1, region 2505704029884282 of the current application.

```
BEGIN
    APEX_IR.CLEAR_REPORT(
        p_page_id      => 1,
        p_region_id    => 2505704029884282,
        p_report_alias => 'CATEGORY_REPORT');
END;
```

## DELETE\_REPORT Procedure

This procedure deletes saved interactive reports. It deletes a specific saved report in the current logged in workspace and application.

### Syntax

```
APEX_IR.DELETE_REPORT(  
    p_report_id IN NUMBER);
```

### Parameters

[Table 12–9](#) describes the available parameters for the DELETE\_REPORT procedure.

**Table 12–9** *DELETE\_REPORT Procedure Parameters*

Parameter	Description
p_report_id	Report ID to delete within the current Application Express application.

### Example

The following example shows how to use the DELETE\_REPORT procedure to delete the saved interactive report with ID of 880629800374638220 in the current application.

```
BEGIN  
    APEX_IR.DELETE_REPORT (  
        p_report_id => 880629800374638220);  
END;
```

---

## DELETE\_SUBSCRIPTION Procedure

This procedure deletes interactive report subscriptions.

### Syntax

```
APEX_IR.DELETE_SUBSCRIPTION(  
    p_subscription_id IN NUMBER);
```

### Parameters

Table 12–10 describes the available parameters for the DELETE\_SUBSCRIPTION procedure.

**Table 12–10** *DELETE\_SUBSCRIPTION Procedure Parameters*

Parameter	Description
p_subscription_id	Subscription ID to delete within the current workspace.

### Example

The following example shows how to use the DELETE\_SUBSCRIPTION procedure to delete the subscription with ID of 567890123 in the current workspace.

```
BEGIN  
    APEX_IR.DELETE_SUBSCRIPTION(  
        p_subscription_id => 567890123);  
END;
```

## GET\_LAST\_VIEWED\_REPORT\_ID Function

This function returns the last viewed base report ID of the specified page and region.

### Syntax

```
APEX_IR.GET_LAST_VIEWED_REPORT_ID(  
    p_page_id    IN NUMBER,  
    p_region_id  IN NUMBER);
```

### Parameters

[Table 12-11](#) describes the available parameters for the GET\_LAST\_VIEWED\_REPORT\_ID function.

**Table 12-11** GET\_LAST\_VIEWED\_REPORT\_ID Function Parameters

Parameter	Description
p_page_id	Page of the current Application Express application that contains an interactive report.
p_region_id	The interactive report region ID.

### Example

The following example shows how to use the GET\_LAST\_VIEWED\_REPORT\_ID function to retrieve the last viewed report ID in page 1, region 2505704029884282 of the current application.

```
DECLARE  
    l_report_id number;  
BEGIN  
    l_report_id := APEX_IR.GET_LAST_VIEWED_REPORT_ID (  
        p_page_id    => 1,  
        p_region_id => 2505704029884282);  
END;
```



## GET\_REPORT Function

This function returns an interactive report runtime query.

### Syntax

```
APEX_IR.GET_REPORT(
  p_page_id   IN NUMBER,
  p_region_id IN NUMBER,
  p_report_id IN NUMBER DEFAULT NULL);
```

### Parameters

Table 12–12 describes the available parameters for the GET\_REPORT function.

**Table 12–12** GET\_REPORT Function Parameters

Parameter	Description
p_page_id	Page of the current Application Express application that contains an interactive report.
p_region_id	The interactive report region ID.
p_report_id	The saved report ID within the current application page. If p_report_id is null, it gets last viewed report query.

### Example

The following example shows how to use the GET\_REPORT function to retrieve the runtime report query with bind variable information with report ID of 880629800374638220 in page 1, region 2505704029884282 of the current application.

```
DECLARE
  l_report apex_ir.t_report;
  l_query varchar2(32767);
BEGIN
  l_report := APEX_IR.GET_REPORT (
    p_page_id => 1,
    p_region_id => 2505704029884282,
    p_report_id => 880629800374638220);
  l_query := l_report.sql_query;
  sys.http.p('Statement = '||l_report.sql_query);
  for i in 1..l_report.binds.count
  loop
    sys.http.p(i||'. '||l_report.binds(i).name||' = '||l_
report.binds(i).value);
  end loop;
END;
```

## RESET\_REPORT Procedure Signature 1

This procedure resets report settings to the developer defined default settings using the report ID.

---

---

**Note:** The use of this procedure in a page rendering process causes report download issues (CSV, HTML, Email, and so on). When a user downloads the report, the interactive report reloads the page with download format in the REQUEST value. Any interactive report settings changes (such as add filter or reset report) are done in partial page refresh. Thus, the download data may not match the report data user is seeing. For this reason, Oracle recommends only using this procedure in a page submit process.

---

---

### Syntax

```
APEX_IR.RESET_REPORT(  
    p_page_id    IN NUMBER,  
    p_region_id  IN NUMBER,  
    p_report_id  IN NUMBER DEFAULT NULL);
```

### Parameters

[Table 12–13](#) describes the available parameters for the RESET\_REPORT procedure signature 1.

**Table 12–13** RESET\_REPORT Procedure Signature 1 Parameters

Parameter	Description
p_page_id	Page of the current Application Express application that contains an interactive report.
p_region_id	The interactive report region ID.
p_report_id	The saved report ID within the current application page. If p_report_id is null, it resets the last viewed report settings.

### Example

The following example shows how to use the RESET\_REPORT procedure signature 1 to reset interactive report settings with report ID of 880629800374638220 in page 1, region 2505704029884282 of the current application.

```
BEGIN  
    APEX_IR.RESET_REPORT(  
        p_page_id    => 1,  
        p_region_id  => 2505704029884282,  
        p_report_id  => 880629800374638220);  
END;
```

## RESET\_REPORT Procedure Signature 2

This procedure resets report settings using the report alias.

---

**Note:** The use of this procedure in a page rendering process causes report download issues (CSV, HTML, Email, and so on). When a user downloads the report, the interactive report reloads the page with download format in the REQUEST value. Any interactive report settings changes (such as add filter or reset report) are done in partial page refresh. Thus, the download data may not match the report data user is seeing. For this reason, Oracle recommends only using this procedure in a page submit process.

---

### Syntax

```
APEX_IR.RESET_REPORT(
    p_page_id      IN NUMBER,
    p_region_id    IN NUMBER,
    p_report_alias IN VARCHAR2 DEFAULT NULL);
```

### Parameters

[Table 12–14](#) describes the available parameters for the RESET\_REPORT procedure signature 2.

**Table 12–14** RESET\_REPORT Procedure Signature 2 Parameters

Parameter	Description
p_page_id	Page of the current Application Express application that contains an interactive report.
p_region_id	The interactive report region ID.
p_report_alias	The saved report alias within the current application page. If p_report_alias is null, it resets the last viewed report settings.

### Example

The following example shows how to use the RESET\_REPORT procedure to reset interactive report settings with a report alias of CATEGORY\_REPORT in page 1, region 2505704029884282 of the current application.

```
BEGIN
    APEX_IR.RESET_REPORT(
        p_page_id      => 1,
        p_region_id    => 2505704029884282,
        p_report_alias => 'CATEGORY_REPORT');
END;
```



You can use the APEX\_ITEM package to create form elements dynamically based on a SQL query instead of creating individual items page by page.

- CHECKBOX2 Function
- DATE\_POPUP Function
- DATE\_POPUP2 Function
- DISPLAY\_AND\_SAVE Function
- HIDDEN Function
- MD5\_CHECKSUM Function
- MD5\_HIDDEN Function
- POPUP\_FROM\_LOV Function
- POPUP\_FROM\_QUERY Function
- POPUPKEY\_FROM\_LOV Function
- POPUPKEY\_FROM\_QUERY Function
- RADIOGROUP Function
- SELECT\_LIST Function
- SELECT\_LIST\_FROM\_LOV Function
- SELECT\_LIST\_FROM\_LOV\_XL Function
- SELECT\_LIST\_FROM\_QUERY Function
- SELECT\_LIST\_FROM\_QUERY\_XL Function
- TEXT Function
- TEXTAREA Function
- TEXT\_FROM\_LOV Function
- TEXT\_FROM\_LOV\_QUERY Function

## CHECKBOX2 Function

This function creates check boxes.

### Syntax

```
APEX_ITEM.CHECKBOX2 (
    p_idx                IN    NUMBER,
    p_value              IN    VARCHAR2 DEFAULT NULL,
    p_attributes        IN    VARCHAR2 DEFAULT NULL,
    p_checked_values    IN    VARCHAR2 DEFAULT NULL,
    p_checked_values_delimiter IN  VARCHAR2 DEFAULT ':',
    p_item_id          IN    VARCHAR2 DEFAULT NULL,
    p_item_label       IN    VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

Table 13–1 describes the parameters available in the CHECKBOX2 function.

**Table 13–1** CHECKBOX2 Parameters

Parameter	Description
p_idx	Number that determines which APEX_APPLICATION global variable is used. Valid range of values is 1 to 50. For example 1 creates F01 and 2 creates F02
p_value	Value of a check box, hidden field, or input form item
p_attributes	Controls HTML tag attributes (such as disabled)
p_checked_values	Values to be checked by default
p_checked_values_delimiter	Delimits the values in the previous parameter, p_checked_values
p_item_id	HTML attribute ID for the <input> tag
p_item_label	Invisible label created for the item

### Examples of Default Check Box Behavior

The following example demonstrates how to create a selected check box for each employee in the emp table.

```
SELECT APEX_ITEM.CHECKBOX2(1,empno,'CHECKED') "Select",
       ename, job
FROM   emp
ORDER BY 1
```

The following example demonstrates how to have all check boxes for employees display without being selected.

```
SELECT APEX_ITEM.CHECKBOX2(1,empno) "Select",
       ename, job
FROM   emp
ORDER BY 1
```

The following example demonstrates how to select the check boxes for employees who work in department 10.

```
SELECT APEX_ITEM.CHECKBOX2(1,empno,DECODE(deptno,10,'CHECKED',NULL)) "Select",
       ename, job
FROM   emp
ORDER BY 1
```

The next example demonstrates how to select the check boxes for employees who work in department 10 or department 20.

```
SELECT APEX_ITEM.CHECKBOX2(1,deptno,NULL,'10:20',':') "Select",
       ename, job
FROM   emp
ORDER BY 1
```

### Creating an On-Submit Process

If you are using check boxes in your application, you might need to create an On Submit process to perform a specific type of action on the selected rows. For example, you could have a Delete button that uses the following logic:

```
SELECT APEX_ITEM.CHECKBOX2(1,empno) "Select",
       ename, job
FROM   emp
ORDER by 1
```

Consider the following sample on-submit process:

```
FOR I in 1..APEX_APPLICATION.G_F01.COUNT LOOP
    DELETE FROM emp WHERE empno = to_number(APEX_APPLICATION.G_F01(i));
END LOOP;
```

The following example demonstrates how to create unselected checkboxes for each employee in the emp table, with a unique ID. This is useful for referencing records from within JavaScript code:

```
SELECT APEX_ITEM.CHECKBOX2(1,empno,NULL,NULL,NULL,'f01_#ROWNUM#') "Select",
       ename, job
FROM   emp
ORDER BY 1
```

## DATE\_POPUP Function

Use this function with forms that include date fields. The DATE\_POPUP function dynamically generates a date field that has a popup calendar button.

### Syntax

```
APEX_ITEM.DATE_POPUP (
    p_idx           IN     NUMBER,
    p_row           IN     NUMBER,
    p_value         IN     VARCHAR2 DEFAULT NULL,
    p_date_format   IN     DATE DEFAULT 'DD-MON-YYYY',
    p_size          IN     NUMBER DEFAULT 20,
    p_maxlength     IN     NUMBER DEFAULT 2000,
    p_attributes    IN     VARCHAR2 DEFAULT NULL,
    p_item_id       IN     VARCHAR2 DEFAULT NULL,
    p_item_label    IN     VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

[Table 13–2](#) describes the parameters available in the DATE\_POPUP function.

**Table 13–2** DATE\_POPUP Parameters

Parameter	Description
p_idx	Number that determines which APEX_APPLICATION global variable is used. Valid range of values is 1 to 50. For example, 1 creates F01 and 2 creates F02
p_row	This parameter is deprecated. Anything specified for this value is ignored
p_value	Value of a field item
p_date_format	Valid database date format
p_size	Controls HTML tag attributes (such as disabled)
p_maxlength	Determines the maximum number of enterable characters. Becomes the maxlength attribute of the <input> HTML tag
p_attributes	Extra HTML parameters you want to add
p_item_id	HTML attribute ID for the <input> tag
p_item_label	Invisible label created for the item

**See Also:** *Oracle Database SQL Language Reference* for information about the TO\_CHAR or TO\_DATE functions

### Example

The following example demonstrates how to use APEX\_ITEM.DATE\_POPUP to create popup calendar buttons for the hiredate column.

```
SELECT
    empno,
    APEX_ITEM.HIDDEN(1, empno) ||
    APEX_ITEM.TEXT(2, ename) ename,
    APEX_ITEM.TEXT(3, job) job,
```



```
mgr,  
APEX_ITEM.DATE_POPUP(4,rownum,hiredate,'dd-mon-yyyy') hd,  
APEX_ITEM.TEXT(5,sal) sal,  
APEX_ITEM.TEXT(6,comm) comm,  
deptno  
FROM emp  
ORDER BY 1
```

## DATE\_POPUP2 Function

Use this function with forms that include date fields. The DATE\_POPUP2 function dynamically generates a date field that has a jQuery based popup calendar with button.

### Syntax

```
APEX_ITEM.DATE_POPUP2 (
    p_idx                in number,
    p_value              in date    default null,
    p_date_format        in varchar2 default null,
    p_size               in number  default 20,
    p_maxLength          in number  default 2000,
    p_attributes         in varchar2 default null,
    p_item_id            in varchar2 default null,
    p_item_label         in varchar2 default null,
    p_default_value      in varchar2 default null,
    p_max_value          in varchar2 default null,
    p_min_value          in varchar2 default null,
    p_show_on            in varchar2 default 'button',
    p_number_of_months   in varchar2 default null,
    p_navigation_list_for in varchar2 default 'NONE',
    p_year_range         in varchar2 default null,
    p_validation_date    in varchar2 default null)
RETURN VARCHAR2;
```

### Parameters

Table 13–3 describes the parameters available in the DATE\_POPUP2 function.

**Table 13–3** DATE\_POPUP2 Parameters

Parameter	Description
p_idx	Number that determines which APEX_APPLICATION global variable is used. Valid range of values is 1 to 50. For example, 1 creates F01 and 2 creates F02.
p_value	Value of a field item
p_date_format	Valid database date format
p_size	Controls HTML tag attributes (such as disabled)
p_maxlength	Determines the maximum number of enterable characters. Becomes the maxlength attribute of the <input> HTML tag
p_attributes	Extra HTML parameters you want to add
p_item_id	HTML attribute ID for the <input> tag
p_item_label	Invisible label created for the item
p_default_value	The default date which should be selected in DatePicker calendar popup
p_max_value	The Maximum date that can be selected from the datepicker
p_min_value	The Minimum date that can be selected from the datepicker.

**Table 13–3 (Cont.) DATE\_POPUP2 Parameters**

<b>Parameter</b>	<b>Description</b>
p_show_on	Determines when the datepicker displays, on button click or on focus of the item or both.
p_number_of_months	Determines number of months displayed. Value should be in array formats follows: [row,column]
p_navigation_list_for	Determines if a select list is displayed for Changing Month, Year or Both. Possible values include: MONTH, YEAR, MONTH_AND_YEAR and default is null.
p_year_range	The range of years displayed in the year selection list.
p_validation_date	Used to store the Date value for the which date validation failed

**See Also:** *Oracle Database SQL Language Reference* for information about the TO\_CHAR or TO\_DATE functions

## DISPLAY\_AND\_SAVE Function

Use this function to display an item as text, but save its value to session state.

### Syntax

```
APEX_ITEM.DISPLAY_AND_SAVE (
  p_idx          IN    NUMBER,
  p_value        IN    VARCHAR2 DEFAULT NULL,
  p_item_id      IN    VARCHAR2 DEFAULT NULL,
  p_item_label   IN    VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

Table 13–4 describes the parameters available in the DISPLAY\_AND\_SAVE function.

**Table 13–4** DISPLAY\_AND\_SAVE Parameters

Parameter	Description
p_idx	Number that determines which APEX_APPLICATION global variable is used. Valid range of values is 1 to 50. For example, 1 creates F01 and 2 creates F02
p_value	Current value
p_item_id	HTML attribute ID for the <span> tag
p_item_label	Invisible label created for the item

### Example

The following example demonstrates how to use the APEX\_ITEM.DISPLAY\_AND\_SAVE function.

```
SELECT APEX_ITEM.DISPLAY_AND_SAVE(10, empno) c FROM emp
```

## HIDDEN Function

This function dynamically generates hidden form items.

### Syntax

```
APEX_ITEM.HIDDEN(
  p_idx          IN    NUMBER,
  p_value        IN    VARCHAR2 DEFAULT
  p_attributes   IN    VARCHAR2 DEFAULT NULL,
  p_item_id      IN    VARCHAR2 DEFAULT NULL,
  p_item_label   IN    VARCHAR2 DEFAULT NULL
) RETURN VARCHAR2;
```

### Parameters

Table 13–5 describes the parameters available in the HIDDEN function.

**Table 13–5 HIDDEN Parameters**

Parameter	Description
p_idx	Number to identify the item you want to generate. The number determines which G_FXX global is populated <b>See Also:</b> "APEX_APPLICATION" on page 1-1
p_value	Value of the hidden input form item
p_attributes	Extra HTML parameters you want to add
p_item_id	HTML attribute ID for the <input> tag
p_item_label	Invisible label created for the item

### Example

Typically, the primary key of a table is stored as a hidden column and used for subsequent update processing, for example:

```
SELECT
  empno,
  APEX_ITEM.HIDDEN(1, empno) ||
  APEX_ITEM.TEXT(2, ename)  ename,
  APEX_ITEM.TEXT(3, job)   job,
  mgr,
  APEX_ITEM.DATE_POPUP(4, rownum, hiredate, 'dd-mon-yyyy') hiredate,
  APEX_ITEM.TEXT(5, sal)   sal,
  APEX_ITEM.TEXT(6, comm) comm,
  deptno
FROM emp
ORDER BY 1
```

The previous query could use the following page process to process the results:

```
BEGIN
  FOR i IN 1..APEX_APPLICATION.G_F01.COUNT LOOP
    UPDATE emp
      SET
        ename=APEX_APPLICATION.G_F02(i),
        job=APEX_APPLICATION.G_F03(i),
```

```
        hiredate=to_date(APEX_APPLICATION.G_F04(i), 'dd-mon-yyyy'),
        sal=APEX_APPLICATION.G_F05(i),
        comm=APEX_APPLICATION.G_F06(i)
    WHERE empno=to_number(APEX_APPLICATION.G_F01(i));
END LOOP;
END;
```

Note that the G\_F01 column (which corresponds to the hidden EMPNO) is used as the key to update each row.

## MD5\_CHECKSUM Function

Use this function for lost update detection. Lost update detection ensures data integrity in applications where data can be accessed concurrently.

This function produces hidden form field(s) with a name attribute equal to 'fcs' and as value a MD5 checksum based on up to 50 inputs. APEX\_ITEM.MD5\_CHECKSUM also produces an MD5 checksum using Oracle database DBMS\_CRYPTO:

```
UTL_RAW.CAST_TO_RAW(DBMS_CRYPTO.MD5())
```

An MD5 checksum provides data integrity through hashing and sequencing to ensure that data is not altered or stolen as it is transmitted over a network.

### Syntax

```
APEX_ITEM.MD5_CHECKSUM(
  p_value01 IN VARCHAR2 DEFAULT NULL,
  p_value02 IN VARCHAR2 DEFAULT NULL,
  p_value03 IN VARCHAR2 DEFAULT NULL,
  ...
  p_value50 IN VARCHAR2 DEFAULT NULL,
  p_col_sep IN VARCHAR2 DEFAULT '|',
  p_item_id IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

Table 13–6 describes the parameters available in the MD5\_CHECKSUM function.

**Table 13–6 MD5\_CHECKSUM Parameters**

Parameter	Description
p_value01	Fifty available inputs. If no parameters are supplied, the default to NULL
...	
p_value50	
p_col_sep	String used to separate p_value inputs. Defaults to the pipe symbol ( )
p_item_id	ID of the HTML form item

### Example

This function generates hidden form elements with the name 'fcs'. The values can subsequently be accessed by using the APEX\_APPLICATION.G\_FCS array.

```
SELECT APEX_ITEM.MD5_CHECKSUM(ename, job, sal) md5_cks,
       ename, job, sal
FROM emp
```

## MD5\_HIDDEN Function

Use this function for lost update detection. Lost update detection ensures data integrity in applications where data can be accessed concurrently.

This function produces a hidden form field with a MD5 checksum as value which is based on up to 50 inputs. `APEX_ITEM.MD5_HIDDEN` also produces an MD5 checksum using Oracle database `DBMS_CRYPTO`:

```
UTL_RAW.CAST_TO_RAW(DBMS_CRYPTO.MD5())
```

An MD5 checksum provides data integrity through hashing and sequencing to ensure that data is not altered or stolen as it is transmitted over a network

### Syntax

```
APEX_ITEM.MD5_HIDDEN(
  p_idx      IN      NUMBER,
  p_value01  IN      VARCHAR2 DEFAULT NULL,
  p_value02  IN      VARCHAR2 DEFAULT NULL,
  p_value03  IN      VARCHAR2 DEFAULT NULL,
  ...
  p_value50  IN      VARCHAR2 DEFAULT NULL,
  p_col_sep  IN      VARCHAR2 DEFAULT '|',
  p_item_id  IN      VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

[Table 13–7](#) describes the parameters available in the `MD5_HIDDEN` function.

**Table 13–7 MD5\_HIDDEN Parameters**

Parameter	Description
<code>p_idx</code>	Indicates the form element to be generated. For example, 1 equals F01 and 2 equals F02. Typically the <code>p_idx</code> parameter is constant for a given column
<code>p_value01</code>	Fifty available inputs. Parameters not supplied default to NULL
...	
<code>p_value50</code>	
<code>p_col_sep</code>	String used to separate <code>p_value</code> inputs. Defaults to the pipe symbol ( )
<code>p_item_id</code>	ID of the HTML form item

### Example

The `p_idx` parameter specifies the FXX form element to be generated. In the following example, 7 generates F07. Also note that an HTML hidden form element is generated.

```
SELECT APEX_ITEM.MD5_HIDDEN(7,ename,job,sal)md5_h, ename, job, sal
FROM emp
```



## POPUP\_FROM\_LOV Function

This function generates an HTML popup select list from an application shared list of values (LOV). Like other available functions in the APEX\_ITEM package, POPUP\_FROM\_LOV function is designed to generate forms with F01 to F50 form array elements.

### Syntax

```
APEX_ITEM.POPUP_FROM_LOV (
    p_idx          IN      NUMBER,
    p_value        IN      VARCHAR2 DEFAULT NULL,
    p_lov_name     IN      VARCHAR2,
    p_width        IN      VARCHAR2 DEFAULT NULL,
    p_max_length   IN      VARCHAR2 DEFAULT NULL,
    p_form_index   IN      VARCHAR2 DEFAULT '0',
    p_escape_html  IN      VARCHAR2 DEFAULT NULL,
    p_max_elements IN      VARCHAR2 DEFAULT NULL,
    p_attributes   IN      VARCHAR2 DEFAULT NULL,
    p_ok_to_query  IN      VARCHAR2 DEFAULT 'YES',
    p_item_id      IN      VARCHAR2 DEFAULT NULL,
    p_item_label   IN      VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

Table 13–8 describes the available parameters in the POPUP\_FROM\_LOV function.

**Table 13–8** POPUP\_FROM\_LOV Parameters

Parameter	Description
p_idx	Form element name. For example, 1 equals F01 and 2 equals F02. Typically, p_idx is a constant for a given column
p_value	Form element current value. This value should be one of the values in the p_lov_name parameter
p_lov_name	Named LOV used for this popup
p_width	Width of the text box
p_max_length	Maximum number of characters that can be entered in the text box
p_form_index	HTML form on the page in which an item is contained. Defaults to 0 and rarely used.  Only use this parameter when it is necessary to embed a custom form in your page template (such as a search field that posts to a different website). If this form comes before the #FORM_OPEN# substitution string, then its index is zero and the form opened automatically by Oracle Application Express must be referenced as form 1. This functionality supports the JavaScript used in the popup LOV that passes a value back to a form element.

**Table 13–8 (Cont.) POPUP\_FROM\_LOV Parameters**

Parameter	Description
p_escape_html	Replacements for special characters that require an escaped equivalent: <ul style="list-style-type: none"> <li>■ &amp;lt; for &lt;</li> <li>■ &amp;gt; for &gt;</li> <li>■ &amp;amp; for &amp;</li> </ul> Range of values is YES and NO. If YES, special characters are escaped. This parameter is useful if you know your query returns illegal HTML.
p_max_elements	Limit on the number of rows that can be returned by your query. Limits the performance impact of user searches. By entering a value in this parameter, you force the user to search for a narrower set of results.
p_attributes	Additional HTML attributes to use for the form item.
p_ok_to_query	Range of values is YES and NO. If YES, a popup returns first set of rows for the LOV. If NO, a search is initiated to return rows.
p_item_id	ID attribute of the form element.
p_item_label	Invisible label created for the item.

**Example**

The following example demonstrates a sample query the generates a popup from an LOV named DEPT\_LOV.

```
SELECT APEX_ITEM.POPUP_FROM_LOV (1,deptno, 'DEPT_LOV') dt
FROM emp
```

## POPUP\_FROM\_QUERY Function

This function generates an HTML popup select list from a query. Like other available functions in the APEX\_ITEM package, the POPUP\_FROM\_QUERY function is designed to generate forms with F01 to F50 form array elements.

### Syntax

```
APEX_ITEM.POPUP_FROM_QUERY (

    p_idx           IN     NUMBER,
    p_value         IN     VARCHAR2 DEFAULT NULL,
    p_lov_query     IN     VARCHAR2,
    p_width         IN     VARCHAR2 DEFAULT NULL,
    p_max_length    IN     VARCHAR2 DEFAULT NULL,
    p_form_index    IN     VARCHAR2 DEFAULT '0',
    p_escape_html   IN     VARCHAR2 DEFAULT NULL,
    p_max_elements  IN     VARCHAR2 DEFAULT NULL,
    p_attributes    IN     VARCHAR2 DEFAULT NULL,
    p_ok_to_query   IN     VARCHAR2 DEFAULT 'YES',
    p_item_id       IN     VARCHAR2 DEFAULT NULL,
    p_item_label    IN     VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

Table 13–9 describes the available parameters in the POPUP\_FROM\_QUERY function.

**Table 13–9 POPUP\_FROM\_QUERY Parameters**

Parameter	Description
p_idx	Form element name. For example, 1 equals F01 and 2 equals F02. Typically, p_idx is a constant for a given column.
p_value	Form element current value. This value should be one of the values in the p_lov_query parameter.
p_lov_query	SQL query that is expected to select two columns (a display column and a return column). For example: SELECT dname, deptno FROM dept
p_width	Width of the text box.
p_max_length	Maximum number of characters that can be entered in the text box.
p_form_index	HTML form on the page in which an item is contained. Defaults to 0 and rarely used.  Only use this parameter when it is necessary to embed a custom form in your page template (such as a search field that posts to a different website). If this form comes before the #FORM_OPEN# substitution string, then its index is zero and the form opened automatically by Oracle Application Express must be referenced as form 1. This functionality supports the JavaScript used in the popup LOV that passes a value back to a form element.

**Table 13–9 (Cont.) POPUP\_FROM\_QUERY Parameters**

Parameter	Description
p_escape_html	<p>Replacements for special characters that require an escaped equivalent.</p> <ul style="list-style-type: none"> <li>■ &amp;lt; for &lt;</li> <li>■ &amp;gt; for &gt;</li> <li>■ &amp;amp; for &amp;</li> </ul> <p>Range of values is YES and NO. If YES, special characters are escaped. This parameter is useful if you know your query returns illegal HTML.</p>
p_max_elements	<p>Limit on the number of rows that can be returned by your query. Limits the performance impact of user searches. By entering a value in this parameter, you force the user to search for a narrower set of results.</p>
p_attributes	<p>Additional HTML attributes to use for the form item.</p>
p_ok_to_query	<p>Range of values is YES and NO. If YES, a popup returns the first set of rows for the LOV. If NO, a search is initiated to return rows.</p>
p_item_id	<p>ID attribute of the form element.</p>
p_item_label	<p>Invisible label created for the item.</p>

**Example**

The following example demonstrates a sample query the generates a popup select list from the emp table.

```
SELECT APEX_ITEM.POPUP_FROM_QUERY (1,deptno,'SELECT dname, deptno FROM dept') dt
FROM emp
```

## POPUPKEY\_FROM\_LOV Function

This function generates a popup key select list from a shared list of values (LOV). Similar to other available functions in the APEX\_ITEM package, the POPUPKEY\_FROM\_LOV function is designed to generate forms with F01 to F50 form array elements.

### Syntax

```
APEX_ITEM.POPUPKEY_FROM_LOV(
    p_idx          IN     NUMBER,
    p_value        IN     VARCHAR2 DEFAULT NULL,
    p_lov_name     IN     VARCHAR2,
    p_width        IN     VARCHAR2 DEFAULT NULL,
    p_max_length   IN     VARCHAR2 DEFAULT NULL,
    p_form_index   IN     VARCHAR2 DEFAULT '0',
    p_escape_html  IN     VARCHAR2 DEFAULT NULL,
    p_max_elements IN     VARCHAR2 DEFAULT NULL,
    p_attributes   IN     VARCHAR2 DEFAULT NULL,
    p_ok_to_query  IN     VARCHAR2 DEFAULT 'YES',
    p_item_id      IN     VARCHAR2 DEFAULT NULL,
    p_item_label   IN     VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

Although the text field associated with the popup displays in the first column in the LOV query, the actual value is specified in the second column in the query.

### Parameters

Table 13–10 describes the available parameters in the POPUPKEY\_FROM\_LOV function.

**Table 13–10 POPUPKEY\_FROM\_LOV Parameters**

Parameter	Description
p_idx	Identifies a form element name. For example, 1 equals F01 and 2 equals F02. Typically, p_idx is a constant for a given column  Because of the behavior of POPUPKEY_FROM_QUERY, the next index value should be p_idx + 1. For example:  SELECT APEX_ITEM.POPUPKEY_FROM_LOV (1,deptno, 'DEPT') dt, APEX_ITEM.HIDDEN (3, empno) eno
p_value	Indicates the current value. This value should be one of the values in the P_LOV_NAME parameter.
p_lov_name	Identifies a named LOV used for this popup.
p_width	Width of the text box.
p_max_length	Maximum number of characters that can be entered in the text box.
p_form_index	HTML form on the page in which an item is contained. Defaults to 0 and rarely used.  Only use this parameter when it is necessary to embed a custom form in your page template (such as a search field that posts to a different website). If this form comes before the #FORM_OPEN# substitution string, then its index is zero and the form opened automatically by Oracle Application Express must be referenced as form 1. This functionality supports the JavaScript used in the popup LOV that passes a value back to a form element.

**Table 13–10 (Cont.) POPUPKEY\_FROM\_LOV Parameters**

Parameter	Description
p_escape_html	Replacements for special characters that require an escaped equivalent. <ul style="list-style-type: none"> <li>▪ &amp;lt; for &lt;</li> <li>▪ &amp;gt; for &gt;</li> <li>▪ &amp;amp; for &amp;</li> </ul> This parameter is useful if you know your query returns illegal HTML.
p_max_elements	Limit on the number of rows that can be returned by your query. Limits the performance impact of user searches. By entering a value in this parameter, you force the user to search for a narrower set of results.
p_attributes	Additional HTML attributes to use for the form item.
p_ok_to_query	Range of values is YES and NO. If YES, a popup returns the first set of rows for the LOV. If NO, a search is initiated to return rows.
p_item_id	HTML attribute ID for the <input> tag
p_item_label	Invisible label created for the item

**Example**

The following example demonstrates how to generate a popup key select list from a shared list of values (LOV).

```
SELECT APEX_ITEM.POPUPKEY_FROM_LOV (1,deptno,'DEPT') dt
FROM emp
```

## POPUPKEY\_FROM\_QUERY Function

This function generates a popup key select list from a SQL query. Similar to other available functions in the APEX\_ITEM package, the POPUPKEY\_FROM\_QUERY function is designed to generate forms with F01 to F50 form array elements.

### Syntax

```
APEX_ITEM.POPUPKEY_FROM_QUERY (
  p_idx          IN     NUMBER,
  p_value        IN     VARCHAR2 DEFAULT NULL,
  p_lov_query    IN     VARCHAR2,
  p_width       IN     VARCHAR2 DEFAULT NULL,
  p_max_length  IN     VARCHAR2 DEFAULT NULL,
  p_form_index  IN     VARCHAR2 DEFAULT '0',
  p_escape_html IN     VARCHAR2 DEFAULT NULL,
  p_max_elements IN    VARCHAR2 DEFAULT NULL,
  p_attributes  IN     VARCHAR2 DEFAULT NULL,
  p_ok_to_query IN     VARCHAR2 DEFAULT 'YES',
  p_item_id     IN     VARCHAR2 DEFAULT NULL,
  p_item_label  IN     VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

Table 13–11 describes the available parameters in the POPUPKEY\_FROM\_QUERY function.

**Table 13–11 POPUPKEY\_FROM\_QUERY Parameters**

Parameter	Description
p_idx	Form element name. For example, 1 equals F01 and 2 equals F02. Typically, p_idx is a constant for a given column.  Because of the behavior of POPUPKEY_FROM_QUERY, the next index value should be p_idx + 1. For example:  SELECT APEX_ITEM.POPUPKEY_FROM_QUERY (1,deptno, 'SELECT dname, deptno FROM dept') dt, APEX_ITEM.HIDDEN(3,empno) eno
p_value	Form element current value. This value should be one of the values in the P_LOV_QUERY parameter.
p_lov_query	LOV query used for this popup.
p_width	Width of the text box.
p_max_length	Maximum number of characters that can be entered in the text box.
p_form_index	HTML form on the page in which an item is contained. Defaults to 0 and rarely used.  Only use this parameter when it is necessary to embed a custom form in your page template (such as a search field that posts to a different website). If this form comes before the #FORM_OPEN# substitution string, then its index is zero and the form opened automatically by Oracle Application Express must be referenced as form 1. This functionality supports the JavaScript used in the popup LOV that passes a value back to a form element.

**Table 13–11 (Cont.) POPUPKEY\_FROM\_QUERY Parameters**

Parameter	Description
p_escape_html	Replacements for special characters that require an escaped equivalent. <ul style="list-style-type: none"> <li>■ &amp;lt; for &lt;</li> <li>■ &amp;gt; for &gt;</li> <li>■ &amp;amp; for &amp;</li> </ul> This parameter is useful if you know your query returns illegal HTML.
p_max_elements	Limit on the number of rows that can be returned by your query. Limits the performance impact of user searches. By entering a value in this parameter, you force the user to search for a narrower set of results.
p_attributes	Additional HTML attributes to use for the form item.
p_ok_to_query	Range of values is YES and NO. If YES, a popup returns first set of rows for the LOV. If NO, a search is initiated to return rows.
p_item_id	ID attribute of the form element.
p_item_label	Invisible label created for the item.

**Example**

The following example demonstrates how to generate a popup select list from a SQL query.

```
SELECT APEX_ITEM.POPUPKEY_FROM_QUERY (1,deptno,'SELECT dname, deptno FROM dept')
dt
FROM emp
```



## RADIOGROUP Function

This function generates a radio group from a SQL query.

### Syntax

```
APEX_ITEM.RADIOGROUP (
  p_idx          IN      NUMBER,
  p_value        IN      VARCHAR2 DEFAULT NULL,
  p_selected_value IN    VARCHAR2 DEFAULT NULL,
  p_display      IN      VARCHAR2 DEFAULT NULL,
  p_attributes   IN      VARCHAR2 DEFAULT NULL,
  p_onblur       IN      VARCHAR2 DEFAULT NULL,
  p_onchange     IN      VARCHAR2 DEFAULT NULL,
  p_onfocus     IN      VARCHAR2 DEFAULT NULL,
  p_item_id      IN      VARCHAR2 DEFAULT NULL,
  p_item_label   IN      VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

Table 13–12 describes the parameters available in the RADIOGROUP function.

**Table 13–12 RADIOGROUP Parameters**

Parameter	Description
p_idx	Number that determines which APEX_APPLICATION global variable is used. Valid range of values is 1 to 50. For example 1 creates F01 and 2 creates F02.
p_value	Value of the radio group.
p_selected_value	Value that should be selected.
p_display	Text to display next to the radio option.
p_attributes	Extra HTML parameters you want to add.
p_onblur	JavaScript to execute in the onBlur event.
p_onchange	JavaScript to execute in the onChange event.
p_onfocus	JavaScript to execute in the onFocus event.
p_item_id	HTML attribute ID for the <input> tag
p_item_label	Invisible label created for the item

### Example

The following example demonstrates how to select department 20 from the emp table as a default in a radio group.

```
SELECT APEX_ITEM.RADIOGROUP (1,deptno, '20',dname) dt
FROM   dept
ORDER BY 1
```

## SELECT\_LIST Function

This function dynamically generates a static select list. Similar to other functions available in the `APEX_ITEM` package, these select list functions are designed to generate forms with F01 to F50 form array elements.

### Syntax

```
APEX_ITEM.SELECT_LIST(
  p_idx          IN    NUMBER,
  p_value        IN    VARCHAR2 DEFAULT NULL,
  p_list_values  IN    VARCHAR2 DEFAULT NULL,
  p_attributes   IN    VARCHAR2 DEFAULT NULL,
  p_show_null   IN    VARCHAR2 DEFAULT 'NO',
  p_null_value   IN    VARCHAR2 DEFAULT '%NULL%',
  p_null_text    IN    VARCHAR2 DEFAULT '%',
  p_item_id      IN    VARCHAR2 DEFAULT NULL,
  p_item_label   IN    VARCHAR2 DEFAULT NULL,
  p_show_extra   IN    VARCHAR2 DEFAULT 'YES')
RETURN VARCHAR2;
```

### Parameters

[Table 13–13](#) describes the parameters available in the `SELECT_LIST` function.

**Table 13–13** *SELECT\_LIST Parameters*

Parameter	Description
<code>p_idx</code>	Form element name. For example, 1 equals F01 and 2 equals F02. Typically the <code>P_IDX</code> parameter is constant for a given column.
<code>p_value</code>	Current value. This value should be a value in the <code>P_LIST_VALUES</code> parameter.
<code>p_list_values</code>	List of static values separated by commas. Displays values and returns values that are separated by semicolons. Note that this is only available in the <code>SELECT_LIST</code> function.
<code>p_attributes</code>	Extra HTML parameters you want to add.
<code>p_show_null</code>	Extra select option to enable the NULL selection. Range of values is <code>YES</code> and <code>NO</code> .
<code>p_null_value</code>	Value to be returned when a user selects the NULL option. Only relevant when <code>p_show_null</code> equals <code>YES</code> .
<code>p_null_text</code>	Value to be displayed when a user selects the NULL option. Only relevant when <code>p_show_null</code> equals <code>YES</code> .
<code>p_item_id</code>	HTML attribute ID for the <code>&lt;input&gt;</code> tag.
<code>p_item_label</code>	Invisible label created for the item.
<code>p_show_extra</code>	Shows the current value even if the value of <code>p_value</code> is not located in the select list.

### Example

The following example demonstrates a static select list that displays `Yes`, returns `Y`, defaults to `Y`, and generates a F01 form item.

```
SELECT APEX_ITEM.SELECT_LIST(1, 'Y', 'Yes;Y,No;N') yn
FROM emp
```

The following example demonstrates the use of `APEX_ITEM.SELECT_LIST` to generate a static select list where:

- A form array element F03 is generated (`p_idx` parameter).
- The initial value for each element is equal to the value for `deptno` for the row from `emp` (`p_value` parameter).
- The select list contains 4 options (`p_list_values` parameter).
- The text within the select list displays in red (`p_attributes` parameter).
- A null option is displayed (`p_show_null`) and this option displays `-Select-` as the text (`p_null_text` parameter).
- An HTML ID attribute is generated for each row, where `#ROWNUM#` is substituted for the current row `rownum` (`p_item_id` parameter). (So an ID of `'f03_4'` is generated for row 4.)
- A HTML label element is generated for each row (`p_item_label` parameter).
- The current value for `deptno` is displayed, even if it is not contained with the list of values passed in the `p_list_values` parameter (`p_show_extra` parameter).

```
SELECT empno "Employee #",
       ename "Name",
       APEX_ITEM.SELECT_LIST(
         p_idx      => 3,
         p_value    => deptno,
         p_list_values => 'ACCOUNTING;10,RESEARCH;20,SALES;30,OPERATIONS;40',
         p_attributes => 'style="color:red;"',
         p_show_null => 'YES',
         p_null_value => NULL,
         p_null_text => '-Select-',
         p_item_id   => 'f03_#ROWNUM#',
         p_item_label => 'Label for f03_#ROWNUM#',
         p_show_extra => 'YES') "Department"
FROM emp;
```

## SELECT\_LIST\_FROM\_LOV Function

This function dynamically generates select lists from a shared list of values (LOV). Similar to other functions available in the APEX\_ITEM package, these select list functions are designed to generate forms with F01 to F50 form array elements.

### Syntax

```
APEX_ITEM.SELECT_LIST_FROM_LOV(
  p_idx          IN    NUMBER,
  p_value        IN    VARCHAR2 DEFAULT NULL,
  p_lov          IN    VARCHAR2,
  p_attributes   IN    VARCHAR2 DEFAULT NULL,
  p_show_null    IN    VARCHAR2 DEFAULT 'YES',
  p_null_value   IN    VARCHAR2 DEFAULT '%NULL%',
  p_null_text    IN    VARCHAR2 DEFAULT '%',
  p_item_id      IN    VARCHAR2 DEFAULT NULL,
  p_item_label   IN    VARCHAR2 DEFAULT NULL,
  p_show_extra   IN    VARCHAR2 DEFAULT 'YES')
RETURN VARCHAR2;
```

### Parameters

Table 13–14 describes the parameters available in the SELECT\_LIST\_FROM\_LOV function.

**Table 13–14** SELECT\_LIST\_FROM\_LOV Parameters

Parameter	Description
p_idx	Form element name. For example, 1 equals F01 and 2 equals F02. Typically, the p_idx parameter is constant for a given column.
p_value	Current value. This value should be a value in the p_lov parameter.
p_lov	Text name of an application list of values. This list of values must be defined in your application. This parameter is used only by the select_list_from_lov function.
p_attributes	Extra HTML parameters you want to add.
p_show_null	Extra select option to enable the NULL selection. Range of values is YES and NO.
p_null_value	Value to be returned when a user selects the NULL option. Only relevant when p_show_null equals YES.
p_null_text	Value to be displayed when a user selects the NULL option. Only relevant when p_show_null equals YES.
p_item_id	HTML attribute ID for the <select> tag.
p_item_label	Invisible label created for the item.
p_show_extra	Shows the current value even if the value of p_value is not located in the select list.

### Example

The following example demonstrates a select list based on an LOV defined in the application.

```
SELECT APEX_ITEM.SELECT_LIST_FROM_LOV(2, job, 'JOB_FLOW_LOV') job
FROM emp
```

## SELECT\_LIST\_FROM\_LOV\_XL Function

This function dynamically generates very large select lists (greater than 32K) from a shared list of values (LOV). Similar to other functions available in the `APEX_ITEM` package, these select list functions are designed to generate forms with F01 to F50 form array elements. This function is the same as `SELECT_LIST_FROM_LOV`, but its return value is CLOB. Use this function in SQL queries where you need to handle a column value longer than 4000 characters.

### Syntax

```
APEX_ITEM.SELECT_LIST_FROM_LOV_XL(
  p_idx          IN  NUMBER,
  p_value        IN  VARCHAR2 DEFAULT NULL,
  p_lov          IN  VARCHAR2,
  p_attributes   IN  VARCHAR2 DEFAULT NULL,
  p_show_null    IN  VARCHAR2 DEFAULT 'YES',
  p_null_value   IN  VARCHAR2 DEFAULT '%NULL%',
  p_null_text    IN  VARCHAR2 DEFAULT '%',
  p_item_id      IN  VARCHAR2 DEFAULT NULL,
  p_item_label   IN  VARCHAR2 DEFAULT NULL,
  p_show_extra   IN  VARCHAR2 DEFAULT 'YES')
RETURN CLOB;
```

### Parameters

[Table 13–15](#) describes the parameters available in the `SELECT_LIST_FROM_LOV_XL` function.

**Table 13–15** *SELECT\_LIST\_FROM\_LOV\_XL Parameters*

Parameter	Description
<code>p_idx</code>	Form element name. For example, 1 equals F01 and 2 equals F02. Typically, the <code>p_idx</code> parameter is constant for a given column.
<code>p_value</code>	Current value. This value should be a value in the <code>p_lov</code> parameter.
<code>p_lov</code>	Text name of a list of values. This list of values must be defined in your application. This parameter is used only by the <code>select_list_from_lov</code> function.
<code>p_attributes</code>	Extra HTML parameters you want to add.
<code>p_show_null</code>	Extra select option to enable the NULL selection. Range of values is YES and NO.
<code>p_null_value</code>	Value to be returned when a user selects the NULL option. Only relevant when <code>p_show_null</code> equals YES.
<code>p_null_text</code>	Value to be displayed when a user selects the NULL option. Only relevant when <code>p_show_null</code> equals YES.
<code>p_item_id</code>	HTML attribute ID for the <code>&lt;select&gt;</code> tag.
<code>p_item_label</code>	Invisible label created for the item.
<code>p_show_extra</code>	Shows the current value even if the value of <code>p_value</code> is not located in the select list.

**Example**

The following example demonstrates how to create a select list based on an LOV defined in the application.

```
SELECT APEX_ITEM.SELECT_LIST_FROM_LOV_XL(2,job,'JOB_FLOW_LOV') job  
FROM emp
```

## SELECT\_LIST\_FROM\_QUERY Function

This function dynamically generates a select list from a query. Similar to other functions available in the APEX\_ITEM package, these select list functions are designed to generate forms with F01 to F50 form array elements.

### Syntax

```
APEX_ITEM.SELECT_LIST_FROM_QUERY (
  p_idx          IN    NUMBER,
  p_value        IN    VARCHAR2 DEFAULT NULL,
  p_query        IN    VARCHAR2,
  p_attributes   IN    VARCHAR2 DEFAULT NULL,
  p_show_null    IN    VARCHAR2 DEFAULT 'YES',
  p_null_value   IN    VARCHAR2 DEFAULT '%NULL%',
  p_null_text    IN    VARCHAR2 DEFAULT '%',
  p_item_id      IN    VARCHAR2 DEFAULT NULL,
  p_item_label   IN    VARCHAR2 DEFAULT NULL,
  p_show_extra   IN    VARCHAR2 DEFAULT 'YES')
RETURN VARCHAR2;
```

### Parameters

[Table 13–16](#) describes the parameters available in the SELECT\_LIST\_FROM\_QUERY function.

**Table 13–16** SELECT\_LIST\_FROM\_QUERY Parameters

Parameter	Description
p_idx	Form element name. For example, 1 equals F01 and 2 equals F02. Typically, the p_idx parameter is constant for a given column.
p_value	Current value. This value should be a value in the p_query parameter.
p_query	SQL query that is expected to select two columns, a display column, and a return column. For example:  SELECT dname, deptno FROM dept  Note that this is used only by the SELECT_LIST_FROM_QUERY function. Also note, if only one column is specified in the select clause of this query, the value for this column is used for both display and return purposes.
p_attributes	Extra HTML parameters you want to add.
p_show_null	Extra select option to enable the NULL selection. Range of values is YES and NO.
p_null_value	Value to be returned when a user selects the NULL option. Only relevant when p_show_null equals YES.
p_null_text	Value to be displayed when a user selects the NULL option. Only relevant when p_show_null equals YES.
p_item_id	HTML attribute ID for the <select> tag.
p_item_label	Invisible label created for the item.
p_show_extra	Show the current value even if the value of p_value is not located in the select list.

**Example**

The following example demonstrates a select list based on a SQL query.

```
SELECT APEX_ITEM.SELECT_LIST_FROM_QUERY(3,job,'SELECT DISTINCT job FROM emp')job  
FROM emp
```



## SELECT\_LIST\_FROM\_QUERY\_XL Function

This function is the same as `SELECT_LIST_FROM_QUERY`, but its return value is a CLOB. This allows its use in SQL queries where you need to handle a column value longer than 4000 characters. Similar to other functions available in the `APEX_ITEM` package, these select list functions are designed to generate forms with F01 to F50 form array elements.

### Syntax

```
APEX_ITEM.SELECT_LIST_FROM_QUERY_XL (
    p_idx          IN      NUMBER,
    p_value        IN      VARCHAR2 DEFAULT NULL,
    p_query        IN      VARCHAR2,
    p_attributes   IN      VARCHAR2 DEFAULT NULL,
    p_show_null    IN      VARCHAR2 DEFAULT 'YES',
    p_null_value   IN      VARCHAR2 DEFAULT '%NULL%',
    p_null_text    IN      VARCHAR2 DEFAULT '%',
    p_item_id      IN      VARCHAR2 DEFAULT NULL,
    p_item_label   IN      VARCHAR2 DEFAULT NULL,
    p_show_extra   IN      VARCHAR2 DEFAULT 'YES')
RETURN CLOB;
```

### Parameters

[Table 13–17](#) describes the parameters available in the `SELECT_LIST_FROM_QUERY_XL` function.

**Table 13–17** *SELECT\_LIST\_FROM\_QUERY\_XL Parameters*

Parameter	Description
<code>p_idx</code>	Form element name. For example, 1 equals F01 and 2 equals F02. Typically the <code>p_idx</code> parameter is constant for a given column.
<code>p_value</code>	Current value. This value should be a value in the <code>p_query</code> parameter.
<code>p_query</code>	SQL query that is expected to select two columns, a display column, and a return column. For example:  <pre>SELECT dname, deptno FROM dept</pre> <p>Note that this is used only by the <code>SELECT_LIST_FROM_QUERY_XL</code> function.</p> <p>Also note, if only one column is specified in the select clause of this query, the value for this column is used for both display and return purposes.</p>
<code>p_attributes</code>	Extra HTML parameters you want to add.
<code>p_show_null</code>	Extra select option to enable the NULL selection. Range of values is YES and NO.
<code>p_null_value</code>	Value to be returned when a user selects the NULL option. Only relevant when <code>p_show_null</code> equals YES.
<code>p_null_text</code>	Value to be displayed when a user selects the NULL option. Only relevant when <code>p_show_null</code> equals YES.
<code>p_item_id</code>	HTML attribute ID for the <code>&lt;select&gt;</code> tag.
<code>p_item_label</code>	Invisible label created for the item.

**Table 13–17 (Cont.) SELECT\_LIST\_FROM\_QUERY\_XL Parameters**

<b>Parameter</b>	<b>Description</b>
<code>p_show_extra</code>	Show the current value even if the value of <code>p_value</code> is not located in the select list.

**Example**

The following example demonstrates a select list based on a SQL query.

```
SELECT APEX_ITEM.SELECT_LIST_FROM_QUERY_XL(3,job,'SELECT DISTINCT job FROM  
emp') job  
FROM emp
```

## TEXT Function

This function generates text fields (or text input form items) from a SQL query.

### Syntax

```
APEX_ITEM.TEXT(
  p_idx      IN      NUMBER,
  p_value    IN      VARCHAR2 DEFAULT NULL,
  p_size     IN      NUMBER DEFAULT NULL,
  p_maxlength IN    NUMBER DEFAULT NULL,
  p_attributes IN    VARCHAR2 DEFAULT NULL,
  p_item_id  IN      VARCHAR2 DEFAULT NULL,
  p_item_label IN    VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

[Table 13–18](#) describes the parameters available in the `TEXT` function.

**Table 13–18** TEXT Parameters

Parameter	Description
<code>p_idx</code>	Number to identify the item you want to generate. The number determines which <code>G_FXX</code> global is populated. <b>See Also:</b> " <a href="#">APEX_APPLICATION</a> " on page 1-1
<code>p_value</code>	Value of a text field item.
<code>p_size</code>	Controls HTML tag attributes (such as disabled).
<code>p_maxlength</code>	Maximum number of characters that can be entered in the text box.
<code>p_attributes</code>	Extra HTML parameters you want to add.
<code>p_item_id</code>	HTML attribute ID for the <code>&lt;input&gt;</code> tag.
<code>p_item_label</code>	Invisible label created for the item.

### Example

The following sample query demonstrates how to generate one update field for each row. Note that the `ename`, `sal`, and `comm` columns use the `APEX_ITEM.TEXT` function to generate an HTML text field for each row. Also, notice that each item in the query is passed a unique `p_idx` parameter to ensure that each column is stored in its own array.

```
SELECT
  empno,
  APEX_ITEM.HIDDEN(1, empno) ||
  APEX_ITEM.TEXT(2, ename) ename,
  APEX_ITEM.TEXT(3, job) job,
  mgr,
  APEX_ITEM.DATE_POPUP(4, rownum, hiredate, 'dd-mon-yyyy') hiredate,
  APEX_ITEM.TEXT(5, sal) sal,
  APEX_ITEM.TEXT(6, comm) comm,
  deptno
FROM emp
ORDER BY 1
```

## TEXTAREA Function

This function creates text areas.

### Syntax

```
APEX_ITEM.TEXTAREA (
  p_idx          IN    NUMBER,
  p_value        IN    VARCHAR2 DEFAULT NULL,
  p_rows         IN    NUMBER DEFAULT 40,
  p_cols         IN    NUMBER DEFAULT 4,
  p_attributes   IN    VARCHAR2 DEFAULT NULL,
  p_item_id      IN    VARCHAR2 DEFAULT NULL,
  p_item_label   IN    VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

[Table 13–19](#) describes the parameters available in the TEXTAREA function.

**Table 13–19** TEXTAREA Parameters

Parameter	Description
p_idx	Number to identify the item you want to generate. The number determines which G_FXX global is populated. <b>See Also:</b> " <a href="#">APEX_APPLICATION</a> " on page 1-1
p_value	Value of the text area item.
p_rows	Height of the text area (HTML rows attribute)
p_cols	Width of the text area (HTML column attribute).
p_attributes	Extra HTML parameters you want to add.
p_item_id	HTML attribute ID for the <textarea> tag.
p_item_label	Invisible label created for the item.

### Example

The following example demonstrates how to create a text area based on a SQL query.

```
SELECT APEX_ITEM.TEXTAREA(3,ename,5,80) a
FROM emp
```

---

## TEXT\_FROM\_LOV Function

Use this function to display an item as text, deriving the display value of the named LOV.

### Syntax

```
APEX_ITEM.TEXT_FROM_LOV (
  p_value      IN   VARCHAR2 DEFAULT NULL,
  p_lov        IN   VARCHAR2,
  p_null_text  IN   VARCHAR2 DEFAULT '%' )
RETURN VARCHAR2;
```

### Parameters

[Table 13–20](#) describes the parameters available in the TEXT\_FROM\_LOV function.

**Table 13–20** TEXT\_FROM\_LOV Parameters

Parameter	Description
p_value	Value of a field item.  Note that if p_value is not located in the list of values, p_null_text is value displayed.
p_lov	Text name of a shared list of values. This list of values must be defined in your application.
p_null_text	Value displayed when the value of the field item is NULL.

### Example

The following example demonstrates how to derive the display value from a named LOV (EMPNO\_ENAME\_LOV).

```
SELECT APEX_ITEM.TEXT_FROM_LOV(empno, 'EMPNO_ENAME_LOV') c FROM emp
```

---

## TEXT\_FROM\_LOV\_QUERY Function

Use this function to display an item as text, deriving the display value from a list of values query.

### Syntax

```
APEX_ITEM.TEXT_FROM_LOV_QUERY (
  p_value      IN   VARCHAR2 DEFAULT NULL,
  p_query      IN   VARCHAR2,
  p_null_text  IN   VARCHAR2 DEFAULT '%' )
RETURN VARCHAR2;
```

### Parameters

[Table 13–21](#) describes the parameters available in the TEXT\_FROM\_LOV\_QUERY function.

**Table 13–21** TEXT\_FROM\_LOV\_QUERY Parameters

Parameter	Description
p_value	Value of a field item.
p_query	SQL query that is expected to select two columns, a display column and a return column. For example:  SELECT dname, deptno FROM dept  Note if only one column is specified in the select clause of this query, the value for this column is used for both display and return purposes.
p_null_text	Value to be displayed when the value of the field item is NULL or a corresponding entry is not located for the value p_value in the list of values query.

### Example

The following example demonstrates how to derive the display value from a query.

```
SELECT APEX_ITEM.TEXT_FROM_LOV_QUERY(empno, 'SELECT ename, empno FROM emp') c from
emp
```

---

---

## APEX\_JAVASCRIPT

The `APEX_JAVASCRIPT` package provides utility functions for adding dynamic JavaScript code to HTTP output. This package is usually used for plug-in development.

- [ADD\\_3RD\\_PARTY\\_LIBRARY\\_FILE Procedure](#)
- [ADD\\_ATTRIBUTE Function Signature 1](#)
- [ADD\\_ATTRIBUTE Function Signature 2](#)
- [ADD\\_ATTRIBUTE Function Signature 3](#)
- [ADD\\_ATTRIBUTE Function Signature 4](#)
- [ADD\\_INLINE\\_CODE Procedure](#)
- [ADD\\_LIBRARY Procedure](#)
- [ADD\\_ONLOAD\\_CODE Procedure](#)
- [ADD\\_VALUE Function Signature 1](#)
- [ADD\\_VALUE Function Signature 2](#)
- [ADD\\_VALUE Function Signature 3](#)
- [ADD\\_VALUE Function Signature 4](#)
- [Escape Function](#)

## ADD\_3RD\_PARTY\_LIBRARY\_FILE Procedure

This procedure adds the script tag to load a 3rd party javascript library file and also takes into account the specified Content Delivery Network for the application. Supported libraries include: jQuery, jQueryUI, and jQuery Mobile.

### Syntax

```
add_3rd_party_library_file (
    p_library in varchar2,
    p_file_name in varchar2,
    p_directory in varchar2 default null,
    p_version in varchar2 default null );
```

### Parameters

[Table 14–1](#) describes the parameters available for the ADD\_3RD\_PARTY\_LIBRARY\_FILE procedure.

**Table 14–1 ADD\_3RD\_PARTY\_LIBRARY\_FILE Parameters**

Parameters	Description
p_library	Use one of the c_library_* constants
p_file_name	Specifies the file name without version, .min and .js
p_directory	Directory where the file p_file_name is located (optional)
p_version	If no value is provided then the same version Application Express ships is used (optional)

### Example

This example loads the JavaScript file of the Draggable feature of jQuery UI.

```
apex_javascript.add_3rd_party_library_file (
    p_library =>apex_javascript.c_library_jquery_ui,
    p_file_name => 'jquery.ui.draggable' )
```



## ADD\_ATTRIBUTE Function Signature 1

This function returns the attribute and the attribute's escaped text surrounded by double quotation marks.

---

**Note:** This function does not escape HTML tags. It only prevents HTML tags from breaking the JavaScript object attribute assignment. To prevent XSS (cross site scripting) attacks, you must also call `SYS.HTF.ESCAPE_SC` to prevent embedded JavaScript code from being executed when you inject the string into the HTML page.

---

### Syntax

```
APEX_JAVASCRIPT.ADD_ATTRIBUTE (
    p_name      IN VARCHAR2,
    p_value     IN VARCHAR2,
    p_omit_null IN BOOLEAN:=TRUE,
    p_add_comma IN BOOLEAN:=TRUE)
RETURN VARCHAR2;
```

### Parameters

Table 14–2 describes the parameters available in the ADD\_ATTRIBUTE function signature 1.

**Table 14–2 ADD\_ATTRIBUTE Signature 1 Parameters**

Parameter	Description
p_name	Name of the JavaScript object attribute.
p_value	Text to be assigned to the JavaScript object attribute.
p_omit_null	If set to TRUE and p_value is empty, returns NULL.
p_add_comma	If set to TRUE, a trailing comma is added when a value is returned.

### Example

Adds a call to the `addEmployee` JavaScript function and passes in a JavaScript object with different attribute values. The output of this call looks like:

```
addEmployee(
  {"FirstName": "John",
   "LastName": "Doe",
   "Salary": 2531.29,
   "Birthday": new Date(1970, 1, 15, 0, 0, 0),
   "isSalesman": true
  });
```

As the last attribute you should use the parameter combination `FALSE` (`p_omit_null`), `FALSE` (`p_add_comma`) so that the last attribute is always generated. This avoids that you have to check for the other parameters if a trailing comma should be added or not.

```
apex_javascript.add_onload_code (
  'addEmployee(' ||
    '{ ' ||
    apex_javascript.add_attribute('FirstName', sys.htf.escape_sc(l_first_
name)) ||
```

```
        apex_javascript.add_attribute('LastName',    sys.htf.escape_sc(l_last_
name)) ||
        apex_javascript.add_attribute('Salary',     l_salary) ||
        apex_javascript.add_attribute('Birthday',   l_birthday) ||
        apex_javascript.add_attribute('isSalesman', l_is_salesman, false, false) ||
        ');' );
```

---

## ADD\_ATTRIBUTE Function Signature 2

This function returns the attribute and the attribute's number.

### Syntax

```
APEX_JAVASCRIPT.ADD_ATTRIBUTE (  
    p_name      IN VARCHAR2,  
    p_value     IN NUMBER,  
    p_omit_null IN BOOLEAN:=TRUE,  
    p_add_comma IN BOOLEAN:=TRUE)  
RETURN VARCHAR2;
```

### Parameters

[Table 14–3](#) describes the parameters available in the ADD\_ATTRIBUTE function signature 2.

**Table 14–3** ADD\_ATTRIBUTE Signature 2 Parameters

Parameter	Description
p_name	Name of the JavaScript object attribute.
p_value	Number which should be assigned to the JavaScript object attribute.
p_omit_null	If set to TRUE and p_value is empty, returns NULL.
p_add_comma	If set to TRUE, a trailing comma is added when a value is returned.

### Example

See example for [ADD\\_ATTRIBUTE Function Signature 1](#) on page 14-3.

## ADD\_ATTRIBUTE Function Signature 3

This function returns the attribute and a JavaScript boolean of TRUE, FALSE, or NULL.

### Syntax

```
APEX_JAVASCRIPT.ADD_ATTRIBUTE (  
    p_name      IN VARCHAR2,  
    p_value     IN BOLLEAN,  
    p_omit_null IN BOOLEAN:=TRUE,  
    p_add_comma IN BOOLEAN:=TRUE)  
RETURN VARCHAR2;
```

### Parameters

[Table 14–4](#) describes the parameters available in the ADD\_ATTRIBUTE function signature 3.

**Table 14–4** ADD\_ATTRIBUTE Signature 3 Parameters

Parameter	Description
p_name	Name of the JavaScript object attribute.
p_value	Boolean assigned to the JavaScript object attribute.
p_omit_null	If p_omit_null is TRUE and p_value is NULL the function returns NULL.
p_add_comma	If set to TRUE a trailing comma is added when a value is returned.

### Example

See example for [ADD\\_ATTRIBUTE Function Signature 1](#) on page 14-3

## ADD\_ATTRIBUTE Function Signature 4

This function returns the attribute and the attribute's date. If p\_value is null the value null is returned.

### Syntax

```
APEX_JAVASCRIPT.ADD_ATTRIBUTE (
    p_name      IN VARCHAR2,
    p_value     IN DATE,
    p_omit_null IN BOOLEAN:=TRUE,
    p_add_comma IN BOOLEAN:=TRUE)
RETURN VARCHAR2;
```

### Parameters

[Table 14–5](#) describes the parameters available in the ADD\_ATTRIBUTE function signature 4.

**Table 14–5 ADD\_ATTRIBUTE Signature 4 Parameters**

Parameter	Description
p_name	Name of the JavaScript object attribute.
p_value	Date assigned to the JavaScript object attribute.
p_omit_null	If p_omit_null is TRUE and p_value is NULL the function returns NULL.
p_add_comma	If set to TRUE a trailing comma is added when a value is returned.

### Example

See example for [ADD\\_ATTRIBUTE Function Signature 1](#) on page 14-3

## ADD\_INLINE\_CODE Procedure

This procedure adds a code snippet that is included inline into the HTML output. For example, you can use this procedure to add new functions or global variable declarations. If you want to execute code you should use [ADD\\_ONLOAD\\_CODE Procedure](#).

### Syntax

```
APEX_JAVASCRIPT.ADD_INLINE_CODE (
    p_code      IN VARCHAR2,
    p_key       IN VARCHAR2 DEFAULT NULL);
```

### Parameters

[Table 14–6](#) describes the parameters available in the ADD\_INLINE\_CODE procedure.

**Table 14–6** ADD\_INLINE\_CODE Parameters

Parameter	Description
p_code	JavaScript code snippet. For example: <code>\$s('P1_TEST',123);</code>
p_key	Identifier for the code snippet. If specified and a code snippet with the same name has already been added, the new code snippet is ignored. If p_key is NULL the snippet is always added.

### Example

The following example includes the JavaScript function `initMySuperWidget` in the HTML output. If the plug-in is used multiple times on the page and the `add_inline_code` is called multiple times, it is added once to the HTML output because all calls have the same value for `p_key`.

```
apex_javascript.add_inline_code (
    p_code => 'function initMySuperWidget(){'||chr(10)||
            ' // do something'||chr(10)||
            '};',
    p_key  => 'my_super_widget_function');
```

## ADD\_LIBRARY Procedure

This procedure adds the script tag to load a JavaScript library. If a library has been added, it is not added a second time.

### Syntax

```
APEX_JAVASCRIPT.ADD_LIBRARY (
    p_name           IN VARCHAR2,
    p_directory      IN VARCHAR2,
    p_version        IN VARCHAR2 DEFAULT NULL,
    p_check_to_add_minified IN BOOLEAN DEFAULT FALSE,
    p_skip_extension IN BOOLEAN  DEFAULT FALSE,
    p_ie_condition   IN VARCHAR2 DEFAULT NULL,
    p_key            IN VARCHAR2 DEFAULT NULL);
```

### Parameters

Table 14–7 describes the parameters available in the ADD\_LIBRARY procedure.

**Table 14–7 ADD\_LIBRARY Parameters**

Parameter	Description
p_name	Name of the JavaScript file. Must not use .js when specifying.
p_directory	Directory where JavaScript library is loaded. Must have a trailing slash.
p_version	Version identifier.
p_check_to_add_minified	If TRUE, the procedure tests if it is appropriate to add .min extension and add it if appropriate. This is added if an application is not running in DEBUG mode, and omitted when in DEBUG mode.
p_skip_extension	If TRUE the extension .js is NOT added.
p_ie_condition	Condition which is used as Internet Explorer condition.
p_key	Name used to indicate if the library has already been loaded. If not specified, defaults to p_directory p_name p_version.

### Example

The following example includes the JavaScript library file named my\_library.1.2.min.js (if the application is not running in DEBUG mode), or my\_library.1.2.js (if the application is running in DEBUG mode), from the directory specified by p\_plugin.file\_prefix. The addition of the .min extension if the application is not running in DEBUG mode is carried out because p\_check\_to\_add\_minified is set to TRUE. Since p\_skip\_extension is not specified, this defaults to .js. Also, since p\_key is not specified, the key defaults to p\_plugin.file\_prefix|mylibrary.1.2.

```
apex_javascript.add_library (
    p_name           => 'mylibrary.1.2',
    p_directory      => p_plugin.file_prefix,
    p_check_to_add_minified => true );
```

---

## ADD\_ONLOAD\_CODE Procedure

This procedure adds a javascript code snippet to the HTML output which is executed by the onload event. If an entry with the same key exists it is ignored. If `p_key` is NULL the snippet is always added.

### Syntax

```
APEX_JAVASCRIPT.ADD_ONLOAD_CODE (  
    p_code          IN VARCHAR2,  
    p_key           IN VARCHAR2 DEFAULT NULL);
```

### Parameters

[Table 14–8](#) describes the parameters available in the `ADD_ONLOAD_CODE` procedure.

**Table 14–8** *ADD\_ONLOAD\_CODE Parameters*

Parameter	Description
<code>p_code</code>	Javascript code snippet to be executed during the onload event.
<code>p_key</code>	Any name to identify the specified code snippet. If specified, the code snippet is added if there has been no other call with the same <code>p_key</code> . If <code>p_key</code> is NULL the code snippet is always added.

### Example

Adds the JavaScript call `initMySuperWidget()` to the onload buffer. If the plug-in is used multiple times on the page and the `add_onload_code` is called multiple times, it is added once to the HTML output because all calls have the same value for `p_key`

```
apex_javascript.add_onload_code (  
    p_code => 'initMySuperWidget();'  
    p_key  => 'my_super_widget' );
```



## ADD\_VALUE Function Signature 1

This function returns the escaped text surrounded by double quotation marks. For example, this string could be returned "That\'s a test".

---

**Note:** This function does not escape HTML tags. It only prevents HTML tags from breaking the JavaScript object attribute assignment. To prevent XSS (cross site scripting) attacks, you must also call `SYS.HTF.ESCAPE_SC` to prevent embedded JavaScript code from being executed when you inject the string into the HTML page.

---

### Syntax

```
APEX_JAVASCRIPT.ADD_VALUE (
    p_value          IN VARCHAR2,
    p_add_comma     IN BOOLEAN :=TRUE)
RETURN VARCHAR2;
```

### Parameters

[Table 14–9](#) describes the parameters available in the `ADD_VALUE` signature 1 function.

**Table 14–9 ADD\_VALUE Signature 1 Parameters**

Parameter	Description
<code>p_value</code>	Text to be escaped and wrapped by double quotation marks.
<code>p_add_comma</code>	If <code>p_add_comma</code> is TRUE a trailing comma is added.

### Example

This example adds some JavaScript code to the onload buffer. The value of `p_item.attribute_01` is first escaped with `htf.escape_sc` to prevent XSS attacks and then assigned to the JavaScript variable `lTest` by calling `apex_javascript.add_value`. `Add_value` takes care of properly escaping the value and wrapping it with double quotation marks. Because commas are not wanted, `p_add_comma` is set to `FALSE`.

```
apex_javascript.add_onload_code (
    'var lTest = ' || apex_javascript.add_value(sys.htf.escape_sc(p_item.attribute_
01), FALSE) || ';' || chr(10) ||
    'showMessage(lTest);' );
```

## ADD\_VALUE Function Signature 2

This function returns `p_value` as JavaScript number, if `p_value` is NULL the value null is returned.

### Syntax

```
APEX_JAVASCRIPT.ADD_VALUE (  
    p_value          IN NUMBER,  
    p_add_comma     IN BOOLEAN :=TRUE)  
RETURN VARCHAR2;
```

### Parameters

[Table 14–9](#) describes the parameters available in the `ADD_VALUE` signature 2 function.

**Table 14–10** *ADD\_VALUE Signature 2 Parameters*

Parameter	Description
<code>p_value</code>	Number which should be returned as JavaScript number.
<code>p_add_comma</code>	If <code>p_add_comma</code> is TRUE a trailing comma is added. Default is TRUE.

### Example

See example for [ADD\\_VALUE Function Signature 1](#) on page 14-11.

---

## ADD\_VALUE Function Signature 3

This function returns p\_value as JavaScript boolean. If p\_value is NULL the value null is returned.

### Syntax

```
APEX_JAVASCRIPT.ADD_VALUE (  
    p_value          IN BOOLEAN,  
    p_add_comma     IN BOOLEAN :=TRUE)  
RETURN VARCHAR2;
```

### Parameters

[Table 14–11](#) describes the parameters available in the ADD\_VALUE signature 3 function.

**Table 14–11** ADD\_VALUE Signature 3 Parameters

Parameter	Description
p_value	Boolean which should be returned as JavaScript boolean.
p_add_comma	If p_add_comma is TRUE a trailing comma is added. Default is TRUE.

### Example

See example for [ADD\\_VALUE Function Signature 1](#) on page 14-11.

## ADD\_VALUE Function Signature 4

This function returns p\_value as JavaScript date object, if p\_value is NULL the value null is returned.

### Syntax

```
APEX_JAVASCRIPT.ADD_VALUE (  
    p_value          IN NUMBER,  
    p_add_comma     IN BOOLEAN :=TRUE)  
RETURN VARCHAR2;
```

### Parameters

[Table 14–12](#) describes the parameters available in the ADD\_VALUE signature 4 function.

**Table 14–12** ADD\_VALUE Signature 4 Parameters

Parameter	Description
p_value	Date which should be returned as JavaScript date object.
p_add_comma	If p_add_comma is TRUE a trailing comma is added. Default is TRUE.

### Example

See example for [ADD\\_VALUE Function Signature 1](#) on page 14-11.

## Escape Function

This function escapes text to be used in JavaScript. This function makes the following replacements:

**Table 14–13 Table of Replacement Values**

Replacement	After replacement
<	\u003c
>	\u003e
\	\\
/	\/
"	\u0022
'	\u0027
tab	\t
chr(10)	\n

**Note:** This function prevents HTML tags from breaking the JavaScript object attribute assignment and also escapes the HTML tags '<' and '>'. It does not escape other HTML tags, therefore to be sure to prevent XSS (cross site scripting) attacks, you must also call `SYS.HTF.ESCAPE_SC` to prevent embedded JavaScript code from being executed when you inject the string into the HTML page.

### Syntax

```
APEX_JAVASCRIPT.ESCAPE (
    p_text IN VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

[Table 14–14](#) describes the parameters available in the ESCAPE function.

**Table 14–14 ESCAPE Parameters**

Parameter	Description
p_text	Text to be escaped.

### Example

Adds some JavaScript code to the onload buffer. The value of `p_item.attribute_01` is first escaped with `htf.escape_sc` to prevent XSS attacks and then escaped with `apex_javascript.escape` to prevent that special characters like a quotation mark break the JavaScript code.

```
apex_javascript.add_onload_code (
    'var lTest = ''||apex_javascript.escape(sys.htf.escape_sc(p_item.attribute_
01))||'';||chr(10)||
    'showMessage(lTest);' );
```



This package includes utilities that parse and generate JSON.

- [Package Overview and Examples](#)
- [Constants and Data Types](#)
- [CLOSE\\_ALL Procedure](#)
- [CLOSE\\_ARRAY Procedure](#)
- [CLOSE\\_OBJECT Procedure](#)
- [DOES\\_EXIST Function](#)
- [FIND\\_PATHS\\_LIKE Function](#)
- [FREE\\_OUTPUT Procedure](#)
- [FLUSH Procedure](#)
- [GET\\_BOOLEAN Function](#)
- [GET\\_CLOB\\_OUTPUT Function](#)
- [GET\\_COUNT Function](#)
- [GET\\_DATE Function](#)
- [GET\\_MEMBERS Function](#)
- [GET\\_NUMBER Function](#)
- [GET\\_VALUE Function](#)
- [GET\\_VARCHAR2 Function](#)
- [INITIALIZE\\_CLOB\\_OUTPUT Procedure](#)
- [INITIALIZE\\_OUTPUT Procedure](#)
- [OPEN\\_ARRAY Procedure](#)
- [OPEN\\_OBJECT Procedure](#)
- [PARSE Procedure Signature 1](#)
- [PARSE Procedure Signature 2](#)
- [STRINGIFY Function Signature 1](#)
- [STRINGIFY Function Signature 2](#)
- [STRINGIFY Function Signature 3](#)
- [STRINGIFY Function Signature 4](#)

- 
- [TO\\_XMLTYPE Function](#)
  - [WRITE Procedure Signature 1](#)
  - [WRITE Procedure Signature 2](#)
  - [WRITE Procedure Signature 3](#)
  - [WRITE Procedure Signature 4](#)
  - [WRITE Procedure Signature 5](#)
  - [WRITE Procedure Signature 6](#)
  - [WRITE Procedure Signature 7](#)
  - [WRITE Procedure Signature 8](#)
  - [WRITE Procedure Signature 9](#)
  - [WRITE Procedure Signature 10](#)
  - [WRITE Procedure Signature 11](#)
  - [WRITE Procedure Signature 12](#)
  - [WRITE Procedure Signature 13](#)
  - [WRITE Procedure Signature 14](#)
  - [WRITE Procedure Signature 15](#)
  - [WRITE Procedure Signature 16](#)



## Package Overview and Examples

To read from a string that contains JSON data, first use `parse()` to convert the string to an internal format. Then use the `get_%` routines (e.g. `get_varchar2()`, `get_number()`, ...) to access the data and `find_paths_like()` to search.

Alternatively, use `to_xmltype()` to convert a JSON string to an `xmltype`.

This package also contains procedures to generate JSON-formatted output. Use the overloaded `open_%`, `close_%` and `write()` procedures for writing.

### Example 1

This example parses a JSON string and prints the value of member variable "a".

```
DECLARE
  s varchar2(32767) := '{ "a": 1, "b": ["hello", "world"]}';
BEGIN
  apex_json.parse(s);
  sys.dbms_output.put_line('a is '||apex_json.get_varchar2(p_path => 'a'));
END;
```

### Example 2

This example converts a JSON string to XML and uses `XMLTABLE` to query member values.

```
select col1, col2
from xmltable (
  '/json/row'
  passing apex_json.to_xmltype('{"col1": 1, "col2": "hello"},' ||
    '{"col1": 2, "col2": "world"}')
  columns
    col1 number path '/row/col1',
    col2 varchar2(5) path '/row/col2' );
```

### Example 3

This example writes a nested JSON object to the HTP buffer.

```
BEGIN
  apex_json.open_object;          -- {
  apex_json.write('a', 1);       --  "a":1
  apex_json.open_array('b');    --  ,"b":[
  apex_json.open_object;        --    {
  apex_json.write('c',2);       --    "c":2
  apex_json.close_object;       --    }
  apex_json.write('hello');     --    ,"hello"
  apex_json.write('world');     --    ,"world"
  apex_json.close_all;          --  ]
                                -- }
END;
```

## Constants and Data Types

### Parser Interface

The following are constants used for the parser interface:

```
subtype t_kind is binary_integer range 1 .. 7;
c_null      constant t_kind := 1;
c_true     constant t_kind := 2;
c_false    constant t_kind := 3;
c_number   constant t_kind := 4;
c_varchar2 constant t_kind := 5;
c_object   constant t_kind := 6;
c_array    constant t_kind := 7;
```

### Storage for JSON Data

JSON data is stored in an index by varchar2 table. The JSON values are stored as records. The discriminator "kind" determines whether the value is null, true, false, a number, a varchar2, an object or an array. It depends on "kind" which record fields are used and how. If not explicitly mentioned below, the other record fields' values are undefined:

- \* c\_null: -
- \* c\_true: -
- \* c\_false: -
- \* c\_number: number\_value contains the number value
- \* c\_varchar2: varchar2\_value contains the varchar2 value
- \* c\_object: object\_members contains the names of the object's members
- \* c\_array: number\_value contains the array length

```
type t_value is record (
    kind          t_kind,
    number_value  number,
    varchar2_value varchar2(32767),
    object_members wwv_flow_t_varchar2 );
type t_values is table of t_value index by varchar2(32767);
```

### Default Format for Dates

```
c_date_iso8601 constant varchar2(30) := 'yyyy-mm-dd"T"hh24:mi:ss"Z";
```

### Default JSON Values Table

```
g_values t_values;
```

### Errors Thrown for PARSE()

```
e_parse_error      exception;
pragma exception_init(e_parse_error, -20987);
```

## CLOSE\_ALL Procedure

This procedure closes all objects and arrays up to the outermost nesting level.

### Syntax

```
APEX_JSON.CLOSE_ALL;
```

### Parameters

None.

### Example

See ["Package Overview and Examples"](#) on page 15-3.

## CLOSE\_ARRAY Procedure

This procedure writes a close bracket symbol as follows:

```
]
```

### Syntax

```
APEX_JSON.CLOSE_ARRAY ();
```

### Parameters

None.

### Example

See "[Package Overview and Examples](#)" on page 15-3.

## CLOSE\_OBJECT Procedure

This procedure writes a close curly bracket symbol as follows:

```
}
```

### Syntax

```
APEX_JSON.CLOSE_OBJECT ();
```

### Parameters

None.

### Example

See ["Package Overview and Examples"](#) on page 15-3.

## DOES\_EXIST Function

This function determines whether the given path points to an existing value.

### Syntax

```
APEX_JSON.DOES_EXIST (
  p_path          IN VARCHAR2,
  p0              IN VARCHAR2 DEFAULT NULL,
  p1              IN VARCHAR2 DEFAULT NULL,
  p2              IN VARCHAR2 DEFAULT NULL,
  p3              IN VARCHAR2 DEFAULT NULL,
  p4              IN VARCHAR2 DEFAULT NULL,
  p_values        IN t_values DEFAULT g_values )
RETURN BOOLEAN;
```

### Parameters

**Table 15–1** DOES\_EXIST Function Parameters

Parameter	Description
p_path	Index into p_values.
p[0-4]	Each %N in p_path is replaced by pN and every i-th %s or %d is replaced by the p[i-1].
p_values	Parsed JSON members. The default is g_values.

### Returns

**Table 15–2** DOES\_EXIST Function Returns

Return	Description
TRUE	Given path points to an existing value.
FALSE	Given path does not point to an existing value

### Example

This example parses a JSON string and prints whether it contains values under a path.

```
DECLARE
  j apex_json.t_values;
BEGIN
  apex_json.parse(j, '{ "items": [ 1, 2, { "foo": true } ] }');
  if apex_json.does_exist(p_path => 'items[%d].foo', p0 => 3, p_values =>
j) then
    dbms_output.put_line('found items[3].foo');
  end if;
END;
```

## FIND\_PATHS\_LIKE Function

This function returns paths into `p_values` that match a given pattern.

### Syntax

```
APEX_JSON.FIND_PATHS_LIKE (
    p_return_path    IN VARCHAR2,
    p_subpath        IN VARCHAR2 DEFAULT NULL,
    p_value           IN VARCHAR2 DEFAULT NULL,
    p_values          IN t_values DEFAULT g_values )
RETURN wwv_flow_t_varchar2;
```

### Parameters

**Table 15–3** FIND\_PATHS\_LIKE Function Parameters

Parameter	Description
<code>p_return_path</code>	Search pattern for the return path..
<code>p_subpath</code>	Search pattern under <code>p_return_path</code> (optional).
<code>p_value</code>	Search pattern for value (optional).
<code>p_values</code>	Parsed JSON members. The default is <code>g_values</code> .

### Returns/Raised Errors

**Table 15–4** FIND\_PATHS\_LIKE Function Returns and Raised Errors

Return	Description
<code>apex_t_varchar2</code>	Table of paths that match the pattern.
<code>VALUE_ERROR</code>	Raises this error if <code>p_values(p_path)</code> is not an array or object.

### Example

This example parses a JSON string, finds paths that match a pattern, and prints the values under the paths.

```
DECLARE
    j          apex_json.t_values;
    l_paths apex_t_varchar2;
BEGIN
    apex_json.parse(j, '{ "items": [ { "name": "Amulet of Yendor", "magical": true
}, ||
                                { "name": "Slippers", "magical": "rather
not" } ]}');
    l_paths := apex_json.find_paths_like (
        p_values          => j,
        p_return_path => 'items[%]',
        p_subpath        => '.magical',
        p_value           => 'true' );
    dbms_output.put_line('Magical items:');
    for i in 1 .. l_paths.count loop
        dbms_output.put_line(apex_json.get_varchar2(p_values => j, p_path => l_
paths(i) || '.name'));
    end loop;
END;
```

## FREE\_OUTPUT Procedure

Frees output resources. Call this procedure after process if you are using `INITIALIZE_CLOB_OUTPUT` to write to a temporary CLOB.

### Syntax

```
free_output;
```

### Example

This example configures `APEX_JSON` for CLOB output, generate JSON, print the CLOB with `DBMS_OUTPUT`, and finally free the CLOB.

```
BEGIN
    apex_json.initialize_clob_output;

    apex_json.open_object;
    apex_json.write('hello', 'world');
    apex_json.close_object;

    dbms_output.put_line(apex_json.get_clob_output);

    apex_json.free_output;
END;
```



## FLUSH Procedure

This procedure flushes pending changes. Note that close procedures automatically flush.

### Syntax

```
APEX_JSON.FLUSH
```

### Parameters

None.

### Example

This example writes incomplete JSON.

```
BEGIN
  apex_json.open_object;
  apex_json.write('attr', 'value');
  apex_json.flush;
  sys.htp.p('the "}" is missing');
END;
```

## GET\_BOOLEAN Function

This function returns a boolean number value.

### Syntax

```
APEX_JSON.GET_BOOLEAN (
    p_path          IN VARCHAR2,
    p0              IN VARCHAR2 DEFAULT NULL,
    p1              IN VARCHAR2 DEFAULT NULL,
    p2              IN VARCHAR2 DEFAULT NULL,
    p3              IN VARCHAR2 DEFAULT NULL,
    p4              IN VARCHAR2 DEFAULT NULL,
    p_default       IN BOOLEAN  DEFAULT NULL,
    p_values        IN t_values DEFAULT g_values )
RETURN BOOLEAN;
```

### Parameters

**Table 15–5** GET\_BOOLEAN Function Parameters

Parameter	Description
p_path	Index into p_values.
p[0-4]	Each %N in p_path is replaced by pN and every i-th %s or %d is replaced by the p[i-1].
p_default	The default value if the member does not exist.
p_values	Parsed JSON members. The default is g_values.

### Returns

**Table 15–6** GET\_BOOLEAN Function Returns

Return	Description
TRUE	.Value at the given path position.
FALSE	Value at the given path position.
NULL	Value at the given path position.
VALUE_ERROR	Raises this error if p_values(p_path) is not boolean.

### Example

This example parses a JSON string and prints the boolean value at a position.

```
DECLARE
    j apex_json.t_values;
BEGIN
    apex_json.parse(j, '{ "items": [ 1, 2, { "foo": true } ] }');
    if apex_json.get_boolean(p_path=>'items[%d].foo', p0=>3,p_values=>j) then
        dbms_output.put_line('items[3].foo is true');
    END IF;
END;
```

## GET\_CLOB\_OUTPUT Function

Returns the temporary CLOB that you created with `INITIALIZE_CLOB_OUTPUT`.

### Syntax

```
function get_clob_output  
    return clob;
```

### Example

This example configures `APEX_JSON` for CLOB output, generate JSON, print the CLOB with `DBMS_OUTPUT`, and finally free the CLOB.

```
BEGIN  
    apex_json.initialize_clob_output;  
  
    apex_json.open_object;  
    apex_json.write('hello', 'world');  
    apex_json.close_object;  
  
    dbms_output.put_line(apex_json.get_clob_output);  
  
    apex_json.free_output;  
END;
```

## GET\_COUNT Function

This function returns the number of array elements or object members.

### Syntax

```
APEX_JSON.GET_COUNT (
    p_path          IN VARCHAR2,
    p0              IN VARCHAR2 DEFAULT NULL,
    p1              IN VARCHAR2 DEFAULT NULL,
    p2              IN VARCHAR2 DEFAULT NULL,
    p3              IN VARCHAR2 DEFAULT NULL,
    p4              IN VARCHAR2 DEFAULT NULL,
    p_values        IN t_values DEFAULT g_values )
RETURN NUMBER;
```

### Parameters

**Table 15–7 GET\_COUNT Function Parameters**

Parameter	Description
p_path	Index into p_values.
p[0-4]	Each %N in p_path is replaced by pN and every i-th %s or %d is replaced by the p[i-1].
p_values	Parsed JSON members. The default is g_values.

### Returns/Raised Errors

**Table 15–8 GET\_COUNT Function Returns and Raised Errors**

Return	Description
NUMBER	The number of array elements or object members or null if the array or object could not be found
VALUE_ERROR	Raises this error if p_values (p_path) is not an array or object.

### Example

This example parses a JSON string and prints the number of members at positions.

```
DECLARE
    j apex_json.t_values;
BEGIN
    apex_json.parse(j, '{ "foo": 3, "bar": [1, 2, 3, 4] }');
    dbms_output.put_line(apex_json.get_count(p_path=>'.',p_values=>j)); -- 2 (foo
and bar)
    dbms_output.put_line(apex_json.get_count(p_path=>'bar',p_values=>j)); -- 4
END;
```

## GET\_DATE Function

This function returns a date member value.

### Syntax

```
APEX_JSON.GET_DATE (
    p_path          IN VARCHAR2,
    p0              IN VARCHAR2 DEFAULT NULL,
    p1              IN VARCHAR2 DEFAULT NULL,
    p2              IN VARCHAR2 DEFAULT NULL,
    p3              IN VARCHAR2 DEFAULT NULL,
    p4              IN VARCHAR2 DEFAULT NULL,
    p_default       IN DATE      DEFAULT NULL,
    p_format        IN VARCHAR2 DEFAULT c_date_iso8601,
    p_values        IN t_values  DEFAULT g_values )
RETURN DATE;
```

### Parameters

**Table 15–9** GET\_DATE Function Parameters

Parameter	Description
p_path	Index into p_values.
p[0-4]	Each %N in p_path is replaced by pN and every i-th %s or %d is replaced by the p[i-1].
p_default	The default value if the member does not exist.
p_format	The date format mask.
p_values	Parsed JSON members. The default is g_values.

### Returns/Raised Errors

**Table 15–10** GET\_DATE Function Returns and Raised Errors

Return	Description
DATE	.Returns the date.
VALUE_ERROR	Raises this error if p_values(p_path) is not a date.

### Example

This example parses a JSON string and prints the value at a position.

```
DECLARE
    j apex_json.t_values;
BEGIN
    apex_json.parse(j, '{ "items": [ 1, 2, { "foo": "2014-04-29T10:08:00Z" } ] }');

    dbms_output.put_line(to_char(apex_json.get_date(p_path=>'items[%d].foo', p0=>3, p_
values=>j), 'DD-Mon-YYYY'));
END;
```

## GET\_MEMBERS Function

This function returns the table of OBJECT\_MEMBERS names for an object.

### Syntax

```
APEX_JSON.GET_MEMBERS (
    p_path          IN VARCHAR2,
    p0              IN VARCHAR2 DEFAULT NULL,
    p1              IN VARCHAR2 DEFAULT NULL,
    p2              IN VARCHAR2 DEFAULT NULL,
    p3              IN VARCHAR2 DEFAULT NULL,
    p4              IN VARCHAR2 DEFAULT NULL,
    p_values        IN t_values DEFAULT g_values )
RETURN WWV_FLOW_T_VARCHAR2;
```

### Parameters

**Table 15–11** GET\_MEMBERS Function Parameters

Parameter	Description
p_path	Index into p_values.
p[0-4]	Each %N in p_path is replaced by pN and every i-th %s or %d is replaced by the p[i-1].
p_values	Parsed JSON members. The default is g_values.

### Returns/Raised Errors

**Table 15–12** GET\_MEMBERS Function Returns and Raised Errors

Return	Description
OBJECT_MEMBERS	The OBJECT_MEMBERS of the object or null if the object could not be found.
VALUE_ERROR	Raises this error if p_values (p_path) is not an array or object.

### Example

This example parses a JSON string and prints members at positions.

```
DECLARE
    j apex_json.t_values;
BEGIN
    apex_json.parse(j, '{ "foo": 3, "bar": [1, 2, 3, 4] }');
    dbms_output.put_line(apex_json.get_members(p_path=>'.',p_values=>j) (1)); --
foo
    dbms_output.put_line(apex_json.get_members(p_path=>'.',p_values=>j) (2)); --
bar
END;
```

## GET\_NUMBER Function

This function returns a numeric number value.

### Syntax

```
APEX_JSON.GET_NUMBER (
    p_path          IN VARCHAR2,
    p0              IN VARCHAR2 DEFAULT NULL,
    p1              IN VARCHAR2 DEFAULT NULL,
    p2              IN VARCHAR2 DEFAULT NULL,
    p3              IN VARCHAR2 DEFAULT NULL,
    p4              IN VARCHAR2 DEFAULT NULL,
    p_default       IN BOOLEAN DEFAULT NULL,
    p_values        IN t_values DEFAULT g_values )
RETURN NUMBER;
```

### Parameters

**Table 15–13** GET\_NUMBER Function Parameters

Parameter	Description
p_path	Index into p_values.
p[0-4]	Each %N in p_path is replaced by pN and every i-th %s or %d is replaced by the p[i-1].
p_default	The default value if the member does not exist.
p_values	Parsed JSON members. The default is g_values.

### Returns/Raised Errors

**Table 15–14** GET\_NUMBER Function Returns and Raised Errors

Return	Description
NUMBER	The value at the given path position.
VALUE_ERROR	Raises this error if p_values(p_path) is not a number.

### Example

This example parses a JSON string and prints the value at a position.

```
DECLARE
    j apex_json.t_values;
BEGIN
    apex_json.parse(j, '{ "items": [ 1, 2, { "foo": 42 } ] }');
    dbms_output.put_line(apex_json.get_number(p_path=>'items[%d].foo', p0=> 3, p_
values=>j));
END;
```

## GET\_VALUE Function

This function returns the `t_value`.

### Syntax

```
APEX_JSON.GET_VALUE (
    p_path          IN VARCHAR2,
    p0              IN VARCHAR2 DEFAULT NULL,
    p1              IN VARCHAR2 DEFAULT NULL,
    p2              IN VARCHAR2 DEFAULT NULL,
    p3              IN VARCHAR2 DEFAULT NULL,
    p4              IN VARCHAR2 DEFAULT NULL,
    p_values        IN t_values DEFAULT g_values )
RETURN t_value;
```

### Parameters

**Table 15–15** GET\_VALUE Function Parameters

Parameter	Description
<code>p_path</code>	Index into <code>p_values</code> .
<code>p[0-4]</code>	Each <code>%N</code> in <code>p_path</code> is replaced by <code>pN</code> and every <code>i</code> -th <code>%s</code> or <code>%d</code> is replaced by the <code>p[i-1]</code> .
<code>p_values</code>	Parsed JSON members. The default is <code>g_values</code> .

### Returns/Raised Errors

**Table 15–16** GET\_VALUE Function Returns and Raised Errors

Return	Description
<code>t_value</code>	The <code>t_value</code> at the given path position. The record attributes are null if no data is found.
<code>VALUE_ERROR</code>	Raises this error if <code>p_values(p_path)</code> is not an array or object.

### Example

This example parses a JSON string and prints attributes of values at positions.

```
DECLARE
    j apex_json.t_values;
    v apex_json.t_value;
BEGIN
    apex_json.parse(j, '{ "foo": 3, "bar": [1, 2, 3, 4] }');
    v := apex_json.get_value(p_path=>'bar[%d]',p0=> 2,p_values=>j); -- returns the
t_value for bar[2]
    dbms_output.put_line(v.number_value); -- 2
    v := apex_json.get_value(p_path=>'does.not.exist',p_values=>j);
    dbms_output.put_line(case when v.kind is null then 'not found!' end);
END;
```



## GET\_VARCHAR2 Function

This function returns a varchar2 member value. This function converts boolean and number values to varchar2 values.

### Syntax

```
APEX_JSON.GET_VARCHAR2 (
    p_path          IN VARCHAR2,
    p0              IN VARCHAR2 DEFAULT NULL,
    p1              IN VARCHAR2 DEFAULT NULL,
    p2              IN VARCHAR2 DEFAULT NULL,
    p3              IN VARCHAR2 DEFAULT NULL,
    p4              IN VARCHAR2 DEFAULT NULL,
    p_default       IN BOOLEAN  DEFAULT NULL,
    p_values        IN t_values DEFAULT g_values )
RETURN VARCHAR2;
```

### Parameters

**Table 15–17** GET\_VARCHAR2 Function Parameters

Parameter	Description
p_path	Index into p_values.
p[0-4]	Each %N in p_path is replaced by pN and every i-th %s or %d is replaced by the p[i-1].
p_default	The default value if the member does not exist.
p_values	Parsed JSON members. The default is g_values.

### Returns/Raised Errors

**Table 15–18** GET\_VARCHAR2 Function Returns and Raised Errors

Return	Description
VARCHAR2	This is the value at the given path position.
VALUE_ERROR	Raises this error if p_values(p_path) is not an array or object.

### Example

This example parses a JSON string and prints the value at a position.

```
DECLARE
    j apex_json.t_values;
BEGIN
    apex_json.parse(j, '{ "items": [ 1, 2, { "foo": 42 } ] }');
    dbms_output.put_line(apex_json.get_varchar2(p_path=>'items[%d].foo',p0=> 3,p_
values=>j));
END;
```

---

## INITIALIZE\_CLOB\_OUTPUT Procedure

Initialize the output interface to write to a temporary CLOB. the default is to write to SYS.HTP. If using CLOB output, you should call `FREE_OUTPUT()` at the end to free the CLOB.

### Syntax

```
APEX_JSON.INITIALIZE_CLOB_OUTPUT (
    p_dur          in pls_integer default sys.dbms_lob.call,
    p_cache        in boolean      default true,
    p_indent       in pls_integer default null );
```

### Parameters

**Table 15–19 INITIALIZE\_CLOB\_OUTPUT Procedure Parameters**

Parameter	Description
p_dur	Duration of the temporary CLOB. this can be DBMS_LOB.SESSION or DBMS_LOB.CALL (the default).
p_cache	Specifies if the lob should be read into buffer cache or not.
p_indent	Indent level. Defaults to 2 if debug is turned on, 0 otherwise.

### Example

This example configures APEX\_JSON for CLOB output, generate JSON, print the CLOB with DBMS\_OUTPUT, and finally free the CLOB.

```
BEGIN
    apex_json.initialize_clob_output;

    apex_json.open_object;
    apex_json.write('hello', 'world');
    apex_json.close_object;

    dbms_output.put_line(apex_json.get_clob_output);

    apex_json.free_output;
END;
```

---

## INITIALIZE\_OUTPUT Procedure

This procedure initializes the output interface. You only have to call this procedure if you want to modify the parameters below. Initially, output is already configured with the defaults mentioned in the parameter table.

### Syntax

```
APEX_JSON.INITIALIZE_OUTPUT (
    p_http_header    in boolean    default true,
    p_http_cache     in boolean    default false,
    p_http_cache_etag in varchar2  default null,
    p_indent         in pls_integer default null );
```

### Parameters

**Table 15–20 INITIALIZE\_OUTPUT Procedure Parameters**

Parameter	Description
p_http_header	If TRUE (the default), write an application/JSON mime type header.
p_http_cache	This parameter is only relevant if p_write_header is TRUE. If TRUE, writes Cache-Control: max-age=315360000. If FALSE (the default), writes Cache-Control: no-cache. Otherwise, does not write Cache-Control.
http_cache_etag	If not null, writes an etag header. This parameter is only used if P_HTTP_CACHE is true.
p_indent	Indent level. Defaults to 2, if debug is turned on, otherwise defaults to 0.

### Example

This example configures APEX\_JSON to not emit default headers, because they are written directly.

```
BEGIN
    apex_json.initialize_output (
        p_http_header => false );

    sys.owa_util.mime_header('application/json', false);
    sys.owa_util.status_line(429, 'Too Many Requests');
    sys.owa_util.http_header_close;
    --
    apex_json.open_object;
    apex_json.write('maxRequestsPerSecond', 10);
    apex_json.close_object;
END;
```

## OPEN\_ARRAY Procedure

This procedure writes an open bracket symbol as follows:

[

### Syntax

```
APEX_JSON.OPEN_ARRAY (  
    p_name      IN VARCHAR2 DEFAULT NULL );
```

### Parameters

**Table 15–21** OPEN\_ARRAY Procedure Parameters

Parameter	Description
p_name	If not null, write an object attribute name and colon before the opening bracket.

### Example

This example performs a write { "array":[ 1 ,[ ] ] }.

```
BEGIN  
    apex_json.open_object; -- {  
    apex_json.open_array('array'); -- "array": [  
    apex_json.write(1); -- 1  
    apex_json.open_array; -- , [  
    apex_json.close_array; -- ]  
    apex_json.close_array; -- ]  
    apex_json.close_object; -- }  
END;
```

---

## OPEN\_OBJECT Procedure

This procedure writes an open curly bracket symbol as follows:

```
{
```

### Syntax

```
APEX_JSON.OPEN_OBJECT (  
    p_name      IN VARCHAR2 DEFAULT NULL );
```

### Parameters

**Table 15–22** OPEN\_OBJECT Procedure Parameters

Parameter	Description
p_name	If not null, write an object attribute name and colon before the opening brace.

### Example

This example performs a write { "obj": { "obj-attr": "value" } }.

```
BEGIN  
    apex_json.open_object; -- {  
    apex_json.open_object('obj'); -- "obj": {  
    apex_json.write('obj-attr', 'value'); -- "obj-attr": "value"  
    apex_json.close_all; -- }  
END;
```

## PARSE Procedure Signature 1

This procedure parses a JSON-formatted varchar2 or clob and puts the members into p\_values.

### Syntax

```
APEX_JSON.PARSE (
    p_values    in out nocopy t_values,
    p_source    in varchar2,
    p_strict    in boolean default true );
```

```
APEX_JSON.PARSE (
    p_values    in out nocopy t_values,
    p_source    in clob,
    p_strict    in boolean default true );
```

### Parameters

**Table 15–23** PARSE Procedure Parameters

Parameter	Description
p_values	An index by varchar2 result array which contains the JSON members and values. The default is g_values.
p_source	The JSON source (varchar2 or clob)
p_strict	If TRUE (default), enforce strict JSON rules

### Example

This example parses JSON and prints member values.

```
DECLARE
    l_values apex_json.t_values;
BEGIN
    apex_json.parse (
        p_values => l_values,
        p_source => '{ "type": "circle", "coord": [10, 20] }' );
    sys.http.p('Point at ' ||
        apex_json.get_number (
            p_values => l_values,
            p_path   => 'coord[1]') ||
        ', ' ||
        apex_json.get_number (
            p_values => l_values,
            p_path   => 'coord[2]'));
END;
```

## PARSE Procedure Signature 2

This procedure parses a JSON-formatted `varchar2` or `clob` and puts the members into the package global `g_values`. This simplified API works similar to the `parse()` procedure for signature 1, but saves the developer from declaring a local variable for parsed JSON data and passing it to each JSON API call.

### Syntax

```
APEX_JSON.PARSE (
    p_source   IN VARCHAR2,
    p_strict   IN BOOLEAN DEFAULT TRUE );
```

```
APEX_JSON.PARSE (
    p_source   IN CLOB,
    p_strict   IN BOOLEAN DEFAULT TRUE );
```

### Parameters

**Table 15–24** PARSE Procedure Parameters

Parameter	Description
<code>p_source</code>	The JSON source ( <code>varchar2</code> or <code>clob</code> ).
<code>p_strict</code>	If TRUE (default), enforce strict JSON rules.

### Example

This example parses JSON and prints member values.

```
apex_json.parse('{ "type": "circle", "coord": [10, 20] }');
sys.htp.p('Point at ' ||
    apex_json.get_number(p_path=>'coord[1]') ||
    ', ' ||
    apex_json.get_number(p_path=>'coord[2]'));
```

## STRINGIFY Function Signature 1

This function converts a string to an escaped JSON value.

### Syntax

```
APEX_JSON.STRINGIFY (  
    p_value IN VARCHAR2 )  
RETURN VARCHAR2;
```

### Parameters

**Table 15–25** *STRINGIFY Function Parameters*

Parameter	Description
p_value	The string to be converted.

### Returns

**Table 15–26** *STRINGIFY Function Returns*

Return	Description
VARCHAR2	The converted and escaped JSON value.

### Example

This example is a query that returns a JSON varchar2 value.

```
select apex_json.stringify('line 1'||chr(10)||'line 2') from dual;
```



---

## STRINGIFY Function Signature 2

This function converts a number to an escaped JSON value.

### Syntax

```
APEX_JSON.STRINGIFY (  
    p_value IN NUMBER )  
RETURN VARCHAR2;
```

### Parameters

**Table 15–27** *STRINGIFY Function Parameters*

Parameter	Description
p_value	The number to be converted.

### Returns

**Table 15–28** *STRINGIFY Function Returns*

Return	Description
VARCHAR2	The converted and escaped JSON value.

### Example

This example is a query that returns a JSON number value.

```
select apex_json.stringify(-1/10) from dual
```

## STRINGIFY Function Signature 3

This function converts a date to an escaped JSON value.

### Syntax

```
APEX_JSON.STRINGIFY (  
    p_value IN DATE,  
    p_format IN VARCHAR2 DEFAULT c_date_iso8601 )  
RETURN VARCHAR2;
```

### Parameters

**Table 15–29** *STRINGIFY Function Parameters*

Parameter	Description
p_value	The date value to be converted.

### Returns

**Table 15–30** *STRINGIFY Function Returns*

Return	Description
VARCHAR2	The converted and escaped JSON value.

### Example

This example is a query that returns a JSON `varchar2` value that is suitable to be converted to dates.

```
select apex_json.stringify(sysdate) from dual
```

---

## STRINGIFY Function Signature 4

This function converts a boolean value to an escaped JSON value.

### Syntax

```
APEX_JSON.STRINGIFY (  
    p_value IN BOOLEAN,  
    RETURN VARCHAR2;
```

### Parameters

**Table 15–31** *STRINGIFY Function Parameters*

Parameter	Description
p_value	The boolean value to be converted.

### Returns

**Table 15–32** *STRINGIFY Function Returns*

Return	Description
VARCHAR2	The converted and escaped JSON value.

### Example

This example demonstrates printing JSON boolean values.

```
BEGIN  
    sys.htp.p(apex_json.stringify(true));  
    sys.htp.p(apex_json.stringify(false));  
END;
```

## TO\_XMLTYPE Function

This procedure parses a JSON-formatted varchar2 or CLOB and converts it to an xmltype.

### Syntax

```
APEX_JSON.TO_XMLTYPE (
    p_source    IN VARCHAR2,
    p_strict    IN BOOLEAN DEFAULT TRUE )
RETURN sys.xmltype;
```

```
APEX_JSON.TO_XMLTYPE (
    p_source    IN CLOB,
    p_strict    IN BOOLEAN DEFAULT TRUE )
RETURN sys.xmltype;
```

### Parameters

**Table 15–33 TO\_XMLTYPE Function Parameters**

Parameter	Description
p_source	The JSON source (VARCHAR2 or CLOB)
p_strict	If TRUE (default), enforce strict JSON rules

### Returns

**Table 15–34 TO\_XMLTYPE Function Returns**

Return	Description
sys.xmltype	An xmltype representation of the JSON data.

### Example

This example parses JSON and prints the XML representation.

```
DECLARE
    l_xml xmltype;
BEGIN
    l_xml := apex_json.to_xmltype('{ "items": [ 1, 2, { "foo": true } ] }');
    dbms_output.put_line(l_xml.getstringval);
END;
```

---

## WRITE Procedure Signature 1

This procedure writes an array attribute of type VARCHAR2.

### Syntax

```
APEX_JSON.WRITE (
    p_value    IN VARCHAR2 );
```

### Parameters

**Table 15–35** WRITE Procedure Parameters

Parameter	Description
p_value	The value to be written.

### Example

This example writes an array containing 1, "two", "long text", false, the current date and a JSON representation of an xml document.

```
DECLARE
    l_clob clob := 'long text';
    l_xml sys.xmltype := sys.xmltype('<obj><foo>1</foo><bar>2</bar></obj>');
BEGIN
    apex_json.open_array; -- [
    apex_json.write(1); -- 1
    apex_json.write('two'); -- , "two"
    apex_json.write(l_clob); -- , "long text"
    apex_json.write(false); -- , false
    apex_json.write(sysdate); -- , "2014-05-05T05:36:08Z"
    apex_json.write(l_xml); -- , { "foo": 1, "bar": 2 }
    apex_json.close_array; -- ]
END;
```

## WRITE Procedure Signature 2

---

This procedure writes an array attribute. of type clob.

### Syntax

```
APEX_JSON.WRITE (  
    p_value    IN CLOB );
```

### Parameters

**Table 15–36** *WRITE Procedure Parameters*

Parameter	Description
p_value	The value to be written.

### Example

See "[WRITE Procedure Signature 1](#)" on page 15-31.

---

## WRITE Procedure Signature 3

This procedure writes an array attribute of type NUMBER.

### Syntax

```
APEX_JSON.WRITE (  
    p_value    IN NUMBER );
```

### Parameters

**Table 15–37** *WRITE Procedure Parameters*

Parameter	Description
p_value	The value to be written.

### Example

See "[WRITE Procedure Signature 1](#)" on page 15-31.

## WRITE Procedure Signature 4

This procedure writes an array attribute. of type date

### Syntax

```
APEX_JSON.WRITE (  
    p_value      IN DATE,  
    p_format     IN VARCHAR2 DEFAULT c_date_iso8601 );
```

### Parameters

**Table 15–38** *WRITE Procedure Parameters*

Parameter	Description
p_value	The value to be written.
p_format	The date format mask (default c_date_iso8601).

### Example

See "[WRITE Procedure Signature 1](#)" on page 15-31.



---

## WRITE Procedure Signature 5

This procedure writes an array attribute of type boolean.

### Syntax

```
APEX_JSON.WRITE (  
    p_value    IN BOOLEAN );
```

### Parameters

**Table 15–39** *WRITE Procedure Parameters*

Parameter	Description
p_value	The value to be written.

### Example

See "[WRITE Procedure Signature 1](#)" on page 15-31.

## WRITE Procedure Signature 6

This procedure writes an array attribute of type `sys.xmltype`. The procedure uses a XSL transformation to generate JSON. To determine the JSON type of values, it uses the following rules:

- If the value is empty, it generates a `NULL` value.
- If `upper(value)` is `TRUE`, it generates a boolean true value.
- If `upper(value)` is `FALSE`, it generates a boolean false value.
- If the XPath number function returns `TRUE`, it emits the value as is. Otherwise, it enquotes the value (that is, treats it as a JSON string).

### Syntax

```
APEX_JSON.WRITE (  
    p_value    IN sys.xmltype );
```

### Parameters

**Table 15–40** *WRITE Procedure Parameters*

Parameter	Description
<code>p_value</code>	The value to be written.

### Example

See "[WRITE Procedure Signature 1](#)" on page 15-31.

---

## WRITE Procedure Signature 7

This procedure writes an array with all rows that the cursor returns. Each row is a separate object. If the query contains object type, collection, or cursor columns, the procedure uses `write(xmltype)` to generate JSON. Otherwise, it uses `DBMS_SQL` to fetch rows and the `write()` procedures for the appropriate column data types for output. If the column type is `varchar2` and the uppercase value is 'TRUE' or 'FALSE', it generates boolean values.

### Syntax

```
APEX_JSON.WRITE (
    p_cursor          IN OUT NOCOPY sys_refcursor );
```

### Parameters

**Table 15–41** WRITE Procedure Parameters

Parameter	Description
<code>p_cursor</code>	The cursor.

### Example 1

This example writes an array containing JSON objects for departments 10 and 20.

```
DECLARE
    c sys_refcursor;
BEGIN
    open c for select deptno, dname, loc from dept where deptno in (10, 20);
    apex_json.write(c);
END;
```

This is the output:

```
[ { "DEPTNO":10 , "DNAME":"ACCOUNTING" , "LOC":"NEW YORK" }
, { "DEPTNO":20 , "DNAME":"RESEARCH" , "LOC":"DALLAS" } ]
```

## WRITE Procedure Signature 8

This procedure writes an object attribute of type VARCHAR2.

### Syntax

```
APEX_JSON.WRITE (
    p_name      IN VARCHAR2,
    p_value     IN VARCHAR2,
    p_write_null IN BOOLEAN DEFAULT FALSE );
```

### Parameters

**Table 15–42** WRITE Procedure Parameters

Parameter	Description
p_name	The attribute name.
p_value	The attribute value to be written.
p_write_null	If true, write NULL values. If false (the default), do not write NULLs.

### Example

This example writes an object with named member attributes of various types. The comments to the right of the statements show the output that they generate.

```
DECLARE
    l_clob clob := 'long text';
    l_xml sys.xmltype := sys.xmltype('<obj><foo>1</foo><bar>2</bar></obj>');
BEGIN
    apex_json.open_object; -- {
    apex_json.write('a1', 1); -- "a1": 1
    apex_json.write('a2', 'two'); -- ,"a2": "two"
    apex_json.write('a3', l_clob); -- ,"a3": "long text"
    apex_json.write('a4', false); -- ,"a4": false
    apex_json.write('a5', sysdate); -- ,"a5": "2014-05-05T05:36:08Z"
    apex_json.write('a6', l_xml); -- ,"a6": { "foo": 1, "bar": 2 }
    apex_json.close_object; -- }
END;
```

---

## WRITE Procedure Signature 9

This procedure writes an object attribute of type CLOB.

### Syntax

```
APEX_JSON.WRITE (  
    p_name      IN VARCHAR2,  
    p_value     IN CLOB,  
    p_write_null IN BOOLEAN DEFAULT FALSE );
```

### Parameters

**Table 15–43** *WRITE Procedure Parameters*

Parameter	Description
p_name	The attribute name.
p_value	The attribute value to be written.
p_write_null	If true, write NULL values. If false (the default), do not write NULLs.

### Example

See example for "[WRITE Procedure Signature 8](#)" on page 15-38.

## WRITE Procedure Signature 10

This procedure writes an object attribute of type NUMBER.

### Syntax

```
APEX_JSON.WRITE (  
    p_name      IN VARCHAR2,  
    p_value     IN NUMBER,  
    p_write_null IN BOOLEAN DEFAULT FALSE );
```

### Parameters

**Table 15–44** *WRITE Procedure Parameters*

Parameter	Description
p_name	The attribute name.
p_value	The attribute value to be written.
p_write_null	If true, write NULL values. If false (the default), do not write NULLs.

### Example

See example for "[WRITE Procedure Signature 8](#)" on page 15-38.

---

## WRITE Procedure Signature 11

This procedure writes an object attribute of type date.

### Syntax

```
APEX_JSON.WRITE (
  p_name      IN VARCHAR2,
  p_value     IN DATE,
  p_format    IN VARCHAR2 DEFAULT c_date_iso8691,
  p_write_null IN BOOLEAN  DEFAULT FALSE );
```

### Parameters

**Table 15–45** *WRITE Procedure Parameters*

Parameter	Description
p_name	The attribute name.
p_value	The attribute value to be written.
p_format	The date format mask (default wwv_flow_json.c_date_iso8601).
p_write_null	If true, write NULL values. If false (the default), do not write NULLs.

### Example

See example for "[WRITE Procedure Signature 8](#)" on page 15-38.

## WRITE Procedure Signature 12

This procedure writes an object attribute of type boolean.

### Syntax

```
APEX_JSON.WRITE (  
    p_name      IN VARCHAR2,  
    p_value     IN BOOLEAN,  
    p_write_null IN BOOLEAN DEFAULT FALSE );
```

### Parameters

**Table 15–46** *WRITE Procedure Parameters*

Parameter	Description
p_name	The attribute name.
p_value	The attribute value to be written.
p_write_null	If true, write NULL values. If false (the default), do not write NULLs.

### Example

See example for "[WRITE Procedure Signature 8](#)" on page 15-38.



## WRITE Procedure Signature 13

This procedure writes an attribute where the value is an array that contains all rows that the cursor returns. Each row is a separate object.

If the query contains object type, collection, or cursor columns, the procedure uses `write(p_name, <xmltype>)`. See "WRITE Procedure Signature 14" on page 15-44. Otherwise, it uses `DBMS_SQL` to fetch rows and the `write()` procedures for the appropriate column data types for output. If the column type is `varchar2` and the uppercase value is 'TRUE' or 'FALSE', it generates boolean values.

### Syntax

```
APEX_JSON.WRITE (
    p_name      IN VARCHAR2,
    p_cursor    IN OUT NOCOPY sys_refcursor );
```

### Parameters

**Table 15–47** WRITE Procedure Parameters

Parameter	Description
<code>p_name</code>	The attribute name.
<code>p_cursor</code>	The cursor.

### Example

This example writes an array containing JSON objects for departments 10 and 20, as an object member attribute.

```
DECLARE
    c sys_refcursor;
BEGIN
    open c for select deptno,
                   dname,
                   cursor(select empno,
                               ename
                          from emp e
                          where e.deptno=d.deptno) emps
    from dept d;
    apex_json.open_object;
    apex_json.write('departments', c);
    apex_json.close_object;
END;
```

```
{ "departments": [
  { "DEPTNO": 10,
    "DNAME": "ACCOUNTING",
    "EMPS": [{"EMPNO": 7839, "ENAME": "KING"}] },
  ...
  { "DEPTNO": 40, "DNAME": "OPERATIONS", "EMPS": null } ] }
```

## WRITE Procedure Signature 14

This procedure writes an array attribute of type `sys.xmltype`. The procedure uses a XSL transformation to generate JSON. To determine the JSON type of values, it uses the following rules:

- If the value is empty, it generates a `NULL` value.
- If `upper(value)` is `TRUE`, it generates a boolean true value.
- If `upper(value)` is `FALSE`, it generates a boolean false value.
- If the XPath number function returns true, it emits the value as is. Otherwise, it quotes the value (that is, treats it as a JSON string).

### Syntax

```
APEX_JSON.WRITE (  
    p_name      IN VARCHAR2,  
    p_value     IN sys.xmltype,  
    p_write_null IN BOOLEAN DEFAULT FALSE );
```

### Parameters

**Table 15–48** *WRITE Procedure Parameters*

Parameter	Description
<code>p_name</code>	The attribute name.
<code>p_value</code>	The value to be written. The XML is converted to JSON
<code>p_write_null</code>	If true, write <code>NULL</code> values. If false (the default), do not write <code>NULL</code> s.

### Example

See example for "[WRITE Procedure Signature 13.](#)"

---

## WRITE Procedure Signature 15

This procedure writes parts of a parsed `APEX_JSON.t_values` table.

### Syntax

```
APEX_JSON.WRITE (
  p_values      IN t_values,
  p_path        IN VARCHAR2 DEFAULT '.',
  p0            IN VARCHAR2 DEFAULT NULL,
  p1            IN VARCHAR2 DEFAULT NULL,
  p2            IN VARCHAR2 DEFAULT NULL,
  p3            IN VARCHAR2 DEFAULT NULL,
  p4            IN VARCHAR2 DEFAULT NULL );
```

### Parameters

**Table 15–49** WRITE Procedure Parameters

Parameter	Description
<code>p_values</code>	The parsed JSON members.
<code>p_path</code>	The index into <code>p_values</code> .
<code>p[0-4]</code>	Each <code>%N</code> in <code>p_path</code> will be replaced by <code>pN</code> and every <code>i</code> -th <code>%s</code> or <code>%d</code> is replaced by <code>p[i-1]</code> .

### Example

This example parses a JSON string and writes parts of it.

```
DECLARE
  j apex_json.t_values;
BEGIN
  apex_json.parse(j, '{ "foo": 3, "bar": { "x": 1, "y": 2 }}');
  apex_json.write(j, 'bar');
END;
```

## WRITE Procedure Signature 16

This procedure writes parts of a parsed `APEX_JSON.t_values` table as an object member attribute.

### Syntax

```
APEX_JSON.WRITE (
  p_name          IN VARCHAR2,
  p_values        IN t_values,
  p_path          IN VARCHAR2 DEFAULT '.',
  p0              IN VARCHAR2 DEFAULT NULL,
  p1              IN VARCHAR2 DEFAULT NULL,
  p2              IN VARCHAR2 DEFAULT NULL,
  p3              IN VARCHAR2 DEFAULT NULL,
  p4              IN VARCHAR2 DEFAULT NULL,
  p_write_null    IN BOOLEAN  DEFAULT FALSE );
```

### Parameters

**Table 15–50** *WRITE Procedure Parameters*

Parameter	Description
<code>p_name</code>	The attribute name.
<code>p_values</code>	The parsed JSON members.
<code>p_path</code>	The index into <code>p_values</code> .
<code>p[0-4]</code>	Each <code>%N</code> in <code>p_path</code> will be replaced by <code>pN</code> and every <code>i</code> -th <code>%s</code> or <code>%d</code> is replaced by <code>p[i-1]</code> .
<code>p_write_null</code>	If true, write NULL values. If false (the default), do not write NULLs.

### Example

This example parses a JSON string and writes parts of it as an object member.

```
DECLARE
  j apex_json.t_values;
BEGIN
  apex_json.parse(j, '{ "foo": 3, "bar": { "x": 1, "y": 2 }}');
  apex_json.open_object; -- {
  apex_json.write('parsed-bar',j,'bar');-- "parsed-bar":{ "x":1 ,"y":2 }
  apex_json.close_object; -- }
END;
```

You can use APEX\_LANG API to translate messages.

- [CREATE\\_LANGUAGE\\_MAPPING Procedure](#)
- [DELETE\\_LANGUAGE\\_MAPPING Procedure](#)
- [LANG Function](#)
- [MESSAGE Function](#)
- [PUBLISH\\_APPLICATION Procedure](#)
- [SEED\\_TRANSLATIONS Procedure](#)
- [UPDATE\\_LANGUAGE\\_MAPPING Procedure](#)
- [UPDATE\\_MESSAGE Procedure](#)
- [UPDATE\\_TRANSLATED\\_STRING Procedure](#)

## CREATE\_LANGUAGE\_MAPPING Procedure

Use this procedure to create the language mapping for the translation of an application. Translated applications are published as new applications, but are not directly editable in the Application Builder.

---

**Note:** This procedure is available in Application Express release 4.2.3 and later.

---

### Syntax

```
APEX_LANG.CREATE_LANGUAGE_MAPPING (
  p_application_id IN NUMBER,
  p_language IN VARCHAR2,
  p_translation_application_id IN NUMBER )
```

### Parameters

**Table 16–1 CREATE\_LANGUAGE\_MAPPING Parameters**

Parameter	Description
p_application_id	The ID of the application for which you want to create the language mapping. This is the ID of the primary language application.
p_language	The IANA language code for the mapping. Examples include en-us, fr-ca, ja, he.
p_translation_application_id	Unique integer value for the ID of the underlying translated application. This number cannot end in 0.

### Example

The following example demonstrates the creation of the language mapping for an existing Application Express application.

```
begin
  --
  -- If running from SQL*Plus, we need to set the environment
  -- for the Application Express workspace associated with this schema. The
  -- call to apex_util.set_security_group_id is not necessary if
  -- you're running within the context of the Application Builder
  -- or an Application Express application.
  --
  for c1 in (select workspace_id
             from apex_workspaces) loop
    apex_util.set_security_group_id( c1.workspace_id );
    exit;
  end loop;

  -- Now, actually create the language mapping
  apex_lang.create_language_mapping(
    p_application_id => 63969,
    p_language => 'ja',
    p_translation_application_id => 778899 );
  commit;
  --
  -- Print what we just created to confirm
```

```
--  
for c1 in (select *  
           from apex_application_trans_map  
           where primary_application_id = 63969) loop  
  dbms_output.put_line( 'translated_application_id: ' || c1.translated_  
application_id );  
  dbms_output.put_line( 'translated_app_language: ' || c1.translated_app_  
language );  
end loop;  
end;  
/
```

## DELETE\_LANGUAGE\_MAPPING Procedure

Use this procedure to delete the language mapping for the translation of an application. This procedure deletes all translated strings in the translation repository for the specified language and mapping. Translated applications are published as new applications, but are not directly editable in the Application Builder.

---

**Note:** This procedure is available in Application Express release 4.2.3 and later.

---

### Syntax

```
APEX_LANG.DELETE_LANGUAGE_MAPPING (
  p_application_id IN NUMBER,
  p_language IN VARCHAR2 )
```

### Parameters

**Table 16–2** *DELETE\_LANGUAGE\_MAPPING Parameters*

Parameter	Description
p_application_id	The ID of the application for which you want to delete the language mapping. This is the ID of the primary language application.
p_language	The IANA language code for the existing mapping. Examples include en-us, fr-ca, ja, he.

### Example

The following example demonstrates the deletion of the language mapping for an existing Application Express application and existing translation mapping.

```
begin
  --
  -- If running from SQL*Plus, we need to set the environment
  -- for the Application Express workspace associated with this schema. The
  -- call to apex_util.set_security_group_id is not necessary if
  -- you're running within the context of the Application Builder
  -- or an Application Express application.
  --
  for c1 in (select workspace_id
             from apex_workspaces) loop
    apex_util.set_security_group_id( c1.workspace_id );
    exit;
  end loop;
  -- Now, delete the language mapping
  apex_lang.delete_language_mapping(
    p_application_id => 63969,
    p_language => 'ja' );
  commit;
  --
  -- Print what we just updated to confirm
  --
  for c1 in (select count(*) thecount
             from apex_application_trans_map
             where primary_application_id = 63969) loop
    dbms_output.put_line( 'Translation mappings found: ' || c1.thecount );
```



```
        end loop;  
    end;  
/
```

## LANG Function

Use this function to return a translated text string for translations defined in dynamic translations.

### Syntax

```
APEX_LANG.LANG (
    p_primary_text_string IN VARCHAR2 DEFAULT NULL,
    p0 IN VARCHAR2 DEFAULT NULL,
    p1 IN VARCHAR2 DEFAULT NULL,
    p2 IN VARCHAR2 DEFAULT NULL,
    ...
    p9 IN VARCHAR2 DEFAULT NULL,
    p_primary_language IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 16–3 LANG Parameters**

Parameter	Description
p_primary_text_string	Text string of the primary language. This is the value of the Translate From Text in the dynamic translation.
p0 through p9	Dynamic substitution value: p0 corresponds to %0 in the translation string; p1 corresponds to %1 in the translation string; p2 corresponds to %2 in the translation string, and so on.
p_primary_language	Language code for the message to be retrieved. If not specified, Oracle Application Express uses the current language for the user as defined in the Application Language Derived From attribute.  See also: Specifying the Primary Language for an Application in the <i>Oracle Application Express Application Builder User's Guide</i> .

### Example

Suppose you have a table that defines all primary colors. You could define a dynamic message for each color and then apply the LANG function to the defined values in a query. For example:

```
SELECT APEX_LANG.LANG(color)
FROM my_colors
```

If you were running the application in German, RED was a value for the color column in the `my_colors` table, and you defined the German word for red, the previous example would return ROT.

## MESSAGE Function

Use this function to translate text strings (or messages) generated from PL/SQL stored procedures, functions, triggers, packaged procedures, and functions.

### Syntax

```
APEX_LANG.MESSAGE (
    p_name IN VARCHAR2 DEFAULT NULL,
    p0 IN VARCHAR2 DEFAULT NULL,
    p1 IN VARCHAR2 DEFAULT NULL,
    p2 IN VARCHAR2 DEFAULT NULL,
    ...
    p9 IN VARCHAR2 DEFAULT NULL,
    p_lang IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 16–4 MESSAGE Parameters**

Parameter	Description
p_name	Name of the message as defined in Text Messages under Shared Components of your application in Oracle Application Express.
p0 through p9	Dynamic substitution value: p0 corresponds to %0 in the translation string; p1 corresponds to %1 in the translation string; p2 corresponds to %2 in the translation string, and so on.
p_lang	Language code for the message to be retrieved. If not specified, Oracle Application Express uses the current language for the user as defined in the Application Language Derived From attribute.  See also: Specifying the Primary Language for an Application in the <i>Oracle Application Express Application Builder User's Guide</i> .

### Example

The following example assumes you have defined a message called GREETING\_MSG in your application in English as "Good morning %0" and in German as "Guten Tag %1". The following example demonstrates how you could invoke this message from PL/SQL:

```
BEGIN
--
-- Print the greeting
--
HTP.P(APEX_LANG.MESSAGE('GREETING_MSG', V('APP_USER')));
END;
```

How the p\_lang attribute is defined depends on how the Application Express engine derives the Application Primary Language. For example, if you are running the application in German and the previous call is made to the APEX\_LANG.MESSAGE API, the Application Express engine first looks for a message called GREETING\_MSG with a LANG\_CODE of de. If it does not find anything, then it is reverted to the Application Primary Language attribute. If it still does not find anything, the Application Express engine looks for a message by this name with a language code of en.

**See also:** Specifying the Primary Language for an Application in the *Oracle Application Express Application Builder User's Guide*.

## PUBLISH\_APPLICATION Procedure

Use this procedure to publish the translated version of an application. This procedure creates an underlying, hidden replica of the primary application and merges the strings from the translation repository in this new application. Perform a seed and publish process each time you want to update the translated version of your application and synchronize it with the primary application.

This application is not visible in the Application Builder. It can be published and exported, but not directly edited.

---

**Note:** This procedure is available in Application Express release 4.2.3 and later.

---

### Syntax

```
APEX_LANG.PUBLISH_APPLICATION (
  p_application_id IN NUMBER,
  p_language IN VARCHAR2 )
```

### Parameters

**Table 16–5 PUBLISH\_APPLICATION Parameters**

Parameter	Description
p_application_id	The ID of the application for which you want to publish and create the translated version. This is the ID of the primary language application.
p_language	The IANA language code for the existing translation mapping. Examples include en-us, fr-ca, ja, he.

### Example

The following example demonstrates the publish process for an Application Express application and language.

```
begin
  --
  -- If running from SQL*Plus, we need to set the environment
  -- for the Application Express workspace associated with this schema. The
  -- call to apex_util.set_security_group_id is not necessary if
  -- you're running within the context of the Application Builder
  -- or an Application Express application.
  --
  for c1 in (select workspace_id
             from apex_workspaces) loop
    apex_util.set_security_group_id( c1.workspace_id );
    exit;
  end loop;
  -- Now, publish the translated version of the application
  apex_lang.publish_application(
    p_application_id => 63969,
    p_language => 'ja' );
  commit;
end;
/
```

## SEED\_TRANSLATIONS Procedure

Use this procedure to seed the translation repository for the specified application and language. This procedure populates the translation repository with all of the new, updated and removed translatable strings from your application. Perform a seed and publish process each time you want to update the translated version of your application and synchronize it with the primary application.

### Syntax

```
APEX_LANG.SEED_TRANSLATIONS (
  p_application_id IN NUMBER,
  p_language IN VARCHAR2 )
```

### Parameters

**Table 16–6 SEED\_TRANSLATIONS Parameters**

Parameter	Description
p_application_id	The ID of the application for which you want to update the translation repository. This is the ID of the primary language application.
p_language	The IANA language code for the existing translation mapping. Examples include en-us, fr-ca, ja, he.

### Example

The following example demonstrates the seeding process of the translation repository for an Application Express application and language.

```
begin
  --
  -- If running from SQL*Plus, we need to set the environment
  -- for the Application Express workspace associated with this schema. The
  -- call to apex_util.set_security_group_id is not necessary if
  -- you're running within the context of the Application Builder
  -- or an Application Express application.
  --
  for c1 in (select workspace_id
             from apex_workspaces) loop
    apex_util.set_security_group_id( c1.workspace_id );
    exit;
  end loop;
  -- Now, seed the translation repository
  apex_lang.seed_translations(
    p_application_id => 63969,
    p_language => 'ja' );
  commit;
  -- Print out the total number of potentially translatable strings
  --
  for c1 in (select count(*) thecount
             from apex_application_trans_repos
             where application_id = 63969) loop
    dbms_output.put_line( 'Potentially translatable strings found: ' ||
c1.thecount );
  end loop;
end;
/
```

## UPDATE\_LANGUAGE\_MAPPING Procedure

Use this procedure to update the language mapping for the translation of an application. Translated applications are published as new applications, but are not directly editable in the Application Builder.

---



---

**Note:** This procedure is available in Application Express release 4.2.3 and later.

---



---

### Syntax

```
APEX_LANG.UPDATE_LANGUAGE_MAPPING (
  p_application_id IN NUMBER,
  p_language IN VARCHAR2,
  p_new_trans_application_id IN NUMBER )
```

### Parameters

**Table 16–7 UPDATE\_LANGUAGE\_MAPPING Parameters**

Parameters	Description
p_application_id	The ID of the application for which you want to update the language mapping. This is the ID of the primary language application.
p_language	The IANA language code for the existing mapping. Examples include en-us, fr-ca, ja, he. The language of the mapping cannot be updated with this procedure, only the new translation application ID.
p_new_trans_application_id	New unique integer value for the ID of the underlying translated application. This number cannot end in 0.

### Example

The following example demonstrates the update of the language mapping for an existing Application Express application and existing translation mapping.

```
begin
  --
  -- If running from SQL*Plus, we need to set the environment
  -- for the Application Express workspace associated with this schema. The
  -- call to apex_util.set_security_group_id is not necessary if
  -- you're running within the context of the Application Builder
  -- or an Application Express application.
  --
  for c1 in (select workspace_id
             from apex_workspaces) loop
    apex_util.set_security_group_id( c1.workspace_id );
    exit;
  end loop;
  -- Now, update the language mapping
  apex_lang.update_language_mapping(
    p_application_id => 63969,
    p_language => 'ja',
    p_new_trans_application_id => 881188 );
  commit;
  --
```

```
-- Print what we just updated to confirm
--
for c1 in (select *
           from apex_application_trans_map
           where primary_application_id = 63969) loop
    dbms_output.put_line( 'translated_application_id: ' || c1.translated_
application_id );
    dbms_output.put_line( 'translated_app_language: ' || c1.translated_app_
language );
end loop;
end;
/
```



---

## UPDATE\_MESSAGE Procedure

Use this procedure to update a translatable text message for the specified application.

---

**Note:** This procedure is available in Application Express release 4.2.3 and later.

---

### Syntax

```
APEX_LANG.UPDATE_MESSAGE (
  p_id IN NUMBER,
  p_message_text IN VARCHAR2 )
```

### Parameters

**Table 16–8 UPDATE\_MESSAGE Parameters**

Parameter	Description
p_id	The ID of the text message.
p_message_text	The new text for the translatable text message.

### Example

The following example demonstrates an update of an existing translatable text message.

```
begin
  --
  -- If running from SQL*Plus, we need to set the environment
  -- for the Application Express workspace associated with this schema. The
  -- call to apex_util.set_security_group_id is not necessary if
  -- you're running within the context of the Application Builder
  -- or an Application Express application.
  --
  for c1 in (select workspace_id
             from apex_workspaces) loop
    apex_util.set_security_group_id( c1.workspace_id );
    exit;
  end loop;
  -- Locate the ID of the specific message and update it with the new text
  for c1 in (select translation_entry_id
             from apex_application_translations
             where application_id = 63969
               and translatable_message = 'TOTAL_COST'
               and language_code = 'ja') loop
    apex_lang.update_message(
      p_id => c1.translation_entry_id,
      p_message_text => 'The total cost is: %0');
    commit;
    exit;
  end loop;
end;
/
```

---

## UPDATE\_TRANSLATED\_STRING Procedure

Use this procedure to update a translated string in the seeded translation repository.

---

**Note:** This procedure is available in Application Express release 4.2.3 and later.

---

### Syntax

```
APEX_LANG.UPDATE_TRANSLATED_STRING (
  p_id IN NUMBER,
  p_language IN VARCHAR2
  p_string IN VARCHAR2 )
```

### Parameters

**Table 16–9 UPDATE\_TRANSLATED\_STRING Parameters**

Parameter	Description
p_id	The ID of the string in the translation repository.
p_language	The IANA language code for the existing translation mapping. Examples include en-us, fr-ca, ja, he. The language of the mapping cannot be updated with this procedure, only the new translation application ID.
p_string	The new value for the string in the translation repository.

### Example

The following example demonstrates an update of an existing string in the translation repository.

```
begin
  --
  -- If running from SQL*Plus, we need to set the environment
  -- for the Application Express workspace associated with this schema. The
  -- call to apex_util.set_security_group_id is not necessary if
  -- you're running within the context of the Application Builder
  -- or an Application Express application.
  --
  for c1 in (select workspace_id
             from apex_workspaces) loop
    apex_util.set_security_group_id( c1.workspace_id );
    exit;
  end loop;
  -- Locate all strings in the repository for the specified application
  -- which are 'Search' and change to 'Find'
  for c1 in (select id
             from apex_application_trans_repos
             where application_id = 63969
               and dbms_lob.compare(from_string, to_nclob('Search')) = 0
               and language_code = 'ja') loop
    apex_lang.update_translated_string(
      p_id => c1.id,
      p_language => 'ja',
      p_string => 'Find');
  commit;
```

```
        exit;  
    end loop;  
end;  
/
```



You can use APEX\_LDAP to perform various operations related to Lightweight Directory Access Protocol (LDAP) authentication.

- [AUTHENTICATE Function](#)
- [GET\\_ALL\\_USER\\_ATTRIBUTES Procedure](#)
- [GET\\_USER\\_ATTRIBUTES Procedure](#)
- [IS\\_MEMBER Function](#)
- [MEMBER\\_OF Function](#)
- [MEMBER\\_OF2 Function](#)
- [APEX\\_LDAP.SEARCH Function](#)

## AUTHENTICATE Function

The `AUTHENTICATE` function returns a boolean `TRUE` if the user name and password can be used to perform a `SIMPLE_BIND_S`, call using the provided search base, host, and port.

### Syntax

```
APEX_LDAP.AUTHENTICATE(
    p_username      IN VARCHAR2 DEFAULT NULL,
    p_password      IN VARCHAR2 DEFAULT NULL,
    p_search_base   IN VARCHAR2,
    p_host          IN VARCHAR2,
    p_port          IN VARCHAR2 DEFAULT 389,
    p_use_ssl       IN VARCHAR2 DEFAULT 'N')
RETURN BOOLEAN;
```

### Parameters

[Table 17–1](#) describes the parameters available in the `AUTHENTICATE` function.

**Table 17–1 AUTHENTICATE Parameters**

Parameter	Description
<code>p_username</code>	Login name of the user.
<code>p_password</code>	Password for <code>p_username</code> .
<code>p_search_base</code>	LDAP search base, for example, <code>dc=users,dc=my,dc=org</code> .
<code>p_host</code>	LDAP server host name.
<code>p_port</code>	LDAP server port number.
<code>p_use_ssl</code>	Set to 'Y' to use SSL in bind to LDAP server. Set to 'A' to use SSL with one way authentication (requires LDAP server certificate configured in an Oracle wallet). Set to 'N' to not use SSL (default).

### Example

The following example demonstrates how to use the `APEX_LDAP.AUTHENTICATE` function to verify user credentials against an LDAP Server.

```
IF APEX_LDAP.AUTHENTICATE(
    p_username =>'firstname.lastname',
    p_password =>'abcdef',
    p_search_base => 'cn=user,l=amer,dc=my_company,dc=com',
    p_host => 'our_ldap_sever.my_company.com',
    p_port => 389) THEN
    dbms_output.put_line('authenticated');
ELSE
    dbms_output.put_line('authentication failed');
END IF;
```

## GET\_ALL\_USER\_ATTRIBUTES Procedure

The GET\_ALL\_USER\_ATTRIBUTES procedure returns two OUT arrays of user\_attribute names and values for the user name designated by p\_username (with password if required) using the provided auth base, host, and port.

### Syntax

```
APEX_LDAP.GET_ALL_USER_ATTRIBUTES (
  p_username      IN VARCHAR2 DEFAULT NULL,
  p_pass          IN VARCHAR2 DEFAULT NULL,
  p_auth_base     IN VARCHAR2 DEFAULT NULL,
  p_host          IN VARCHAR2,
  p_port          IN VARCHAR2 DEFAULT 389,
  p_use_ssl       IN VARCHAR2 DEFAULT 'N',
  p_attributes    OUT apex_application_global.vc_arr2,
  p_attribute_values OUT apex_application_global.vc_arr2);
```

### Parameters

Table 17–2 describes the parameters for the GET\_ALL\_USER\_ATTRIBUTES procedure.

**Table 17–2 GET\_ALL\_USER\_ATTRIBUTES Parameters**

Parameter	Description
p_username	Login name of the user.
p_pass	Password for p_username.
p_auth_base	LDAP search base, for example, dc=users,dc=my,dc=org.
p_host	LDAP server host name.
p_port	LDAP server port number.
p_use_ssl	Set to 'Y' to use SSL in bind to LDAP server. Set to 'A' to use SSL with one way authentication (requires LDAP server certificate configured in an Oracle wallet). Set to 'N' to not use SSL (default).
p_attributes	An array of attribute names returned.
p_attribute_values	An array of values returned for each corresponding attribute name returned in p_attributes.

### Example

The following example demonstrates how to use the APEX\_LDAP.GET\_ALL\_USER\_ATTRIBUTES procedure to retrieve all attribute value's associated to a user.

```
DECLARE
  L_ATTRIBUTES      apex_application_global.vc_arr2;
  L_ATTRIBUTE_VALUES apex_application_global.vc_arr2;
BEGIN
  APEX_LDAP.GET_ALL_USER_ATTRIBUTES (
    p_username      => 'firstname.lastname',
    p_pass          => 'abcdef',
    p_auth_base     => 'cn=user,l=amer,dc=my_company,dc=com',
    p_host          => 'our_ldap_sever.my_company.com',
    p_port          => '389',
    p_attributes    => L_ATTRIBUTES,
    p_attribute_values => L_ATTRIBUTE_VALUES);

  FOR i IN L_ATTRIBUTES.FIRST..L_ATTRIBUTES.LAST LOOP
```

```
        http.p('attribute name: ' || L_ATTRIBUTES(i));  
        http.p('attribute value: ' || L_ATTRIBUTE_VALUES(i));  
    END LOOP;  
END;
```



## GET\_USER\_ATTRIBUTES Procedure

The GET\_USER\_ATTRIBUTES procedure returns an OUT array of user\_attribute values for the user name designated by p\_username (with password if required) corresponding to the attribute names passed in p\_attributes using the provided auth base, host, and port.

### Syntax

```
APEX_LDAP.GET_USER_ATTRIBUTES (
    p_username          IN VARCHAR2 DEFAULT NULL,
    p_pass              IN VARCHAR2 DEFAULT NULL,
    p_auth_base         IN VARCHAR2,
    p_host              IN VARCHAR2,
    p_port              IN VARCHAR2 DEFAULT 389,
    p_use_ssl           IN VARCHAR2 DEFAULT 'N',
    p_attributes        IN apex_application_global.vc_arr2,
    p_attribute_values  OUT apex_application_global.vc_arr2);
```

### Parameters

Table 17–3 describes the parameters available in the GET\_USER\_ATTRIBUTES procedure.

**Table 17–3 GET\_USER\_ATTRIBUTES Parameters**

Parameter	Description
p_username	Login name of the user.
p_pass	Password for p_username.
p_auth_base	LDAP search base, for example, dc=users, dc=my, dc=org.
p_host	LDAP server host name.
p_port	LDAP server port number.
p_use_ssl	Set to 'Y' to use SSL in bind to LDAP server. Set to 'A' to use SSL with one way authentication (requires LDAP server certificate configured in an Oracle wallet). Set to 'N' to not use SSL (default).
p_attributes	An array of attribute names for which values are to be returned.
p_attribute_values	An array of values returned for each corresponding attribute name in p_attributes.

### Example

The following example demonstrates how to use the APEX\_LDAP.GET\_USER\_ATTRIBUTES procedure to retrieve a specific attribute value associated to a user.

```
DECLARE
    L_ATTRIBUTES apex_application_global.vc_arr2;
    L_ATTRIBUTE_VALUES apex_application_global.vc_arr2;
BEGIN
    L_ATTRIBUTES(1) := 'xxxxxxxxxx'; /* name of the employee number attribute */
    APEX_LDAP.GET_USER_ATTRIBUTES (
        p_username => 'firstname.lastname',
        p_pass => NULL,
        p_auth_base => 'cn=user,l=amer,dc=my_company,dc=com',
        p_host => 'our_ldap_sever.my_company.com',
        p_port => '389',
        p_attributes => L_ATTRIBUTES,
        p_attribute_values => L_ATTRIBUTE_VALUES);
```

END;

## IS\_MEMBER Function

The IS\_MEMBER function returns a boolean TRUE if the user named by p\_username (with password if required) is a member of the group specified by the p\_group and p\_group\_base parameters using the provided auth base, host, and port.

### Syntax

```
APEX_LDAP.IS_MEMBER(
    p_username      IN VARCHAR2,
    p_pass          IN VARCHAR2 DEFAULT NULL,
    p_auth_base     IN VARCHAR2,
    p_host          IN VARCHAR2,
    p_port          IN VARCHAR2 DEFAULT 389,
    p_use_ssl       IN VARCHAR2 DEFAULT 'N',
    p_group         IN VARCHAR2,
    p_group_base    IN VARCHAR2)
RETURN BOOLEAN;
```

### Parameters

Table 17-4 describes the parameters available in the IS\_MEMBER function.

**Table 17-4 IS\_MEMBER Parameters**

Parameter	Description
p_username	Login name of the user.
p_pass	Password for p_username.
p_auth_base	LDAP search base, for example, dc=users, dc=my, dc=org.
p_host	LDAP server host name.
p_port	LDAP server port number.
p_use_ssl	Set to 'Y' to use SSL in bind to LDAP server. Set to 'A' to use SSL with one way authentication (requires LDAP server certificate configured in an Oracle wallet). Set to 'N' to not use SSL.
p_group	Name of the group to be search for membership.
p_group_base	The base from which the search should be started.

### Example

The following example demonstrates how to use the APEX\_LDAP.IS\_MEMBER function to verify whether a user is a member of a group against an LDAP server.

```
DECLARE
    L_VAL boolean;
BEGIN
    L_VAL := APEX_LDAP.IS_MEMBER(
        p_username => 'firstname.lastname',
        p_pass => 'abcdef',
        p_auth_base => 'cn=user,l=amer,dc=my_company,dc=com',
        p_host => 'our_ldap_sever.my_company.com',
        p_port => 389,
        p_group => 'group_name',
        p_group_base => 'group_base');
    IF L_VAL THEN
        htp.p('Is a member.');
```

```
        http.p('Not a member.');
```

```
    END IF;
```

```
END;
```

## MEMBER\_OF Function

The `MEMBER_OF` function returns an array of groups the user name designated by `p_username` (with password if required) belongs to, using the provided auth base, host, and port.

### Syntax

```
APEX_LDAP.MEMBER_OF(
    p_username      IN VARCHAR2 DEFAULT NULL,
    p_pass          IN VARCHAR2 DEFAULT NULL,
    p_auth_base     IN VARCHAR2,
    p_host          IN VARCHAR2,
    p_port          IN VARCHAR2 DEFAULT 389,
    p_use_ssl       IN VARCHAR2 DEFAULT 'N')
RETURN apex_application_global.vc_arr2;
```

### Parameters

[Table 17-5](#) describes the parameters available in the `MEMBER_OF` function.

**Table 17-5 MEMBER\_OF Parameters**

Parameter	Description
<code>p_username</code>	Login name of the user.
<code>p_pass</code>	Password for <code>p_username</code> .
<code>p_auth_base</code>	LDAP search base, for example, <code>dc=users,dc=my,dc=org</code> .
<code>p_host</code>	LDAP server host name.
<code>p_port</code>	LDAP server port number.
<code>p_use_ssl</code>	Set to 'Y' to use SSL in bind to LDAP server. Set to 'A' to use SSL with one way authentication (requires LDAP server certificate configured in an Oracle wallet). Set to 'N' to not use SSL (default).

### Example

The following example demonstrates how to use the `APEX_LDAP.MEMBER_OF` function to retrieve all the groups designated by the specified username.

```
DECLARE
    L_MEMBERSHIP apex_application_global.vc_arr2;
BEGIN
    L_MEMBERSHIP := APEX_LDAP.MEMBER_OF(
        p_username      => 'firstname.lastname',
        p_pass          => 'abcdef',
        p_auth_base     => 'cn=user,l=amer,dc=my_company,dc=com',
        p_host          => 'our_ldap_sever.my_company.com',
        p_port          => '389');
    FOR i IN L_MEMBERSHIP.FIRST..L_MEMBERSHIP.LAST LOOP
        htp.p('Member of: '||L_MEMBERSHIP(i));
    END LOOP;
END;
```

## MEMBER\_OF2 Function

The `MEMBER_OF2` function returns a `VARCHAR2` colon delimited list of groups the user name designated by `p_username` (with password if required) belongs to, using the provided auth base, host, and port.

### Syntax

```
APEX_LDAP.MEMBER_OF2 (
    p_username    IN VARCHAR2 DEFAULT NULL,
    p_pass        IN VARCHAR2 DEFAULT NULL,
    p_auth_base   IN VARCHAR2,
    p_host        IN VARCHAR2,
    p_port        IN VARCHAR2 DEFAULT 389,
    p_use_ssl     IN VARCHAR2 DEFAULT 'N')
RETURN VARCHAR2;
```

### Parameters

[Table 17–6](#) describes the parameters available in the `MEMBER_OF2` function.

**Table 17–6** *MEMBER\_OF2 Parameters*

Parameter	Description
<code>p_username</code>	Login name of the user.
<code>p_pass</code>	Password for <code>p_username</code> .
<code>p_auth_base</code>	LDAP search base, for example, <code>dc=users,dc=my,dc=org</code> .
<code>p_host</code>	LDAP server host name.
<code>p_port</code>	LDAP server port number.
<code>p_use_ssl</code>	Set to 'Y' to use SSL in bind to LDAP server. Set to 'A' to use SSL with one way authentication (requires LDAP server certificate configured in an Oracle wallet). Set to 'N' to not use SSL (default).

### Example

The following example demonstrates how to use the `APEX_LDAP.MEMBER_OF2` function to retrieve all the groups designated by the specified username.

```
DECLARE
    L_VAL varchar2(4000);
BEGIN
    L_VAL := APEX_LDAP.MEMBER_OF2 (
        p_username => 'firstname.lastname',
        p_pass => 'abcdef',
        p_auth_base => 'cn=user,l=amer,dc=my_company,dc=com',
        p_host => 'our_ldap_sever.my_company.com',
        p_port => 389);
    http.p('Is Member of: ' || L_VAL);
END;
```

## APEX\_LDAP.SEARCH Function

The `APEX_LDAP.SEARCH` function searches the LDAP repository. The result is an object table of (dn, name, val) that can be used in table queries.

### Syntax

```
function search (
    p_username          IN VARCHAR2 DEFAULT NULL,
    p_pass              IN VARCHAR2 DEFAULT NULL,
    p_auth_base         IN VARCHAR2 DEFAULT NULL,
    p_host              IN VARCHAR2,
    p_port              IN NUMBER DEFAULT 389,
    p_use_ssl           IN VARCHAR2 DEFAULT 'N',
    p_search_base       IN VARCHAR2,
    p_search_filter     IN VARCHAR2,
    p_scope             IN BINARY_INTEGER DEFAULT SYS.DBMS_LDAP.SCOPE_SUBTREE,
    p_timeout_sec       IN BINARY_INTEGER DEFAULT 3,
    p_attribute_names   IN VARCHAR2 )
RETURN APEX_T_LDAP_ATTRIBUTES PIPELINED;
```

### Parameters

[Table 17–7](#) describes the parameters available in `SEARCH` function.

**Table 17–7 Search Parameters**

Parameter	Descriptions
<code>p_username</code>	Login name of the user (can be NULL for anonymous binds).
<code>p_pass</code>	The password for <code>p_username</code> (can be NULL for anonymous binds)
<code>p_auth_base</code>	The authentication base dn for <code>p_username</code> (for example, <code>dc=users,dc=my,dc=org</code> ). Can be NULL for anonymous binds.
<code>p_host</code>	The LDAP server host name.
<code>p_port</code>	The LDAP server port number.
<code>p_use_ssl</code>	Set to 'Y' to use SSL in bind to LDAP server. Set to 'A' to use SSL with one way authentication (requires LDAP server certificate configured in an Oracle wallet). Set to 'N' to not use SSL (default).
<code>p_search_base</code>	dn base for the search.
<code>p_search_filter</code>	LDAP search filter expression.
<code>p_scope</code>	Search scope (default descends into subtrees).
<code>p_timeout_sec</code>	Timeout for the search (default is 3 seconds)
<code>p_attribute_names</code>	Comma separated list of return attribute names

### Example 1

```
SELECT val group_dns
FROM table(apex_ldap.search (
    p_host          => 'ldap.example.com',
    p_search_base   => 'dc=example,dc=com',
    p_search_filter => 'uid=' || apex_escape.ldap_search_filter(:APP_USER),
    p_attribute_names => 'memberof' ));
```

**Example 2**

```
SELECT dn, mail, dispname, phone
  from ( select dn, name, val
         from table(apex_ldap.search (
                   p_host      => 'ldap.example.com',
                   p_search_base => 'dc=example,dc=com',
                   p_search_filter => '&(objectClass=person) (ou=Test)',
                   p_attribute_names => 'mail,displayname,telephonenumber'
         )))
  pivot (min(val) for name in ( 'mail'           mail,
                               'displayname'     dispname,
                               'telephonenumber' phone ))
```



You can use the `APEX_MAIL` package to send an email from an Oracle Application Express application. This package is built on top of the Oracle supplied `UTL_SMTP` package. Because of this dependence, the `UTL_SMTP` package must be installed and functioning to use `APEX_MAIL`.

**See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information about the `UTL_SMTP` package

`APEX_MAIL` contains three procedures. Use `APEX_MAIL.SEND` to send an outbound email message from your application. Use `APEX_MAIL.PUSH_QUEUE` to deliver mail messages stored in `APEX_MAIL_QUEUE`. Use `APEX_MAIL.ADD_ATTACHMENT` to send an outbound email message from your application as an attachment.

- [Configuring Oracle Application Express to Send Email](#)
- [ADD\\_ATTACHMENT Procedure](#)
- [GET\\_IMAGES\\_URL Function](#)
- [GET\\_INSTANCE\\_URL Function](#)
- [PUSH\\_QUEUE Procedure](#)
- [SEND Procedure](#)

---

---

**Note:** Oracle Application Express installs the database job `ORACLE_APEX_MAIL_QUEUE`, which periodically sends all mail messages stored in the active mail queue.:

---

---

**See Also:** "Sending Email from an Application" in *Oracle Application Express Application Builder User's Guide*

---

## Configuring Oracle Application Express to Send Email

Before you can send email from an Application Builder application, you must:

1. Log in to Oracle Application Express Administration Services and configure the email settings on the Instance Settings page. See "Configuring Email" in *Oracle Application Express Administration Guide*.
2. If you are running Oracle Application Express with Oracle Database 11g release 1 (11.1), you must enable outbound mail. In Oracle Database 11g release 1 (11.1), the ability to interact with network services is disabled by default. See "Enabling Network Services in Oracle Database 11g" in *Oracle Application Express Application Builder User's Guide*.

**Tip:** You can configure Oracle Application Express to automatically email users their login credentials when a new workspace request has been approved. To learn more, see "Selecting a Provisioning Mode" in *Oracle Application Express Administration Guide*.

## ADD\_ATTACHMENT Procedure

This procedure sends an outbound email message from an application as an attachment. To add multiple attachments to a single email, `APEX_MAIL.ADD_ATTACHMENT` can be called repeatedly for a single email message.

### Syntax

```
APEX_MAIL.ADD_ATTACHMENT (
    p_mail_id           IN     NUMBER,
    p_attachment        IN     BLOB,
    p_filename          IN     VARCHAR2,
    p_mime_type         IN     VARCHAR2);
```

### Parameters

Table 18–1 describes the parameters available in the `ADD_ATTACHMENT` procedure.

**Table 18–1** *ADD\_ATTACHMENT Parameters*

Parameter	Description
<code>p_mail_id</code>	The numeric ID associated with the email. This is the numeric identifier returned from the call to <code>APEX_MAIL.SEND</code> to compose the email body.
<code>p_attachment</code>	A BLOB variable containing the binary content to be attached to the email message.
<code>p_filename</code>	The filename associated with the email attachment.
<code>p_mime_type</code>	A valid MIME type (or Internet media type) to associate with the email attachment.

### Examples

The following example demonstrates how to access files stored in `APEX_APPLICATION_FILES` and add them to an outbound email message

```
DECLARE
    l_id NUMBER;
BEGIN
    l_id := APEX_MAIL.SEND(
        p_to      => 'fred@flintstone.com',
        p_from    => 'barney@rubble.com',
        p_subj    => 'APEX_MAIL with attachment',
        p_body    => 'Please review the attachment.',
        p_body_html => '<b>Please</b> review the attachment');
    FOR c1 IN (SELECT filename, blob_content, mime_type
               FROM APEX_APPLICATION_FILES
               WHERE ID IN (123,456)) LOOP
        APEX_MAIL.ADD_ATTACHMENT(
            p_mail_id => l_id,
            p_attachment => c1.blob_content,
            p_filename => c1.filename,
            p_mime_type => c1.mime_type);
    END LOOP;
    COMMIT;
END;
/
```

---

## GET\_IMAGES\_URL Function

Use this function to get the image prefixed URL, if the email includes Application Express instance images.

### Syntax

```
APEX_MAIL.GET_IMAGES_URL return VARCHAR2;
```

### Parameters

None.

### Example

The following example sends an Order Confirmation email which includes the Oracle Logo image.

```
declare
    l_body      clob;
    l_body_html clob;
begin
    l_body := 'To view the content of this message, please use an HTML enabled
mail client.' || utl_tcp.crlf;

    l_body_html := '<html><body>' || utl_tcp.crlf ||
        '<p>Please confirm your order on the <a href="' ||
        apex_mail.get_instance_url || '?p=100:10">Order
Confirmation</a> page.</p>' || utl_tcp.crlf ||
        '<p>Sincerely,<br />' || utl_tcp.crlf ||
        'The Application Express Dev Team<br />' || utl_tcp.crlf ||
        '</p>' || utl_tcp.crlf ||
        '</body></html>';

    apex_mail.send (
        p_to      => 'some_user@somewhere.com', -- change to your email
address
        p_from     => 'some_sender@somewhere.com', -- change to a real senders
email address
        p_body     => l_body,
        p_body_html => l_body_html,
        p_subj    => 'Order Confirmation' );
end;
```

---

## GET\_INSTANCE\_URL Function

If an email includes a link to an Application Express instance, use this function to get the instance URL.

---

**Note:** This function requires that the instance setting Application Express Instance URL for emails is set.

---

### Syntax

```
APEX_MAIL.GET_INSTANCE_URL return VARCHAR2;
```

### Parameters

None.

### Example

The following example sends an Order Confirmation email which includes an absolute URL to page 10 of application 100.

```
declare
    l_body      clob;
    l_body_html clob;
begin
    l_body := 'To view the content of this message, please use an HTML enabled
mail client.' || utl_tcp.crlf;

    l_body_html := '<html><body>' || utl_tcp.crlf ||
        '<p>Please confirm your order on the <a href="' ||
        apex_mail.get_instance_url || 'f?p=100:10">Order
Confirmation</a> page.</p>' || utl_tcp.crlf ||
        '</body></html>';
    apex_mail.send (
        p_to      => 'some_user@somewhere.com', -- change to your email
address
        p_from    => 'some_sender@somewhere.com', -- change to a real senders
email address
        p_body    => l_body,
        p_body_html => l_body_html,
        p_subj    => 'Order Confirmation' );
end;
```

## PUSH\_QUEUE Procedure

Oracle Application Express stores unsent email messages in a table named `APEX_MAIL_QUEUE`. You can manually deliver mail messages stored in this queue to the specified SMTP gateway by invoking the `APEX_MAIL.PUSH_QUEUE` procedure.

Oracle Application Express logs successfully submitted message in the table `APEX_MAIL_LOG` with the timestamp reflecting your server's local time.

**See Also:** "Sending an Email from an Application" in *Oracle Application Express Application Builder User's Guide*

### Syntax

```
APEX_MAIL.PUSH_QUEUE (
    p_smtp_hostname          IN    VARCHAR2 DEFAULT NULL,
    p_smtp_portno           IN    NUMBER   DEFAULT NULL);
```

### Parameters

Table 18–2 describes the parameters available in the `PUSH_QUEUE` procedure.

**Table 18–2** *PUSH\_QUEUE Parameters*

Parameters	Description
<code>p_smtp_hostname</code>	SMTP gateway host name
<code>p_smtp_portno</code>	SMTP gateway port number

Note that these parameter values are provided for backward compatibility, but their respective values are ignored. The SMTP gateway hostname and SMTP gateway port number are exclusively derived from values entered on the Manage Environment Settings when sending email.

**See Also:** "Configuring Email Settings" in *Oracle Application Express Administration Guide*

### Example

The following example demonstrates the use of the `APEX_MAIL.PUSH_QUEUE` procedure using a shell script. This example only applies to UNIX/LINUX installations.

```
SQLPLUS / <<EOF
APEX_MAIL.PUSH_QUEUE;
DISCONNECT
EXIT
EOF
```

**See Also:** "Sending Email from an Application" in *Oracle Application Express Application Builder User's Guide*

## SEND Procedure

This procedure sends an outbound email message from an application. Although you can use this procedure to pass in either a `VARCHAR2` or a `CLOB` to `p_body` and `p_body_html`, the data types must be the same. In other words, you cannot pass a `CLOB` to `P_BODY` and a `VARCHAR2` to `p_body_html`.

When using `APEX_MAIL.SEND`, remember the following:

- **No single line may exceed 1000 characters.** The SMTP/MIME specification dictates that no single line shall exceed 1000 characters. To comply with this restriction, you must add a carriage return or line feed characters to break up your `p_body` or `p_body_html` parameters into chunks of 1000 characters or less. Failing to do so results in erroneous email messages, including partial messages or messages with extraneous exclamation points.
- **Plain text and HTML email content.** Passing a value to `p_body`, but not `p_body_html` results in a plain text message. Passing a value to `p_body` and `p_body_html` yields a multi-part message that includes both plain text and HTML content. The settings and capabilities of the recipient's email client determine what displays. Although most modern email clients can read an HTML formatted email, remember that some users disable this functionality to address security issues.
- **Avoid images.** When referencing images in `p_body_html` using the `<img />` tag, remember that the images must be accessible to the recipient's email client in order for them to see the image.

For example, suppose you reference an image on your network called `hello.gif` as follows:

```

```

In this example, the image is not attached to the email, but is referenced by the email. For the recipient to see it, they must be able to access the image using a web browser. If the image is inside a firewall and the recipient is outside of the firewall, the image is not displayed. For this reason, avoid using images. If you must include images, be sure to include the ALT attribute to provide a textual description in the event the image is not accessible.

### Syntax

```
APEX_MAIL.SEND(
  p_to           IN    VARCHAR2,
  p_from        IN    VARCHAR2,
  p_body        IN    [ VARCHAR2 | CLOB ],
  p_body_html   IN    [ VARCHAR2 | CLOB ] DEFAULT NULL,
  p_subj       IN    VARCHAR2 DEFAULT NULL,
  p_cc         IN    VARCHAR2 DEFAULT NULL,
  p_bcc        IN    VARCHAR2 DEFAULT NULL,
  p_replyto    IN    VARCHAR2);
```

### Parameters

[Table 18–3](#) describes the parameters available in the `SEND` procedure.

**Table 18–3 SEND Parameters**

Parameter	Description
<code>p_to</code>	Valid email address to which the email is sent (required). For multiple email addresses, use a comma-separated list
<code>p_from</code>	Email address from which the email is sent (required). This email address must be a valid address. Otherwise, the message is not sent
<code>p_body</code>	Body of the email in plain text, not HTML (required). If a value is passed to <code>p_body_html</code> , then this is the only text the recipient sees. If a value is not passed to <code>p_body_html</code> , then this text only displays for email clients that do not support HTML or have HTML disabled. A carriage return or line feed (CRLF) must be included every 1000 characters.
<code>p_body_html</code>	Body of the email in HTML format. This must be a full HTML document including the <code>&lt;html&gt;</code> and <code>&lt;body&gt;</code> tags. A single line cannot exceed 1000 characters without a carriage return or line feed (CRLF)
<code>p_subj</code>	Subject of the email
<code>p_cc</code>	Valid email addresses to which the email is copied. For multiple email addresses, use a comma-separated list
<code>p_bcc</code>	Valid email addresses to which the email is blind copied. For multiple email addresses, use a comma-separated list
<code>p_replyto</code>	Address of the Reply-To mail header. You can use this parameter as follows: <ul style="list-style-type: none"> <li>▪ If you omit the <code>p_replyto</code> parameter, the Reply-To mail header is set to the value specified in the <code>p_from</code> parameter</li> <li>▪ If you include the <code>p_replyto</code> parameter, but provide a NULL value, the Reply-To mail header is set to NULL. This results in the suppression of automatic email replies</li> <li>▪ If you include <code>p_replyto</code> parameter, but provide a non-null value (for example, a valid email address), you send these messages, but the automatic replies go to the value specified (for example, the email address)</li> </ul>

### Examples

The following example demonstrates how to use `APEX_MAIL.SEND` to send a plain text email message from an application.

```
-- Example One: Plain Text only message
DECLARE
    l_body      CLOB;
BEGIN
    l_body := 'Thank you for your interest in the APEX_MAIL
package.'||utl_tcp.crlf||utl_tcp.crlf;
    l_body := l_body ||' Sincerely,'||utl_tcp.crlf;
    l_body := l_body ||' The Application Express Dev Team'||utl_tcp.crlf;
    apex_mail.send(
        p_to      => 'some_user@somewhere.com', -- change to your email address
        p_from    => 'some_sender@somewhere.com', -- change to a real senders
        email address
        p_body    => l_body,
        p_subj    => 'APEX_MAIL Package - Plain Text message');
END;
/
```



The following example demonstrates how to use `APEX_MAIL.SEND` to send an HTML email message from an application. Remember, you must include a carriage return or line feed (CRLF) every 1000 characters. The example that follows uses `utl_tcp.crlf`.

```
-- Example Two: Plain Text / HTML message
DECLARE
    l_body      CLOB;
    l_body_html CLOB;
BEGIN
    l_body := 'To view the content of this message, please use an HTML enabled
mail client.'||utl_tcp.crlf;

    l_body_html := '<html>
<head>
<style type="text/css">
    body{font-family: Arial, Helvetica, sans-serif;
        font-size:10pt;
        margin:30px;
        background-color:#ffffff;}

    span.sig{font-style:italic;
        font-weight:bold;
        color:#811919;}
</style>
</head>
<body>'||utl_tcp.crlf;
    l_body_html := l_body_html || '<p>Thank you for your interest in the
<strong>APEX_MAIL</strong> package.</p>'||utl_tcp.crlf;
    l_body_html := l_body_html || ' Sincerely,<br />'||utl_tcp.crlf;
    l_body_html := l_body_html || ' <span class="sig">The Application Express Dev
Team</span><br />'||utl_tcp.crlf;
    l_body_html := l_body_html || '</body></html>';
    apex_mail.send(
        p_to   => 'some_user@somewhere.com', -- change to your email address
        p_from => 'some_sender@somewhere.com', -- change to a real senders email
address
        p_body      => l_body,
        p_body_html => l_body_html,
        p_subj      => 'APEX_MAIL Package - HTML formatted message');
END;
/
```



The APEX\_PAGE package is the public API for handling pages.

- [Global Constants](#)
- [IS\\_DESKTOP\\_UI Function](#)
- [IS\\_JQM\\_SMARTPHONE\\_UI Function](#)
- [IS\\_JQM\\_TABLET\\_UI Function](#)
- [GET\\_UL\\_TYPE Function](#)
- [IS\\_READ\\_ONLY Function](#)
- [GET\\_PAGE\\_MODE Function](#)
- [PURGE\\_CACHE Procedure](#)
- [GET\\_URL Function](#)

## Global Constants

The following constants are used by this package.

```
c_ui_type_desktop          constant varchar2(10) := 'DESKTOP';  
c_ui_type_jqm_smartphone  constant varchar2(15) := 'JQM_SMARTPHONE';
```

## IS\_DESKTOP\_UI Function

This function returns TRUE if the current page has been designed for desktop browsers.

### **Syntax**

```
FUNCTION IS_DESKTOP_UI  
RETURN BOOLEAN;
```

---

## IS\_JQM\_SMARTPHONE\_UI Function

This function returns TRUE if the current page has been designed for smartphone devices using jQuery Mobile.

### **Syntax**

```
FUNCTION IS_JQM_SMARTPHONE_UI  
RETURN BOOLEAN;
```

## IS\_JQM\_TABLET\_UI Function

This function returns TRUE if the current page has been designed for tablet devices using jQuery Mobile.

### **Syntax**

```
FUNCTION IS_JQM_TABLET_UI  
RETURN BOOLEAN;
```

---

## GET\_UI\_TYPE Function

This function returns the user interface (UI) type for which the current page has been designed.

### **Syntax**

```
FUNCTION GET_UI_TYPE  
RETURN VARCHAR2;
```



## IS\_READ\_ONLY Function

This function returns TRUE if the current page is rendered read-only and FALSE if it is not.

### Syntax

```
FUNCTION IS_READ_ONLY  
RETURN BOOLEAN;
```

## GET\_PAGE\_MODE Function

This function returns the page mode for the current page.

### Syntax

```
FUNCTION GET_PAGE_MODE (  
    p_application_id      IN NUMBER,  
    p_page_id  
    RETURN VARCHAR2;
```

### Parameters

**Table 19–1** *GET\_PAGE\_MODE Parameters*

Parameter	Description
p_application_id	ID of the application. Defaults to the current application.
p_page_id	ID of the page. Defaults to the current page.

---

## PURGE\_CACHE Procedure

This procedure purges the cache of the specified application, page, and region for the specified user. If the user is not specified, the procedure purges all cached versions of the page.

### Syntax

```
PROCEDURE PURGE_CACHE (
    p_application_id      IN NUMBER DEFAULT wwv_flow.g_flow_id,
    p_page_id             IN NUMBER DEFAULT wwv_flow.g_flow_step_id,
    p_user_name           IN VARCHAR2 DEFAULT NULL,
    p_current_session_only IN BOOLEAN  DEFAULT FALSE );
```

### Parameters

**Table 19–2** *PURGE\_CACHE Parameters*

Parameter	Description
p_application_id	ID of the application. Defaults to the current application.
p_page_id	ID of the page. Defaults to the current page. If you pass NULL, Oracle Application Express purges the cache on all pages of the application.
p_user_name	Specify a user name if you only want to purge entries that were saved for the given user.
p_current_session_only	Specify TRUE if you only want to purge entries that were saved for the current session. Defaults to FALSE.

### Example

This example purges session specific cache on the current page.

```
BEGIN
    APEX_PAGE.PURGE_CACHE (
        p_current_session_only => true );
END;
```

## GET\_URL Function

This function returns an Oracle Application Express `f?p=` URL. It is sometimes clearer to read a function call than a concatenated URL. See the example below for a comparison.

### Syntax

```
FUNCTION GET_URL (
    p_application      IN VARCHAR2 DEFAULT NULL,
    p_page            IN VARCHAR2 DEFAULT NULL,
    p_session         IN NUMBER   DEFAULT WWW_FLOW.G_INSTANCE,
    p_request         IN VARCHAR2 DEFAULT NULL,
    p_debug           IN VARCHAR2 DEFAULT NULL,
    p_clear_cache     IN VARCHAR2 DEFAULT NULL,
    p_items           IN VARCHAR2 DEFAULT NULL,
    p_values          IN VARCHAR2 DEFAULT NULL,
    p_printer_friendly IN VARCHAR2 DEFAULT NULL,
    p_trace           IN VARCHAR2 DEFAULT NULL )
RETURN VARCHAR2;
```

### Parameters

**Table 19–3** GET\_URL Parameters

Parameter	Description
<code>p_application</code>	The application ID or alias. Defaults to the current application.
<code>p_page</code>	Page ID or alias. Defaults to the current page.
<code>p_session</code>	Session ID. Defaults to the current session ID.
<code>p_request</code>	URL request parameter.
<code>p_debug</code>	URL debug parameter. Defaults to the current debug mode.
<code>p_clear_cache</code>	URL clear cache parameter.
<code>p_items</code>	Comma-delimited list of item names to set session state.
<code>p_values</code>	Comma-delimited list of item values to set session state.
<code>p_printer_friendly</code>	URL printer friendly parameter. Defaults to the current request's printer friendly mode.
<code>p_trace</code>	SQL trace parameter.

### Example

This query uses `APEX_PAGE.GET_URL` and its alternative `APEX_UTIL.PREPARE_URL` to produce two identical URLs.

```
SELECT APEX_PAGE.GET_URL (
    p_page => 1,
    p_items => 'P1_X,P1_Y',
    p_values => 'somevalue,othervalue' ) f_url_1,
    APEX_UTIL.PREPARE_URL('f?p=&APP_ID.:1:&APP_SESSION.::::P1_X,P1_
Y:somevalue,othervalue')
FROM DUAL
```

The `APEX_PLUGIN` package provides the interface declarations and some utility functions to work with plug-ins.

- [Data Types](#)
- [GET\\_AJAX\\_IDENTIFIER Function](#)
- [GET\\_INPUT\\_NAME\\_FOR\\_PAGE\\_ITEM Function](#)

## Data Types

The data types used by the APEX\_PLUGIN package are described in this section.

### Data Types:

- `c_*`
- `t_authentication`
- `t_authentication_ajax_result`
- `t_authentication_auth_result`
- `t_authentication_inval_result`
- `t_authentication_logout_result`
- `t_authentication_sentry_result`
- `t_authorization`
- `t_authorization_exec_result`
- `t_dynamic_action`
- `t_dynamic_action_ajax_result`
- `t_dynamic_action_render_result`
- `t_page_item`
- `t_page_item_ajax_result`
- `t_page_item_render_result`
- `t_page_item_validation_result`
- `t_plugin`
- `t_process`
- `t_process_exec_result`
- `type t_region_column (`
- `type t_region_columns is table of t_region_column index by pls_integer;`
- `type t_region_column (`
- `type t_region_columns is table of t_region_column index by pls_integer;`
- `t_region`
- `t_region_ajax_result`
- `t_region_render_result`

### `c_*`

The following constants are used for `display_location` in the page item validation function result type `t_page_item_validation_result`.

```
c_inline_with_field          constant varchar2(40) := 'INLINE_WITH_FIELD';
c_inline_with_field_and_notif constant varchar2(40) := 'INLINE_WITH_FIELD_AND_
NOTIFICATION';
c_inline_in_notification     constant varchar2(40) := 'INLINE_IN_NOTIFICATION';
c_on_error_page              constant varchar2(40) := 'ON_ERROR_PAGE';
```

**t\_authentication**

```

type t_authentication is record (
  id                number,
  name              varchar2(255),
  invalid_session_url varchar2(4000),
  logout_url        varchar2(4000),
  plssql_code       clob,
  attribute_01      varchar2(32767),
  attribute_02      varchar2(32767),
  attribute_03      varchar2(32767),
  attribute_04      varchar2(32767),
  attribute_05      varchar2(32767),
  attribute_06      varchar2(32767),
  attribute_07      varchar2(32767),
  attribute_08      varchar2(32767),
  attribute_09      varchar2(32767),
  attribute_10      varchar2(32767),
  attribute_11      varchar2(32767),
  attribute_12      varchar2(32767),
  attribute_13      varchar2(32767),
  attribute_14      varchar2(32767),
  attribute_15      varchar2(32767),
  --
  session_id        number,
  username           varchar2(255) );

```

**t\_authentication\_ajax\_result**

```

type t_authentication_ajax_result is record (
  dummy            boolean );

```

**t\_authentication\_auth\_result**

```

type t_authentication_auth_result is record (
  is_authenticated boolean,
  redirect_url      varchar2(4000),
  log_code          number,
  log_text          varchar2(4000),
  display_text      varchar2(4000) );

```

**t\_authentication\_inval\_result**

```

type t_authentication_inval_result is record (
  redirect_url      varchar2(4000) );

```

**t\_authentication\_logout\_result**

```

type t_authentication_logout_result is record (
  redirect_url      varchar2(4000) );

```

**t\_authentication\_sentry\_result**

```

type t_authentication_sentry_result is record (
  is_valid          boolean );

```

**t\_authorization**

The following type is passed to all authorization plug-in functions and contains information about the current authorization.

```

type t_authorization is record (
  id                number,
  name              varchar2(255),

```

```
username          varchar2(255),
caching           varchar2(20),
component         wwv_flow.t_component,
attribute_01      varchar2(32767),
attribute_02      varchar2(32767),
attribute_03      varchar2(32767),
attribute_04      varchar2(32767),
attribute_05      varchar2(32767),
attribute_06      varchar2(32767),
attribute_07      varchar2(32767),
attribute_08      varchar2(32767),
attribute_09      varchar2(32767),
attribute_10      varchar2(32767),
attribute_11      varchar2(32767),
attribute_12      varchar2(32767),
attribute_13      varchar2(32767),
attribute_14      varchar2(32767),
attribute_15      varchar2(32767),
```

### **t\_authorization\_exec\_result**

The `t_authorization_exec_result` data type has been added to the `APEX_PLUGIN` package.

```
type t_authorization_exec_result is record (
  is_authorized      boolean
);
```

### **t\_dynamic\_action**

The following type is passed into all dynamic action plug-in functions and contains information about the current dynamic action.

```
type t_dynamic_action is record (
  id          number,
  action      varchar2(50),
  attribute_01 varchar2(32767),
  attribute_02 varchar2(32767),
  attribute_03 varchar2(32767),
  attribute_04 varchar2(32767),
  attribute_05 varchar2(32767),
  attribute_06 varchar2(32767),
  attribute_07 varchar2(32767),
  attribute_08 varchar2(32767),
  attribute_09 varchar2(32767),
  attribute_10 varchar2(32767),
  attribute_11 varchar2(32767),
  attribute_12 varchar2(32767),
  attribute_13 varchar2(32767),
  attribute_14 varchar2(32767),
  attribute_15 varchar2(32767) );
```

### **t\_dynamic\_action\_ajax\_result**

The following type is used as the result type for the Ajax function of a dynamic action type plug-in.

```
type t_dynamic_action_ajax_result is record (
  dummy boolean /* not used yet */
);
```



**t\_dynamic\_action\_render\_result**

The following type is used as the result type for the rendering function of a dynamic action plug-in.

```
type t_dynamic_action_render_result is record (
  javascript_function varchar2(32767),
  ajax_identifier     varchar2(255),
  attribute_01       varchar2(32767),
  attribute_02       varchar2(32767),
  attribute_03       varchar2(32767),
  attribute_04       varchar2(32767),
  attribute_05       varchar2(32767),
  attribute_06       varchar2(32767),
  attribute_07       varchar2(32767),
  attribute_08       varchar2(32767),
  attribute_09       varchar2(32767),
  attribute_10       varchar2(32767),
  attribute_11       varchar2(32767),
  attribute_12       varchar2(32767),
  attribute_13       varchar2(32767),
  attribute_14       varchar2(32767),
  attribute_15       varchar2(32767) );
```

**t\_page\_item**

The following type is passed into all item type plug-in functions and contains information about the current page item.

```
type t_page_item is record (
  id                number,
  name              varchar2(255),
  label            varchar2(4000),
  plain_label      varchar2(4000),
  label_id         varchar2(255), /* label id is set if 'Standard
Form Element' = no and label template uses #LABEL_ID# substitution */
  placeholder      varchar2(255),
  format_mask      varchar2(255),
  is_required      boolean,
  lov_definition   varchar2(4000),
  lov_display_extra boolean,
  lov_display_null boolean,
  lov_null_text    varchar2(255),
  lov_null_value   varchar2(255),
  lov_cascade_parent_items varchar2(255),
  ajax_items_to_submit varchar2(255),
  ajax_optimize_refresh boolean,
  element_width    number,
  element_max_length number,
  element_height   number,
  element_css_classes varchar2(255),
  element_attributes varchar2(2000),
  element_option_attributes varchar2(4000),
  escape_output    boolean,
  attribute_01     varchar2(32767),
  attribute_02     varchar2(32767),
  attribute_03     varchar2(32767),
  attribute_04     varchar2(32767),
  attribute_05     varchar2(32767),
  attribute_06     varchar2(32767),
  attribute_07     varchar2(32767),
  attribute_08     varchar2(32767),
```

```
attribute_09          varchar2(32767),
attribute_10          varchar2(32767),
attribute_11          varchar2(32767),
attribute_12          varchar2(32767),
attribute_13          varchar2(32767),
attribute_14          varchar2(32767),
attribute_15          varchar2(32767) );
```

### **t\_page\_item\_ajax\_result**

The following type is used as the result type for the Ajax function of an item type plug-in.

```
type t_page_item_ajax_result is record (
    dummy boolean /* not used yet */
);
```

### **t\_page\_item\_render\_result**

The following type is used as the result type for the rendering function of an item type plug-in.

```
type t_page_item_render_result is record (
    is_navigable      boolean default false,
    navigable_dom_id  varchar2(255)          /* should only be set if navigable
element is not equal to item name */
);
```

### **t\_page\_item\_validation\_result**

The following type is used as the result type for the validation function of an item type plug-in.

```
type t_page_item_validation_result is record (
    message           varchar2(32767),
    display_location  varchar2(40),        /* if not set the application default is
used */
    page_item_name    varchar2(255) ); /* if not set the validated page item name
is used */
```

### **t\_plugin**

The following type is passed into all plug-in functions and contains information about the current plug-in.

```
type t_plugin is record (
    name              varchar2(45),
    file_prefix       varchar2(4000),
    attribute_01      varchar2(32767),
    attribute_02      varchar2(32767),
    attribute_03      varchar2(32767),
    attribute_04      varchar2(32767),
    attribute_05      varchar2(32767),
    attribute_06      varchar2(32767),
    attribute_07      varchar2(32767),
    attribute_08      varchar2(32767),
    attribute_09      varchar2(32767),
    attribute_10      varchar2(32767),
    attribute_11      varchar2(32767),
    attribute_12      varchar2(32767),
    attribute_13      varchar2(32767),
    attribute_14      varchar2(32767),
```

```
attribute_15 varchar2(32767) );
```

### **t\_process**

The following type is passed into all process type plug-in functions and contains information about the current process.

```
type t_process is record (
    id                number,
    name              varchar2(255),
    success_message   varchar2(32767),
    attribute_01      varchar2(32767),
    attribute_02      varchar2(32767),
    attribute_03      varchar2(32767),
    attribute_04      varchar2(32767),
    attribute_05      varchar2(32767),
    attribute_06      varchar2(32767),
    attribute_07      varchar2(32767),
    attribute_08      varchar2(32767),
    attribute_09      varchar2(32767),
    Attribute_10      varchar2(32767),
    attribute_11      varchar2(32767),
    attribute_12      varchar2(32767),
    attribute_13      varchar2(32767),
    attribute_14      varchar2(32767),
    attribute_15      varchar2(32767) );
```

### **t\_process\_exec\_result**

The following type is used as the result type for the execution function of a process type plug-in.

```
type t_process_exec_result is record (
    success_message varchar2(32767)
    execution_skipped boolean default false /* set to TRUE if process execution
has been skipped by plug-in because of additional condition checks */
);
```

### **type t\_region\_column (**

The following type is passed into all region type plug-in functions and contains information about the current region.

```
type t_region_column is record (
    id                number,
    name              t_region_column_name,
    is_displayed      boolean,
    heading           wwv_flow_region_columns.heading%type,
    heading_alignment wwv_flow_region_columns.heading_alignment%type,
    value_alignment   wwv_flow_region_columns.value_alignment%type,
    value_css_classes wwv_flow_region_columns.value_css_classes%type,
    value_attributes  wwv_flow_region_columns.value_attributes%type,
    format_mask       wwv_flow_region_columns.format_mask%type,
    escape_output     boolean,
    attribute_01      varchar2(32767),
    attribute_02      varchar2(32767),
    attribute_03      varchar2(32767),
    attribute_04      varchar2(32767),
    attribute_05      varchar2(32767),
    attribute_06      varchar2(32767),
```

```
attribute_07      varchar2(32767),
attribute_08      varchar2(32767),
attribute_09      varchar2(32767),
attribute_10      varchar2(32767),
attribute_11      varchar2(32767),
attribute_12      varchar2(32767),
attribute_13      varchar2(32767),
attribute_14      varchar2(32767),
attribute_15      varchar2(32767),
attribute_16      varchar2(32767),
attribute_17      varchar2(32767),
attribute_18      varchar2(32767),
attribute_19      varchar2(32767),
attribute_20      varchar2(32767),
attribute_21      varchar2(32767),
attribute_22      varchar2(32767),
attribute_23      varchar2(32767),
attribute_24      varchar2(32767),
attribute_25      varchar2(32767);
```

**type t\_region\_columns is table of t\_region\_column index by pls\_integer;**

### **t\_region**

The following type is passed into all region type plug-in functions and contains information about the current region.

```
type t_region is record (
  id                number,
  static_id         varchar2(255),
  name              varchar2(255),
  type              varchar2(255),
  source            varchar2(32767),
  ajax_items_to_submit varchar2(32767),
  fetched_rows      pls_integer,
  escape_output     boolean,
  error_message     varchar2(32767), /* obsolete */
  no_data_found_message varchar2(32767),
  attribute_01      varchar2(32767),
  attribute_02      varchar2(32767),
  attribute_03      varchar2(32767),
  attribute_04      varchar2(32767),
  attribute_05      varchar2(32767),
  attribute_06      varchar2(32767),
  attribute_07      varchar2(32767),
  attribute_08      varchar2(32767),
  attribute_09      varchar2(32767),
  attribute_10      varchar2(32767),
  attribute_11      varchar2(32767),
  attribute_12      varchar2(32767),
  attribute_13      varchar2(32767),
  attribute_14      varchar2(32767),
  attribute_15      varchar2(32767),
  attribute_16      varchar2(32767),
  attribute_17      varchar2(32767),
  attribute_18      varchar2(32767),
  attribute_19      varchar2(32767),
  attribute_20      varchar2(32767),
  attribute_21      varchar2(32767),
  attribute_22      varchar2(32767),
```

```
attribute_23      varchar2(32767),
attribute_24      varchar2(32767),
attribute_25      varchar2(32767),
region_columns    t_region_columns );
```

### **t\_region\_ajax\_result**

The following type is used as result type for the Ajax function of a region type plug-in.

```
type t_region_ajax_result is record (
    dummy boolean /* not used yet */
);
```

### **t\_region\_render\_result**

The following type is used as the result type for the rendering function of a region type plug-in.

```
type t_region_render_result is record (
    navigable_dom_id varchar2(255) /* can be used to put focus to an input field
(that is, search field) the region renders as part of the plug-in output */
);
```

## GET\_AJAX\_IDENTIFIER Function

This function returns the Ajax identifier used to call the Ajax callback function defined for the plug-in.

---

---

**Note:** This function only works in the context of a plug-in rendering function call and only if the plug-in has defined an Ajax function callback in the plug-in definition.

---

---

### Syntax

```
APEX_PLUGIN.GET_AJAX_IDENTIFIER  
RETURN VARCHAR2;
```

### Parameters

None.

### Example

This is an example of a dynamic action plug-in rendering function that supports an Ajax callback.

```
function render_set_value (  
    p_dynamic_action in apex_plugin.t_dynamic_action )  
    return apex_plugin.t_dynamic_action_render_result  
is  
    l_result          apex_plugin.t_dynamic_action_render_result;  
begin  
    l_result.javascript_function := 'com_oracle_apex_set_value';  
    l_result.ajax_identifier     := wwv_flow_plugin.get_ajax_identifier;  
    return l_result;  
end;
```

## GET\_INPUT\_NAME\_FOR\_PAGE\_ITEM Function

Use this function when you want to render an HTML input element in the rendering function of an item type plug-in.

For the HTML input element, for example, `<input type="text" id="P1_TEST" name="xxx">`, you have to provide a value for the name attribute so that Oracle Application Express can map the submitted value to the actual page item in session state. This function returns the mapping name for your page item. If the HTML input element has multiple values, such as a select list with `multiple="multiple"`, then set `p_is_multi_value` to `TRUE`.

---

**Note:** This function is only useful when called in the rendering function of an item type plug-in.

---

### Syntax

```
APEX_PLUGIN.GET_INPUT_NAME_FOR_PAGE_ITEM (
    p_is_multi_value IN BOOLEAN)
RETURN VARCHAR2;
```

### Parameters

Table 20–1 describes the parameters available in the GET\_INPUT\_NAME\_FOR\_PAGE\_ITEM function.

**Table 20–1** GET\_INPUT\_NAME\_FOR\_PAGE\_ITEM Parameters

Parameter	Description
<code>p_is_multi_value</code>	Set to <code>TRUE</code> if the HTML input element has multiple values. If not, set to <code>FALSE</code> . HTML input elements with multiple values can be checkboxes and multi select lists.

### Example

The following example outputs the necessary HTML code to render a text field where the value gets stored in session state when the page is submitted.

```
sys.ftp.prn (
    '<input type="text" id="' || p_item.name || '" ' ||
    'name="' || wwv_flow_plugin.get_input_name_for_page_item(false) || '" ' ||
    'value="' || sys.htf.escape_sc(p_value) || '" ' ||
    'size="' || p_item.element_width || '" ' ||
    'maxlength="' || p_item.element_max_length || '" ' ||
    coalesce(p_item.element_attributes, 'class="text_field"') ||' /> ');
```





---

---

## APEX\_PLUGIN\_UTIL

The APEX\_PLUGIN\_UTIL package provides utility functions that solve common problems when writing a plug-in.

- [DEBUG\\_DYNAMIC\\_ACTION Procedure](#)
- [DEBUG\\_PAGE\\_ITEM Procedure Signature 1](#)
- [DEBUG\\_PAGE\\_ITEM Procedure Signature 2](#)
- [DEBUG\\_PROCESS Procedure](#)
- [DEBUG\\_REGION Procedure Signature 1](#)
- [DEBUG\\_REGION Procedure Signature 2](#)
- [ESCAPE Function](#)
- [EXECUTE\\_PLSQL\\_CODE Procedure](#)
- [GET\\_ATTRIBUTE\\_AS\\_NUMBER Function](#)
- [GET\\_DATA Function Signature 1](#)
- [GET\\_DATA Function Signature 2](#)
- [GET\\_DATA2 Function Signature 1](#)
- [GET\\_DATA2 Function Signature 2](#)
- [GET\\_DISPLAY\\_DATA Function Signature 1](#)
- [GET\\_DISPLAY\\_DATA Function Signature 2](#)
- [GET\\_ELEMENT\\_ATTRIBUTES Function](#)
- [GET\\_PLSQL\\_EXPRESSION\\_RESULT Function](#)
- [GET\\_PLSQL\\_FUNCTION\\_RESULT Function](#)
- [GET\\_POSITION\\_IN\\_LIST Function](#)
- [GET\\_SEARCH\\_STRING Function](#)
- [IS\\_EQUAL Function](#)
- [PAGE\\_ITEM\\_NAMES\\_TO\\_JQUERY Function](#)
- [PRINT\\_DISPLAY\\_ONLY Procedure](#)
- [PRINT\\_ESCAPED\\_VALUE Procedure](#)
- [PRINT\\_HIDDEN\\_IF\\_READONLY Procedure](#)
- [PRINT\\_JSON\\_HTTP\\_HEADER Procedure](#)

- 
- [PRINT\\_LOV\\_AS\\_JSON Procedure](#)
  - [PRINT\\_OPTION Procedure](#)
  - [REPLACE\\_SUBSTITUTIONS Function](#)

---

## DEBUG\_DYNAMIC\_ACTION Procedure

This procedure writes the data of the dynamic action meta data to the debug output if debugging is enabled.

### Syntax

```
APEX_PLUGIN_UTIL.DEBUG_DYNAMIC_ACTION (  
    p_plugin          IN apex_plugin.t_plugin,  
    p_dynamic_action IN apex_plugin.t_dynamic_action);
```

### Parameters

Table 21–2 describes the parameters available in the DEBUG\_DYNAMIC\_ACTION procedure.

**Table 21–1** *DEBUG\_DYNAMIC\_ACTION Parameters*

Parameter	Description
p_plugin	This is the p_plugin parameter of your plug-in function.
p_dynamic_action	This is the p_dynamic_action parameter of your plug-in function.

### Example

This example shows how to collect helpful debug information during the plug-in development cycle to see what values are actually passed into the rendered function or Ajax callback function of the plug-in.

```
apex_plugin_util.debug_dynamic_action (  
    p_plugin          => p_plugin,  
    p_dynamic_action => p_dynamic_action );
```

## DEBUG\_PAGE\_ITEM Procedure Signature 1

This procedure writes the data of the page item meta data to the debug output if debugging is enabled.

### Syntax

```
APEX_PLUGIN_UTIL.DEBUG_PAGE_ITEM (  
    p_plugin      IN apex_plugin.t_plugin,  
    p_page_item  IN apex_plugin.t_page_item);
```

### Parameters

[Table 21-2](#) describes the parameters available in the `DEBUG_PAGE_ITEM` procedure.

**Table 21-2** *DEBUG\_PAGE\_ITEM Parameters*

Parameter	Description
<code>p_plugin</code>	This is the <code>p_plugin</code> parameter of your plug-in function.
<code>p_page_item</code>	This is the <code>p_page_item</code> parameter of your plug-in function.

### Example

This example shows how to collect helpful debug information during the plug-in development cycle to see what values are actually passed into the renderer, Ajax callback or validation function.

```
apex_plugin_util.debug_page_item (  
    p_plugin      => p_plugin,  
    p_page_item  => p_page_item );
```

## DEBUG\_PAGE\_ITEM Procedure Signature 2

This procedure writes the data of the page item meta data to the debug output if debugging is enabled.

### Syntax

```
APEX_PLUGIN_UTIL.DEBUG_PAGE_ITEM (
    p_plugin           IN apex_plugin.t_plugin,
    p_page_item       IN apex_plugin.t_page_item,
    p_value            IN VARCHAR2,
    p_is_readonly     IN BOOLEAN,
    p_is_printer_friendly IN BOOLEAN);
```

### Parameters

Table 21–3 describes the parameters available in the `DEBUG_PAGE_ITEM` procedure.

**Table 21–3** *DEBUG\_PAGE\_ITEM Parameters*

Parameter	Description
<code>p_plugin</code>	This is the <code>p_plugin</code> parameter of your plug-in function.
<code>p_page_item</code>	This is the <code>p_page_item</code> parameter of your plug-in function.
<code>p_value</code>	This is the <code>p_value</code> parameter of your plug-in function.
<code>p_is_readonly</code>	This is the <code>p_is_readonly</code> parameter of your plug-in function.
<code>p_is_printer_friendly</code>	This is the <code>p_is_printer_friendly</code> parameter of your plug-in function.

### Example

This example shows how to collect helpful debug information during the plug-in development cycle to see what values are actually passed into the renderer, Ajax callback or validation function.

```
apex_plugin_util.debug_page_item (
    p_plugin           => p_plugin,
    p_page_item       => p_page_item,
    p_value            => p_value,
    p_is_readonly     => p_is_readonly,
    p_is_printer_friendly => p_is_printer_friendly);
```

---

## DEBUG\_PROCESS Procedure

This procedure writes the data of the process meta data to the debug output if debugging is enabled.

### Syntax

```
APEX_PLUGIN_UTIL.DEBUG_PROCESS (  
    p_plugin          IN apex_plugin.t_plugin,  
    p_process         IN apex_plugin.t_process);
```

### Parameters

[Table 21–4](#) describes the parameters available in the `DEBUG_PROCESS` procedure.

**Table 21–4** *DEBUG\_PROCESS Parameters*

Parameter	Description
<code>p_plugin</code>	This is the <code>p_plugin</code> parameter of your plug-in function.
<code>p_process</code>	This is the <code>p_process</code> parameter of your plug-in function.

### Example

This example shows how to collect helpful debug information during the plug-in development cycle to see what values are actually passed into the execution function of the plug-in.

```
apex_plugin_util.debug_process (  
    p_plugin          => p_plugin,  
    p_process         => p_process);
```

---

## DEBUG\_REGION Procedure Signature 1

This procedure writes the data of the region meta data to the debug output if debugging is enabled.

### Syntax

```
APEX_PLUGIN_UTIL.DEBUG_REGION (  
    p_plugin          IN apex_plugin.t_plugin,  
    p_region          IN apex_plugin.t_region);
```

### Parameters

[Table 21-5](#) describes the parameters available in the `DEBUG_REGION` procedure.

**Table 21-5** *DEBUG\_REGION Signature 1 Parameters*

Parameter	Description
<code>p_plugin</code>	This is the <code>p_plugin</code> parameter of your plug-in function.
<code>p_region</code>	This is the <code>p_region</code> parameter of your plug-in function.

### Example

This example shows how to collect helpful debug information during the plug-in development cycle to see what values are actually passed into the render function or Ajax callback function of the plug-in.

```
apex_plugin_util.debug_region (  
    p_plugin => p_plugin,  
    p_region => p_region);
```

## DEBUG\_REGION Procedure Signature 2

This procedure writes the data of the region meta data to the debug output if debugging is enabled. This is the advanced version of the debugging procedure which is used for the rendering function of a region plug-in.

### Syntax

```
APEX_PLUGIN_UTIL.DEBUG_REGION (  
    p_plugin          IN apex_plugin.t_plugin,  
    p_region         IN apex_plugin.t_region,  
    p_is_printer_friendly IN BOOLEAN);
```

### Parameters

Table 21–6 describes the parameters available in the DEBUG\_REGION procedure.

**Table 21–6** *DEBUG\_REGION Signature 2 Parameters*

Parameter	Description
p_plugin	This is the p_plugin parameter of your plug-in function
p_region	This is the p_region parameter of your plug-in function
p_is_printer_friendly	This is the p_is_printer_friendly parameter of your plug-in function

### Example

This example shows how to collect helpful debug information during the plug-in development cycle to see what values are actually passed into the render function or Ajax callback function of the plug-in.

```
apex_plugin_util.debug_region (  
    p_plugin          => p_plugin,  
    p_region         => p_region,  
    p_is_printer_friendly => p_is_printer_friendly);
```



## ESCAPE Function

This function is used if you have checked the standard attribute "Has Escape Output Attribute" option for your item type plug-in which allows a developer to decide if the output should be escaped or not.

### Syntax

```
APEX_PLUGIN_UTIL.ESCAPE (
    p_value IN VARCHAR2,
    p_escape IN BOOLEAN)
RETURN VARCHAR2;
```

### Parameters

Table 21-7 describes the parameters available in the ESCAPE function.

**Table 21-7 ESCAPE Parameters**

Parameter	Description
p_value	This is the value you want to escape depending on the p_escape parameter.
p_escape	If set to TRUE, the return value is escaped. If set to FALSE, the value is not escaped.

### Example

This example outputs all values of the array l\_display\_value\_list as a HTML list and escapes the value of the array depending on the setting the developer as picked when using the plug-in.

```
for i in 1 .. l_display_value_list.count
loop
    sys.htp.prn (
        '<li>' ||
        apex_plugin_util.escape (
            p_value => l_display_value_list(i),
            p_escape => p_item.escape_output ) ||
        '</li>' );
end loop;
```

---

## EXECUTE\_PLSQL\_CODE Procedure

This procedure executes a PL/SQL code block and performs binding of bind variables in the provided PL/SQL code. This procedure is usually used for plug-in attributes of type PL/SQL Code.

### Syntax

```
APEX_PLUGIN_UTIL.EXECUTE_PLSQL_CODE (  
    p_plsql_code IN VARCHAR2);
```

### Parameters

[Table 21–8](#) describes the parameters available in the EXECUTE\_PLSQL\_CODE procedure.

**Table 21–8 EXECUTE\_PLSQL\_CODE Parameters**

Parameter	Description
p_plsql_code	PL/SQL code to be executed.

### Example

Text which should be escaped and then printed to the HTTP buffer.

```
declare  
    l_plsql_code VARCHAR(32767) := p_process.attribute_01;  
begin  
    apex_plugin_util.execute_plsql_code (  
        p_plsql_code => l_plsql_code );  
end;
```

## GET\_ATTRIBUTE\_AS\_NUMBER Function

This function returns the value of a plug-in attribute as a number, taking into account NLS decimal separator effective for the current database session. Use this function in plug-in PL/SQL source for custom attributes of type NUMBER instead of the built-in to\_number function.

### Syntax

```
APEX_PLUGIN_UTIL.GET_ATTRIBUTE_AS_NUMBER (
  p_value IN VARCHAR2 ),
  p_attribute_label IN VARCHAR2 )
return NUMBER;
```

### Parameters

Table 21-9 describes the parameters available in the GET\_ATTRIBUTE\_AS\_NUMBER function.

**Table 21-9** GET\_ATTRIBUTE\_AS\_NUMBER Function Parameters

Parameter	Description
p_attribute_label	The label of the custom plug-in attribute.
p_value	The value of a custom attribute of type NUMBER.

### Example

```
declare
  l_value number;
begin
  -- The following may fail for languages that don't use dot as the NLS decimal
  separator
  l_value := to_number( p_region.attribute_04 );

  -- The following will work correctly regardless of the effective NLS decimal
  separator
  l_value := apex_plugin_util.get_attribute_as_number( p_region.attribute_04,
'Minimum Amount' );
end;
/
```

## GET\_DATA Function Signature 1

Executes the specified SQL query restricted by the provided search string (optional) and returns the values for each column. All column values are returned as a string, independent of their data types. The search column is identified by providing a column number in the `p_search_column_no` parameter. This function takes into account character value comparison globalization attributes defined for the application.

### Syntax

```
APEX_PLUGIN_UTIL.GET_DATA (
    p_sql_statement      IN VARCHAR2,
    p_min_columns       IN NUMBER,
    p_max_columns       IN NUMBER,
    p_component_name    IN VARCHAR2,
    p_search_type       IN VARCHAR2 DEFAULT 2,
    p_search_column_no  IN VARCHAR2 DEFAULT 2,
    p_search_string     IN VARCHAR2 DEFAULT NULL,
    p_first_row        IN NUMBER DEFAULT NULL,
    p_max_rows         IN NUMBER DEFAULT NULL)
RETURN t_column_value_list;
```

### Parameters

[Table 21–10](#) describes the parameters available in the `GET_DATA` function signature 1.

**Table 21–10** *GET\_DATA Function Signature 1 Parameters*

Parameters	Description
<code>p_sql_statement</code>	SQL statement used for the lookup.
<code>p_min_columns</code>	Minimum number of return columns.
<code>p_max_columns</code>	Maximum number of return columns.
<code>p_component_name</code>	In case an error is returned, this is the name of the page item or report column used to display the error message.
<code>p_search_type</code>	Must be one of the <code>c_search_*</code> constants. They are as follows: <code>c_search_contains_case</code> , <code>c_search_contains_ignore</code> , <code>c_search_exact_case</code> , <code>c_search_exact_ignore</code>
<code>p_search_column_no</code>	Number of the column used to restrict the SQL statement. Must be within the <code>p_min_columns</code> through <code>p_max_columns</code> range.
<code>p_search_string</code>	Value used to restrict the query.
<code>p_first_row</code>	Start query at the specified row. All rows before the specified row are skipped.
<code>p_max_rows</code>	Maximum number of return rows allowed.

### Return

[Table 21–11](#) describes the return value by the `GET_DATA` function signature 1.

**Table 21–11** *GET\_DATA Function Signature 1 Return*

Return	Description
<code>t_column_value_list</code>	Table of <code>apex_application_global.vc_arr2</code> indexed by column number.

**Example**

The following example shows a simple item type plug-in rendering function which executes the LOV defined for the page item and does a case sensitive LIKE filtering with the current value of the page item. The result is then generated as a HTML list.

```
function render_list (
    p_item          in apex_plugin.t_page_item,
    p_value         in varchar2,
    p_is_readonly   in boolean,
    p_is_printer_friendly in boolean )
return apex_plugin.t_page_item_render_result
is
    l_column_value_list apex_plugin_util.t_column_value_list;
begin
    l_column_value_list :=
        apex_plugin_util.get_data (
            p_sql_statement => p_item.lov_definition,
            p_min_columns  => 2,
            p_max_columns  => 2,
            p_component_name => p_item.name,
            p_search_type   => apex_plugin_util.c_search_contains_case,
            p_search_column_no => 1,
            p_search_string => p_value );

    sys.htp.p('<ul>');
    for i in 1 .. l_column_value_list(1).count
    loop
        sys.htp.p(
            '<li>' ||
            sys.htf.escape_sc(l_column_value_list(1)(i)) || -- display column
            '-' ||
            sys.htf.escape_sc(l_column_value_list(2)(i)) || -- return column
            '</li>');
    end loop;
    sys.htp.p('</ul>');
end render_list;
```

## GET\_DATA Function Signature 2

Executes the specified SQL query restricted by the provided search string (optional) and returns the values for each column. All column values are returned as a string, independent of their data types. The search column is identified by providing a column name in the `p_search_column_name` parameter. This function takes into account character value comparison globalization attributes defined for the application.

### Syntax

```
APEX_PLUGIN_UTIL.GET_DATA (
    p_sql_statement      IN VARCHAR2,
    p_min_columns        IN NUMBER,
    p_max_columns        IN NUMBER,
    p_component_name     IN VARCHAR2,
    p_search_type        IN VARCHAR2 DEFAULT NULL,
    p_search_column_name IN VARCHAR2 DEFAULT NULL,
    p_search_string      IN VARCHAR2 DEFAULT NULL,
    p_first_row          IN NUMBER DEFAULT NULL,
    p_max_rows           IN NUMBER DEFAULT NULL)
RETURN t_column_value_list;
```

### Parameters

[Table 21–12](#) describes the parameters available for GET\_DATA function signature 2.

**Table 21–12** GET\_DATA Function Signature 2 Parameters

Parameters	Description
<code>p_sql_statement</code>	SQL statement used for the lookup.
<code>p_min_columns</code>	Minimum number of return columns.
<code>p_max_columns</code>	Maximum number of return columns.
<code>p_component_name</code>	In case an error is returned, this is the name of the page item or report column used to display the error message.
<code>p_search_type</code>	Must be one of the <code>c_search_*</code> constants. They are as follows: <code>c_search_contains_case</code> , <code>c_search_contains_ignore</code> , <code>c_search_exact_case</code> , <code>c_search_exact_ignore</code>
<code>p_search_column_name</code>	This is the column name used to restrict the SQL statement.
<code>p_search_string</code>	Value used to restrict the query.
<code>p_first_row</code>	Start query at the specified row. All rows before the specified row are skipped.
<code>p_max_rows</code>	Maximum number of return rows allowed.

### Return

[Table 21–13](#) describes the return value by the GET\_DATA function signature 2.

**Table 21–13** GET\_TABLE Function Signature 2

Parameter	Description
<code>t_column_value_list</code>	Table of <code>apex_application_global.vc_arr2</code> indexed by column number.

**Example**

The following example shows a simple item type plug-in rendering function which executes the LOV defined for the page item and does a case sensitive LIKE filtering with the current value of the page item. The result is then generated as a HTML list.

```
function render_list (
    p_item          in apex_plugin.t_page_item,
    p_value         in varchar2,
    p_is_readonly   in boolean,
    p_is_printer_friendly in boolean )
return apex_plugin.t_page_item_render_result
is
    l_column_value_list apex_plugin_util.t_column_value_list;
begin
    l_column_value_list :=
        apex_plugin_util.get_data (
            p_sql_statement => p_item.lov_definition,
            p_min_columns   => 2,
            p_max_columns   => 2,
            p_component_name => p_item.name,
            p_search_type    => apex_plugin_util.c_search_contains_case,
            p_search_column_name => 'ENAME',
            p_search_string  => p_value );

    sys.htp.p('<ul>');
    for i in 1 .. l_column_value_list(1).count
    loop
        sys.htp.p(
            '<li>' ||
            sys.htf.escape_sc(l_column_value_list(1)(i)) || -- display column
            '-' ||
            sys.htf.escape_sc(l_column_value_list(2)(i)) || -- return column
            '</li>');
    end loop;
    sys.htp.p('</ul>');
end render_list;
```

## GET\_DATA2 Function Signature 1

Executes the specified SQL query restricted by the provided search string (optional) and returns the values for each column. All column values are returned along with their original data types. The search column is identified by providing a column number in the `p_search_column_no` parameter. This function takes into account character value comparison globalization attributes defines for the application.

### Syntax

```
APEX_PLUGIN_UTIL.GET_DATA2 (
    p_sql_statement      IN VARCHAR2,
    p_min_columns       IN NUMBER,
    p_max_columns       IN NUMBER,
    p_data_type_list    IN WWV_GLOBAL.VC_ARR2 DEFAULT C_EMPTY_DATA_TYPE_LIST,
    p_component_name    IN VARCHAR2,
    p_search_type       IN VARCHAR2 DEFAULT 2,
    p_search_column_no  IN VARCHAR2 DEFAULT 2,
    p_search_string     IN VARCHAR2 DEFAULT NULL,
    p_first_row        IN NUMBER DEFAULT NULL,
    p_max_rows         IN NUMBER DEFAULT NULL)
RETURN t_column_value_list2;
```

### Parameters

[Table 21–14](#) describes the parameters available in the `GET_DATA2` function.

**Table 21–14** *GET\_DATA2 Parameters*

Parameter	Description
<code>p_sql_statement</code>	SQL statement used for the lookup.
<code>p_min_columns</code>	Minimum number of return columns.
<code>p_max_columns</code>	Maximum number of return columns.
<code>p_data_type_list</code>	If provided, checks to make sure the data type for each column matches the specified data type in the array. Use the constants <code>c_data_type_*</code> for available data types.
<code>p_component_name</code>	In case an error is returned, this is the name of the page item or report column used to display the error message.
<code>p_search_type</code>	Must be one of the <code>c_search_*</code> constants. They are as follows: <code>c_search_contains_case</code> , <code>c_search_contains_ignore</code> , <code>c_search_exact_case</code> , <code>c_search_exact_ignore</code>
<code>p_search_column_no</code>	Number of the column used to restrict the SQL statement. Must be within the <code>p_min_columns</code> though <code>p_max_columns</code> range.
<code>p_search_string</code>	Value used to restrict the query.
<code>p_first_row</code>	Start query at the specified row. All rows before the specified row are skipped.
<code>p_max_rows</code>	Maximum number of return rows allowed.

### Return

[Table 21–15](#) describes the return value by the `GET_DATA2` function.



**Table 21–15 GET\_DATA2 Return**

Return	Description
t_column_value_list2	Table of t_column_values indexed by column number.

**Example**

The following example is a simple item type plug-in rendering function which executes the LOV defined for the page item and does a case sensitive LIKE filtering with the current value of the page item. The result is then generated as a HTML list. This time, the first column of the LOV SQL statement is checked if it is of type VARCHAR2 and the second is of type number.

```
function render_list (
    p_item          in apex_plugin.t_page_item,
    p_value         in varchar2,
    p_is_readonly   in boolean,
    p_is_printer_friendly in boolean )
return apex_plugin.t_page_item_render_result
is
    l_data_type_list apex_application_global.vc_arr2;
    l_column_value_list apex_plugin_util.t_column_value_list2;
begin
    -- The first LOV column has to be a string and the second a number
    l_data_type_list(1) := apex_plugin_util.c_data_type_varchar2;
    l_data_type_list(2) := apex_plugin_util.c_data_type_number;
    --
    l_column_value_list :=
        apex_plugin_util.get_data2 (
            p_sql_statement => p_item.lov_definition,
            p_min_columns   => 2,
            p_max_columns   => 2,
            p_data_type_list => l_data_type_list,
            p_component_name => p_item.name,
            p_search_type    => apex_plugin_util.c_search_contains_case,
            p_search_column_no => 1,
            p_search_string  => p_value );
    --
    sys.htp.p('<ul>');
    for i in 1 .. l_column_value_list.count(1)
    loop
        sys.htp.p(
            '<li>' ||
            sys.htf.escape_sc(l_column_value_list(1).value_list(i).varchar2_
value) || -- display column
            '-' ||
            sys.htf.escape_sc(l_column_value_list(2).value_list(i).number_value) ||
-- return column
            '</li>');
    end loop;
    sys.htp.p('</ul>');
end render_list;
```

## GET\_DATA2 Function Signature 2

Executes the specified SQL query restricted by the provided search string (optional) and returns the values for each column. All column values are returned along with their original data types. The search column is identified by providing a column number in the `p_search_column_no` parameter. This function takes into account character value comparison globalization attributes defines for the application.

### Syntax

```
APEX_PLUGIN_UTIL.GET_DATA2 (
    p_sql_statement      IN VARCHAR2,
    p_min_columns        IN NUMBER,
    p_max_columns        IN NUMBER,
    p_data_type_list     IN WWV_GLOBAL.VC_ARR2 DEFAULT C_EMPTY_DATA_TYPE_LIST,
    p_component_name     IN VARCHAR2,
    p_search_type        IN VARCHAR2 DEFAULT 2,
    p_search_column_name IN VARCHAR2 DEFAULT 2,
    p_search_string      IN VARCHAR2 DEFAULT NULL,
    p_first_row          IN NUMBER DEFAULT NULL,
    p_max_rows           IN NUMBER DEFAULT NULL)
RETURN t_column_value_list2;
```

### Parameters

[Table 21–16](#) describes the parameters available in the GET\_DATA2 function signature 2.

**Table 21–16** GET\_DATA2 Function Signature 2

Parameter	Description
<code>p_sql_statement</code>	SQL statement used for the lookup.
<code>p_min_columns</code>	Minimum number of return columns.
<code>p_max_columns</code>	Maximum number of return columns.
<code>p_data_type_list</code>	If provided, checks to make sure the data type for each column matches the specified data type in the array. Use the constants <code>c_data_type_*</code> for available data types.
<code>p_component_name</code>	In case an error is returned, this is the name of the page item or report column used to display the error message.
<code>p_search_type</code>	Must be one of the <code>c_search_*</code> constants. They are as follows: <code>c_search_contains_case</code> , <code>c_search_contains_ignore</code> , <code>c_search_exact_case</code> , <code>c_search_exact_ignore</code>
<code>p_search_column_name</code>	The column name used to restrict the SQL statement.
<code>p_search_string</code>	Value used to restrict the query.
<code>p_first_row</code>	Start query at the specified row. All rows before the specified row are skipped.
<code>p_max_rows</code>	Maximum number of return rows allowed.

### Return

[Table 21–17](#) describes the return value by the GET\_DATA2 function signature 2.

**Table 21–17 GET\_DATA2 Function Signature 2 Return**

Parameter	Description
t_column_value_list2	Table of t_column_values indexed by column number.

**Example**

The following example is a simple item type plug-in rendering function which executes the LOV defined for the page item and does a case sensitive LIKE filtering with the current value of the page item. The result is then generated as a HTML list. This time, the first column of the LOV SQL statement is checked if it is of type VARCHAR2 and the second is of type number.

```
function render_list (
    p_item          in apex_plugin.t_page_item,
    p_value         in varchar2,
    p_is_readonly   in boolean,
    p_is_printer_friendly in boolean )
return apex_plugin.t_page_item_render_result
is
    l_data_type_list apex_application_global.vc_arr2;
    l_column_value_list apex_plugin_util.t_column_value_list2;
begin
    -- The first LOV column has to be a string and the second a number
    l_data_type_list(1) := apex_plugin_util.c_data_type_varchar2;
    l_data_type_list(2) := apex_plugin_util.c_data_type_number;
    --
    l_column_value_list :=
        apex_plugin_util.get_data2 (
            p_sql_statement => p_item.lov_definition,
            p_min_columns  => 2,
            p_max_columns  => 2,
            p_data_type_list => l_data_type_list,
            p_component_name => p_item.name,
            p_search_type   => apex_plugin_util.c_search_contains_case,
            p_search_column_name => 'ENAME',
            p_search_string  => p_value );
    --
    sys.htp.p('<ul>');
    for i in 1 .. l_column_value_list.count(1)
    loop
        sys.htp.p(
            '<li>' ||
            sys.htf.escape_sc(l_column_value_list(1).value_list(i).varchar2_
value) || -- display column
            '-' ||
            sys.htf.escape_sc(l_column_value_list(2).value_list(i).number_value) ||
-- return column
            '</li>');
    end loop;
    sys.htp.p('</ul>');
end render_list;
```

## GET\_DISPLAY\_DATA Function Signature 1

This function gets the display lookup value for the value specified in `p_search_string`.

### Syntax

```
APEX_PLUGIN_UTIL.GET_DISPLAY_DATA (
    p_sql_statement      IN VARCHAR2,
    p_min_columns       IN NUMBER,
    p_max_columns       IN NUMBER,
    p_component_name    IN VARCHAR2,
    p_display_column_no IN BINARY_INTEGER DEFAULT 1,
    p_search_column_no  IN BINARY_INTEGER DEFAULT 2,
    p_search_string     IN VARCHAR2 DEFAULT NULL,
    p_display_extra     IN BOOLEAN DEFAULT TRUE)
RETURN VARCHAR2;
```

### Parameters

[Table 21–18](#) describes the parameters available in the `GET_DISPLAY_DATA` function signature 1.

**Table 21–18** *GET\_DISPLAY\_DATA Signature 1 Parameters*

Parameter	Description
<code>p_sql_statement</code>	SQL statement used for the lookup.
<code>p_min_columns</code>	Minimum number of return columns.
<code>p_max_columns</code>	Maximum number of return columns.
<code>p_component_name</code>	In case an error is returned, this is the name of the page item or report column used to display the error message.
<code>p_display_column_no</code>	Number of the column returned from the SQL statement. Must be within the <code>p_min_columns</code> through <code>p_max_columns</code> range.
<code>p_search_column_no</code>	Number of the column used to restrict the SQL statement. Must be within the <code>p_min_columns</code> through <code>p_max_columns</code> range.
<code>p_search_string</code>	Value used to restrict the query.
<code>p_display_extra</code>	If set to <code>TRUE</code> , and a value is not found, the search value is added to the result instead.

### Return

[Table 21–19](#) describes the return value by the `GET_DISPLAY_DATA` function signature 1.

**Table 21–19** *GET\_DISPLAY\_DATA Signature 1 Return*

Return	Description
VARCHAR2	Value of the first record of the column specified by <code>p_display_column_no</code> . If no record was found it contains the value of <code>p_search_string</code> if the parameter <code>p_display_extra</code> is set to <code>TRUE</code> . Otherwise <code>NULL</code> is returned.

**Example**

The following example does a lookup with the value provided in p\_value and returns the display column of the LOV query.

```
function render_value (
    p_item          in apex_plugin.t_page_item,
    p_value         in varchar2,
    p_is_readonly   in boolean,
    p_is_printer_friendly in boolean )
return apex_plugin.t_page_item_render_result
is
begin
    sys.htp.p(sys.htf.escape_sc(
        apex_plugin_util.get_display_data (
            p_sql_statement => p_item.lov_definition,
            p_min_columns  => 2,
            p_max_columns  => 2,
            p_component_name => p_item.name,
            p_display_column_no => 1,
            p_search_column_no => 2,
            p_search_string  => p_value )));
end render_value;
```

## GET\_DISPLAY\_DATA Function Signature 2

This function looks up all the values provided in the `p_search_value_list` instead of just a single value lookup.

### Syntax

```
APEX_PLUGIN_UTIL.GET_DISPLAY_DATA (
    p_sql_statement      IN VARCHAR2,
    p_min_columns       IN NUMBER,
    p_max_columns       IN NUMBER,
    p_component_name    IN VARCHAR2,
    p_display_column_no IN BINARY_INTEGER DEFAULT 1,
    p_search_column_no  IN BINARY_INTEGER DEFAULT 2,
    p_search_value_list IN ww_flow_global.vc_arr2,
    p_display_extra     IN BOOLEAN DEFAULT TRUE)
RETURN apex_application_global.vc_arr2;
```

### Parameters

[Table 21–20](#) describes the parameters available in the `GET_DISPLAY_DATA` function signature 2.

**Table 21–20** *GET\_DISPLAY\_DATA Signature 2 Parameters*

Parameter	Description
<code>p_sql_statement</code>	SQL statement used for the lookup.
<code>p_min_columns</code>	Minimum number of return columns.
<code>p_max_columns</code>	Maximum number of return columns.
<code>p_component_name</code>	In case an error is returned, this is the name of the page item or report column used to display the error message.
<code>p_display_column_no</code>	Number of the column returned from the SQL statement. Must be within the <code>p_min_columns</code> through <code>p_max_columns</code> range.
<code>p_search_column_no</code>	Number of the column used to restrict the SQL statement. Must be within the <code>p_min_columns</code> through <code>p_max_columns</code> range.
<code>p_search_value_list</code>	Array of values to look up.
<code>p_display_extra</code>	If set to <code>TRUE</code> , and a value is not found, the search value is added to the result instead.

### Return

[Table 21–21](#) describes the return value by the `GET_DISPLAY_DATA` function signature 2.

**Table 21–21** *GET\_DISPLAY\_DATA Signature 2 Return*

Return	Description
<code>apex_application_global.vc_arr2</code>	List of <code>VARCHAR2</code> indexed by <code>pls_integer</code> . For each entry in <code>p_search_value_list</code> the resulting array contains the value of the first record of the column specified by <code>p_display_column_no</code> in the same order as in <code>p_search_value_list</code> . If no record is found it contains the value of <code>p_search_string</code> if the parameter <code>p_display_extra</code> is set to <code>TRUE</code> . Otherwise the value is skipped.

**Example**

Looks up the values 7863, 7911 and 7988 and generates a HTML list with the value of the corresponding display column in the LOV query.

```
function render_list (
    p_plugin          in apex_plugin.t_plugin,
    p_item            in apex_plugin.t_page_item,
    p_value           in varchar2,
    p_is_readonly     in boolean,
    p_is_printer_friendly in boolean )
    return apex_plugin.t_page_item_render_result
is
    l_search_list apex_application_global.vc_arr2;
    l_result_list apex_application_global.vc_arr2;
begin
    l_search_list(1) := '7863';
    l_search_list(2) := '7911';
    l_search_list(3) := '7988';
    --
    l_result_list :=
        apex_plugin_util.get_display_data (
            p_sql_statement => p_item.lov_definition,
            p_min_columns  => 2,
            p_max_columns  => 2,
            p_component_name => p_item.name,
            p_search_column_no => 1,
            p_search_value_list => l_search_list );
    --
    sys.htp.p('<ul>');
    for i in 1 .. l_result_list.count
    loop
        sys.htp.p(
            '<li>' ||
            sys.htf.escape_sc(l_result_list(i)) ||
            '</li>');
    end loop;
    sys.htp.p('</ul>');
end render_list;
```

## GET\_ELEMENT\_ATTRIBUTES Function

This function returns some of the standard attributes of an HTML element (for example, id, name, required, placeholder, aria-error-attributes, class) which is used if a HTML input/select/textarea/... tag is generated to get a consistent set of attributes.

### Syntax

```
APEX_PLUGIN_UTIL.GET_ELEMENT_ATTRIBUTES (
    p_item in apex_plugin.t_page_item,
    p_name in varchar2 default null,
    p_default_class in varchar2 default null,
    p_add_id in boolean default true,
    p_add_labelledby in boolean default true )
return varchar2;
```

### Parameters

Table 21–22 describes the available parameters for GET\_ELEMENT\_ATTRIBUTES function.

**Table 21–22 GET\_ELEMENT\_ATTRIBUTES Function Parameters**

Parameters	Description
p_add_labelled_by	Returns some of the general attributes of an HTML element (for example, the ID, name, required, placeholder, aria-error-attributes, class) which should be used if an HTML input, select, or textarea tag is generated to get a consistent set of attributes.  Set to FALSE if you render a HTML input element like input, select, or textarea which does not require specifying the aria-labelledby attribute because the label's for attribute works for those HTML input elements. Set it to TRUE for all 'non-standard form element widgets (that is, those using div, span, and so on.) which do allow focus to make them accessible to screen readers.  <b>Note:</b> Inclusion of aria-labelledby is also dependent on the item plug-in having Standard Form Element set to No and that there is a #LABEL_ID# substitution defined in the item's corresponding label template.
p_item	This is the p_item parameter of your plug-in function.
p_name	This is the value which has been return by apex_plugin.get_input_name_or_page_item
p_default_class	Default CSS class which which should be contained in the result string.
p_add_id	If set to TRUE then the id attribute is also contained in the result string.

### Example

This example emits an INPUT tag of type text which uses apex\_plugin\_util.get\_element\_attributes to automatically include the most common attributes.

```
sys.http.prn (
    '<input type="text" ' ||
    apex_plugin_util.get_element_attributes(p_item, l_name, 'text_field') ||
    'value="' || l_escaped_value || '" ' ||
    'size="' || p_item.element_width || '" ' ||
```



```
'maxlength="' || p_item.element_max_length || '" ' ||  
' />');
```

## GET\_PLSQL\_EXPRESSION\_RESULT Function

This function executes a PL/SQL expression and returns a result. This function also performs the binding of any bind variables in the provided PL/SQL expression. This function is usually used for plug-in attributes of type PL/SQL Expression.

### Syntax

```
APEX_PLUGIN_UTIL.GET_PLSQL_EXPRESSION_RESULT (
    p_plsql_expression IN VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

[Table 21-23](#) describes the parameters available in the GET\_PLSQL\_EXPRESSION\_RESULT function.

**Table 21-23 GET\_PLSQL\_EXPRESSION\_RESULT Parameters**

Parameter	Description
p_plsql_expression_result	A PL/SQL expression that returns a string.

### Return

[Table 21-24](#) describes the return value by the function GET\_PLSQL\_EXPRESSION\_RESULT.

**Table 21-24 GET\_PLSQL\_EXPRESSION\_RESULT Return**

Return	Description
VARCHAR2	String result value returned by the PL/SQL Expression.

### Example

This example executes and returns the result of the PL/SQL expression which is specified in attribute\_03 of an item type plug-in attribute of type "PL/SQL Expression".

```
l_result := apex_plugin_util.get_plsql_expression_result (
    p_plsql_expression => p_item.attribute_03 );
```

## GET\_PLSQL\_FUNCTION\_RESULT Function

This function executes a PL/SQL function block and returns the result. This function also performs binding of bind variables in the provided PL/SQL Function Body. This function is usually used for plug-in attributes of type PL/SQL Function Body.

### Syntax

```
APEX_PLUGIN_UTIL.GET_PLSQL_FUNCTION_RESULT (
    p_plsql_function IN VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

[Table 21–25](#) describes the parameters available in the GET\_PLSQL\_FUNCTION\_RESULT function.

**Table 21–25** GET\_PLSQL\_FUNCTION\_RESULT Parameters

Parameter	Description
p_plsql_function	A PL/SQL function block that returns a result of type string.

### Return

[Table 21–26](#) describes the return value by the function GET\_PLSQL\_FUNCTION\_RESULT.

**Table 21–26** GET\_PLSQL\_FUNCTION\_RESULT Return

Return	Description
VARCHAR2	String result value returned by the PL/SQL function block.

### Example

The following example executes and returns the result of the PL/SQL function body that is specified in attribute\_03 of an item type plug-in attribute of type PL/SQL Function Body.

```
l_result := apex_plugin_util.get_plsql_function_result (
    p_plsql_function => p_item.attribute_03 );
```

## GET\_POSITION\_IN\_LIST Function

This function returns the position in the list where `p_value` is stored. If it is not found, null is returned.

### Syntax

```
APEX_PLUGIN_UTIL.GET_POSITION_IN_LIST(
    p_list IN apex_application_global.vc_arr2,
    p_value IN VARCHAR2)
RETURN NUMBER;
```

### Parameters

[Table 21-27](#) describes the parameters available in the `GET_POSITION_IN_LIST` function.

**Table 21-27** *GET\_POSITION\_IN\_LIST Parameters*

Parameter	Description
<code>p_list</code>	Array of type <code>apex_application_global.vc_arr2</code> that contains entries of type <code>VARCHAR2</code> .
<code>p_value</code>	Value located in the <code>p_list</code> array.

### Return

[Table 21-28](#) describes the return value by the `GET_POISTION_IN_LIST` function.

**Table 21-28** *GET\_POSITION\_IN\_LIST Return*

Return	Description
NUMBER	Returns the position of <code>p_value</code> in the array <code>p_list</code> . If it is not found NULL is returned.

### Example

The following example searches for "New York" in the provided list and returns 2 into `l_position`.

```
declare
    l_list      apex_application_global.vc_arr2;
    l_position number;
begin
    l_list(1) := 'Rome';
    l_list(2) := 'New York';
    l_list(3) := 'Vienna';

    l_position := apex_plugin_util.get_position_in_list (
        p_list => l_list,
        p_value => 'New York' );
end;
```

## GET\_SEARCH\_STRING Function

Based on the provided value in `p_search_type` the passed in value of `p_search_string` is returned unchanged or is converted to uppercase. Use this function with the `p_search_string` parameter of `get_data` and `get_data2`.

### Syntax

```
APEX_PLUGIN_UTIL.GET_SEARCH_STRING(
    p_search_type IN VARCHAR2,
    p_search_string IN VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

[Table 21–29](#) describes the parameters available in the `GET_SEARCH_STRING` function.

**Table 21–29** *GET\_SEARCH\_STRING Parameters*

Parameter	Description
<code>p_search_type</code>	Type of search when used with <code>get_data</code> and <code>get_data2</code> . Use one of the <code>c_search_*</code> constants.
<code>p_search_string</code>	Search string used for the search with <code>get_data</code> and <code>get_data2</code> .

### Return

[Table 21–30](#) describes the return value by the function `GET_SEARCH_STRING`.

**Table 21–30** *GET\_SEARCH\_STRING Return*

Return	Description
<code>VARCHAR2</code>	Returns <code>p_search_string</code> unchanged or in uppercase if <code>p_search_type</code> is of type <code>c_search_contains_ignore</code> or <code>c_search_exact_ignore</code> .

### Example

This example uses a call to `get_data` or `get_data2` to make sure the search string is using the correct case.

```
l_column_value_list :=
    apex_plugin_util.get_data (
        p_sql_statement => p_item.lov_definition,
        p_min_columns   => 2,
        p_max_columns   => 2,
        p_component_name => p_item.name,
        p_search_type    => apex_plugin_util.c_search_contains_ignore,
        p_search_column_no => 1,
        p_search_string  => apex_plugin_util.get_search_string (
            p_search_type => apex_plugin_util.c_search_contains_ignore,
            p_search_string => p_value ) );
```

---

## IS\_EQUAL Function

This function returns `TRUE` if both values are equal and `FALSE` if not. If both values are `NULL`, `TRUE` is returned.

### Syntax

```
APEX_PLUGIN_UTIL.IS_EQUAL (  
    p_value1 IN VARCHAR2  
    p_value2 IN VARCHAR2)  
RETURN BOOLEAN;
```

### Parameters

[Table 21–31](#) describes the parameters available in the `IS_EQUAL` function.

**Table 21–31** *IS\_EQUAL Parameters*

Parameter	Description
<code>p_value1</code>	First value to compare.
<code>p_value2</code>	Second value to compare.

### Return

[Table 21–32](#) describes the return value by the function `IS_EQUAL`.

**Table 21–32** *IS\_EQUAL Return*

Return	Description
<code>BOOLEAN</code>	Returns <code>TRUE</code> if both values are equal or both values are <code>NULL</code> , otherwise it returns <code>FALSE</code> .

### Example

In the following example, if the value in the database is different from what is entered, the code in the `if` statement is executed.

```
if NOT apex_plugin_util.is_equal(l_database_value, l_current_value) then  
    -- value has changed, do something  
    null;  
end if;
```

## PAGE\_ITEM\_NAMES\_TO\_JQUERY Function

This function returns a jQuery selector based on a comma delimited string of page item names. For example, you could use this function for a plug-in attribute called "Page Items to Submit" where the JavaScript code has to read the values of the specified page items.

### Syntax

```
APEX_PLUGIN_UTIL.PAGE_ITEM_NAMES_TO_JQUERY (
    p_page_item_names IN VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

[Table 21–33](#) describes the parameters available in the PAGE\_ITEM\_NAMES\_TO\_JQUERY function.

**Table 21–33** PAGE\_ITEM\_NAMES\_TO\_JQUERY Parameters

Parameter	Description
p_page_item_names	Comma delimited list of page item names.

### Return

[Table 21–32](#) describes the return value by the PAGE\_ITEM\_NAMES\_TO\_JQUERY function.

**Table 21–34** PAGE\_ITEM\_NAMES\_TO\_JQUERY Return

Return	Description
VARCHAR2	Transforms the page items specified in p_page_item_names into a jQuery selector.

### Example

The following example shows the code to construct the initialization call for a JavaScript function called myOwnWidget. This function gets an object with several attributes where one attribute is pageItemsToSubmit which is expected to be a jQuery selector.

```
apex_javascript.add_onload_code (
    p_code => 'myOwnWidget('||
        '#'||p_item.name||','||
        '{'||
            apex_javascript.add_attribute('ajaxIdentifier', apex_
plugin.get_ajax_identifier)||
            apex_javascript.add_attribute('dependingOnSelector', apex_
plugin_util.page_item_names_to_jquery(p_item.lov_cascade_parent_items)||
            apex_javascript.add_attribute('optimizeRefresh', p_
item.ajax_optimize_refresh)||
            apex_javascript.add_attribute('pageItemsToSubmit', apex_
plugin_util.page_item_names_to_jquery(p_item.ajax_items_to_submit)||
            apex_javascript.add_attribute('nullValue', p_item.lov_
null_value, false, false)||
        '});' );
```

---

## PRINT\_DISPLAY\_ONLY Procedure

This procedure outputs a SPAN tag for a display only field.

### Syntax

```
APEX_PLUGIN_UTIL.PRINT_DISPLAY_ONLY (
  p_item_name      IN VARCHAR2,
  p_display_value  IN VARCHAR2,
  p_show_line_breaks IN BOOLEAN,
  p_attributes     IN VARCHAR2,
  p_id_postfix    IN VARCHAR2 DEFAULT '_DISPLAY');
```

### Parameters

[Table 21–35](#) describes the parameters available in the PRINT\_DISPLAY\_ONLY procedure.

**Table 21–35** PRINT\_DISPLAY\_ONLY Parameter

Parameter	Description
p_item_name	Name of the page item. This parameter should be called with p_item.name.
p_display_value	Text to be displayed.
p_show_line_breaks	If set to TRUE line breaks in p_display_value are changed to   so that the browser renders them as line breaks.
p_attributes	Additional attributes added to the SPAN tag.
p_id_postfix	Postfix which is getting added to the value in p_item_name to get the ID for the SPAN tag. Default is _DISPLAY.

### Example

The following code could be used in an item type plug-in to render a display only page item.

```
apex_plugin_util.print_display_only (
  p_item_name      => p_item.name,
  p_display_value  => p_value,
  p_show_line_breaks => false,
  p_escape        => true,
  p_attributes     => p_item.element_attributes );
```



---

## PRINT\_ESCAPED\_VALUE Procedure

This procedure outputs the value in an escaped form and chunks big strings into smaller outputs.

### Syntax

```
APEX_PLUGIN_UTIL.PRINT_ESCAPED_VALUE (  
    p_value IN VARCHAR2);
```

### Parameters

Table 21-36 describes the parameters available in the PRINT\_ESCAPED\_VALUE procedure.

**Table 21-36 PRINT\_ESCAPED\_VALUE Parameter**

Parameter	Description
p_value	Text which should be escaped and then printed to the HTTP buffer.

### Example

Prints a hidden field with the current value of the page item.

```
sys.http.prn('<input type="hidden" name="'||l_name||'" id="'||p_item_name||'"  
value="');  
print_escaped_value(p_value);  
sys.http.prn('>');
```

## PRINT\_HIDDEN\_IF\_READONLY Procedure

This procedure outputs a hidden field to store the page item value if the page item is rendered as readonly and is not printer friendly. If this procedure is called in an item type plug-in, the parameters of the plug-in interface should directly be passed in.

### Syntax

```
APEX_PLUGIN_UTIL.PRINT_HIDDEN_IF_READ_ONLY (
    p_item_name    IN VARCHAR2,
    p_value        IN VARCHAR2,
    p_is_readonly  IN BOOLEAN,
    p_is_printer_friendly IN BOOLEAN,
    p_id_postfix   IN VARCHAR2 DEFAULT NULL);
```

### Parameters

Table 21–37 describes the parameters available in the PRINT\_HIDDEN\_IF\_READONLY procedure.

**Table 21–37 PRINT\_HIDDEN\_IF\_READONLY Parameters**

Parameter	Description
p_item_name	Name of the page item. For this parameter the p_item.name should be passed in.
p_value	Current value of the page item. For this parameter p_value should be passed in.
p_is_readonly	Is the item rendered readonly. For this parameter p_is_readonly should be passed in.
p_is_printer_friendly	Is the item rendered in printer friendly mode. For this parameter p_is_printer_friendly should be passed in.
p_id_postfix	Used to generate the ID attribute of the hidden field. It is build based on p_item_name and the value in p_id_postfix.

### Example

Writes a hidden field with the current value to the HTTP output if p\_is\_readonly is TRUE and p\_printer\_friendly is FALSE.

```
apex_plugin_util.print_hidden_if_readonly (
    p_item_name    => p_item.name,
    p_value        => p_value,
    p_is_readonly  => p_is_readonly,
    p_is_printer_friendly => p_is_printer_friendly );
```

---

## PRINT\_JSON\_HTTP\_HEADER Procedure

This procedure outputs a standard HTTP header for a JSON output.

### Syntax

```
APEX_PLUGIN_UTIL.PRINT_JSON_HTTP_HEADER;
```

### Parameters

None.

### Example

This example shows how to use this procedure in the Ajax callback function of a plugin. This code outputs a JSON structure in the following format: [{"d": "Display 1", "r": "Return 1"}, {"d": "Display 2", "r": "Return 2"}]

```
-- Write header for the JSON stream.
apex_plugin_util.print_json_http_header;
-- initialize the JSON structure
sys.htp.p('');
-- loop through the value array
for i in 1 .. l_values.count
loop
    -- add array entry
    sys.htp.p (
        case when i > 1 then ',' end||
        '{'||
        apex_javascript.add_attribute('d', sys.htf.escape_sc(l_values(i).display_
value), false, true)||
        apex_javascript.add_attribute('r', sys.htf.escape_sc(l_values(i).return_
value), false, false)||
        '}' );
end loop;
-- close the JSON structure
sys.htp.p('');
```

## PRINT\_LOV\_AS\_JSON Procedure

This procedure outputs a JSON response based on the result of a two column LOV in the format:

```
[{"d":"display","r":"return"}, {"d":.....,"r":.....},.....]
```

---

**Note:** The HTTP header is initialized with MIME type "application/json" as well.

---

### Syntax

```
APEX_PLUGIN_UTIL.PRINT_LOV_AS_JSON (
  p_sql_statement      IN VARCHAR2,
  p_component_name    IN VARCHAR2,
  p_escape             IN BOOLEAN,
  p_replace_substitutions IN BOOLEAN DEFAULT FALSE);
```

### Parameters

[Table 21–38](#) describes the parameters available in the PRINT\_LOV\_AS\_JSON procedure.

**Table 21–38** PRINT\_LOV\_AS\_JSON Parameters

Parameter	Description
p_sql_statement	A SQL statement which returns two columns from the SELECT.
p_component_name	The name of the page item or report column that is used in case an error is displayed.
p_escape	If set to TRUE the value of the display column is escaped, otherwise it is output as is.
p_replace_substitutions	If set to TRUE, apex_plugin_util.replace_substitutions is called for the value of the display column, otherwise, it is output as is.

### Example

This example shows how to use the procedure in an Ajax callback function of an item type plug-in. The following call writes the LOV result as a JSON array to the HTTP output.

```
apex_plugin_util.print_lov_as_json (
  p_sql_statement => p_item.lov_definition,
  p_component_name => p_item.name,
  p_escape        => true );
```

---

## PRINT\_OPTION Procedure

This procedure outputs an OPTION tag.

### Syntax

```
APEX_PLUGIN_UTIL.PRINT_OPTION (
  p_display_value      IN VARCHAR2,
  p_return_value       IN VARCHAR2,
  p_is_selected        IN BOOLEAN,
  p_attributes         IN VARCHAR2,
  p_escape             IN BOOLEAN DEFAULT TRUE);
```

### Parameters

Table 21–39 describes the parameters available in the PRINT\_OPTION procedure.

**Table 21–39 PRINT\_OPTION Parameters**

Parameter	Description
p_display_value	Text which is displayed by the option.
p_return_value	Value which is set when the option is picked.
p_is_selected	Set to TRUE if the selected attribute should be set for this option.
p_attributes	Additional HTML attributes which should be set for the OPTION tag.
p_escape	Set to TRUE if special characters in p_display_value should be escaped.

### Example

The following example could be used in an item type plug-in to create a SELECT list. Use apex\_plugin\_util.is\_equal to find out which list entry should be marked as current.

```
sys.htp.p('<select id="'||p_item.name||'" size="'||nvl(p_item.element_height,
5)||'" '||coalesce(p_item.element_attributes, 'class="new_select_list"')||'>');
-- loop through the result and add list entries
for i in 1 .. l_values.count
loop
  apex_plugin_util.print_option (
    p_display_value => l_values(i).display_value,
    p_return_value  => l_values(i).return_value,
    p_is_selected   => apex_plugin_util.is_equal(l_values(i).return_value, p_
value),
    p_attributes    => p_item.element_option_attributes,
    p_escape        => true );
end loop;
sys.htp.p('</select>');
```

## REPLACE\_SUBSTITUTIONS Function

This function replaces any `&ITEM.` substitution references with their actual value. If `p_escape` is set to `TRUE`, any special characters contained in the value of the referenced item are escaped to prevent Cross-site scripting (XSS) attacks.

### Syntax

```
apex_plugin_util.replace_substitutions (
    p_value      in varchar2,
    p_escape     in boolean default true )
return varchar2;
```

### Parameters

[Table 21–40](#) describes the parameters available in the `REPLACE_SUBSTITUTION` function.

**Table 21–40** *REPLACE\_SUBSTITUTION Parameters*

Parameter	Description
<code>p_value</code>	This value is a string which can contain several <code>&amp;ITEM.</code> references which are replaced by their actual page item values.
<code>p_escape</code>	If set to <code>TRUE</code> any special characters contained in the value of the referenced item are escaped to prevent Cross-site scripting (XSS) attacks. If set to <code>FALSE</code> , the referenced items are not escaped.

### Example

The following example replaces any substitution syntax references in the region plug-in attribute `05` with their actual values. Any special characters in the values are escaped.

```
l_advanced_formatting := apex_plugin_util.replace_substitutions (
    p_value => p_region.attribute_05,
    p_escape => true );
```

---

---

## APEX\_REGION

The APEX\_REGION package is the public API for handling regions.

- [IS\\_READ\\_ONLY](#) Function
- [PURGE\\_CACHE](#) Procedure

## IS\_READ\_ONLY Function

This function returns TRUE if the current region is rendered read-only and FALSE if region is not rendered read-only. If the function is called from a context where no region is currently processed, it returns NULL. For example, you can use this function in conditions of a region or its underlying items and buttons.

### Syntax

```
FUNCTION IS_READ_ONLY  
RETURN BOOLEAN;
```

### Parameters

None.

### Example

This examples purges the session for a specific region cache for the whole application.

```
RETURN APEX_REGION.IS_READ_ONLY;
```



---

## PURGE\_CACHE Procedure

This procedure purges the region cache of the specified application, page, and region.

### Syntax

```
PROCEDURE PURGE_CACHE (
    p_application_id      IN NUMBER DEFAULT wv_flow.g_flow_id,
    p_page_id            IN NUMBER DEFAULT NULL,
    p_region_id          IN NUMBER DEFAULT NULL,
    p_current_session_only IN BOOLEAN DEFAULT FALSE );
```

### Parameters

**Table 22–1** *PURGE\_CACHE Parameters*

Parameter	Description
p_application_id	ID of the application where the region caches should be purged. Defaults to the current application.
p_page_id	ID of the page where the region caches should be purged. If no value is specified (which is the default), all regions of the application are purged.
p_region_id	ID of a specific region on a page. If no value is specified, all regions of the specified page are purged.
p_current_session_only	Specify true if you only want to purge entries that were saved for the current session. defaults to false.

### Example

This example purges session specific region cache for the whole application.

```
BEGIN
    APEX_REGION.PURGE_CACHE (
        p_current_session_only => true );
END;
```



This package enables you to use Oracle Locator and the Spatial Option within Application Express. In an Application Express context, the logon user of the database session is typically `APEX_PUBLIC_USER` or `ANONYMOUS`. Spatial developers can not directly use DML on `USER_SDO_GEOM_METADATA` within such a session, for example, in SQL Commands within SQL Workshop. The Spatial view's trigger performs DML as the logon user, but it has to run as the application owner or workspace user. With the `APEX_SPATIAL` API, developers can use the procedures and functions below to insert, update and delete rows of `USER_SDO_GEOM_METADATA` as the current Application Express user. The package also provides a few utilities that simplify the use of Spatial in Application Express.

- [Data Types](#)
- [CHANGE\\_GEOM\\_METADATA Procedure](#)
- [CIRCLE\\_POLYGON Function](#)
- [DELETE\\_GEOM\\_METADATA Procedure](#)
- [INSERT\\_GEOM\\_METADATA Procedure](#)
- [INSERT\\_GEOM\\_METADATA\\_LONLAT Procedure](#)
- [POINT Function](#)
- [RECTANGLE Function](#)

## Data Types

The data types used by this package are described in this section.

### **t\_srid**

```
subtype t_srid is number;
```

### **c\_no\_reference\_system**

```
c_no_reference_system constant t_srid := null;
```

### **c\_wgs\_84**

```
c_wgs_84 constant t_srid := 4326; -- World Geodetic System, EPSG:4326
```

## CHANGE\_GEOM\_METADATA Procedure

This procedure modifies a spatial metadata record.

### Syntax

```
APEX_SPATIAL.CHANGE_GEOM_METADATA (
  p_table_name      IN VARCHAR2,
  p_column_name     IN VARCHAR2,
  p_new_table_name  IN VARCHAR2 DEFAULT NULL,
  p_new_column_name IN VARCHAR2 DEFAULT NULL,
  p_diminfo         IN mdsys.sdo_dim_array,
  p_srid            IN t_srid );
```

### Parameters

**Table 23–1** CHANGE\_GEOM\_METADATA Parameters

Parameter	Description
p_table_name	Name of the feature table.
p_column_name	Name of the column of type mdsys.sdo_geometry.
p_new_table_name	New name of a feature table (or null, to keep the current value).
p_new_column_name	New name of the column of type mdsys.sdo_geometry (or null, to keep the current value).
p_diminfo	SDO_DIM_ELEMENT array, ordered by dimension, with one entry for each dimension.
p_srid	SRID value for the coordinate system for all geometries in the column.

### Example

The code below modifies the dimensions of column CITIES.SHAPE.

```
begin
  for l_meta in ( select *
                  from user_sdo_geom_metadata
                  where table_name = 'CITIES'
                    and column_name = 'SHAPE' )
  loop
    apex_spatial.change_geom_metadata (
      p_table_name => l_meta.table_name,
      p_column_name => l_meta.column_name,
      p_diminfo    => SDO_DIM_ARRAY (
                    SDO_DIM_ELEMENT('X', -180, 180, 0.1),
                    SDO_DIM_ELEMENT('Y', -90, 90, 0.1) ),
      p_srid       => l_meta.srid );
  end loop;
end;
```

## CIRCLE\_POLYGON Function

This function creates a polygon that approximates a circle at (p\_lon, p\_lat) with radius of p\_radius. See `mdsys.sdo_util.circle_polygon` for details.

### Syntax

```
APEX_SPATIAL.CIRCLE_POLYGON (
  p_lon      IN NUMBER,
  p_lat      IN NUMBER,
  p_radius   IN NUMBER,
  p_arc_tolerance IN NUMBER DEFAULT 20,
  p_srid     IN t_srid DEFAULT c_wgs_84 )
RETURN mdsys.sdo_geometry;
```

### Parameters

**Table 23–2 CIRCLE\_POLYGON Parameters**

Parameter	Description
p_lon	Longitude position of the lower left point.
p_lat	Latitude position of the lower left point.
p_radius	Radius of the circle in meters.
p_arc_tolerance	Arc tolerance (default 20).
p_srid	Reference system (default c_wgs_84).

### Returns

**Table 23–3 CIRCLE\_POLYGON Function Returns**

Return	Description
<code>mdsys.sdo_geometry</code>	The geometry for the polygon that approximates the circle.

### Example

This example is a query that returns a polygon that approximates a circle at (0, 0) with radius 1.

```
select apex_spatial.circle_polygon(0, 0, 1) from dual
```

---

## DELETE\_GEOM\_METADATA Procedure

This procedure deletes a spatial metadata record.

### Syntax

```
APEX_SPATIAL.DELETE_GEOM_METADATA (  
    p_table_name      IN VARCHAR2,  
    p_column_name     IN VARCHAR2,  
    p_drop_index      IN BOOLEAN DEFAULT FALSE );
```

### Parameters

**Table 23–4** *DELETE\_GEOM\_METADATA Parameters*

Parameter	Description
p_table_name	Name of the feature table.
p_column_name	Name of the column of type <code>mdsys.sdo_geometry</code> .
p_drop_index	If TRUE (default is FALSE), drop the spatial index on the column.

### Example

This example deletes metadata on column `CITIES.SHAPE` and drops the spatial index on this column.

```
begin  
    apex_spatial.delete_geom_metadata (  
        p_table_name => 'CITIES',  
        p_column_name => 'SHAPE',  
        p_drop_index => true );  
end;
```

## INSERT\_GEOM\_METADATA Procedure

This procedure inserts a spatial metadata record and optionally creates a spatial index.

### Syntax

```
APEX_SPATIAL.INSERT_GEOM_METADATA (
  p_table_name      IN VARCHAR2,
  p_column_name     IN VARCHAR2,
  p_diminfo         in mdsys.sdo_dim_array,
  p_srid            in t_srid,
  p_create_index_name IN VARCHAR2 DEFAULT NULL );
```

### Parameters

**Table 23–5** INSERT\_GEOM\_METADATA\_LONLAT Parameters

Parameter	Description
p_table_name	The name of the feature table.
p_column_name	The name of the column of type mdsys.sdo_geometry.
p_diminfo	The SDO_DIM_ELEMENT array, ordered by dimension, with one entry for each dimension.
p_srid	The SRID value for the coordinate system for all geometries in the column.
p_create_index_name	If not null, a spatial index on the column is created with this name. Only simple column names are supported, function based indexes or indexes on object attributes cause an error. For more complex requirements, leave this parameter null (the default) and manually create the index.

### Example

This example creates table CITIES, spatial metadata and an index on column CITIES.SHAPE.

```
create table cities (
  city_id  number primary key,
  city_name varchar2(30),
  shape    mdsys.sdo_geometry )
/
begin
  apex_spatial.insert_geom_metadata (
    p_table_name => 'CITIES',
    p_column_name => 'SHAPE',
    p_diminfo    => SDO_DIM_ARRAY (
      SDO_DIM_ELEMENT('X', -180, 180, 1),
      SDO_DIM_ELEMENT('Y', -90, 90, 1) ),
    p_srid       => apex_spatial.c_wgs_84 );
end;
/
create index cities_idx_shape on cities(shape) indextype is mdsys.spatial_index
/
```



---

## INSERT\_GEOM\_METADATA\_LONLAT Procedure

This procedure inserts a spatial metadata record that is suitable for longitude/latitude and optionally creates a spatial index.

### Syntax

```
APEX_SPATIAL.INSERT_GEOM_METADATA_LONLAT (
  p_table_name      IN VARCHAR2,
  p_column_name     IN VARCHAR2,
  p_tolerance       IN NUMBER DEFAULT 1,
  p_create_index_name IN VARCHAR2 DEFAULT NULL );
```

### Parameters

**Table 23–6** *INSERT\_GEOM\_METADATA\_LONLAT Parameters*

Parameter	Description
p_table_name	Name of the feature table.
p_column_name	Name of the column of type <code>mdsys.sdo_geometry</code> .
p_tolerance	Tolerance value in each dimension, in meters (default 1).
p_create_index_name	if not null, a spatial index on the column is created with this name. Only simple column names are supported, function based indexes or indexes on object attributes cause an error. For more complex requirements, leave this parameter null (the default) and manually create the index.

### Example

The code below creates table `CITIES` and spatial metadata for the column `CITIES.SHAPE`. By passing `CITIES_IDX_SHAPE` to `p_create_index_name`, the API call automatically creates an index on the spatial column.

```
create table cities (
  city_id  number primary key,
  city_name varchar2(30),
  shape    mdsys.sdo_geometry )
/
begin
  apex_spatial.insert_geom_metadata_lonlat (
    p_table_name      => 'CITIES',
    p_column_name     => 'SHAPE',
    p_create_index_name => 'CITIES_IDX_SHAPE' );
end;
/
```

## POINT Function

This function creates a point at (p\_lon, p\_lat).

### Syntax

```
APEX_SPATIAL.POINT (
  p_lon      IN NUMBER,
  p_lat      IN NUMBER,
  p_srid     IN t_srid DEFAULT c_wgs_84 )
RETURN mdsys.sdo_geometry;
```

### Parameters

**Table 23–7** POINT parameters

Parameter	Description
p_lon	Longitude position.
p_lat	Latitude position.
p_srid	Reference system (default c_wgs_84).

### Returns

**Table 23–8** POINT Function Returns

Return	Description
mdsys.sdo_geometry	The geometry for the point.

### Example

This example is a query that returns a point at (10, 50).

```
select apex_spatial.point(10, 50) from dual;
```

This example is equivalent to:

```
select mdsys.sdo_geometry(2001, 4326, sdo_point_type(10, 50, null), null, null)
from dual;
```

## RECTANGLE Function

This function creates a rectangle from point at (p\_lon1, p\_lat1) to (p\_lon2, p\_lat2).

### Syntax

```
APEX_SPATIAL.RECTANGLE (
  p_lon1      IN NUMBER,
  p_lat1      IN NUMBER,
  p_lon2      IN NUMBER,
  p_lat2      IN NUMBER,
  p_srid      IN t_srid DEFAULT c_wgs_84 )
RETURN mdsys.sdo_geometry;
```

### Parameters

**Table 23–9** *RECTANGLE Parameters*

Parameter	Description
p_lon1	Longitude position of the lower left point.
p_lat1	Latitude position of the lower left point.
p_lon2	Longitude position of the upper right point.
p_lat2	Latitude position of the upper right point.
p_srid	Reference system (default c_wgs_84).

### Returns

**Table 23–10** *RECTANGLE Function Returns*

Return	Description
mdsys.sdo_geometry	The geometry for the rectangle (p_lon1, p_lon2, p_lat1, p_lat2).

### Example

This example is a query that returns a rectangle from (10, 50) to (11, 51).

```
select apex_spatial.rectangle(10, 50, 11, 51) from dual
```

This example is equivalent to:

```
select mdsys.sdo_geometry(
  2003, 4326, null,
  sdo_elem_info_array(1, 1003, 1),
  sdo_ordinate_array(10, 50, 11, 50, 11, 51, 10, 51, 10, 50))
from dual;
```



---

---

## APEX\_UI\_DEFAULT\_UPDATE

The APEX\_UI\_DEFAULT\_UPDATE package provides procedures to access user interface defaults from within SQL Developer or SQL\*Plus.

You can use this package to set the user interface defaults associated with a table within a schema. The package must be called from within the schema that owns the table you are updating.

User interface defaults enable you to assign default user interface properties to a table, column, or view within a specified schema. When you create a form or report using a wizard, the wizard uses this information to create default values for region and item properties. Utilizing user interface defaults can save valuable development time and has the added benefit of providing consistency across multiple pages in an application.

- [ADD\\_AD\\_COLUMN Procedure](#)
- [ADD\\_AD\\_SYNONYM Procedure](#)
- [DEL\\_AD\\_COLUMN Procedure](#)
- [DEL\\_AD\\_SYNONYM Procedure](#)
- [DEL\\_COLUMN Procedure](#)
- [DEL\\_GROUP Procedure](#)
- [DEL\\_TABLE Procedure](#)
- [SYNCH\\_TABLE Procedure](#)
- [UPD\\_AD\\_COLUMN Procedure](#)
- [UPD\\_AD\\_SYNONYM Procedure](#)
- [UPD\\_COLUMN Procedure](#)
- [UPD\\_DISPLAY\\_IN\\_FORM Procedure](#)
- [UPD\\_DISPLAY\\_IN\\_REPORT Procedure](#)
- [UPD\\_FORM\\_REGION\\_TITLE Procedure](#)
- [UPD\\_GROUP Procedure](#)
- [UPD\\_ITEM\\_DISPLAY\\_HEIGHT Procedure](#)
- [UPD\\_ITEM\\_DISPLAY\\_WIDTH Procedure](#)
- [UPD\\_ITEM\\_FORMAT\\_MASK Procedure](#)
- [UPD\\_ITEM\\_HELP Procedure](#)
- [UPD\\_LABEL Procedure](#)

- 
- [UPD\\_REPORT\\_ALIGNMENT Procedure](#)
  - [UPD\\_REPORT\\_FORMAT\\_MASK Procedure](#)
  - [UPD\\_REPORT\\_REGION\\_TITLE Procedure](#)
  - [UPD\\_TABLE Procedure](#)

**See Also:** "Managing User Interface Defaults" in *Oracle Application Express SQL Workshop Guide*

## ADD\_AD\_COLUMN Procedure

Adds a User Interface Default Attribute Dictionary entry with the provided definition. Up to three synonyms can be provided during the creation. Additional synonyms can be added post-creation using `apex_ui_default_update.add_ad_synonym`. Synonyms share the column definition of their base column.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.ADD_AD_COLUMN (
    p_column_name          IN  VARCHAR2,
    p_label                 IN  VARCHAR2  DEFAULT NULL,
    p_help_text            IN  VARCHAR2  DEFAULT NULL,
    p_format_mask          IN  VARCHAR2  DEFAULT NULL,
    p_default_value        IN  VARCHAR2  DEFAULT NULL,
    p_form_format_mask     IN  VARCHAR2  DEFAULT NULL,
    p_form_display_width   IN  VARCHAR2  DEFAULT NULL,
    p_form_display_height  IN  VARCHAR2  DEFAULT NULL,
    p_form_data_type       IN  VARCHAR2  DEFAULT NULL,
    p_report_format_mask   IN  VARCHAR2  DEFAULT NULL,
    p_report_col_alignment IN  VARCHAR2  DEFAULT NULL,
    p_syn_name1            IN  VARCHAR2  DEFAULT NULL,
    p_syn_name2            IN  VARCHAR2  DEFAULT NULL,
    p_syn_name3            IN  VARCHAR2  DEFAULT NULL);
```

### Parameters

Table 24–5 describes the parameters available in the `ADD_AD_COLUMN` procedure.

**Table 24–1 ADD\_AD\_COLUMN Parameters**

Parameter	Description
<code>p_column_name</code>	Name of column to be created.
<code>p_label</code>	Used for item label and report column heading.
<code>p_help_text</code>	Used for help text for items and interactive report columns.
<code>p_format_mask</code>	Used as the format mask for items and report columns. Can be overwritten by report for form specific format masks.
<code>p_default_value</code>	Used as the default value for items.
<code>p_form_format_mask</code>	If provided, used as the format mask for items, overriding any value for the general format mask.
<code>p_form_display_width</code>	Used as the width of any items using this Attribute Definition.
<code>p_form_display_height</code>	Used as the height of any items using this Attribute Definition (only used by item types such as text areas and shuttles).
<code>p_form_data_type</code>	Used as the data type for items (results in an automatic validation). Valid values are VARCHAR, NUMBER and DATE.
<code>p_report_format_mask</code>	If provided, used as the format mask for report columns, overriding any value for the general format mask.
<code>p_report_col_alignment</code>	Used as the alignment for report column data (for example, number are usually right justified). Valid values are LEFT, CENTER, and RIGHT.

**Table 24–1 (Cont.) ADD\_AD\_COLUMN Parameters**

Parameter	Description
p_syn_name1	Name of synonym to be created along with this column. For more than 3, use APEX_UI_DEFAULT_UPDATE.ADD_AD_SYNONYM.
p_syn_name2	Name of second synonym to be created along with this column. For more than 3, use APEX_UI_DEFAULT_UPDATE.ADD_AD_SYNONYM.
p_syn_name3	Name of third synonym to be created along with this column. For more than 3, use APEX_UI_DEFAULT_UPDATE.ADD_AD_SYNONYM.

**Example**

The following example creates a new attribute to the UI Defaults Attribute Dictionary within the workspace associated with the current schema. It also creates a synonym for that attribute.

```
BEGIN
  apex_ui_default_update.add_ad_column (
    p_column_name      => 'CREATED_BY',
    p_label            => 'Created By',
    p_help_text        => 'User that created the record.',
    p_form_display_width => 30,
    p_form_data_type   => 'VARCHAR',
    p_report_col_alignment => 'LEFT',
    p_syn_name1        => 'CREATED_BY_USER' );
END;
```



## ADD\_AD\_SYNONYM Procedure

If the column name is found within the User Interface Default Attribute Dictionary, the synonym provided is created and associated with that column. Synonyms share the column definition of their base column.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.ADD_AD_SYNONYM (
    p_column_name      IN VARCHAR2,
    p_syn_name         IN VARCHAR2);
```

### Parameters

Table 24–2 describes the parameters available in the ADD\_AD\_SYNONYM procedure.

**Table 24–2 ADD\_AD\_SYNONYM Parameters**

Parameter	Description
p_column_name	Name of column with the Attribute Dictionary that the synonym is being created for.
p_syn_name	Name of synonym to be created.

### Example

The following example add the synonym CREATED\_BY\_USER to the CREATED\_BY attribute of the UI Defaults Attribute Dictionary within the workspace associated with the current schema.

```
BEGIN
    apex_ui_default_update.add_ad_synonym (
        p_column_name => 'CREATED_BY',
        p_syn_name     => 'CREATED_BY_USER' );
END;
```

## DEL\_AD\_COLUMN Procedure

If the column name is found within the User Interface Default Attribute Dictionary, the column, along with any associated synonyms, is deleted.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.DEL_AD_COLUMN (  
    p_column_name          IN VARCHAR2);
```

### Parameters

[Table 24–3](#) describes the parameters available in the DEL\_AD\_COLUMN procedure.

**Table 24–3 DEL\_AD\_COLUMN Parameters**

Parameter	Description
p_column_name	Name of column to be deleted

### Example

The following example deletes the attribute CREATED\_BY from the UI Defaults Attribute Dictionary within the workspace associated with the current schema.

```
BEGIN  
    apex_ui_default_update.del_ad_column (  
        p_column_name => 'CREATED_BY' );  
END;
```

---

## DEL\_AD\_SYNONYM Procedure

If the synonym name is found within the User Interface Default Attribute Dictionary, the synonym name is deleted.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.DEL_AD_SYNONYM (  
    p_syn_name          IN VARCHAR2);
```

### Parameters

[Table 24–4](#) describes the parameters available in the DEL\_AD\_SYNONYM procedure.

**Table 24–4 DEL\_AD\_SYNONYM Parameters**

Parameter	Description
p_syn_name	Name of synonym to be deleted

### Example

The following example deletes the synonym CREATED\_BY\_USER from the UI Defaults Attribute Dictionary within the workspace associated with the current schema.

```
BEGIN  
    apex_ui_default_update.del_ad_synonym (  
        p_syn_name => 'CREATED_BY_USER' );  
END;
```

## DEL\_COLUMN Procedure

If the provided table and column exists within the user's schema's table based User Interface Defaults, the UI Defaults for it are deleted.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.DEL_COLUMN (  
    p_table_name          IN VARCHAR2,  
    p_column_name        IN VARCHAR2);
```

### Parameters

[Table 24–5](#) describes the parameters available in the DEL\_COLUMN procedure.

**Table 24–5 DEL\_COLUMN Parameters**

Parameter	Description
p_table_name	Name of table whose column's UI Defaults are to be deleted.
p_column_name	Name of columns whose UI Defaults are to be deleted.

### Example

The following example deletes the column `CREATED_BY` from the `EMP` table definition within the UI Defaults Table Dictionary within the current schema.

```
BEGIN  
    apex_ui_default_update.del_column (  
        p_table_name => 'EMP',  
        p_column_name => 'CREATED_BY' );  
END;
```

---

## DEL\_GROUP Procedure

If the provided table and group exists within the user's schema's table based User Interface Defaults, the UI Defaults for it are deleted and any column within the table that references that group has the group\_id set to null.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.DEL_GROUP (  
    p_table_name          IN VARCHAR2,  
    p_group_name         IN VARCHAR2);
```

### Parameters

Table 24–6 describes the parameters available in the DEL\_GROUP procedure.

**Table 24–6** DEL\_GROUP Parameters

Parameter	Description
p_table_name	Name of table whose group UI Defaults are to be deleted
p_group_name	Name of group whose UI Defaults are to be deleted

### Example

The following example deletes the group AUDIT\_INFO from the EMP table definition within the UI Defaults Table Dictionary within the current schema.

```
BEGIN  
    apex_ui_default_update.del_group (  
        p_table_name => 'EMP',  
        p_group_name => 'AUDIT_INFO' );  
END;
```

## DEL\_TABLE Procedure

If the provided table exists within the user's schema's table based User Interface Defaults, the UI Defaults for it is deleted. This includes the deletion of any groups defined for the table and all the columns associated with the table.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.DEL_TABLE (  
    p_table_name          IN VARCHAR2);
```

### Parameters

[Table 24–7](#) describes the parameters available in the DEL\_TABLE procedure.

**Table 24–7 DEL\_TABLE Parameters**

Parameter	Description
p_table_name	Table name

### Example

The following example removes the UI Defaults for the EMP table that are associated with the current schema.

```
begin  
    apex_ui_default_update.del_table (  
        p_table_name => 'EMP' );  
end;  
/
```

---

## SYNCH\_TABLE Procedure

If the Table Based User Interface Defaults for the table do not already exist within the user's schema, they are defaulted. If they do exist, they are synchronized, meaning, the columns in the table is matched against the column in the UI Defaults Table Definitions. Additions and deletions are used to make them match.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.SYNCH_TABLE (  
    p_table_name          IN VARCHAR2);
```

### Parameters

[Table 24–8](#) describes the parameters available in the SYNCH\_TABLE procedure.

**Table 24–8 SYNCH\_TABLE Parameters**

Parameter	Description
p_table_name	Table name

### Example

The following example synchronizes the UI Defaults for the EMP table that are associated with the current schema.

```
BEGIN  
    apex_ui_default_update.synch_table (  
        p_table_name => 'EMP' );  
END;
```

## UPD\_AD\_COLUMN Procedure

If the column name is found within the User Interface Default Attribute Dictionary, the column entry is updated using the provided parameters. If 'null%' is passed in, the value of the associated parameter is set to null.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_AD_COLUMN (
  p_column_name          IN  VARCHAR2,
  p_new_column_name      IN  VARCHAR2  DEFAULT NULL,
  p_label                IN  VARCHAR2  DEFAULT NULL,
  p_help_text            IN  VARCHAR2  DEFAULT NULL,
  p_format_mask          IN  VARCHAR2  DEFAULT NULL,
  p_default_value        IN  VARCHAR2  DEFAULT NULL,
  p_form_format_mask     IN  VARCHAR2  DEFAULT NULL,
  p_form_display_width   IN  VARCHAR2  DEFAULT NULL,
  p_form_display_height  IN  VARCHAR2  DEFAULT NULL,
  p_form_data_type       IN  VARCHAR2  DEFAULT NULL,
  p_report_format_mask   IN  VARCHAR2  DEFAULT NULL,
  p_report_col_alignment IN  VARCHAR2  DEFAULT NULL);
```

### Parameters

Table 24–9 describes the parameters available in the UPD\_AD\_COLUMN procedure.

**Table 24–9 UPD\_AD\_COLUMN Parameters**

Parameter	Description
p_column_name	Name of column to be updated
p_new_column_name	New name for column, if column is being renamed
p_label	Used for item label and report column heading
p_help_text	Used for help text for items and interactive report columns
p_format_mask	Used as the format mask for items and report columns. Can be overwritten by report for form specific format masks.
p_default_value	Used as the default value for items.
p_form_format_mask	If provided, used as the format mask for items, overriding any value for the general format mask.
p_form_display_width	Used as the width of any items using this Attribute Definition.
p_form_display_height	Used as the height of any items using this Attribute Definition (only used by item types such as text areas and shuttles).
p_form_data_type	Used as the data type for items (results in an automatic validation). Valid values are VARCHAR, NUMBER and DATE.
p_report_format_mask	If provided, used as the format mask for report columns, overriding any value for the general format mask.
p_report_col_alignment	Used as the alignment for report column data (for example, number are usually right justified). Valid values are LEFT, CENTER, and RIGHT.



---

---

**Note:** If `p_label` through `p_report_col_alignment` are set to 'null%', the value is nullified. If no value is passed in, that column is not updated.

---

---

**Example**

The following example updates the `CREATED_BY` column in the UI Defaults Attribute Dictionary within the workspace associated with the current schema, setting the `form_format_mask` to null.

```
BEGIN
  apex_ui_default_update.upd_ad_column (
    p_column_name      => 'CREATED_BY',
    p_form_format_mask => 'null%');
END;
```

## UPD\_AD\_SYNONYM Procedure

If the synonym name is found within the User Interface Default Attribute Dictionary, the synonym name is updated.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_AD_SYNONYM (  
    p_syn_name          IN VARCHAR2,  
    p_new_syn_name      IN VARCHAR2 DEFAULT NULL);
```

### Parameters

[Table 24–10](#) describes the parameters available in the UPD\_AD\_SYNONYM procedure.

**Table 24–10** UPD\_AD\_SYNONYM Parameters

Parameter	Description
p_syn_name	Name of synonym to be updated
p_new_syn_name	New name for synonym

### Example

The following example updates the CREATED\_BY\_USER synonym in the UI Defaults Attribute Dictionary within the workspace associated with the current schema.

```
BEGIN  
    apex_ui_default_update.upd_ad_synonym (  
        p_syn_name      => 'CREATED_BY_USER',  
        p_new_syn_name => 'USER_CREATED_BY');  
END;
```

## UPD\_COLUMN Procedure

If the provided table and column exists within the user's schema's table based User Interface Defaults, the provided parameters are updated. If 'null%' is passed in, the value of the associated parameter is set to null.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_COLUMN (
  p_table_name          IN VARCHAR2,
  p_column_name         IN VARCHAR2,
  p_group_id            IN VARCHAR2  DEFAULT NULL,
  p_label               IN VARCHAR2  DEFAULT NULL,
  p_help_text           IN VARCHAR2  DEFAULT NULL,
  p_display_in_form     IN VARCHAR2  DEFAULT NULL,
  p_display_seq_form    IN VARCHAR2  DEFAULT NULL,
  p_mask_form           IN VARCHAR2  DEFAULT NULL,
  p_default_value       IN VARCHAR2  DEFAULT NULL,
  p_required            IN VARCHAR2  DEFAULT NULL,
  p_display_width       IN VARCHAR2  DEFAULT NULL,
  p_max_width           IN VARCHAR2  DEFAULT NULL,
  p_height              IN VARCHAR2  DEFAULT NULL,
  p_display_in_report   IN VARCHAR2  DEFAULT NULL,
  p_display_seq_report  IN VARCHAR2  DEFAULT NULL,
  p_mask_report         IN VARCHAR2  DEFAULT NULL,
  p_alignment           IN VARCHAR2  DEFAULT NULL);
```

### Parameters

[Table 24–11](#) describes the parameters available in the UPD\_COLUMN procedure.

**Table 24–11** UPD\_COLUMN Parameters

Parameter	Description
p_table_name	Name of table whose column's UI Defaults are being updated
p_column_name	Name of column whose UI Defaults are being updated
p_group_id	id of group to be associated with the column
p_label	When creating a form against this table or view, this is used as the label for the item if this column is included. When creating a report or tabular form, this is used as the column heading if this column is included.
p_help_text	When creating a form against this table or view, this becomes the help text for the resulting item.
p_display_in_form	When creating a form against this table or view, this determines whether this column is displayed in the resulting form page. Valid values are Y and N.
p_display_seq_form	When creating a form against this table or view, this determines the sequence in which the columns is displayed in the resulting form page.
p_mask_form	When creating a form against this table or view, this specifies the mask that is applied to the item, such as 999-99-9999. This is not used for character based items.
p_default_value	When creating a form against this table or view, this specifies the default value for the item resulting from this column.

**Table 24–11 (Cont.) UPD\_COLUMN Parameters**

Parameter	Description
p_required	When creating a form against this table or view, this specifies to generate a validation in which the resulting item must be NOT NULL. Valid values are Y and N.
p_display_width	When creating a form against this table or view, this specifies the display width of the item resulting from this column.
p_max_width	When creating a form against this table or view, this specifies the maximum string length that a user is allowed to enter in the item resulting from this column.
p_height	When creating a form against this table or view, this specifies the display height of the item resulting from this column.
p_display_in_report	When creating a report against this table or view, this determines whether this column is displayed in the resulting report. Valid values are Y and N.
p_display_seq_report	When creating a report against this table or view, this determines the sequence in which the columns are displayed in the resulting report.
p_mask_report	When creating a report against this table or view, this specifies the mask that is applied against the data, such as 999-99-9999. This is not used for character based items.
p_alignment	When creating a report against this table or view, this determines the alignment for the resulting report column. Valid values are L for Left, C for Center, and R for Right.

---



---

**Note:** If p\_group\_id through p\_alignment are set to 'null%', the value is nullified. If no value is passed in, that column is not updated.

---



---

**Example**

The following example updates the column DEPT\_NO within the EMP table definition within the UI Defaults Table Dictionary within the current schema, setting the group\_id to null.

```
BEGIN
  apex_ui_default_update.upd_column (
    p_table_name => 'EMP',
    p_column_name => 'DEPT_NO',
    p_group_id   => 'null%' );
END;
```

---

## UPD\_DISPLAY\_IN\_FORM Procedure

The UPD\_DISPLAY\_IN\_FORM procedure sets the display in form user interface defaults. This user interface default is used by wizards when you select to create a form based upon the table. It controls whether the column is included by default or not.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_DISPLAY_IN_FORM (
  p_table_name          IN VARCHAR2,
  p_column_name        IN VARCHAR2,
  p_display_in_form     IN VARCHAR2);
```

### Parameters

Table 24–12 describes the parameters available in the UPD\_DISPLAY\_IN\_FORM procedure.

**Table 24–12** UPD\_DISPLAY\_IN\_FORM Parameters

Parameter	Description
p_table_name	Table name
p_column_name	Column name
p_display_in_form	Determines whether to display in the form by default, valid values are Y and N

### Example

In the following example, when creating a Form against the DEPT table, the display option on the DEPTNO column defaults to 'No'.

```
APEX_UI_DEFAULT_UPDATE.UPD_DISPLAY_IN_FORM(
  p_table_name => 'DEPT',
  p_column_name => 'DEPTNO',
  p_display_in_form => 'N');
```

---

## UPD\_DISPLAY\_IN\_REPORT Procedure

The UPD\_DISPLAY\_IN\_REPORT procedure sets the display in report user interface default. This user interface default is used by wizards when you select to create a report based upon the table and controls whether the column is included by default or not.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_DISPLAY_IN_REPORT (  
    p_table_name          IN VARCHAR2,  
    p_column_name        IN VARCHAR2,  
    p_display_in_report   IN VARCHAR2);
```

### Parameters

Table 24–13 describes the parameters available in the UPD\_DISPLAY\_IN\_REPORT procedure.

**Table 24–13** UPD\_DISPLAY\_IN\_REPORT Parameters

Parameter	Description
p_table_name	Table name
p_column_name	Column name
p_display_in_report	Determines whether to display in the report by default, valid values are Y and N

### Example

In the following example, when creating a Report against the DEPT table, the display option on the DEPTNO column defaults to 'No'.

```
APEX_UI_DEFAULT_UPDATE.UPD_DISPLAY_IN_REPORT(  
    p_table_name => 'DEPT',  
    p_column_name => 'DEPTNO',  
    p_display_in_report => 'N');
```

---

## UPD\_FORM\_REGION\_TITLE Procedure

The UPD\_FORM\_REGION\_TITLE procedure updates the Form Region Title user interface default. User interface defaults are used in wizards when you create a form based upon the specified table.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_FORM_REGION_TITLE (  
    p_table_name          IN VARCHAR2,  
    p_form_region_title   IN VARCHAR2 DEFAULT NULL);
```

### Parameters

[Table 24–14](#) describes the parameters available in the UPD\_FORM\_REGION\_TITLE procedure.

**Table 24–14** UPDATE\_FORM\_REGION\_TITLE Parameters

Parameter	Description
p_table_name	Table name
p_form_region_title	Desired form region title

### Example

This example demonstrates how to set the Forms Region Title user interface default on the DEPT table.

```
APEX_UI_DEFAULT_UPDATE.UPD_FORM_REGION_TITLE (  
    p_table_name          => 'DEPT',  
    p_form_region_title   => 'Department Details');
```

## UPD\_GROUP Procedure

If the provided table and group exist within the user's schema's table based User Interface Defaults, the group name, description and display sequence of the group are updated. If 'null%' is passed in for p\_description or p\_display\_sequence, the value is set to null.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_GROUP (
    p_table_name          IN VARCHAR2,
    p_group_name          IN VARCHAR2,
    p_new_group_name      IN VARCHAR2 DEFAULT NULL,
    p_description         IN VARCHAR2 DEFAULT NULL,
    p_display_sequence    IN VARCHAR2 DEFAULT NULL);
```

### Parameters

[Table 24–15](#) describes the parameters available in the UPD\_GROUP procedure.

**Table 24–15** UPD\_GROUP Parameters

Parameter	Description
p_table_name	Name of table whose group is being updated
p_group_name	Group being updated
p_new_group_name	New name for group, if group is being renamed
p_description	Description of group
p_display_sequence	Display sequence of group.

---



---

**Note:** If p\_description or p\_display\_sequence are set to 'null%', the value is nullified. If no value is passed in, that column is not updated.

---



---

### Example

The following example updates the description of the group AUDIT\_INFO within the EMP table definition within the UI Defaults Table Dictionary within the current schema.

```
BEGIN
    apex_ui_default_update.upd_group (
        p_table_name => 'EMP',
        p_group_name => 'AUDIT_INFO',
        p_description => 'Audit columns' );
END;
```



---

## UPD\_ITEM\_DISPLAY\_HEIGHT Procedure

The UPD\_ITEM\_DISPLAY\_HEIGHT procedure sets the item display height user interface default. This user interface default is used by wizards when you select to create a form based upon the table and include the specified column. Display height controls if the item is a text box or a text area.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_ITEM_DISPLAY_HEIGHT (  
    p_table_name          IN VARCHAR2,  
    p_column_name        IN VARCHAR2,  
    p_display_height     IN NUMBER);
```

### Parameters

Table 24–16 describes the parameters available in the UPD\_ITEM\_DISPLAY\_HEIGHT procedure.

**Table 24–16** UPD\_ITEM\_DISPLAY\_HEIGHT Parameters

Parameter	Description
p_table_name	Table name
p_column_name	Column name
p_display_height	Display height of any items created based upon this column

### Example

The following example sets a default item height of 3 when creating an item on the DNAME column against the DEPT table.

```
APEX_UI_DEFAULT_UPDATE.UPD_ITEM_DISPLAY_HEIGHT (  
    p_table_name => 'DEPT',  
    p_column_name => 'DNAME',  
    p_display_height => 3);
```

---

## UPD\_ITEM\_DISPLAY\_WIDTH Procedure

The `UPD_ITEM_DISPLAY_WIDTH` procedure sets the item display width user interface default. This user interface default is used by wizards when you select to create a form based upon the table and include the specified column.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_ITEM_DISPLAY_WIDTH (  
    p_table_name          IN VARCHAR2,  
    p_column_name        IN VARCHAR2,  
    p_display_width      IN NUMBER);
```

### Parameters

[Table 24–17](#) describes the parameters available in the `UPD_ITEM_DISPLAY_WIDTH` procedure.

**Table 24–17** *UPD\_ITEM\_DISPLAY\_WIDTH Parameters*

Parameter	Description
<code>p_table_name</code>	Table name
<code>p_column_name</code>	Column name
<code>p_display_width</code>	Display width of any items created based upon this column

### Example

The following example sets a default item width of 5 when creating an item on the `DEPTNO` column against the `DEPT` table.

```
APEX_UI_DEFAULT_UPDATE.UPD_ITEM_DISPLAY_WIDTH(  
    p_table_name => 'DEPT',  
    p_column_name => 'DEPTNO',  
    p_display_width => 5);
```

---

## UPD\_ITEM\_FORMAT\_MASK Procedure

The UPD\_ITEM\_FORMAT\_MASK procedure sets the item format mask user interface default. This user interface default is used by wizards when you select to create a form based upon the table and include the specified column. Item format mask is typically used to format numbers and dates.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_ITEM_FORMAT_MASK (
    p_table_name          IN VARCHAR2,
    p_column_name         IN VARCHAR2,
    p_format_mask         IN VARCHAR2 DEFAULT NULL);
```

### Parameters

Table 24–18 describes the parameters available in the UPD\_ITEM\_FORMAT\_MASK procedure.

**Table 24–18** UPD\_ITEM\_FORMAT\_MASK Parameters

Parameter	Description
p_table_name	Table name
p_column_name	Column name
p_format_mask	Format mask to be associated with the column

### Example

In the following example, when creating a Form against the EMP table, the default item format mask on the HIREDATE column is set to 'DD-MON-YYYY'.

```
APEX_UI_DEFAULT_UPDATE.UPD_ITEM_FORMAT_MASK (
    p_table_name => 'EMP',
    p_column_name => 'HIREDATE',
    p_format_mask=> 'DD-MON-YYYY');
```

## UPD\_ITEM\_HELP Procedure

The UPD\_ITEM\_HELP procedure updates the help text for the specified table and column. This user interface default is used when you create a form based upon the table and select to include the specified column.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_ITEM_HELP (  
    p_table_name           IN VARCHAR2,  
    p_column_name         IN VARCHAR2,  
    p_help_text           IN VARCHAR2 DEFAULT NULL);
```

### Parameters

Table 24–19 describes the parameters available in the UPD\_ITEM\_HELP procedure.

**Table 24–19** UPD\_ITEM\_HELP Parameters

Parameter	Description
p_table_name	Table name
p_column_name	Column name
p_help_text	Desired help text

### Example

This example demonstrates how to set the User Interface Item Help Text default for the DEPTNO column in the DEPT table.

```
APEX_UI_DEFAULT_UPDATE.UPD_ITEM_HELP(  
    p_table_name => 'DEPT',  
    p_column_name => 'DEPTNO',  
    p_help_text => 'The number assigned to the department.');
```

---

## UPD\_LABEL Procedure

The UPD\_LABEL procedure sets the label used for items. This user interface default is used when you create a form or report based on the specified table and include a specific column.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_LABEL (  
    p_table_name          IN VARCHAR2,  
    p_column_name        IN VARCHAR2,  
    p_label               IN VARCHAR2 DEFAULT NULL);
```

### Parameters

Table 24–20 describes the parameters available in the UPD\_LABEL procedure.

**Table 24–20** UPD\_LABEL Parameters

Parameter	Description
p_table_name	Table name
p_column_name	Column name
p_label	Desired item label

### Example

This example demonstrates how to set the User Interface Item Label default for the DEPTNO column in the DEPT table.

```
APEX_UI_DEFAULT_UPDATE.UPD_LABEL (  
    p_table_name => 'DEPT',  
    p_column_name => 'DEPTNO',  
    p_label => 'Department Number');
```

---

## UPD\_REPORT\_ALIGNMENT Procedure

The UPD\_REPORT\_ALIGNMENT procedure sets the report alignment user interface default. This user interface default is used by wizards when you select to create a report based upon the table and include the specified column and determines if the report column should be left, center, or right justified.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_REPORT_ALIGNMENT (
    p_table_name          IN VARCHAR2,
    p_column_name         IN VARCHAR2,
    p_report_alignment    IN VARCHAR2);
```

### Parameters

[Table 24–21](#) describes the parameters available in the UPD\_REPORT\_ALIGNMENT procedure.

**Table 24–21** UPD\_REPORT\_ALIGNMENT Parameters

Parameter	Description
p_table_name	Table name.
p_column_name	Column name.
p_report_alignment	Defines the alignment of the column in a report. Valid values are L (left), C (center) and R (right).

### Example

In the following example, when creating a Report against the DEPT table, the default column alignment on the DEPTNO column is set to Right justified.

```
APEX_UI_DEFAULT_UPDATE.UPD_REPORT_ALIGNMENT (
    p_table_name => 'DEPT',
    p_column_name => 'DEPTNO',
    p_report_alignment => 'R');
```

## UPD\_REPORT\_FORMAT\_MASK Procedure

The UPD\_REPORT\_FORMAT\_MASK procedure sets the report format mask user interface default. This user interface default is used by wizards when you select to create a report based upon the table and include the specified column. Report format mask is typically used to format numbers and dates.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_REPORT_FORMAT_MASK (
    p_table_name          IN VARCHAR2,
    p_column_name        IN VARCHAR2,
    p_format_mask        IN VARCHAR2 DEFAULT NULL);
```

### Parameters

Table 24–22 describes the parameters available in the UPD\_REPORT\_FORMAT\_MASK procedure.

**Table 24–22** UPD\_REPORT\_FORMAT\_MASK Parameters

Parameter	Description
p_table_name	Table name
p_column_name	Column name
p_format_mask	Format mask to be associated with the column whenever it is included in a report

### Example

In the following example, when creating a Report against the EMP table, the default format mask on the HIREDATE column is set to 'DD-MON-YYYY'.

```
APEX_UI_DEFAULT_UPDATE.UPD_REPORT_FORMAT_MASK (
    p_table_name => 'EMP',
    p_column_name => 'HIREDATE',
    p_format_mask=> 'DD-MON-YYYY');
```

---

## UPD\_REPORT\_REGION\_TITLE Procedure

The UPD\_REPORT\_REGION\_TITLE procedure sets the Report Region Title. User interface defaults are used in wizards when a report is created on a table.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_REPORT_REGION_TITLE (  
    p_table_name          IN VARCHAR2,  
    p_report_region_title IN VARCHAR2 DEFAULT NULL);
```

### Parameters

[Table 24–23](#) describes the parameters available in the UPD\_REPORT\_REGION\_TITLE procedure.

**Table 24–23** UPD\_REPORT\_REGION\_TITLE Parameters

Parameter	Description
p_table_name	Table name
p_report_region_title	Desired report region title

### Example

This example demonstrates how to set the Reports Region Title user interface default on the DEPT table.

```
APEX_UI_DEFAULT_UPDATE.UPD_REPORT_REGION_TITLE (  
    p_table_name          => 'DEPT',  
    p_report_region_title => 'Departments');
```



## UPD\_TABLE Procedure

If the provided table exists within the user's schema's table based User Interface Defaults, the form region title and report region title are updated to match those provided. If 'null%' is passed in for p\_form\_region\_title or p\_report\_region\_title, the value is set to null.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_TABLE (
  p_table_name           IN VARCHAR2,
  p_form_region_title    IN VARCHAR2 DEFAULT NULL,
  p_report_region_title  IN VARCHAR2 DEFAULT NULL);
```

### Parameters

Table 24–24 describes the parameters available in the UPD\_TABLE procedure.

**Table 24–24** UPD\_TABLE Parameters

Parameter	Description
p_table_name	Name of table being updated.
p_form_region_title	Region title used for forms.
p_report_region_title	Region title used for reports and tabular forms.

---

**Note:** if 'null%' is passed in for p\_form\_region\_title or p\_report\_region\_title, the value is set to null. If no value is passed in, that column is not updated.

---

### Example

The following example updates the EMP table definition within the UI Defaults Table Dictionary within the current schema.

```
begin
  apex_ui_default_update.upd_table (
    p_table_name           => 'EMP',
    p_form_region_title    => 'Employee Details',
    p_report_region_title => 'Employees' );
end;
/
```



The `APEX_UTIL` package provides utilities you can use when programming in the Oracle Application Express environment. You can use the `APEX_UTIL` package to get and set session state, get files, check authorizations for users, reset different states for users, get and purge cache information and also to get and set preferences for users.

- `CACHE_GET_DATE_OF_PAGE_CACHE` Function
- `CACHE_GET_DATE_OF_REGION_CACHE` Function
- `CACHE_PURGE_BY_APPLICATION` Procedure
- `CACHE_PURGE_BY_PAGE` Procedure
- `CACHE_PURGE_STALE` Procedure
- `CHANGE_CURRENT_USER_PW` Procedure
- `CHANGE_PASSWORD_ON_FIRST_USE` Function
- `CLOSE_OPEN_DB_LINKS` Procedure
- `CLEAR_APP_CACHE` Procedure
- `CLEAR_PAGE_CACHE` Procedure
- `CLEAR_USER_CACHE` Procedure
- `COUNT_CLICK` Procedure
- `CREATE_USER` Procedure
- `CREATE_USER_GROUP` Procedure
- `CURRENT_USER_IN_GROUP` Function
- `CUSTOM_CALENDAR` Procedure
- `DELETE_USER_GROUP` Procedure Signature 1
- `DELETE_USER_GROUP` Procedure Signature 2
- `DOWNLOAD_PRINT_DOCUMENT` Procedure Signature 1
- `DOWNLOAD_PRINT_DOCUMENT` Procedure Signature 2
- `DOWNLOAD_PRINT_DOCUMENT` Procedure Signature 3
- `DOWNLOAD_PRINT_DOCUMENT` Procedure Signature 4
- `EDIT_USER` Procedure
- `END_USER_ACCOUNT_DAYS_LEFT` Function
- `EXPIRE_END_USER_ACCOUNT` Procedure

- 
- EXPIRE\_WORKSPACE\_ACCOUNT Procedure
  - EXPORT\_USERS Procedure
  - FETCH\_APP\_ITEM Function
  - FETCH\_USER Procedure Signature 1
  - FETCH\_USER Procedure Signature 2
  - FETCH\_USER Procedure Signature 3
  - FIND\_SECURITY\_GROUP\_ID Function
  - FIND\_WORKSPACE Function
  - GET\_ACCOUNT\_LOCKED\_STATUS Function
  - GET\_ATTRIBUTE Function
  - GET\_AUTHENTICATION\_RESULT Function
  - GET\_BLOB\_FILE\_SRC Function
  - GET\_BUILD\_OPTION\_STATUS Function Signature 1
  - GET\_BUILD\_OPTION\_STATUS Function Signature 2
  - GET\_CURRENT\_USER\_ID Function
  - GET\_DEFAULT\_SCHEMA Function
  - GET\_EDITION Function
  - GET\_EMAIL Function
  - GET\_FEEDBACK\_FOLLOW\_UP Function
  - GET\_FILE Procedure
  - GET\_FILE\_ID Function
  - GET\_FIRST\_NAME Function
  - GET\_GROUPS\_USER\_BELONGS\_TO Function
  - GET\_GROUP\_ID Function
  - GET\_GROUP\_NAME Function
  - GET\_HASH Function
  - GET\_HIGH\_CONTRAST\_MODE\_TOGGLE Function
  - GET\_LAST\_NAME Function
  - GET\_NUMERIC\_SESSION\_STATE Function
  - GET\_PREFERENCE Function
  - GET\_PRINT\_DOCUMENT Function Signature 1
  - GET\_PRINT\_DOCUMENT Function Signature 2
  - GET\_PRINT\_DOCUMENT Function Signature 3
  - GET\_PRINT\_DOCUMENT Function Signature 4
  - GET\_SCREEN\_READER\_MODE\_TOGGLE Function
  - GET\_SESSION\_LANG Function
  - GET\_SESSION\_STATE Function

- 
- GET\_SESSION\_TERRITORY Function
  - GET\_SESSION\_TIME\_ZONE Function
  - GET\_SINCE Function
  - GET\_USER\_ID Function
  - GET\_USER\_ROLES Function
  - GET\_USERNAME Function
  - HOST\_URL Function
  - HTML\_PCT\_GRAPH\_MASK Function
  - INCREMENT\_CALENDAR Procedure
  - IR\_CLEAR Procedure [DEPRECATED]
  - IR\_DELETE\_REPORT Procedure [DEPRECATED]
  - IR\_DELETE\_SUBSCRIPTION Procedure [DEPRECATED]
  - IR\_FILTER Procedure [DEPRECATED]
  - IR\_RESET Procedure [DEPRECATED]
  - IS\_HIGH\_CONTRAST\_SESSION Function
  - IS\_HIGH\_CONTRAST\_SESSION\_YN Function
  - IS\_LOGIN\_PASSWORD\_VALID Function
  - IS\_SCREEN\_READER\_SESSION Function
  - IS\_SCREEN\_READER\_SESSION\_YN Function
  - IS\_USERNAME\_UNIQUE Function
  - KEYVAL\_NUM Function
  - KEYVAL\_VC2 Function
  - LOCK\_ACCOUNT Procedure
  - PASSWORD\_FIRST\_USE\_OCCURRED Function
  - PREPARE\_URL Function
  - PUBLIC\_CHECK\_AUTHORIZATION Function [DEPRECATED]
  - PURGE\_REGIONS\_BY\_APP Procedure
  - PURGE\_REGIONS\_BY\_NAME Procedure
  - PURGE\_REGIONS\_BY\_PAGE Procedure
  - REDIRECT\_URL Procedure
  - REMOVE\_PREFERENCE Procedure
  - REMOVE\_SORT\_PREFERENCES Procedure
  - REMOVE\_USER Procedure
  - RESET\_AUTHORIZATIONS Procedure [DEPRECATED]
  - RESET\_PASSWORD Procedure
  - RESET\_PW Procedure
  - SAVEKEY\_NUM Function

- 
- SAVEKEY\_VC2 Function
  - SET\_ATTRIBUTE Procedure
  - SET\_AUTHENTICATION\_RESULT Procedure
  - SET\_BUILD\_OPTION\_STATUS Procedure
  - SET\_CURRENT\_THEME\_STYLE Procedure
  - SET\_CUSTOM\_AUTH\_STATUS Procedure
  - SET\_EDITION Procedure
  - SET\_EMAIL Procedure
  - SET\_FIRST\_NAME Procedure
  - SET\_GROUP\_GROUP\_GRANTS Procedure
  - SET\_GROUP\_USER\_GRANTS Procedure
  - SET\_LAST\_NAME Procedure
  - SET\_PREFERENCE Procedure
  - SET\_SECURITY\_GROUP\_ID Procedure
  - SET\_SESSION\_HIGH\_CONTRAST\_OFF Procedure
  - SET\_SESSION\_HIGH\_CONTRAST\_ON Procedure
  - SET\_SESSION\_LANG Procedure
  - SET\_SESSION\_LIFETIME\_SECONDS Procedure
  - SET\_SESSION\_MAX\_IDLE\_SECONDS Procedure
  - SET\_SESSION\_SCREEN\_READER\_OFF Procedure
  - SET\_SESSION\_SCREEN\_READER\_ON Procedure
  - SET\_SESSION\_STATE Procedure
  - SET\_SESSION\_TERRITORY Procedure
  - SET\_SESSION\_TIME\_ZONE Procedure
  - SET\_USERNAME Procedure
  - SHOW\_HIGH\_CONTRAST\_MODE\_TOGGLE Procedure
  - SHOW\_SCREEN\_READER\_MODE\_TOGGLE Procedure
  - STRING\_TO\_TABLE Function
  - STRONG\_PASSWORD\_CHECK Procedure
  - STRONG\_PASSWORD\_VALIDATION Function
  - SUBMIT\_FEEDBACK Procedure
  - SUBMIT\_FEEDBACK\_FOLLOWUP Procedure
  - TABLE\_TO\_STRING Function
  - UNEXPIRE\_END\_USER\_ACCOUNT Procedure
  - UNEXPIRE\_WORKSPACE\_ACCOUNT Procedure
  - UNLOCK\_ACCOUNT Procedure
  - URL\_ENCODE Function

- 
- [WORKSPACE\\_ACCOUNT\\_DAYS\\_LEFT](#) Function

---

## CACHE\_GET\_DATE\_OF\_PAGE\_CACHE Function

This function returns the date and time a specified application page was cached either for the user issuing the call, or for all users if the page was not set to be cached by user.

### Syntax

```
APEX_UTIL.CACHE_GET_DATE_OF_PAGE_CACHE (
    p_application IN NUMBER,
    p_page        IN NUMBER)
RETURN DATE;
```

### Parameters

**Table 25–1** *CACHE\_GET\_DATE\_OF\_PAGE\_CACHE Parameters*

Parameter	Description
p_application	The identification number (ID) of the application.
p_page	The page number (ID).

### Example

The following example demonstrates how to use the `CACHE_GET_DATE_OF_PAGE_CACHE` function to retrieve the cache date and time for page 9 of the currently executing application. If page 9 has been cached, the cache date and time is output using the HTP package. The page could have been cached either by the user issuing the call, or for all users if the page was not to be cached by the user.

```
DECLARE
    l_cache_date DATE DEFAULT NULL;
BEGIN
    l_cache_date := APEX_UTIL.CACHE_GET_DATE_OF_PAGE_CACHE(
        p_application => :APP_ID,
        p_page => 9);
    IF l_cache_date IS NOT NULL THEN
        HTP.P('Cached on ' || TO_CHAR(l_cache_date, 'DD-MON-YY HH24:MI:SS'));
    END IF;
END;
```



## CACHE\_GET\_DATE\_OF\_REGION\_CACHE Function

This function returns the date and time a specified region was cached either for the user issuing the call, or for all users if the page was not set to be cached by user.

### Syntax

```
APEX_UTIL.CACHE_GET_DATE_OF_REGION_CACHE (
    p_application IN NUMBER,
    p_page        IN NUMBER,
    p_region_name IN VARCHAR2)
RETURN DATE;
```

### Parameters

**Table 25–2** *CACHE\_GET\_DATE\_OF\_REGION\_CACHE Parameters*

Parameter	Description
p_application	The identification number (ID) of the application
p_page	The page number (ID)
p_region_name	The region name

### Example

The following example demonstrates how to use the `CACHE_GET_DATE_OF_REGION_CACHE` function to retrieve the cache date and time for the region named `Cached Region` on page 13 of the currently executing application. If the region has been cached, the cache date and time is output using the HTP package. The region could have been cached either by the user issuing the call, or for all users if the page was not to be cached by user.

```
DECLARE
    l_cache_date DATE DEFAULT NULL;
BEGIN
    l_cache_date := APEX_UTIL.CACHE_GET_DATE_OF_REGION_CACHE(
        p_application => :APP_ID,
        p_page => 13,
        p_region_name => 'Cached Region');
    IF l_cache_date IS NOT NULL THEN
        HTP.P('Cached on ' || TO_CHAR(l_cache_date, 'DD-MON-YY HH24:MI:SS'));
    END IF;
END;
```

## CACHE\_PURGE\_BY\_APPLICATION Procedure

This procedure purges all cached pages and regions for a given application.

### Syntax

```
APEX_UTIL.CACHE_PURGE_BY_APPLICATION (  
    p_application IN NUMBER);
```

### Parameters

**Table 25–3** *CACHE\_PURGE\_BY\_APPLICATION Parameters*

Parameter	Description
p_application	The identification number (ID) of the application.

### Example

The following example demonstrates how to use the `CACHE_PURGE_BY_APPLICATION` procedure to purge all the cached pages and regions for the application currently executing.

```
BEGIN  
    APEX_UTIL.CACHE_PURGE_BY_APPLICATION(p_application => :APP_ID);  
END;
```

---

## CACHE\_PURGE\_BY\_PAGE Procedure

This procedure purges the cache for a given application and page. If the page itself is not cached but contains one or more cached regions, then the cache for these is also purged.

### Syntax

```
APEX_UTIL.CACHE_PURGE_BY_PAGE (  
    p_application  IN NUMBER,  
    p_page         IN NUMBER,  
    p_user_name    IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 25–4** *CACHE\_PURGE\_BY\_PAGE Parameters*

Parameter	Description
p_application	The identification number (ID) of the application.
p_page	The page number (ID).
p_user_name	The user associated with cached pages and regions.

### Example

The following example demonstrates how to use the `CACHE_PURGE_BY_PAGE` procedure to purge the cache for page 9 of the application currently executing. Additionally, if the `p_user_name` parameter is supplied, this procedure would be further restricted by a specific users cache (only relevant if the cache is set to be by user).

```
BEGIN  
    APEX_UTIL.CACHE_PURGE_BY_PAGE(  
        p_application => :APP_ID,  
        p_page => 9);  
END;
```

## CACHE\_PURGE\_STALE Procedure

This procedure deletes all cached pages and regions for a specified application that have passed the defined active time period. When you cache a page or region, you specify an active time period (or Cache Timeout). Once that period has passed, the cache is no longer used, thus removing those unusable pages or regions from the cache.

### Syntax

```
APEX_UTIL.CACHE_PURGE_STALE (  
    p_application IN NUMBER);
```

### Parameters

**Table 25–5** *CACHE\_PURGE\_STALE Parameters*

Parameter	Description
p_application	The identification number (ID) of the application.

### Example

The following example demonstrates how to use the `CACHE_PURGE_STALE` procedure to purge all the stale pages and regions in the application currently executing.

```
BEGIN  
    APEX_UTIL.CACHE_PURGE_STALE(p_application => :APP_ID);  
END;
```

---

## CHANGE\_CURRENT\_USER\_PW Procedure

This procedure changes the password of the currently authenticated user, assuming Application Express user accounts are in use.

### Syntax

```
APEX_UTIL.CHANGE_CURRENT_USER_PW(  
    p_new_password IN VARCHAR2);
```

### Parameters

**Table 25–6** CHANGE\_CURRENT\_USER\_PW Parameters

Parameter	Description
p_new_password	The new password value in clear text

### Example

The following example demonstrates how to use the CHANGE\_CURRENT\_USER\_PW procedure to change the password for the user who is currently authenticated, assuming Application Express accounts are in use.

```
BEGIN  
    APEX_UTIL.CHANGE_CURRENT_USER_PW ('secret99');  
END;
```

**See Also:** ["RESET\\_PW Procedure"](#) on page 25-120

## CHANGE\_PASSWORD\_ON\_FIRST\_USE Function

Enables a developer to check whether this property is enabled or disabled for an end user account. This function returns TRUE if the account password must be changed upon first use (after successful authentication) after the password is initially set and after it is changed on the Administration Service, Edit User page. This function returns FALSE if the account does not have this property.

This function may be run in a page request context by any authenticated user.

### Syntax

```
APEX_UTIL.CHANGE_PASSWORD_ON_FIRST_USE (
    p_user_name IN VARCHAR2)
RETURN BOOLEAN;
```

### Parameters

**Table 25–7** CHANGE\_PASSWORD\_ON\_FIRST\_USE Parameters

Parameter	Description
p_user_name	The user name of the user account

### Example

The following example demonstrates how to use the CHANGE\_PASSWORD\_ON\_FIRST\_USE function. Use this function to check if the password of an Application Express user account (workspace administrator, developer, or end user) in the current workspace must be changed by the user the first time it is used.

```
BEGIN
    FOR c1 IN (SELECT user_name FROM wwv_flow_users) LOOP
        IF APEX_UTIL.CHANGE_PASSWORD_ON_FIRST_USE(p_user_name => c1.user_name)
        THEN
            http.p('User: ' || c1.user_name || ' requires password to be changed the
first time it is used.');
```

```
            END IF;
        END LOOP;
    END;
```

**See Also:** ["PASSWORD\\_FIRST\\_USE\\_OCCURRED Function"](#) on page 25-107

---

## CLOSE\_OPEN\_DB\_LINKS Procedure

This procedure closes all open database links for the current database session.

It is rare that this procedure would ever be called programatically in an application. The primary purpose of this procedure is for the middleware technology in an Oracle Application Express environment (for example, Oracle REST Data Service, `mod_plsql`) to be configured such that it closes all of the open database links in a session, either before a request is made to the Application Express engine, or after a request to the Application Express engine is completed but before the database session is returned to the pool.

### Syntax

```
APEX_UTIL.CLOSE_OPEN_DB_LINKS
```

### Parameters

None.

### Example

In this example, the configuration of Oracle REST Data Services closes any open database links both before the request is made to the Application Express engine and after the request is complete.

```
<entry key="procedure.postProcess">apex_util.close_open_db_links</entry>  
<entry key="procedure.preProcess">apex_util.close_open_db_links</entry>
```

When using Oracle HTTP Server and `mod_plsql`, this configuration would look like this:

```
PlsqlBeforeProcedureapex_util.close_open_db_links  
PlsqlAfterProcedureapex_util.close_open_db_links
```

## CLEAR\_APP\_CACHE Procedure

This procedure removes session state for a given application for the current session.

### Syntax

```
APEX_UTIL.CLEAR_APP_CACHE (  
    p_app_id    IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 25–8** CLEAR\_APP\_CACHE Parameters

Parameter	Description
p_app_id	The ID of the application for which session state is cleared for current session

### Example

The following example demonstrates how to use the CLEAR\_APP\_CACHE procedure to clear all the current sessions state for the application with an ID of 100.

```
BEGIN  
    APEX_UTIL.CLEAR_APP_CACHE('100');  
END;
```



---

## CLEAR\_PAGE\_CACHE Procedure

This procedure removes session state for a given page for the current session.

### Syntax

```
APEX_UTIL.CLEAR_PAGE_CACHE (  
    p_page_id IN NUMBER DEFAULT NULL);
```

### Parameters

**Table 25–9** CLEAR\_PAGE\_CACHE Parameters

Parameter	Description
p_page_id	The ID of the page in the current application for which session state is cleared for current session.

### Example

The following example demonstrates how to use the CLEAR\_PAGE\_CACHE procedure to clear the current session's state for the page with an ID of 10.

```
BEGIN  
    APEX_UTIL.CLEAR_PAGE_CACHE(10);  
END;
```

## CLEAR\_USER\_CACHE Procedure

This procedure removes session state and application system preferences for the current user's session. Run this procedure if you reuse session IDs and want to run applications without the benefit of existing session state.

### Syntax

```
APEX_UTIL.CLEAR_USER_CACHE;
```

### Parameters

None.

### Example

The following example demonstrates how to use the `CLEAR_USER_CACHE` procedure to clear all session state and application system preferences for the current user's session.

```
BEGIN
    APEX_UTIL.CLEAR_USER_CACHE;
END;
```

## COUNT\_CLICK Procedure

This procedure counts clicks from an application built in Application Builder to an external site. You can also use the shorthand version, procedure `Z`, in place of `APEX_UTIL.COUNT_CLICK`.

### Syntax

```
APEX_UTIL.COUNT_CLICK (
    p_url      IN    VARCHAR2,
    p_cat      IN    VARCHAR2,
    p_id       IN    VARCHAR2    DEFAULT NULL,
    p_user     IN    VARCHAR2    DEFAULT NULL,
    p_workspace IN    VARCHAR2    DEFAULT NULL);
```

### Parameters

**Table 25–10** COUNT\_CLICK Parameters

Parameter	Description
<code>p_url</code>	The URL to which to redirect
<code>p_cat</code>	A category to classify the click
<code>p_id</code>	Secondary ID to associate with the click (optional)
<code>p_user</code>	The application user ID (optional)
<code>p_workspace</code>	The workspace associated with the application (optional)

### Example

The following example demonstrates how to use the `COUNT_CLICK` procedure to log how many user's click on the `http://yahoo.com` link specified. Note that once this information is logged, you can view it by using the `APEX_WORKSPACE_CLICKS` view and in the reports on this view available to workspace and site administrators.

```
DECLARE
    l_url VARCHAR2(255);
    l_cat VARCHAR2(30);
    l_workspace_id VARCHAR2(30);
BEGIN
    l_url := 'http://yahoo.com';
    l_cat := 'yahoo';
    l_workspace_id := TO_CHAR(APEX_UTIL.FIND_SECURITY_GROUP_ID('MY_WORKSPACE'));

    HTP.P('<a href=APEX_UTIL.COUNT_CLICK?p_url=' || l_url || '&p_cat=' || l_cat ||
    '&p_workspace=' || l_workspace_id || '>Click</a>');
END;
```

**See Also:** ["FIND\\_SECURITY\\_GROUP\\_ID Function"](#) on page 25-49 in this document and "Deleting Click Counting Log Entries" in *Oracle Application Express Administration Guide*, "Managing Authorized URLs" in *Oracle Application Express Administration Guide*

## CREATE\_USER Procedure

This procedure creates a new account record in the Application Express user account table. To execute this procedure, the current user must have administrative privileges.

### Syntax

```
APEX_UTIL.CREATE_USER (
    p_user_id                IN          NUMBER          DEFAULT NULL,
    p_user_name              IN          VARCHAR2,
    p_first_name             IN          VARCHAR2        DEFAULT NULL,
    p_last_name              IN          VARCHAR2        DEFAULT NULL,
    p_description            IN          VARCHAR2        DEFAULT NULL,
    p_email_address          IN          VARCHAR2        DEFAULT NULL,
    p_web_password           IN          VARCHAR2,
    p_web_password_format   IN          VARCHAR2        DEFAULT 'CLEAR_TEXT',
    p_group_ids              IN          VARCHAR2,
    p_developer_privs       IN          VARCHAR2        DEFAULT NULL,
    p_default_schema        IN          VARCHAR2        DEFAULT NULL,
    p_allow_access_to_schemas IN          VARCHAR2        DEFAULT NULL,
    p_account_expiry        IN          DATE            DEFAULT TRUNC (SYSDATE),
    p_account_locked        IN          VARCHAR2        DEFAULT 'N',
    p_failed_access_attempts IN          NUMBER         DEFAULT 0,
    p_change_password_on_first_use IN          VARCHAR2  DEFAULT 'Y',
    p_first_password_use_occurred IN          VARCHAR2  DEFAULT 'N',
    p_attribute_01          IN          VARCHAR2        DEFAULT NULL,
    p_attribute_02          IN          VARCHAR2        DEFAULT NULL,
    p_attribute_03          IN          VARCHAR2        DEFAULT NULL,
    p_attribute_04          IN          VARCHAR2        DEFAULT NULL,
    p_attribute_05          IN          VARCHAR2        DEFAULT NULL,
    p_attribute_06          IN          VARCHAR2        DEFAULT NULL,
    p_attribute_07          IN          VARCHAR2        DEFAULT NULL,
    p_attribute_08          IN          VARCHAR2        DEFAULT NULL,
    p_attribute_09          IN          VARCHAR2        DEFAULT NULL,
    p_attribute_10          IN          VARCHAR2        DEFAULT NULL,
    p_allow_app_building_yn IN          VARCHAR2        DEFAULT NULL,
    p_allow_sql_workshop_yn IN          VARCHAR2        DEFAULT NULL,
    p_allow_websheet_dev_yn IN          VARCHAR2        DEFAULT NULL,
    p_allow_team_development_yn IN          VARCHAR2    DEFAULT NULL);
```

### Parameters

**Table 25–11** CREATE\_USER Procedure Parameters

Parameter	Description
p_user_id	Numeric primary key of user account
p_user_name	Alphanumeric name used for login
p_first_name	Informational
p_last_name	Informational
p_description	Informational
p_email_address	Email address
p_web_password	Clear text password

**Table 25–11 (Cont.) CREATE\_USER Procedure Parameters**

Parameter	Description
p_web_password_format	If the value your passing for the p_web_password parameter is in clear text format then use CLEAR_TEXT, otherwise use HEX_ENCODED_DIGEST_V2.
p_group_ids	Colon separated list of numeric group IDs
p_developer_privs	<p>Colon separated list of developer privileges. If p_developer_privs is not null, the user is given access to Team Development. If p_developer_privs contains ADMIN, the user is given Application Builder and SQL Workshop access. If p_developer_privs does not contain ADMIN but contains EDIT, the user is given Application Builder Access. If p_developer_privs does not contain ADMIN but contains SQL, the user is given SQL Workshop access. The following are acceptable values for this parameter:</p> <p><b>null</b> - To create an end user (a user who can only authenticate to developed applications).</p> <p><b>CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL</b> - To create a user with developer privileges with access to Application Builder and SQL Workshop.</p> <p><b>ADMIN:CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL</b> - To create a user with full workspace administrator and developer privileges with access to Application Builder, SQL Workshop and Team Development.</p> <p>Note: Currently this parameter is named inconsistently between the CREATE_USER, EDIT_USER and FETCH_USER APIs, although they all relate to the DEVELOPER_ROLE field stored in the named user account record. CREATE_USER uses p_developer_privs, EDIT_USER uses p_developer_roles and FETCH_USER uses p_developer_role.</p>
p_default_schema	A database schema assigned to the user's workspace, used by default for browsing.
p_allow_access_to_schemas	Colon separated list of schemas assigned to the user's workspace to which the user is restricted (leave null for all).
p_account_expiry	Date password was last updated, which defaults to today's date on creation.
p_account_locked	'Y' or 'N' indicating if account is locked or unlocked.
p_failed_access_attempts	Number of consecutive login failures that have occurred, defaults to 0 on creation.
p_change_password_on_first_use	'Y' or 'N' to indicate whether password must be changed on first use, defaults to 'Y' on creation.
p_first_password_use_occurred	'Y' or 'N' to indicate whether login has occurred since password change, defaults to 'N' on creation.
p_attribute_01	Arbitrary text accessible with an API
...	
p_attribute_10	
p_allow_app_building_yn	'Y' or 'N' to indicate whether access is allowed to Application Builder.

**Table 25–11 (Cont.) CREATE\_USER Procedure Parameters**

Parameter	Description
p_allow_sql_workshop_yn	'Y' or 'N' to indicate whether access is allowed to SQL Workshop.
p_allow_websheet_dev_yn	'Y' or 'N' to indicate whether access is allowed to Websheet development.
p_allow_team_development_yn	'Y' or 'N' to indicate whether access is allowed to Team Development.

**Example 1**

The following simple example creates an 'End User' called 'NEWUSER1' with a password of 'secret99'. Note an 'End User' can only authenticate to developed applications.

```
BEGIN
  APEX_UTIL.CREATE_USER(
    p_user_name      => 'NEWUSER1',
    p_web_password => 'secret99');
END;
```

**Example 2**

The following example creates a 'Workspace Administrator' called 'NEWUSER2'. Where the user 'NEWUSER2':

- Has full workspace administration and developer privilege (p\_developer\_privs parameter set to 'ADMIN:CREATE:DATA\_LOADER:EDIT:HELP:MONITOR:SQL').
- Has access to 2 schemas, both their browsing default 'MY\_SCHEMA' (p\_default\_schema parameter set to 'MY\_SCHEMA') and also 'MY\_SCHEMA2' (p\_allow\_access\_to\_schemas parameter set to 'MY\_SCHEMA2').
- Does not have to change their password when they first login (p\_change\_password\_on\_first\_use parameter set to 'N').
- Has their phone number stored in the first additional attribute (p\_attribute\_01 parameter set to '123 456 7890').

```
BEGIN
  APEX_UTIL.CREATE_USER(
    p_user_name           => 'NEWUSER2',
    p_first_name         => 'FRANK',
    p_last_name          => 'SMITH',
    p_description         => 'Description...',
    p_email_address      => 'frank@smith.com',
    p_web_password       => 'password',
    p_developer_privs    => 'ADMIN:CREATE:DATA_
LOADER:EDIT:HELP:MONITOR:SQL',
    p_default_schema     => 'MY_SCHEMA',
    p_allow_access_to_schemas => 'MY_SCHEMA2',
    p_change_password_on_first_use => 'N',
    p_attribute_01      => '123 456 7890');
END;
```

**See Also:** ["FETCH\\_USER Procedure Signature 3"](#) on page 25-46, ["EDIT\\_USER Procedure"](#) on page 25-32, and ["GET\\_GROUP\\_ID Function"](#) on page 25-68

---

## CREATE\_USER\_GROUP Procedure

Assuming you are using Application Express authentication, this procedure creates a user group. To execute this procedure, the current user must have administrative privileges in the workspace.

### Syntax

```
APEX_UTIL.CREATE_USER_GROUP(
  p_id                IN NUMBER default null,
  p_group_name        IN ARCHAR2,
  p_security_group_id IN NUMBER default null,
  p_group_desc        IN VARCHAR2 default null,);
```

### Parameter

**Table 25–12** CREATE\_USER\_GROUP Parameters

Parameter	Description
p_id	Primary key of group
p_group_name	Name of group
p_security_group_id	Workspace ID
p_group_desc	Descriptive text

### Example

The following example demonstrates how to use the CREATE\_USER\_GROUP procedure to create a new group called 'Managers' with a description of 'text'. Pass null for the p\_id parameter to allow the database trigger to assign the new primary key value. Pass null for the p\_security\_group\_id parameter to default to the current workspace ID.

```
BEGIN
  APEX_UTIL.CREATE_USER_GROUP (
    p_id                => null,           -- trigger assigns PK
    p_group_name        => 'Managers',
    p_security_group_id => null,         -- defaults to current workspace ID
    p_group_desc        => 'text');
END;
```

## CURRENT\_USER\_IN\_GROUP Function

This function returns a Boolean result based on whether the current user is a member of the specified group. You can use the group name or group ID to identify the group.

### Syntax

```
APEX_UTIL.CURRENT_USER_IN_GROUP(  
    p_group_name IN VARCHAR2)  
RETURN BOOLEAN;
```

```
APEX_UTIL.CURRENT_USER_IN_GROUP(  
    p_group_id IN NUMBER)  
RETURN BOOLEAN;
```

### Parameters

**Table 25–13** *CURRENT\_USER\_IN\_GROUP Parameters*

Parameter	Description
p_group_name	Identifies the name of an existing group in the workspace
p_group_id	Identifies the numeric ID of an existing group in the workspace

### Example

The following example demonstrates how to use the `CURRENT_USER_IN_GROUP` function to check if the user currently authenticated belongs to the group 'Managers'.

```
DECLARE  
    VAL BOOLEAN;  
BEGIN  
    VAL := APEX_UTIL.CURRENT_USER_IN_GROUP (p_group_name=>'Managers');  
END;
```



---

## CUSTOM\_CALENDAR Procedure

Use this procedure to change the existing calendar view to Custom Calendar.

### Syntax

```
APEX_UTIL.CUSTOM_CALENDAR(  
    p_date_type_field IN VARCHAR2);
```

### Parameters

**Table 25–14** CUSTOM\_CALENDAR Parameters

Parameter	Description
p_date_type_field	Identifies the item name used to define the type of calendar to be displayed.

### Example 1

The following example defines a custom calendar based on the hidden calendar type field. Assuming the Calendar is created in Page 9, the following example hides the column called P9\_CALENDAR\_TYPE.

```
APEX_UTIL.CUSTOM_CALENDAR(  
    'P9_CALENDAR_TYPE');
```

## DELETE\_USER\_GROUP Procedure Signature 1

Assuming you are using Application Express authentication, this procedure deletes a user group by providing the primary key of the group. To execute this procedure, the current user must have administrative privileges in the workspace.

### Syntax

```
APEX_UTIL.DELETE_USER_GROUP(  
    p_group_id IN NUMBER);
```

### Parameter

**Table 25–15** *DELETE\_USER\_GROUP Procedure Signature 1 Parameters*

Parameter	Description
p_group_id	Primary key of group

### Example

The following example demonstrates how to use the DELETE\_USER\_GROUP procedure signature 1 to remove the user group called 'Managers', by providing the user group's primary key.

```
DECLARE  
    VAL NUMBER;  
BEGIN  
    VAL := APEX_UTIL.GET_GROUP_ID (  
        p_group_name => 'Managers');  
    APEX_UTIL.DELETE_USER_GROUP (  
        p_group_id => VAL);  
END;
```

---

## DELETE\_USER\_GROUP Procedure Signature 2

Assuming you are using Application Express authentication, this procedure deletes a user group by providing the name of the group. To execute this procedure, the current user must have administrative privileges in the workspace.

### Syntax

```
APEX_UTIL.DELETE_USER_GROUP(  
    p_group_name IN VARCHAR2);
```

### Parameter

**Table 25–16** *DELETE\_USER\_GROUP Procedure Signature 2 Parameters*

Parameter	Description
p_group_name	Name of group

### Example

The following example demonstrates how to use the DELETE\_USER\_GROUP procedure signature 2 to remove the user group called 'Managers', by providing the name of the user group.

```
BEGIN  
    APEX_UTIL.DELETE_USER_GROUP (  
        p_group_name => 'Managers');  
END;
```

## DOWNLOAD\_PRINT\_DOCUMENT Procedure Signature 1

This procedure initiates the download of a print document using XML based report data (as a BLOB) and RTF or XSL-FO based report layout.

### Syntax

```
APEX_UTIL.DOWNLOAD_PRINT_DOCUMENT (
    p_file_name          IN VARCHAR,
    p_content_disposition IN VARCHAR,
    p_report_data        IN BLOB,
    p_report_layout      IN CLOB,
    p_report_layout_type IN VARCHAR2 default 'xsl-fo',
    p_document_format    IN VARCHAR2 default 'pdf',
    p_print_server       IN VARCHAR2 default null);
```

### Parameters

**Table 25–17** DOWNLOAD\_PRINT\_DOCUMENT Parameters

Parameter	Description
p_file_name	Defines the filename of the print document
p_content_disposition	Specifies whether to download the print document or display inline ("attachment", "inline")
p_report_data	XML based report data
p_report_layout	Report layout in XSL-FO or RTF format
p_report_layout_type	Defines the report layout type, that is "xsl-fo" or "rtf"
p_document_format	Defines the document format, that is "pdf", "rtf", "xls", "htm", or "xml"
p_print_server	URL of the print server. If not specified, the print server is derived from preferences.

**See Also:** "Printing Report Regions" in *Oracle Application Express Application Builder User's Guide*.

## DOWNLOAD\_PRINT\_DOCUMENT Procedure Signature 2

This procedure initiates the download of a print document using pre-defined report query and RTF and XSL-FO based report layout.

### Syntax

```
APEX_UTIL.DOWNLOAD_PRINT_DOCUMENT (
    p_file_name          IN VARCHAR,
    p_content_disposition IN VARCHAR,
    p_application_id     IN NUMBER,
    p_report_query_name  IN VARCHAR2,
    p_report_layout      IN CLOB,
    p_report_layout_type IN VARCHAR2 default 'xsl-fo',
    p_document_format    IN VARCHAR2 default 'pdf',
    p_print_server       IN VARCHAR2 default null);
```

### Parameters

**Table 25–18** DOWNLOAD\_PRINT\_DOCUMENT Parameters

Parameter	Description
p_file_name	Defines the filename of the print document
p_content_disposition	Specifies whether to download the print document or display inline ("attachment", "inline")
p_application_id	Defines the application ID of the report query
p_report_query_name	Name of the report query (stored under application's Shared Components)
p_report_layout	Report layout in XSL-FO or RTF format
p_report_layout_type	Defines the report layout type, that is "xsl-fo" or "rtf"
p_document_format	Defines the document format, that is "pdf", "rtf", "xls", "htm", or "xml"
p_print_server	URL of the print server. If not specified, the print server is derived from preferences.

### Example for Signature 2

The following example shows how to use the DOWNLOAD\_PRINT\_DOCUMENT using Signature 2 (Pre-defined report query and RTF or XSL-FO based report layout.). In this example, the data for the report is taken from a Report Query called 'ReportQueryAndXSL' stored in the current application's Shared Components > Report Queries. The report layout is taken from a value stored in a page item (P1\_XSL).

```
BEGIN
    APEX_UTIL.DOWNLOAD_PRINT_DOCUMENT (
        p_file_name          => 'mydocument',
        p_content_disposition => 'attachment',
        p_application_id     => :APP_ID,
        p_report_query_name  => 'ReportQueryAndXSL',
        p_report_layout      => :P1_XSL,
        p_report_layout_type => 'xsl-fo',
        p_document_format    => 'pdf');
END;
```

**See Also:** "Printing Report Regions" in *Oracle Application Express Application Builder User's Guide*.

## DOWNLOAD\_PRINT\_DOCUMENT Procedure Signature 3

This procedure initiates the download of a print document using pre-defined report query and pre-defined report layout.

### Syntax

```
APEX_UTIL.DOWNLOAD_PRINT_DOCUMENT (
    p_file_name          IN VARCHAR,
    p_content_disposition IN VARCHAR,
    p_application_id     IN NUMBER,
    p_report_query_name  IN VARCHAR2,
    p_report_layout_name IN VARCHAR2,
    p_report_layout_type IN VARCHAR2 default 'xsl-fo',
    p_document_format    IN VARCHAR2 default 'pdf',
    p_print_server       IN VARCHAR2 default null);
```

### Parameters

**Table 25–19** DOWNLOAD\_PRINT\_DOCUMENT Parameters

Parameter	Description
p_file_name	Defines the filename of the print document
p_content_disposition	Specifies whether to download the print document or display inline ("attachment", "inline")
p_application_id	Defines the application ID of the report query
p_report_query_name	Name of the report query (stored under application's Shared Components)
p_report_layout_name	Name of the report layout (stored under application's Shared Components)
p_report_layout_type	Defines the report layout type, that is "xsl-fo" or "rtf"
p_document_format	Defines the document format, that is "pdf", "rtf", "xls", "htm", or "xml"
p_print_server	URL of the print server. If not specified, the print server is derived from preferences.

### Example for Signature 3

The following example shows how to use the `DOWNLOAD_PRINT_DOCUMENT` using Signature 3 (Pre-defined report query and pre-defined report layout). In this example, the data for the report is taken from a Report Query called 'ReportQuery' stored in the current application's Shared Components > Report Queries. The report layout is taken from a Report Layout called 'ReportLayout' stored in the current application's Shared Components > Report Layouts. Note that if you want to provision dynamic layouts, instead of specifying 'ReportLayout' for the `p_report_layout_name` parameter, you could reference a page item that allowed the user to select one of multiple saved Report Layouts. This example also provides a way for the user to specify how they want to receive the document (as an attachment or inline), through passing the value of `P1_CONTENT_DISP` to the `p_content_disposition` parameter. `P1_CONTENT_DISP` is a page item of type 'Select List' with the following List of Values Definition:

```
STATIC2:In Browser;inline,Save / Open in separate Window;attachment
```

```
BEGIN
  APEX_UTIL.DOWNLOAD_PRINT_DOCUMENT (
    p_file_name          => 'myreport123',
    p_content_disposition => :P1_CONTENT_DISP,
    p_application_id     => :APP_ID,
    p_report_query_name  => 'ReportQuery',
    p_report_layout_name => 'ReportLayout',
    p_report_layout_type => 'rtf',
    p_document_format    => 'pdf');
END;
```

**See Also:** "Printing Report Regions" in *Oracle Application Express Application Builder User's Guide*.



## DOWNLOAD\_PRINT\_DOCUMENT Procedure Signature 4

This procedure initiates the download of a print document using XML based report data (as a CLOB) and RTF or XSL-FO based report layout.

### Syntax

```
APEX_UTIL.DOWNLOAD_PRINT_DOCUMENT (
    p_file_name          IN VARCHAR,
    p_content_disposition IN VARCHAR,
    p_report_data        IN CLOB,
    p_report_layout      IN CLOB,
    p_report_layout_type IN VARCHAR2 default 'xsl-fo',
    p_document_format    IN VARCHAR2 default 'pdf',
    p_print_server       IN VARCHAR2 default null);
```

### Parameters

**Table 25–20** DOWNLOAD\_PRINT\_DOCUMENT Parameters

Parameter	Description
p_file_name	Defines the filename of the print document
p_content_disposition	Specifies whether to download the print document or display inline ("attachment", "inline")
p_report_data	XML based report data, must be encoded in UTF-8
p_report_layout	Report layout in XSL-FO or RTF format
p_report_layout_type	Defines the report layout type, that is "xsl-fo" or "rtf"
p_document_format	Defines the document format, that is "pdf", "rtf", "xls", "htm", or "xml"
p_print_server	URL of the print server. If not specified, the print server is derived from preferences.

### Example for Signature 4

The following example shows how to use the DOWNLOAD\_PRINT\_DOCUMENT using Signature 4 (XML based report data (as a CLOB) and RTF or XSL-FO based report layout). In this example both the report data (XML) and report layout (XSL-FO) are taken from values stored in page items.

```
BEGIN
    APEX_UTIL.DOWNLOAD_PRINT_DOCUMENT (
        p_file_name          => 'mydocument',
        p_content_disposition => 'attachment',
        p_report_data        => :P1_XML,
        p_report_layout      => :P1_XSL,
        p_report_layout_type => 'xsl-fo',
        p_document_format    => 'pdf');
END;
```

**See Also:** "Printing Report Regions" in *Oracle Application Express Application Builder User's Guide*.

## EDIT\_USER Procedure

This procedure enables a user account record to be altered. To execute this procedure, the current user must have administrative privileges in the workspace.

### Syntax

```
APEX_UTIL.EDIT_USER (
    p_user_id                IN                NUMBER,
    p_user_name              IN                VARCHAR2,
    p_first_name             IN                VARCHAR2    DEFAULT NULL,
    p_last_name              IN                VARCHAR2    DEFAULT NULL,
    p_web_password           IN                VARCHAR2    DEFAULT NULL,
    p_new_password           IN                VARCHAR2    DEFAULT NULL,
    p_email_address          IN                VARCHAR2    DEFAULT NULL,
    p_start_date             IN                VARCHAR2    DEFAULT NULL,
    p_end_date               IN                VARCHAR2    DEFAULT NULL,
    p_employee_id            IN                VARCHAR2    DEFAULT NULL,
    p_allow_access_to_schemas IN              VARCHAR2    DEFAULT NULL,
    p_person_type            IN                VARCHAR2    DEFAULT NULL,
    p_default_schema         IN                VARCHAR2    DEFAULT NULL,
    p_group_ids              IN                VARCHAR2    DEFAULT NULL,
    p_developer_roles        IN                VARCHAR2    DEFAULT NULL,
    p_description            IN                VARCHAR2    DEFAULT NULL,
    p_account_expiry         IN                DATE        DEFAULT NULL,
    p_account_locked         IN                VARCHAR2    DEFAULT 'N',
    p_failed_access_attempts IN                NUMBER     DEFAULT 0,
    p_change_password_on_first_use IN          VARCHAR2    DEFAULT 'Y',
    p_first_password_use_occurred IN          VARCHAR2    DEFAULT 'N');
```

### Parameters

**Table 25–21** EDIT\_USER Parameters

Parameter	Description
p_user_id	Numeric primary key of the user account
p_user_name	Alphanumeric name used for login. <b>See Also:</b> <a href="#">"SET_USERNAME Procedure"</a> on page 25-148
p_first_name	Informational. <b>See Also:</b> <a href="#">"SET_FIRST_NAME Procedure"</a> on page 25-131
p_last_name	Informational. <b>See Also:</b> <a href="#">"SET_LAST_NAME Procedure"</a> on page 25-134
p_web_password	Clear text password. If using this procedure to update the password for the user, values for both p_web_password and p_new_password must not be null and must be identical.
p_new_password	Clear text new password. If using this procedure to update the password for the user, values for both p_web_password and p_new_password must not be null and must be identical.
p_email_address	Informational. <b>See Also:</b> <a href="#">"SET_EMAIL Procedure"</a> on page 25-130

**Table 25–21 (Cont.) EDIT\_USER Parameters**

Parameter	Description
p_start_date	Unused
p_end_date	Unused
p_employee_id	Unused
p_allow_access_to_schemas	A list of schemas assigned to the user's workspace to which the user is restricted
p_person_type	Unused
p_default_schema	A database schema assigned to the user's workspace, used by default for browsing
p_group_ids	Colon-separated list of numeric group IDs
p_developer_roles	Colon-separated list of developer privileges. The following are acceptable values for this parameter: <ul style="list-style-type: none"> <li>·<b>null</b> - To update the user to be an end user (a user who can only authenticate to developed applications)</li> <li>·<b>CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL</b> - To update the user to have developer privilege</li> <li>·<b>ADMIN:CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL</b> - To update the user to have full workspace administrator and developer privilege</li> </ul> <p><b>Note:</b> Currently this parameter is named inconsistently between the CREATE_USER, EDIT_USER and FETCH_USER APIs, although they all relate to the DEVELOPER_ROLE field stored in the named user account record. CREATE_USER uses p_developer_privs, EDIT_USER uses p_developer_roles and FETCH_USER uses p_developer_role.</p> <p><b>See Also:</b> <a href="#">"GET_USER_ROLES Function"</a> on page 25-87</p>
p_description	Informational
p_account_expiry	Date password was last updated. <p><b>See Also:</b> <a href="#">"EXPIRE_END_USER_ACCOUNT Procedure"</a> on page 25-37, <a href="#">"EXPIRE_WORKSPACE_ACCOUNT Procedure"</a> on page 25-38, <a href="#">"UNEXPIRE_END_USER_ACCOUNT Procedure"</a> on page 25-161, <a href="#">"UNEXPIRE_WORKSPACE_ACCOUNT Procedure"</a> on page 25-162</p>
p_account_locked	'Y' or 'N' indicating if account is locked or unlocked. <p><b>See Also:</b> <a href="#">"LOCK_ACCOUNT Procedure"</a> on page 25-106, <a href="#">"UNLOCK_ACCOUNT Procedure"</a> on page 25-163</p>
p_failed_access_attempts	Number of consecutive login failures that have occurred.
p_change_password_on_first_use	'Y' or 'N' to indicate whether password must be changed on first use. <p><b>See Also:</b> <a href="#">"CHANGE_PASSWORD_ON_FIRST_USE Function"</a> on page 25-12</p>
p_first_password_use_occurred	'Y' or 'N' to indicate whether login has occurred since password change. <p><b>See Also:</b> <a href="#">"PASSWORD_FIRST_USE_OCCURRED Function"</a> on page 25-107</p>

**Example**

The following example shows how to use the `EDIT_USER` procedure to update a user account. This example shows how you can use the `EDIT_USER` procedure to change the user 'FRANK' from a user with just developer privilege to a user with workspace administrator and developer privilege. Firstly, the `FETCH_USER` procedure is called to assign account details for the user 'FRANK' to local variables. These variables are then used in the call to `EDIT_USER` to preserve the details of the account, with the exception of the value for the `p_developer_roles` parameter, which is set to 'ADMIN:CREATE:DATA\_LOADER:EDIT:HELP:MONITOR:SQL'.

```

DECLARE
    l_user_id                NUMBER;
    l_workspace              VARCHAR2(255);
    l_user_name              VARCHAR2(100);
    l_first_name             VARCHAR2(255);
    l_last_name              VARCHAR2(255);
    l_web_password           VARCHAR2(255);
    l_email_address          VARCHAR2(240);
    l_start_date             DATE;
    l_end_date               DATE;
    l_employee_id            NUMBER(15,0);
    l_allow_access_to_schemas VARCHAR2(4000);
    l_person_type            VARCHAR2(1);
    l_default_schema         VARCHAR2(30);
    l_groups                  VARCHAR2(1000);
    l_developer_role         VARCHAR2(60);
    l_description            VARCHAR2(240);
    l_account_expiry         DATE;
    l_account_locked         VARCHAR2(1);
    l_failed_access_attempts NUMBER;
    l_change_password_on_first_use VARCHAR2(1);
    l_first_password_use_occurred VARCHAR2(1);
BEGIN
    l_user_id := APEX_UTIL.GET_USER_ID('FRANK');

    APEX_UTIL.FETCH_USER(
        p_user_id                => l_user_id,
        p_workspace              => l_workspace,
        p_user_name              => l_user_name,
        p_first_name             => l_first_name,
        p_last_name              => l_last_name,
        p_web_password           => l_web_password,
        p_email_address          => l_email_address,
        p_start_date             => l_start_date,
        p_end_date               => l_end_date,
        p_employee_id            => l_employee_id,
        p_allow_access_to_schemas => l_allow_access_to_schemas,
        p_person_type            => l_person_type,
        p_default_schema         => l_default_schema,
        p_groups                  => l_groups,
        p_developer_role         => l_developer_role,
        p_description            => l_description,
        p_account_expiry         => l_account_expiry,
        p_account_locked         => l_account_locked,
        p_failed_access_attempts => l_failed_access_attempts,
        p_change_password_on_first_use => l_change_password_on_first_use,
        p_first_password_use_occurred => l_first_password_use_occurred);
    APEX_UTIL.EDIT_USER (
        p_user_id                => l_user_id,
        p_user_name              => l_user_name,

```

```
p_first_name           => l_first_name,
p_last_name            => l_last_name,
p_web_password         => l_web_password,
p_new_password         => l_web_password,
p_email_address        => l_email_address,
p_start_date           => l_start_date,
p_end_date             => l_end_date,
p_employee_id          => l_employee_id,
p_allow_access_to_schemas => l_allow_access_to_schemas,
p_person_type          => l_person_type,
p_default_schema       => l_default_schema,
p_group_ids            => l_groups,
p_developer_roles      => 'ADMIN:CREATE:DATA_
LOADER:EDIT:HELP:MONITOR:SQL',
p_description          => l_description,
p_account_expiry       => l_account_expiry,
p_account_locked       => l_account_locked,
p_failed_access_attempts => l_failed_access_attempts,
p_change_password_on_first_use => l_change_password_on_first_use,
p_first_password_use_occurred => l_first_password_use_occurred);
END;
```

**See Also:** ["FETCH\\_USER Procedure Signature 3"](#) on page 25-46

## END\_USER\_ACCOUNT\_DAYS\_LEFT Function

Returns the number of days remaining before a end user account password expires. This function may be run in a page request context by any authenticated user.

### Syntax

```
APEX_UTIL.END_USER_ACCOUNT_DAYS_LEFT (
    p_user_name IN VARCHAR2)
RETURN NUMBER;
```

### Parameters

**Table 25–22** *END\_USER\_ACCOUNT\_DAYS\_LEFT Parameters*

Parameter	Description
p_user_name	The user name of the user account

### Example

The following example shows how to use the END\_USER\_ACCOUNT\_DAYS\_LEFT function. Use this function to determine the number of days remaining before an Application Express end user account in the current workspace expires.

```
DECLARE
    l_days_left NUMBER;
BEGIN
    FOR c1 IN (SELECT user_name from wwv_flow_users) LOOP
        l_days_left := APEX_UTIL.END_USER_ACCOUNT_DAYS_LEFT(p_user_name =>
c1.user_name);
        htp.p('End User Account: ' || c1.user_name || ' expires in ' || l_days_left ||
days. ');
    END LOOP;
END;
```

**See Also:** ["EXPIRE\\_END\\_USER\\_ACCOUNT Procedure"](#) on page 25-37 and ["UNEXPIRE\\_END\\_USER\\_ACCOUNT Procedure"](#) on page 25-161

---

## EXPIRE\_END\_USER\_ACCOUNT Procedure

Expires the login account for use as a workspace end user. Must be run by an authenticated workspace administrator in a page request context.

### Syntax

```
APEX_UTIL.EXPIRE_END_USER_ACCOUNT (
    p_user_name IN VARCHAR2
);
```

### Parameters

**Table 25–23** *EXPIRE\_END\_USER\_ACCOUNT Parameters*

Parameter	Description
p_user_name	The user name of the user account

### Example

The following example shows how to use the EXPIRE\_END\_USER\_ACCOUNT procedure. Use this procedure to expire an Oracle Application Express account (workspace administrator, developer, or end user) in the current workspace. This action specifically expires the account for its use by end users to authenticate to developed applications, but it may also expire the account for its use by developers or administrators to log in to a workspace.

Note that this procedure must be run by a user having administration privileges in the current workspace.

```
BEGIN
    FOR c1 IN (select user_name from wwv_flow_users) LOOP
        APEX_UTIL.EXPIRE_END_USER_ACCOUNT(p_user_name => c1.user_name);
        http.p('End User Account: ' || c1.user_name || ' is now expired.');
```

```
    END LOOP;
```

```
END;
```

**See Also:** ["UNEXPIRE\\_END\\_USER\\_ACCOUNT Procedure"](#) on page 25-161

## EXPIRE\_WORKSPACE\_ACCOUNT Procedure

Expires developer or workspace administrator login accounts. Must be run by an authenticated workspace administrator in a page request context.

### Syntax

```
APEX_UTIL.EXPIRE_WORKSPACE_ACCOUNT (
    p_user_name IN VARCHAR2
);
```

### Parameters

**Table 25–24** *EXPIRE\_WORKSPACE\_ACCOUNT Parameters*

Parameter	Description
p_user_name	The user name of the user account

### Example

The following example shows how to use the EXPIRE\_WORKSPACE\_ACCOUNT procedure. Use this procedure to expire an Application Express account (workspace administrator, developer, or end user) in the current workspace. This action specifically expires the account for its use by developers or administrators to log in to a workspace, but it may also expire the account for its use by end users to authenticate to developed applications.

```
BEGIN
    FOR c1 IN (SELECT user_name FROM wwv_flow_users) LOOP
        APEX_UTIL.EXPIRE_WORKSPACE_ACCOUNT(p_user_name => c1.user_name);
        http.p('Workspace Account: '||c1.user_name||' is now expired.');
```

```
END LOOP;
END;
```

**See Also:** ["UNEXPIRE\\_WORKSPACE\\_ACCOUNT Procedure"](#) on page 25-162



---

## EXPORT\_USERS Procedure

When called from a page, this procedure produces an export file of the current workspace definition, workspace users, and workspace groups. To execute this procedure, the current user must have administrative privilege in the workspace.

### Syntax

```
APEX_UTIL.EXPORT_USERS(
    p_export_format IN VARCHAR2 DEFAULT 'UNIX');
```

### Parameters

**Table 25–25** EXPORT\_USERS Parameters

Parameter	Description
p_export_format	Indicates how rows in the export file are formatted. Specify 'UNIX' to have the resulting file contain rows delimited by line feeds. Specify 'DOS' to have the resulting file contain rows delimited by carriage returns and line feeds

### Example

The following example shows how to use the EXPORT\_USERS procedure. Call this procedure from a page to produce an export file containing the current workspace definition, list of workspace users and list of workspace groups. The file is formatted with rows delimited by line feeds.

```
BEGIN
    APEX_UTIL.EXPORT_USERS;
END;
```

## FETCH\_APP\_ITEM Function

This function fetches session state for the current or specified application in the current or specified session.

### Syntax

```
APEX_UTIL.FETCH_APP_ITEM(
    p_item    IN VARCHAR2,
    p_app     IN NUMBER DEFAULT NULL,
    p_session IN NUMBER DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 25–26** *FETCH\_APP\_ITEM Parameters*

Parameter	Description
p_item	The name of an application-level item (not a page item) whose current value is to be fetched
p_app	The ID of the application that owns the item (leave null for the current application)
p_session	The session ID from which to obtain the value (leave null for the current session)

### Example

The following example shows how to use the `FETCH_APP_ITEM` function to obtain the value of the application item 'F300\_NAME' in application 300. As no value is passed for `p_session`, this defaults to the current session state value.

```
DECLARE
    VAL VARCHAR2(30);
BEGIN
    VAL := APEX_UTIL.FETCH_APP_ITEM(
        p_item => 'F300_NAME',
        p_app  => 300);
END;
```

## FETCH\_USER Procedure Signature 1

This procedure fetches a user account record. To execute this procedure, the current user must have administrative privileges in the workspace. Three overloaded versions of this procedure exist, each with a distinct set of allowed parameters or signatures.

### Syntax for Signature 1

```
APEX_UTIL.FETCH_USER (
    p_user_id           IN           NUMBER,
    p_workspace        OUT          VARCHAR2,
    p_user_name        OUT          VARCHAR2,
    p_first_name       OUT          VARCHAR2,
    p_last_name        OUT          VARCHAR2,
    p_web_password     OUT          VARCHAR2,
    p_email_address    OUT          VARCHAR2,
    p_start_date       OUT          VARCHAR2,
    p_end_date         OUT          VARCHAR2,
    p_employee_id      OUT          VARCHAR2,
    p_allow_access_to_schemas OUT    VARCHAR2,
    p_person_type      OUT          VARCHAR2,
    p_default_schema   OUT          VARCHAR2,
    p_groups           OUT          VARCHAR2,
    p_developer_role   OUT          VARCHAR2,
    p_description      OUT          VARCHAR2 );
```

### Parameters for Signature 1

**Table 25–27 Fetch\_User Parameters Signature 1**

Parameter	Description
p_user_id	Numeric primary key of the user account
p_workspace	The name of the workspace
p_user_name	Alphanumeric name used for login. <b>See Also:</b> " <a href="#">GET_USERNAME Function</a> " on page 25-88
p_first_name	Informational. <b>See Also:</b> " <a href="#">GET_FIRST_NAME Function</a> " on page 25-66
p_last_name	Informational. <b>See Also:</b> " <a href="#">GET_LAST_NAME Function</a> " on page 25-72
p_web_password	Obfuscated account password
p_email_address	Email address. <b>See Also:</b> " <a href="#">GET_EMAIL Function</a> " on page 25-61
p_start_date	Unused
p_end_date	Unused
p_employee_id	Unused
p_allow_access_to_schemas	A list of schemas assigned to the user's workspace to which user is restricted
p_person_type	Unused

**Table 25–27 (Cont.) Fetch\_User Parameters Signature 1**

Parameter	Description
p_default_schema	A database schema assigned to the user's workspace, used by default for browsing.  <b>See Also:</b> " <a href="#">GET_DEFAULT_SCHEMA Function</a> " on page 25-59
p_groups	List of groups of which user is a member.  <b>See Also:</b> " <a href="#">GET_GROUPS_USER_BELONGS_TO Function</a> " on page 25-67 and " <a href="#">CURRENT_USER_IN_GROUP Function</a> " on page 25-22
p_developer_role	Colon-separated list of developer roles. The following are acceptable values for this parameter:  null - Indicates an end user (a user who can only authenticate to developed applications).  CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL - Indicates a user with developer privilege.  ADMIN:CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL - Indicates a user with full workspace administrator and developer privilege.  <b>Note:</b> Currently this parameter is named inconsistently between the CREATE_USER, EDIT_USER and FETCH_USER APIs, although they all relate to the DEVELOPER_ROLE field stored in the named user account record. CREATE_USER uses p_developer_privs, EDIT_USER uses p_developer_roles and FETCH_USER uses p_developer_role.  <b>See Also:</b> " <a href="#">GET_USER_ROLES Function</a> " on page 25-87
p_description	Informational

**Example for Signature 1**

The following example shows how to use the FETCH\_USER procedure with Signature 1. This procedure is passed the ID of the currently authenticated user for the only IN parameter p\_user\_id. The code then stores all the other OUT parameter values in local variables.

```

DECLARE
    l_workspace          VARCHAR2(255);
    l_user_name          VARCHAR2(100);
    l_first_name         VARCHAR2(255);
    l_last_name          VARCHAR2(255);
    l_web_password       VARCHAR2(255);
    l_email_address      VARCHAR2(240);
    l_start_date         DATE;
    l_end_date           DATE;
    l_employee_id        NUMBER(15,0);
    l_allow_access_to_schemas VARCHAR2(4000);
    l_person_type        VARCHAR2(1);
    l_default_schema     VARCHAR2(30);
    l_groups             VARCHAR2(1000);
    l_developer_role     VARCHAR2(60);
    l_description        VARCHAR2(240);
BEGIN
    APEX_UTIL.FETCH_USER (
        p_user_id          => APEX_UTIL.GET_CURRENT_USER_ID,
        p_workspace       => l_workspace,
        p_user_name       => l_user_name,
        p_first_name      => l_first_name,

```

```
p_last_name           => l_last_name,  
p_web_password        => l_web_password,  
p_email_address       => l_email_address,  
p_start_date          => l_start_date,  
p_end_date            => l_end_date,  
p_employee_id         => l_employee_id,  
p_allow_access_to_schemas => l_allow_access_to_schemas,  
p_person_type         => l_person_type,  
p_default_schema      => l_default_schema,  
p_groups              => l_groups,  
p_developer_role      => l_developer_role,  
p_description         => l_description);  
  
END;
```

**See Also:** ["EDIT\\_USER Procedure"](#) on page 25-32 and ["GET\\_CURRENT\\_USER\\_ID Function"](#) on page 25-58

## FETCH\_USER Procedure Signature 2

This procedure fetches a user account record. To execute this procedure, the current user must have administrative privileges in the workspace. Three overloaded versions of this procedure exist, each with a distinct set of allowed parameters or signatures.

### Syntax for Signature 2

```
APEX_UTIL.FETCH_USER (
    p_user_id           IN           NUMBER,
    p_user_name        OUT          VARCHAR2,
    p_first_name       OUT          VARCHAR2,
    p_last_name        OUT          VARCHAR2,
    p_email_address    OUT          VARCHAR2,
    p_groups            OUT          VARCHAR2,
    p_developer_role   OUT          VARCHAR2,
    p_description      OUT          VARCHAR2 );
```

### Parameters for Signature 2

**Table 25–28 Fetch\_User Parameters Signature 2**

Parameter	Description
p_user_id	Numeric primary key of the user account
p_user_name	Alphanumeric name used for login. <b>See Also:</b> <a href="#">"GET_USERNAME Function"</a> on page 25-88
p_first_name	Informational. <b>See Also:</b> <a href="#">"GET_FIRST_NAME Function"</a> on page 25-66
p_last_name	Informational. <b>See Also:</b> <a href="#">"GET_LAST_NAME Function"</a> on page 25-72
p_email_address	Email address. <b>See Also:</b> <a href="#">"GET_EMAIL Function"</a> on page 25-61
p_groups	List of groups of which user is a member. <b>See Also:</b> <a href="#">"GET_GROUPS_USER_BELONGS_TO Function"</a> on page 25-67 and <a href="#">"CURRENT_USER_IN_GROUP Function"</a> on page 25-22
p_developer_role	Colon-separated list of developer roles. The following are acceptable values for this parameter:  null - Indicates an end user (a user who can only authenticate to developed applications).  CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL - Indicates a user with developer privilege.  ADMIN:CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL - Indicates a user with full workspace administrator and developer privilege.  Note: Currently this parameter is named inconsistently between the CREATE_USER, EDIT_USER and FETCH_USER APIs, although they all relate to the DEVELOPER_ROLE field stored in the named user account record. CREATE_USER uses p_developer_privs, EDIT_USER uses p_developer_roles and FETCH_USER uses p_developer_role.  <b>See Also:</b> <a href="#">"GET_USER_ROLES Function"</a> on page 25-87

**Table 25–28 (Cont.) Fetch\_User Parameters Signature 2**

Parameter	Description
p_description	Informational

**Example for Signature 2**

The following example shows how to use the `FETCH_USER` procedure with Signature 2. This procedure is passed the ID of the currently authenticated user for the only `IN` parameter `p_user_id`. The code then stores all the other `OUT` parameter values in local variables.

```

DECLARE
    l_user_name          VARCHAR2(100);
    l_first_name        VARCHAR2(255);
    l_last_name         VARCHAR2(255);
    l_email_address     VARCHAR2(240);
    l_groups            VARCHAR2(1000);
    l_developer_role    VARCHAR2(60);
    l_description       VARCHAR2(240);
BEGIN
    APEX_UTIL.FETCH_USER(
        p_user_id        => APEX_UTIL.GET_CURRENT_USER_ID,
        p_user_name      => l_user_name,
        p_first_name     => l_first_name,
        p_last_name      => l_last_name,
        p_email_address  => l_email_address,
        p_groups         => l_groups,
        p_developer_role => l_developer_role,
        p_description    => l_description);
END;
```

**See Also:** ["EDIT\\_USER Procedure"](#) on page 25-32 and ["GET\\_CURRENT\\_USER\\_ID Function"](#) on page 25-58

## FETCH\_USER Procedure Signature 3

This procedure fetches a user account record. To execute this procedure, the current user must have administrative privileges in the workspace. Three overloaded versions of this procedure exist, each with a distinct set of allowed parameters or signatures.

### Syntax for Signature 3

```
APEX_UTIL.FETCH_USER (
    p_user_id                IN          NUMBER,
    p_workspace              OUT        VARCHAR2,
    p_user_name              OUT        VARCHAR2,
    p_first_name             OUT        VARCHAR2,
    p_last_name              OUT        VARCHAR2,
    p_web_password           OUT        VARCHAR2,
    p_email_address          OUT        VARCHAR2,
    p_start_date             OUT        VARCHAR2,
    p_end_date               OUT        VARCHAR2,
    p_employee_id            OUT        VARCHAR2,
    p_allow_access_to_schemas OUT        VARCHAR2,
    p_person_type            OUT        VARCHAR2,
    p_default_schema         OUT        VARCHAR2,
    p_groups                 OUT        VARCHAR2,
    p_developer_role         OUT        VARCHAR2,
    p_description            OUT        VARCHAR2,
    p_account_expiry         OUT        DATE,
    p_account_locked         OUT        VARCHAR2,
    p_failed_access_attempts OUT        NUMBER,
    p_change_password_on_first_use OUT    VARCHAR2,
    p_first_password_use_occurred OUT    VARCHAR2 );
```

### Parameters for Signature 3

**Table 25–29** *Fetch\_User Parameters Signature 3*

Parameter	Description
p_user_id	Numeric primary key of the user account
p_workspace	The name of the workspace
p_user_name	Alphanumeric name used for login. <b>See Also:</b> <a href="#">"GET_USERNAME Function"</a> on page 25-88
p_first_name	Informational. <b>See Also:</b> <a href="#">"GET_FIRST_NAME Function"</a> on page 25-66
p_last_name	Informational. <b>See Also:</b> <a href="#">"GET_LAST_NAME Function"</a> on page 25-72
p_web_password	Obfuscated account password
p_email_address	Email address. <b>See Also:</b> <a href="#">"GET_EMAIL Function"</a> on page 25-61
p_start_date	Unused
p_end_date	Unused



**Table 25–29 (Cont.) Fetch\_User Parameters Signature 3**

Parameter	Description
p_employee_id	Unused
p_allow_access_to_schemas	A list of schemas assigned to the user's workspace to which user is restricted
p_person_type	Unused
p_default_schema	A database schema assigned to the user's workspace, used by default for browsing. <b>See Also:</b> <a href="#">"GET_DEFAULT_SCHEMA Function"</a> on page 25-59
p_groups	List of groups of which user is a member. <b>See Also:</b> <a href="#">"GET_GROUPS_USER_BELONGS_TO Function"</a> on page 25-67 and <a href="#">"CURRENT_USER_IN_GROUP Function"</a> on page 25-22
p_developer_role	Colon-separated list of developer roles. The following are acceptable values for this parameter: null - Indicates an end user (a user who can only authenticate to developed applications). CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL - Indicates a user with developer privilege. ADMIN:CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL - Indicates a user with full workspace administrator and developer privilege. Note: Currently this parameter is named inconsistently between the CREATE_USER, EDIT_USER and FETCH_USER APIs, although they all relate to the DEVELOPER_ROLE field stored in the named user account record. CREATE_USER uses p_developer_privs, EDIT_USER uses p_developer_roles and FETCH_USER uses p_developer_role. <b>See Also:</b> <a href="#">"GET_USER_ROLES Function"</a> on page 25-87
p_description	Informational
p_account_expiry	Date account password was last reset. <b>See Also:</b> <a href="#">"END_USER_ACCOUNT_DAYS_LEFT Function"</a> on page 25-36 and <a href="#">"WORKSPACE_ACCOUNT_DAYS_LEFT Function"</a> on page 25-166
p_account_locked	Locked/Unlocked indicator Y or N. <b>See Also:</b> <a href="#">"GET_ACCOUNT_LOCKED_STATUS Function"</a> on page 25-51
p_failed_access_attempts	Counter for consecutive login failures
p_change_password_on_first_use	Setting to force password change on first use Y or N
p_first_password_use_occurred	Indicates whether login with password occurred Y or N

**Example for Signature 3**

The following example shows how to use the FETCH\_USER procedure with Signature 3. This procedure is passed the ID of the currently authenticated user for the only IN parameter p\_user\_id. The code then stores all the other OUT parameter values in local variables.

```

DECLARE
  l_workspace          VARCHAR2(255);
  l_user_name          VARCHAR2(100);
  l_first_name         VARCHAR2(255);
  l_last_name          VARCHAR2(255);
  l_web_password       VARCHAR2(255);
  l_email_address      VARCHAR2(240);
  l_start_date         DATE;
  l_end_date           DATE;
  l_employee_id        NUMBER(15,0);
  l_allow_access_to_schemas VARCHAR2(4000);
  l_person_type        VARCHAR2(1);
  l_default_schema     VARCHAR2(30);
  l_groups             VARCHAR2(1000);
  l_developer_role     VARCHAR2(60);
  l_description        VARCHAR2(240);
  l_account_expiry     DATE;
  l_account_locked     VARCHAR2(1);
  l_failed_access_attempts NUMBER;
  l_change_password_on_first_use VARCHAR2(1);
  l_first_password_use_occurred VARCHAR2(1);
BEGIN
  APEX_UTIL.FETCH_USER(
    p_user_id          => APEX_UTIL.GET_CURRENT_USER_ID,
    p_workspace        => l_workspace,
    p_user_name        => l_user_name,
    p_first_name       => l_first_name,
    p_last_name        => l_last_name,
    p_web_password     => l_web_password,
    p_email_address    => l_email_address,
    p_start_date       => l_start_date,
    p_end_date         => l_end_date,
    p_employee_id      => l_employee_id,
    p_allow_access_to_schemas => l_allow_access_to_schemas,
    p_person_type      => l_person_type,
    p_default_schema   => l_default_schema,
    p_groups           => l_groups,
    p_developer_role   => l_developer_role,
    p_description      => l_description,
    p_account_expiry   => l_account_expiry,
    p_account_locked   => l_account_locked,
    p_failed_access_attempts => l_failed_access_attempts,
    p_change_password_on_first_use => l_change_password_on_first_use,
    p_first_password_use_occurred => l_first_password_use_occurred);
END;
```

**See Also:** ["EDIT\\_USER Procedure"](#) on page 25-32 and ["GET\\_CURRENT\\_USER\\_ID Function"](#) on page 25-58

---

## FIND\_SECURITY\_GROUP\_ID Function

This function returns the numeric security group ID of the named workspace.

### Syntax

```
APEX_UTIL.FIND_SECURITY_GROUP_ID(  
    p_workspace    IN VARCHAR2)  
RETURN NUMBER;
```

### Parameters

**Table 25–30** FIND\_SECURITY\_GROUP\_ID Parameters

Parameter	Description
p_workspace	The name of the workspace

### Example

The following example demonstrates how to use the FIND\_SECURITY\_GROUP\_ID function to return the security group ID for the workspace called 'DEMOS'.

```
DECLARE  
    VAL NUMBER;  
BEGIN  
    VAL := APEX_UTIL.FIND_SECURITY_GROUP_ID (p_workspace=>'DEMOS');  
END;
```

## FIND\_WORKSPACE Function

This function returns the workspace name associated with a security group ID.

### Syntax

```
APEX_UTIL.FIND_WORKSPACE(  
    p_security_group_id    IN VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 25–31** FIND\_WORKSPACE Parameters

Parameter	Description
p_security_group_id	The security group ID of a workspace

### Example

The following example demonstrates how to use the FIND\_WORKSPACE function to return the workspace name for the workspace with a security group ID of 20.

```
DECLARE  
    VAL VARCHAR2(255);  
BEGIN  
    VAL := APEX_UTIL.FIND_WORKSPACE (p_security_group_id =>'20');  
END;
```

## GET\_ACCOUNT\_LOCKED\_STATUS Function

Returns TRUE if the account is locked and FALSE if the account is unlocked. Must be run by an authenticated workspace administrator in a page request context.

### Syntax

```
APEX_UTIL.GET_ACCOUNT_LOCKED_STATUS (
    p_user_name IN VARCHAR2
) RETURN BOOLEAN;
```

### Parameters

**Table 25–32** GET\_ACCOUNT\_LOCKED\_STATUS Parameters

Parameter	Description
p_user_name	The user name of the user account

### Example

The following example shows how to use the GET\_ACCOUNT\_LOCKED\_STATUS function. Use this function to check if an Application Express user account (workspace administrator, developer, or end user) in the current workspace is locked.

```
BEGIN
    FOR c1 IN (SELECT user_name FROM wwv_flow_users) loop
        IF APEX_UTIL.GET_ACCOUNT_LOCKED_STATUS(p_user_name => c1.user_name) THEN
            HTP.P('User Account: ' || c1.user_name || ' is locked.');
```

**See Also:** [LOCK\\_ACCOUNT Procedure](#) on page 25-106 and [UNLOCK\\_ACCOUNT Procedure](#) on page 25-163.

## GET\_ATTRIBUTE Function

This function returns the value of one of the attribute values (1 through 10) of a named user in the Application Express accounts table. Please note these are only accessible by using the APIs.

### Syntax

```
APEX_UTIL.GET_ATTRIBUTE(  
    p_username           IN VARCHAR2,  
    p_attribute_number   IN NUMBER)  
RETURN VARCHAR2;
```

### Parameters

**Table 25–33** GET\_ATTRIBUTE Parameters

Parameter	Description
p_username	User name in the account.
p_attribute_number	Number of attributes in the user record (1 through 10)

### Example

The following example shows how to use the GET\_ATTRIBUTE function to return the value for the 1st attribute for the user 'FRANK'.

```
DECLARE  
    VAL VARCHAR2(4000);  
BEGIN  
    VAL := APEX_UTIL.GET_ATTRIBUTE (  
        p_username => 'FRANK',  
        p_attribute_number => 1);  
END;
```

**See Also:** ["SET\\_ATTRIBUTE Procedure"](#) on page 25-123

## GET\_AUTHENTICATION\_RESULT Function

Use this function to retrieve the authentication result of the current session. Any authenticated user can call this function in a page request context.

### Syntax

```
APEX_UTIL.GET_AUTHENTICATION_RESULT  
RETURN NUMBER;
```

### Parameters

None.

### Example

The following example demonstrates how to use the post-authentication process of an application's authentication scheme to retrieve the authentication result code set during authentication.

```
APEX_UTIL.SET_SESSION_STATE('MY_AUTH_STATUS',  
    'Authentication result:' || APEX_UTIL.GET_AUTHENTICATION_RESULT);
```

**See Also:** ["SET\\_AUTHENTICATION\\_RESULT Procedure"](#) on page 25-124 and ["SET\\_CUSTOM\\_AUTH\\_STATUS Procedure"](#) on page 25-128

## GET\_BLOB\_FILE\_SRC Function

As an alternative to using the built-in methods of providing a download link, you can use the `APEX_UTIL.GET_BLOB_FILE_SRC` function. One advantage of this approach, is the ability to more specifically format the display of the image (with height and width tags). Please note that this approach is only valid if called from a valid Oracle Application Express session. Also, this method requires that the parameters that describe the BLOB to be listed as the format of a valid item within the application. That item is then referenced by the function.

**See Also:** "About BLOB Support in Forms and Reports" in *Oracle Application Express Application Builder User's Guide*

### Syntax

```
APEX_UTIL.GET_BLOB_FILE_SRC (
    p_item_name          IN VARCHAR2 DEFAULT NULL,
    p_v1                 IN VARCHAR2 DEFAULT NULL,
    p_v2                 IN VARCHAR2 DEFAULT NULL,
    p_content_disposition IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 25–34** GET\_BLOB\_FILE\_SRC Parameters

Parameter	Description
p_item_name	Name of valid application page ITEM that with type FILE that contains the source type of DB column.
p_v1	Value of primary key column 1.
p_v2	Value of primary key column 2.
p_content_disposition	Specify inline or attachment, all other values ignored

### Example

As a PLSQL Function Body:

```
RETURN '';
```

As a Region Source of type SQL:

```
SELECT ID, NAME,CASE WHEN NVL(dbms_lob.getlength(document),0) = 0
    THEN NULL
    ELSE CASE WHEN attach_mimetype like 'image%'
    THEN ''
    ELSE
    '<a href="||apex_util.get_blob_file_src('P4_DOCUMENT',id)||'">Download</a>'
    end
END new_img
FROM TEST_WITH_BLOB
```

The previous example illustrates how to display the BLOB within the report, if it can be displayed, and provide a download link, if it cannot be displayed.



**See Also:** "Running a Demonstration Application" in *Oracle Application Express Application Builder User's Guide*.

## GET\_BUILD\_OPTION\_STATUS Function Signature 1

Use this function to get the build option status of a specified application by providing the ID of the application build option.

### Syntax

```
APEX_UTIL.GET_BUILD_OPTION_STATUS (  
    p_application_id IN NUMBER  
    p_id             IN NUMBER;
```

### Parameters

**Table 25–35** GET\_BUILD\_OPTION\_STATUS Function Signature 1 Parameters

Parameters	Description
p_application_id	The ID of the application that owns the build option under shared components.
p_id	The ID of the build option in the application.

### Example

The following code retrieves the current status of the specified build option that is identified by ID.

```
DECLARE  
    l_status VARCHAR2(255);  
BEGIN  
    l_status := APEX_UTIL.GET_BUILD_OPTION_STATUS(  
        P_APPLICATION_ID => 101,  
        P_ID => 245935500311121039);  
END;  
/
```

## GET\_BUILD\_OPTION\_STATUS Function Signature 2

Use this function to get the build option status of a specified application by providing the name of the application build option.

### Syntax

```
APEX_UTIL.GET_BUILD_OPTION_STATUS (
    p_application_id    IN NUMBER
    p_build_option_name IN VARCHAR2);
```

### Parameters

**Table 25–36** GET\_BUILD\_OPTION\_STATUS Function Signature 2 Parameters

Parameters	Description
p_application_id	The ID of the application that owns the build option under shared components.
p_build_option_name	The name of the build option in the application.

### Example

The following code retrieves the current status of the specified build option that is identified by name.

```
DECLARE
    l_status VARCHAR2(255);
BEGIN
    l_status := APEX_UTIL.GET_BUILD_OPTION_STATUS(
        P_APPLICATION_ID => 101,
        P_BUILD_OPTION_NAME => 'EXCLUDE_FROM_PRODUCTION');
END;
/
```

---

## GET\_CURRENT\_USER\_ID Function

This function returns the numeric user ID of the current user.

### Syntax

```
APEX_UTIL.GET_CURRENT_USER_ID  
RETURN NUMBER;
```

### Parameters

None.

### Example

This following example shows how to use the GET\_CURRENT\_USER\_ID function. It returns the numeric user ID of the current user into a local variable.

```
DECLARE  
    VAL NUMBER;  
BEGIN  
    VAL := APEX_UTIL.GET_CURRENT_USER_ID;  
END;
```

## GET\_DEFAULT\_SCHEMA Function

This function returns the default schema name associated with the current user.

### Syntax

```
APEX_UTIL.GET_DEFAULT_SCHEMA  
RETURN VARCHAR2;
```

### Parameters

None.

### Example

The following example shows how to use the GET\_DEFAULT\_SCHEMA function. It returns the default schema name associated with the current user into a local variable.

```
DECLARE  
    VAL VARCHAR2(30);  
BEGIN  
    VAL := APEX_UTIL.GET_DEFAULT_SCHEMA;  
END;
```

## GET\_EDITION Function

This function returns the edition for the current page view.

### Syntax

```
APEX_UTIL.GET_EDITION  
RETURN VARCHAR2;
```

### Parameters

None.

### Example

The following example shows how to use the `GET_EDITION` function. It returns the edition name for the current page view into a local variable.

```
DECLARE  
    VAL VARCHAR2(30);  
BEGIN  
    VAL := APEX_UTIL.GET_EDITION;  
END;
```

---

## GET\_EMAIL Function

This function returns the email address associated with the named user.

### Syntax

```
APEX_UTIL.GET_EMAIL(  
    p_username IN VARCHAR2);  
RETURN VARCHAR2;
```

### Parameters

**Table 25–37** GET\_EMAIL Parameters

Parameter	Description
p_username	The user name in the account

### Example

The following example shows how to use the GET\_EMAIL function to return the email address of the user 'FRANK'.

```
DECLARE  
    VAL VARCHAR2(240);  
BEGIN  
    VAL := APEX_UTIL.GET_EMAIL(p_username => 'FRANK');  
END;
```

**See Also:** ["SET\\_EMAIL Procedure"](#) on page 25-130

## GET\_FEEDBACK\_FOLLOW\_UP Function

Use this function to retrieve any remaining follow up associated with a specific feedback.

### Syntax

```
APEX_UTIL.GET_FEEDBACK_FOLLOW_UP (
    p_feedback_id    IN NUMBER,
    p_row            IN NUMBER DEFAULT 1,
    p_template       IN VARCHAR2 DEFAULT '<br />#CREATED_ON# (#CREATED_BY#)
#FOLLOW_UP#')
RETURN VARCHAR2;
```

### Parameters

**Table 25–38** GET\_FEEDBACK\_FOLLOW\_UP Parameters

Parameter	Description
p_feedback_id	The unique identifier of the feedback item.
p_row	Identifies which follow-up to retrieve and is ordered by created_on_desc.
p_template	The template to use to return the follow up. Given the   in the default template, the function can be used in a loop to return all the follow up to a feedback.

### Example

The following example displays all the remaining follow-up for feedback with the ID of 123.

```
declare
    l_feedback_count number;
begin
    select count(*)
    into l_feedback_count
    from apex_team_feedback_followup
    where feedback_id = 123;

    for i in 1..l_feedback_count loop
        http.p(apex_util.get_feedback_follow_up (
            p_feedback_id => 123,
            p_row         => i,
            p_template    => '<br />#FOLLOW_UP# was created on #CREATED_ON# by
#CREATED_BY#') );
    end loop;
end;
/
```



## GET\_FILE Procedure

This procedure downloads files from the Oracle Application Express file repository. Please note if you are invoking this procedure during page processing, you must ensure that no page branch is invoked under the same condition, as it interferes with the file retrieval. This means that branches with any of the following conditions should not be set to fire:

- Branches with a 'When Button Pressed' attribute equal to the button that invokes the procedure.
- Branches with conditional logic defined that would succeed during page processing when the procedure is being invoked.
- As unconditional.

### Syntax

```
APEX_UTIL.GET_FILE (
    p_file_id    IN    VARCHAR2,
    p_inline     IN    VARCHAR2 DEFAULT 'NO');
```

### Parameters

**Table 25–39** GET\_FILE Parameters

Parameter	Description
p_file_id	ID in APEX_APPLICATION_FILES of the file to be downloaded. APEX_APPLICATION_FILES is a view on all files uploaded to your workspace. The following example demonstrates how to use APEX_APPLICATION_FILES:  <pre>DECLARE     l_file_id NUMBER; BEGIN     SELECT id     INTO l_file_id     FROM APEX_APPLICATION_FILES     WHERE filename = 'myxml';     --     APEX_UTIL.GET_FILE(         p_file_id =&gt; l_file_id,         p_inline  =&gt; 'YES'); END;</pre>
p_inline	Valid values include YES and NO. YES to display inline in a browser. NO to download as attachment

### Example

The following example shows how to use the GET\_FILE function to return the file identified by the ID 8675309. This is displayed inline in the browser.

```
BEGIN
    APEX_UTIL.GET_FILE(
        p_file_id => '8675309',
        p_inline  => 'YES');
END;
```

**See Also:** ["GET\\_FILE\\_ID Function"](#) on page 25-65

---

## GET\_FILE\_ID Function

This function obtains the primary key of a file in the Oracle Application Express file repository.

### Syntax

```
APEX_UTIL.GET_FILE_ID (  
    p_name    IN    VARCHAR2)  
RETURN NUMBER;
```

### Parameters

**Table 25–40** GET\_FILE\_ID Parameters

Parameter	Description
p_name	The NAME in APEX_APPLICATION_FILES of the file to be downloaded. APEX_APPLICATION_FILES is a view on all files uploaded to your workspace.

### Example

The following example shows how to use the GET\_FILE\_ID function to retrieve the database ID of the file with a filename of 'F125.sql'.

```
DECLARE  
    l_name VARCHAR2(255);  
    l_file_id NUMBER;  
BEGIN  
    SELECT name  
        INTO l_name  
    FROM APEX_APPLICATION_FILES  
    WHERE filename = 'F125.sql';  
    --  
    l_file_id := APEX_UTIL.GET_FILE_ID(p_name => l_name);  
END;
```

## GET\_FIRST\_NAME Function

This function returns the `FIRST_NAME` field stored in the named user account record.

### Syntax

```
APEX_UTIL.GET_FIRST_NAME  
    p_username IN VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 25–41** *GET\_FIRST\_NAME Parameters*

Parameter	Description
<code>p_username</code>	Identifies the user name in the account

### Example

The following example shows how to use the `GET_FIRST_NAME` function to return the `FIRST_NAME` of the user 'FRANK'.

```
DECLARE  
    VAL VARCHAR2(255);  
BEGIN  
    VAL := APEX_UTIL.GET_FIRST_NAME(p_username => 'FRANK');  
END;
```

**See Also:** ["SET\\_FIRST\\_NAME Procedure"](#) on page 25-131

---

## GET\_GROUPS\_USER\_BELONGS\_TO Function

This function returns a comma then a space separated list of group names to which the named user is a member.

### Syntax

```
APEX_UTIL.GET_GROUPS_USER_BELONGS_TO(  
    p_username IN VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 25–42** GET\_GROUPS\_USER\_BELONGS\_TO Parameters

Parameter	Description
p_username	Identifies the user name in the account

### Example

The following example shows how to use the GET\_GROUPS\_USER\_BELONGS\_TO to return the list of groups to which the user 'FRANK' is a member.

```
DECLARE  
    VAL VARCHAR2(32765);  
BEGIN  
    VAL := APEX_UTIL.GET_GROUPS_USER_BELONGS_TO(p_username => 'FRANK');  
END;
```

**See Also:** ["EDIT\\_USER Procedure"](#) on page 25-32

## GET\_GROUP\_ID Function

This function returns the numeric ID of a named group in the workspace.

### Syntax

```
APEX_UTIL.GET_GROUP_ID(  
    p_group_name IN VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 25–43** *GET\_GROUP\_ID Parameters*

Parameter	Description
<code>p_group_name</code>	Identifies the user name in the account

### Example

The following example shows how to use the `GET_GROUP_ID` function to return the ID for the group named 'Managers'.

```
DECLARE  
    VAL NUMBER;  
BEGIN  
    VAL := APEX_UTIL.GET_GROUP_ID(p_group_name => 'Managers');  
END;
```

---

## GET\_GROUP\_NAME Function

This function returns the name of a group identified by a numeric ID.

### Syntax

```
APEX_UTIL.GET_GROUP_NAME(  
    p_group_id IN NUMBER)  
RETURN VARCHAR2;
```

### Parameters

**Table 25-44** GET\_GROUP\_NAME Parameters

Parameter	Description
p_group_id	Identifies a numeric ID of a group in the workspace

### Example

The following example shows how to use the GET\_GROUP\_NAME function to return the name of the group with the ID 8922003.

```
DECLARE  
    VAL VARCHAR2(255);  
BEGIN  
    VAL := APEX_UTIL.GET_GROUP_NAME(p_group_id => 8922003);  
END;
```

## GET\_HASH Function

This function computes a hash value for all given values. Use this function to implement lost update detection for data records.

### Syntax

```
APEX_UTIL.GET_HASH (
    p_values in wwv_flow_t_varchar2,
    p_saltd in boolean default true )
RETURN VARCHAR2;
```

### Parameters

**Table 25–45** GET\_HASH Parameters

Parameter	Description
p_values	The input values.
p_saltd	If true (the default), salt hash with internal session information.

### Example

```
declare
    l_hash varchar2(4000);
begin
    select apex_util.get_hash(apex_t_varchar2 (
        empno, sal, comm ))
        into l_hash
        from emp
        where empno = :P1_EMPNO;

    if :P1_HASH <> l_hash then
        raise_application_error(-20001, 'Somebody already updated SAL/COMM');
    end if;

    update emp
        set sal = :P1_SAL,
            comm = :P1_COMM
        where empno = :P1_EMPNO;
exception when no_data_found then
    raise_application_error(-20001, 'Employee not found');
end;
```



---

## GET\_HIGH\_CONTRAST\_MODE\_TOGGLE Function

This function returns a link to the current page that enables you to turn on or off, toggle, the mode. For example, if you are in standard mode, this function displays a link that when clicked switches high contrast mode on.

### Syntax

```
APEX_UTIL.GET_HIGH_CONTRAST_MODE_TOGGLE (
    p_on_message IN VARCHAR2 DEFAULT NULL,
    p_off_message IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 25–46** *GET\_HIGH\_CONTRAST\_MODE\_TOGGLE Parameters*

Parameter	Description
p_on_message	Optional text used for the link to switch to high contrast mode, when you are in standard mode. If this parameter is not passed, the default 'Set High Contrast Mode On' text is returned in the link.
p_off_message	Optional text used for the link to switch to standard mode, when you are in high contrast mode. If this parameter is not passed, the default 'Set High Contrast Mode Off' text is returned in the link.

### Example

When running in standard mode, this function returns a link with the text 'Set High Contrast Mode On'. When the link is clicked the current page is refreshed and high contrast mode is switched on. When running in high contrast mode, a link 'Set High Contrast Mode Off' is returned. When the link is clicked the current page is refreshed and switched back to standard mode.

```
BEGIN
    http.p(apex_util.get_high_contrast_mode_toggle);
END;
```

**See Also:** ["SHOW\\_HIGH\\_CONTRAST\\_MODE\\_TOGGLE Procedure"](#) on page 25-149

---

**Note:** There are also 2 translatable system messages that can be overridden at application level to change the default link text that is returned for this toggle. They include:

- APEX.SET\_HIGH\_CONTRAST\_MODE\_OFF - Default text = Set High Contrast Mode Off
  - APEX.SET\_HIGH\_CONTRAST\_MODE\_ON - Default text = Set High Contrast Mode On
-

## GET\_LAST\_NAME Function

This function returns the LAST\_NAME field stored in the named user account record.

### Syntax

```
APEX_UTIL.GET_LAST_NAME(  
    p_username IN VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 25–47** GET\_LAST\_NAME Parameters

Parameter	Description
p_username	The user name in the user account record

### Example

The following example shows how to use the function to return the LAST\_NAME for the user 'FRANK'.

```
DECLARE  
    VAL VARCHAR2(255);  
BEGIN  
    VAL := APEX_UTIL.GET_LAST_NAME(p_username => 'FRANK');  
END;
```

**See Also:** ["SET\\_LAST\\_NAME Procedure"](#) on page 25-134

---

## GET\_NUMERIC\_SESSION\_STATE Function

This function returns a numeric value for a numeric item. You can use this function in Oracle Application Express applications wherever you can use PL/SQL or SQL. You can also use the shorthand, function NV, in place of APEX\_UTIL.GET\_NUMERIC\_SESSION\_STATE.

### Syntax

```
APEX_UTIL.GET_NUMERIC_SESSION_STATE (  
    p_item      IN VARCHAR2)  
RETURN NUMBER;
```

### Parameters

**Table 25-48** GET\_NUMERIC\_SESSION\_STATE Parameters

Parameter	Description
p_item	The case insensitive name of the item for which you want to have the session state fetched

### Example

The following example shows how to use the function to return the numeric value stored in session state for the item 'my\_item'.

```
DECLARE  
    l_item_value    NUMBER;  
BEGIN  
    l_item_value := APEX_UTIL.GET_NUMERIC_SESSION_STATE('my_item');  
END;
```

**See Also:** ["GET\\_SESSION\\_STATE Function"](#) on page 25-82 and ["SET\\_SESSION\\_STATE Procedure"](#) on page 25-145

---

## GET\_PREFERENCE Function

This function retrieves the value of a previously saved preference for a given user.

### Syntax

```
APEX_UTIL.GET_PREFERENCE (  
    p_preference IN    VARCHAR2 DEFAULT NULL,  
    p_user      IN    VARCHAR2 DEFAULT V('USER'))  
RETURN VARCHAR2;
```

### Parameters

**Table 25–49** GET\_PREFERENCE Parameters

Parameter	Description
p_preference	Name of the preference to retrieve the value
p_value	Value of the preference
p_user	User for whom the preference is being retrieved

### Example

The following example shows how to use the GET\_PREFERENCE function to return the value for the currently authenticated user's preference named default\_view.

```
DECLARE  
    l_default_view    VARCHAR2(255);  
BEGIN  
    l_default_view := APEX_UTIL.GET_PREFERENCE(  
        p_preference => 'default_view',  
        p_user      => :APP_USER);  
END;
```

**See Also:** ["SET\\_PREFERENCE Procedure"](#) on page 25-135, ["REMOVE\\_PREFERENCE Procedure"](#) on page 25-115 and ["Managing User Preferences"](#) in *Oracle Application Express Administration Guide*.

---

## GET\_PRINT\_DOCUMENT Function Signature 1

This function returns a document as BLOB using XML based report data and RTF or XSL-FO based report layout.

### Syntax

```
APEX_UTIL.GET_PRINT_DOCUMENT (
    p_report_data          IN BLOB,
    p_report_layout       IN CLOB,
    p_report_layout_type  IN VARCHAR2 default 'xsl-fo',
    p_document_format     IN VARCHAR2 default 'pdf',
    p_print_server        IN VARCHAR2 default NULL)
RETURN BLOB;
```

### Parameters

**Table 25–50** GET\_PRINT\_DOCUMENT Signature 1 Parameters

Parameter	Description
p_report_data	XML based report data
p_report_layout	Report layout in XSL-FO or RTF format
p_report_layout_type	Defines the report layout type, that is "xsl-fo" or "rtf"
p_document_format	Defines the document format, that is "pdf", "rtf", "xls", "htm", or "xml"
p_print_server	URL of the print server. If not specified, the print server is derived from preferences.

For a GET\_PRINT\_DOCUMENT example see "[GET\\_PRINT\\_DOCUMENT Function Signature 4](#)".

---

## GET\_PRINT\_DOCUMENT Function Signature 2

This function returns a document as BLOB using pre-defined report query and pre-defined report layout.

### Syntax

```
APEX_UTIL.GET_PRINT_DOCUMENT (
    p_application_id      IN NUMBER,
    p_report_query_name   IN VARCHAR2,
    p_report_layout_name  IN VARCHAR2 default null,
    p_report_layout_type  IN VARCHAR2 default 'xsl-fo',
    p_document_format     IN VARCHAR2 default 'pdf' ,
    p_print_server        IN VARCHAR2 default null)
RETURN BLOB;
```

### Parameters

**Table 25–51** GET\_PRINT\_DOCUMENT Signature 2 Parameters

Parameter	Description
p_application_id	Defines the application ID of the report query
p_report_query_name	Name of the report query (stored under application's shared components)
p_report_layout_name	Name of the report layout (stored under application's Shared Components)
p_report_layout_type	Defines the report layout type, that is "xsl-fo" or "rtf"
p_document_format	Defines the document format, that is "pdf", "rtf", "xls", "htm", or "xml"
p_print_server	URL of the print server. If not specified, the print server is derived from preferences.

For a GET\_PRINT\_DOCUMENT example see ["GET\\_PRINT\\_DOCUMENT Function Signature 4"](#).

## GET\_PRINT\_DOCUMENT Function Signature 3

This function returns a document as BLOB using a pre-defined report query and RTF or XSL-FO based report layout.

### Syntax

```
APEX_UTIL.GET_PRINT_DOCUMENT (
    p_application_id      IN NUMBER,
    p_report_query_name   IN VARCHAR2,
    p_report_layout       IN CLOB,
    p_report_layout_type  IN VARCHAR2 default 'xsl-fo',
    p_document_format     IN VARCHAR2 default 'pdf',
    p_print_server        IN VARCHAR2 default null)
RETURN BLOB;
```

### Parameters

**Table 25–52** GET\_PRINT\_DOCUMENT Signature 3 Parameters

Parameter	Description
p_application_id	Defines the application ID of the report query
p_report_query_name	Name of the report query (stored under application's shared components)
p_report_layout	Defines the report layout in XSL-FO or RTF format
p_report_layout_type	Defines the report layout type, that is "xsl-fo" or "rtf"
p_document_format	Defines the document format, that is "pdf", "rtf", "xls", "htm", or "xml"
p_print_server	URL of the print server. If not specified, the print server is derived from preferences.

For a GET\_PRINT\_DOCUMENT example see ["GET\\_PRINT\\_DOCUMENT Function Signature 4"](#).

## GET\_PRINT\_DOCUMENT Function Signature 4

This function returns a document as BLOB using XML based report data and RTF or XSL-FO based report layout.

### Syntax

```
APEX_UTIL.GET_PRINT_DOCUMENT (
    p_report_data          IN CLOB,
    p_report_layout       IN CLOB,
    p_report_layout_type  IN VARCHAR2 default 'xsl-fo',
    p_document_format     IN VARCHAR2 default 'pdf',
    p_print_server        IN VARCHAR2 default NULL)
RETURN BLOB;
```

### Parameters

**Table 25–53** GET\_PRINT\_DOCUMENT Signature 4 Parameters

Parameter	Description
p_report_data	XML based report data, must be encoded in UTF-8
p_report_layout	Report layout in XSL-FO or RTF format
p_report_layout_type	Defines the report layout type, that is "xsl-fo" or "rtf"
p_document_format	Defines the document format, that is "pdf", "rtf", "xls", "htm", or "xml"
p_print_server	URL of the print server. If not specified, the print server is derived from preferences

### Example for Signature 4

The following example shows how to use the GET\_PRINT\_DOCUMENT using Signature 4 (Document returns as a BLOB using XML based report data and RTF or XSL-FO based report layout). In this example, GET\_PRINT\_DOCUMENT is used with APEX\_MAIL.SEND and APEX\_MAIL.ADD\_ATTACHMENT to send an email with an attachment of the file returned by GET\_PRINT\_DOCUMENT. Both the report data and layout are taken from values stored in page items (P1\_XML and P1\_XSL).

```
DECLARE
    l_id number;
    l_document BLOB;
BEGIN
    l_document := APEX_UTIL.GET_PRINT_DOCUMENT (
        p_report_data          => :P1_XML,
        p_report_layout       => :P1_XSL,
        p_report_layout_type  => 'xsl-fo',
        p_document_format     => 'pdf');

    l_id := APEX_MAIL.SEND(
        p_to          => :P35_MAIL_TO,
        p_from        => 'noreplies@oracle.com',
        p_subj        => 'sending PDF by using print API',
        p_body         => 'Please review the attachment.',
        p_body_html   => 'Please review the attachment');

    APEX_MAIL.ADD_ATTACHMENT (
```



```
p_mail_id    => l_id,  
p_attachment => l_document,  
p_filename   => 'mydocument.pdf',  
p_mime_type  => 'application/pdf');  
END;
```

---

## GET\_SCREEN\_READER\_MODE\_TOGGLE Function

This function returns a link to the current page to turn on or off, toggle, the mode. For example, if you are in standard mode, this function displays a link that when clicked switches screen reader mode on.

### Syntax

```
APEX_UTIL.GET_SCREEN_READER_MODE_TOGGLE (  
    p_on_message IN VARCHAR2 DEFAULT NULL,  
    p_off_message IN VARCHAR2 DEFAULT NULL)  
RETURN VARCHAR2;
```

### Parameters

**Table 25–54** GET\_SCREEN\_READER\_MODE\_TOGGLE Parameters

Parameter	Description
p_on_message	Optional text used for the link to switch to screen reader mode, when you are in standard mode. If this parameter is not passed, the default 'Set Screen Reader Mode On' text is returned in the link.
p_off_message	Optional text used for the link to switch to standard mode, when you are in screen reader mode. If this parameter is not passed, the default 'Set Screen Reader Mode Off' text is returned in the link.

### Example

When running in standard mode, this function returns a link with the text 'Set Screen Reader Mode On'. When the link is clicked the current page is refreshed and screen reader mode is switched on. When running in screen reader mode, a link 'Set Screen Reader Mode Off' is returned. When the link is clicked the current page is refreshed and switched back to standard mode.

```
BEGIN  
    http.p(apex_util.get_screen_reader_mode_toggle);  
END;
```

**See Also:** ["SHOW\\_SCREEN\\_READER\\_MODE\\_TOGGLE Procedure"](#) on page 25-150

## GET\_SESSION\_LANG Function

This function returns the language setting for the current user in the current Application Express session.

### Syntax

```
APEX_UTIL.GET_SESSION_LANG  
RETURN VARCHAR2;
```

### Parameters

None.

### Example

The following example shows how to use the GET\_SESSION\_LANG function. It returns the session language for the current user in the current Application Express session into a local variable.

```
DECLARE  
    VAL VARCHAR2(5);  
BEGIN  
    VAL := APEX_UTIL.GET_SESSION_LANG;  
END;
```

## GET\_SESSION\_STATE Function

This function returns the value for an item. You can use this function in your Oracle Application Express applications wherever you can use PL/SQL or SQL. You can also use the shorthand, function `v`, in place of `APEX_UTIL.GET_SESSION_STATE`.

### Syntax

```
APEX_UTIL.GET_SESSION_STATE (  
    p_item    IN    VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 25–55** *GET\_SESSION\_STATE Parameters*

Parameter	Description
<code>p_item</code>	The case insensitive name of the item for which you want to have the session state fetched

### Example

The following example shows how to use the `GET_SESSION_STATE` function to return the value stored in session state for the item 'my\_item'.

```
DECLARE  
    l_item_value VARCHAR2(255);  
BEGIN  
    l_item_value := APEX_UTIL.GET_SESSION_STATE('my_item');  
END;
```

**See Also:** ["GET\\_NUMERIC\\_SESSION\\_STATE Function"](#) on page 25-73 and ["SET\\_SESSION\\_STATE Procedure"](#) on page 25-145

## GET\_SESSION\_TERRITORY Function

This function returns the territory setting for the current user in the current Application Express session.

### Syntax

```
APEX_UTIL.GET_SESSION_TERRITORY  
RETURN VARCHAR2;
```

### Parameters

None.

### Example

The following example shows how to use the GET\_SESSION\_TERRITORY function. It returns the session territory setting for the current user in the current Application Express session into a local variable.

```
DECLARE  
    VAL VARCHAR2(30);  
BEGIN  
    VAL := APEX_UTIL.GET_SESSION_TERRITORY;  
END;
```

## GET\_SESSION\_TIME\_ZONE Function

This function returns the time zone for the current user in the current Application Express session. This value is null if the time zone is not explicitly set by using `APEX_UTIL.SET_SESSION_TIME_ZONE` or if an application's automatic time zone attribute is enabled.

### Syntax

```
APEX_UTIL.GET_SESSION_TIME_ZONE  
RETURN VARCHAR2;
```

### Parameters

None.

### Example

The following example shows how to use the `GET_SESSION_TIME_ZONE` function. It returns the session time zone for the current user in the current Application Express session into a local variable.

```
BEGIN  
    VAL := APEX_UTIL.GET_SESSION_TIME_ZONE;  
END;
```

---

## GET\_SINCE Function

This function returns the relative date in words (for example, two days from now, thirty minutes ago). This function is equivalent to using the SINCE format mask available within Oracle Application Express and is useful within PL/SQL routines.

### Syntax

```
APEX_UTIL.GET_SINCE (
    p_date date )
RETURN VARCHAR2;
```

### Parameters

**Table 25–56** GET\_SINCE Parameters

Parameter	Description
p_date	The date you want formatted.

### Example

```
begin
    for c1 in (
        select application_id, application_name,
               apex_util.get_since(last_updated_on) last_update
        from apex_applications
        order by application_id)
    loop
        htp.p('Application: ' || to_char(c1.application_id) || ' - ' ||
              c1.application_name || '<br/>');
        htp.p('Last Updated: ' || c1.last_update);
    end loop;
end;
```

## GET\_USER\_ID Function

This function returns the numeric ID of a named user in the workspace.

### Syntax

```
APEX_UTIL.GET_USER_ID(  
    p_username    IN VARCHAR2)  
RETURN NUMBER;
```

### Parameters

**Table 25–57** GET\_USER\_ID Parameters

Parameter	Description
p_username	Identifies the name of a user in the workspace

### Example

The following example shows how to use the GET\_USER\_ID function to return the ID for the user named 'FRANK'.

```
DECLARE  
    VAL NUMBER;  
BEGIN  
    VAL := APEX_UTIL.GET_USER_ID(p_username => 'FRANK');  
END;
```



---

## GET\_USER\_ROLES Function

This function returns the DEVELOPER\_ROLE field stored in the named user account record. Please note that currently this parameter is named inconsistently between the CREATE\_USER, EDIT\_USER and FETCH\_USER APIs, although they all relate to the DEVELOPER\_ROLE field. CREATE\_USER uses p\_developer\_privs, EDIT\_USER uses p\_developer\_roles and FETCH\_USER uses p\_developer\_role.

### Syntax

```
APEX_UTIL.GET_USER_ROLES (
    p_username IN VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 25–58** GET\_USER\_ROLES Parameters

Parameter	Description
p_username	Identifies a user name in the account

### Example

The following example shows how to use the GET\_USER\_ROLES function to return colon separated list of roles stored in the DEVELOPER\_ROLE field for the user 'FRANK'.

```
DECLARE
    VAL VARCHAR2(4000);
BEGIN
    VAL := APEX_UTIL.GET_USER_ROLES(p_username=>'FRANK');
END;
```

## GET\_USERNAME Function

This function returns the user name of a user account identified by a numeric ID.

### Syntax

```
APEX_UTIL.GET_USERNAME(  
    p_userid IN NUMBER)  
RETURN VARCHAR2;
```

### Parameters

**Table 25–59** GET\_USERNAME Parameters

Parameter	Description
p_userid	Identifies the numeric ID of a user account in the workspace

### Example

The following example shows how to use the GET\_USERNAME function to return the user name for the user with an ID of 228922003.

```
DECLARE  
    VAL VARCHAR2(100);  
BEGIN  
    VAL := APEX_UTIL.GET_USERNAME(p_userid => 228922003);  
END;
```

**See Also:** ["SET\\_USERNAME Procedure"](#) on page 25-148

## HOST\_URL Function

This function returns the URL to the Application Express instance, depending on the option passed.

### Syntax

```
APEX_UTIL.HOST_URL (
    p_option IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 25–60** *HOST\_URL Parameters*

Parameter	Description
p_option	Specifies the parts of the URL to include.  Possible values for p_option include: <ul style="list-style-type: none"> <li>■ NULL - Return URL up to port number. For example: http://myserver.com:7778</li> <li>■ SCRIPT - Return URL to include script name. For example: https://myserver.com:7778/pls/apex/</li> <li>■ IMGPRE - Return URL to include image prefix. For example: https://myserver.com:7778/i/</li> </ul>

### Example

The following example demonstrates how to use the HOST\_URL function to return the URL, including the script name, to the current Application Express instance.

```
declare
    l_host_url    varchar2(4000);
    l_url        varchar2(4000);
    l_application varchar2(30) := 'f?p=100:1';
    l_email_body  varchar2(32000);
begin
    l_host_url := apex_util.host_url('SCRIPT');
    l_url := l_host_url||l_application;
    l_email_body := 'The URL to the application is: '||l_url;
end;
```

## HTML\_PCT\_GRAPH\_MASK Function

Use this function to scale a graph. This function can also be used by classic and interactive reports with format mask of GRAPH. This generates a <div> tag with inline styles.

### Syntax

```
APEX_UTIL.HTML_PCT_GRAPH_MASK (
    p_number          IN NUMBER    DEFAULT NULL,
    p_size            IN NUMBER    DEFAULT 100,
    p_background      IN VARCHAR2  DEFAULT NULL,
    p_bar_background  IN VARCHAR2  DEFAULT NULL,
    p_format          IN VARCHAR2  DEFAULT NULL)
RETURN VARCHAR2;
```

### Parameters

**Table 25–61 HTML\_PCT\_GRAPH\_MASK Parameters**

Parameter	Description
p_number	Number between 0 and 100.
p_size	Width of graph in pixels.
p_background	Six character hexadecimal background color of chart bar (not bar color)
p_bar_background	Six character hexadecimal background color of chart bar (bar color)
p_format	<p>If this parameter is supplied, p_size, p_background and p_bar_background are ignored.</p> <p>This parameter uses the following format:</p> <p>PCT_GRAPH:&lt;BACKGROUND&gt;:&lt;FOREGROUND&gt;:&lt;CHART_WIDTH&gt;</p> <p>position 1: PCT_GRAPH format mask indicator</p> <p>position 2: Background color in hexadecimal, 6 characters (optional)</p> <p>position 3: Foreground "bar" color in hexadecimal, 6 characters (optional)</p> <p>position 4: Chart width in pixels. Numeric and defaults to 100.</p> <p>p_number is automatically scaled so that 50 is half of chart_width (optional).</p>

### Example

The following is an SQL example.

```
select apex_util.html_pct_graph_mask(33) from dual
```

The following is a report numeric column format mask example.

```
PCT_GRAPH:777777:111111:200
```

## INCREMENT\_CALENDAR Procedure

Use this procedure to navigate to the next set of days in the calendar. Depending on what the calendar view is, this procedure navigates to the next month, week or day. If it is a Custom Calendar the total number of days between the start date and end date are navigated.

### Syntax

```
APEX_UTIL.INCREMENT_CALENDAR;
```

### Parameter

None.

### Example

In this example, if you create a button called NEXT in the Calendar page and create a process that fires when the create button is clicked the following code navigates the calendar.

```
APEX_UTIL.INCREMENT_CALENDAR
```

---

## IR\_CLEAR Procedure [DEPRECATED]

---



---

**Note:** The use of this procedure is not recommended. This procedure has been replaced by the procedure in APEX\_IR.

---



---

This procedure clears report settings.

---



---

**Note:** This procedure should be used only in a page submit process.

---



---

### Syntax

```
APEX_UTIL.IR_CLEAR(
  p_page_id IN NUMBER,
  p_report_alias IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 25–62** IR\_CLEAR Parameters

Parameter	Description
p_page_id	Page of the current Application Express application that contains an interactive report.
p_report_alias	Identifies the saved report alias within the current application page. To clear a Primary report, p_report_alias must be 'PRIMARY' or leave as NULL. To clear a saved report, p_report_alias must be the name of the saved report. For example, to clear report '1234', p_report_alias must be '1234'.

### Example

The following example shows how to use the IR\_CLEAR procedure to clear Interactive report settings with alias of '8101021' in page 1 of the current application.

```
BEGIN
  APEX_UTIL.IR_CLEAR(
    p_page_id      => 1,
    p_report_alias => '8101021'
  );
END;
```

---

## IR\_DELETE\_REPORT Procedure [DEPRECATED]

---

---

---

**Note:** The use of this procedure is not recommended. This procedure has been replaced by the procedure in APEX\_IR.

---

---

This procedure deletes saved Interactive reports. It deletes all saved reports except the Primary Default report.

### Syntax

```
APEX_UTIL.IR_DELETE_REPORT(  
    p_report_id IN NUMBER);
```

### Parameters

**Table 25–63** IR\_DELETE\_REPORT Parameters

Parameter	Description
p_report_id	Report ID to delete within the current Application Express application.

### Example

The following example shows how to use the IR\_DELETE\_REPORT procedure to delete the saved Interactive report with ID of '880629800374638220' in the current application.

```
BEGIN  
    APEX_UTIL.IR_DELETE_REPORT(  
        p_report_id => '880629800374638220');  
END;
```

---

## IR\_DELETE\_SUBSCRIPTION Procedure [DEPRECATED]

---

---

**Note:** The use of this procedure is not recommended. This procedure has been replaced by the procedure in APEX\_IR.

---

---

This procedure deletes Interactive subscriptions.

### Syntax

```
APEX_UTIL.IR_DELETE_SUBSCRIPTION(  
    p_subscription_id IN NUMBER);
```

### Parameters

**Table 25–64** IR\_DELETE\_SUBSCRIPTION Parameters

Parameter	Description
p_subscription_id	Subscription ID to delete within the current workspace.

### Example

The following example shows how to use the IR\_DELETE\_SUBSCRIPTION procedure to delete the subscription with ID of '880629800374638220' in the current workspace.

```
BEGIN  
    APEX_UTIL.IR_DELETE_SUBSCRIPTION(  
        p_subscription_id => '880629800374638220');  
END;
```



---

## IR\_FILTER Procedure [DEPRECATED]

---

**Note:** The use of this procedure is not recommended. This procedure has been replaced by the procedure in APEX\_IR.

---

This procedure creates a filter on an interactive report.

---

**Note:** This procedure should be used only in a page submit process.

---

### Syntax

```
APEX_UTIL.IR_FILTER(
  p_page_id      IN NUMBER,
  p_report_column IN VARCHAR2,
  p_operator_abbr IN VARCHAR2 DEFAULT NULL,
  p_filter_value  IN VARCHAR2,
  p_report_alias  IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 25–65** IR\_FILTER Parameters

Parameter	Description
p_page_id	Page of the current Application Express application that contains an interactive report.
p_report_column	Name of the report SQL column, or column alias, to be filtered.
p_operator_abbr	Filter type. Valid values are as follows: <ul style="list-style-type: none"> <li>▪ EQ = Equals</li> <li>▪ NEQ = Not Equals</li> <li>▪ LT = Less than</li> <li>▪ LTE = Less then or equal to</li> <li>▪ GT = Greater Than</li> <li>▪ GTE = Greater than or equal to</li> <li>▪ LIKE = SQL Like operator</li> <li>▪ N = Null</li> <li>▪ NN = Not Null</li> <li>▪ C = Contains</li> <li>▪ NC = Not Contains</li> <li>▪ IN = SQL In Operator</li> <li>▪ NIN = SQL Not In Operator</li> </ul>
p_filter_value	Filter value. This value is not used for 'N' and 'NN'.
p_report_alias	Identifies the saved report alias within the current application page. To create a filter on a Primary report, p_report_alias must be 'PRIMARY' or leave as NULL. To create a filter on a saved report, p_report_alias must be the name of the saved report. For example, to create a filter on report '1234', p_report_alias must be '1234'.

---

**Example**

The following example shows how to use the IR\_FILTER procedure to filter Interactive report with alias of '8101021' in page 1 of the current application with DEPTNO equals 30.

```
BEGIN
  APEX_UTIL.IR_FILTER (
    p_page_id      => 1,
    p_report_column => 'DEPTNO',
    p_operator_abbr => 'EQ',
    p_filter_value  => '30',
    p_report_alias  => '8101021'
  );
END;
```

## IR\_RESET Procedure [DEPRECATED]

---



---

**Note:** The use of this procedure is not recommended. This procedure has been replaced by the procedure in APEX\_IR.

---



---

This procedure resets report settings back to the default report settings. Resetting a report removes any customizations you have made.

---



---

**Note:** This procedure should be used only in a page submit process.

---



---

### Syntax

```
APEX_UTIL.IR_RESET(
  p_page_id IN NUMBER,
  p_report_alias IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 25–66** IR\_RESET Parameters

Parameter	Description
p_page_id	Page of the current Application Express application that contains an interactive report.
p_report_alias	Identifies the saved report alias within the current application page. To reset a Primary report, p_report_alias must be 'PRIMARY' or leave as NULL. To reset a saved report, p_report_alias must be the name of the saved report. For example, to reset report '1234', p_report_alias must be '1234'.

### Example

The following example shows how to use the IR\_RESET procedure to reset Interactive report settings with alias of '8101021' in page 1 of the current application.

```
BEGIN
  APEX_UTIL.IR_RESET(
    p_page_id      => 1,
    p_report_alias => '8101021'
  );
END;
```

## IS\_HIGH\_CONTRAST\_SESSION Function

This function returns a boolean TRUE if the session is in high contrast mode and returns a boolean FALSE if not in high contrast mode.

### Syntax

```
APEX_UTIL.IS_HIGH_CONTRAST_SESSION  
RETURN BOOLEAN;
```

### Parameters

None.

### Example

In this example, if the current session is running in high contrast mode, a high contrast specific CSS file 'my\_app\_hc.css' is added to the HTML output of the page.

```
BEGIN  
  IF apex_util.is_high_contrast_session THEN  
    apex_css.add_file (  
      p_name => 'my_app_hc');  
  END IF;  
END;
```

## IS\_HIGH\_CONTRAST\_SESSION\_YN Function

This function returns Y if the session is in high contrast mode and N if not in high contrast mode.

### Syntax

```
APEX_UTIL.IS_HIGH_CONTRAST_SESSION_YN  
RETURN VARCHAR2;
```

### Parameters

None.

### Example

In this example, if the current session is running in high contrast mode, a high contrast specific CSS file, `my_app_hc.css`, is added to the HTML output of the page.

```
BEGIN  
  IF apex_util.is_high_contrast_session_yn = 'Y' THEN  
    apex_css.add_file (  
      p_name => 'my_app_hc');  
  END IF;  
END;
```

## IS\_LOGIN\_PASSWORD\_VALID Function

This function returns a Boolean result based on the validity of the password for a named user account in the current workspace. This function returns TRUE if the password matches and it returns FALSE if the password does not match.

### Syntax

```
APEX_UTIL.IS_LOGIN_PASSWORD_VALID(  
    p_username IN VARCHAR2,  
    p_password IN VARCHAR2)  
RETURN BOOLEAN;
```

### Parameters

**Table 25–67** IS\_LOGIN\_PASSWORD\_VALID Parameters

Parameter	Description
p_username	User name in account
p_password	Password to be compared with password stored in the account

### Example

The following example shows how to use the IS\_LOGIN\_PASSWORD\_VALID function to check if the user 'FRANK' has the password 'tiger'. TRUE is returned if this is a valid password for 'FRANK', FALSE is returned if not.

```
DECLARE  
    VAL BOOLEAN;  
BEGIN  
    VAL := APEX_UTIL.IS_LOGIN_PASSWORD_VALID (  
        p_username=>'FRANK',  
        p_password=>'tiger');  
END;
```

## IS\_SCREEN\_READER\_SESSION Function

This function returns a boolean TRUE if the session is in screen reader mode and returns a boolean FALSE if not in screen reader mode.

### Syntax

```
APEX_UTIL.IS_SCREEN_READER_SESSION  
RETURN BOOLEAN;
```

### Parameters

None

### Example

```
BEGIN  
  IF apex_util.is_screen_reader_session then  
    http.p('Screen Reader Mode');  
  END IF;  
END;
```

## IS\_SCREEN\_READER\_SESSION\_YN Function

This function returns 'Y' if the session is in screen reader mode and 'N' if not in screen reader mode.

### Syntax

```
APEX_UTIL.IS_SCREEN_READER_SESSION_YN  
RETURN VARCHAR2;
```

### Parameters

None

### Example

```
BEGIN  
    IF apex_util.is_screen_reader_session_yn = 'Y' then  
        http.p('Screen Reader Mode');  
    END IF;  
END;
```



---

## IS\_USERNAME\_UNIQUE Function

This function returns a Boolean result based on whether the named user account is unique in the workspace.

### Syntax

```
APEX_UTIL.IS_USERNAME_UNIQUE(  
    p_username IN VARCHAR2)  
RETURN BOOLEAN;
```

### Parameters

**Table 25–68** IS\_USERNAME\_UNIQUE Parameters

Parameter	Description
p_username	Identifies the user name to be tested

### Example

The following example shows how to use the IS\_USERNAME\_UNIQUE function. If the user 'FRANK' already exists in the current workspace, FALSE is returned, otherwise TRUE is returned.

```
DECLARE  
    VAL BOOLEAN;  
BEGIN  
    VAL := APEX_UTIL.IS_USERNAME_UNIQUE(  
        p_username=>'FRANK');  
END;
```

## KEYVAL\_NUM Function

This function gets the value of the package variable (`wwv_flow_utilities.g_val_num`) set by `APEX_UTIL.SAVEKEY_NUM`.

### Syntax

```
APEX_UTIL.KEYVAL_NUM  
RETURN NUMBER;
```

### Parameters

None

### Example

The following example shows how to use the `KEYVAL_NUM` function to return the current value of the package variable `wwv_flow_utilities.g_val_num`.

```
DECLARE  
    VAL NUMBER;  
BEGIN  
    VAL := APEX_UTIL.KEYVAL_NUM;  
END;
```

**See Also:** ["SAVEKEY\\_NUM Function"](#) on page 25-121

## KEYVAL\_VC2 Function

This function gets the value of the package variable (`wwv_flow_utilities.g_val_vc2`) set by `APEX_UTIL.SAVEKEY_VC2`.

### Syntax

```
APEX_UTIL.KEYVAL_VC2;
```

### Parameters

None.

### Example

The following example shows how to use the `KEYVAL_VC2` function to return the current value of the package variable `wwv_flow_utilities.g_val_vc2`.

```
DECLARE
    VAL VARCHAR2(4000);
BEGIN
    VAL := APEX_UTIL.KEYVAL_VC2;
END;
```

**See Also:** ["SAVEKEY\\_VC2 Function"](#) on page 25-122

---

## LOCK\_ACCOUNT Procedure

Sets a user account status to locked. Must be run by an authenticated workspace administrator in the context of a page request.

### Syntax

```
APEX_UTIL.LOCK_ACCOUNT (  
    p_user_name IN VARCHAR2);
```

### Parameters

**Table 25–69** LOCK\_ACCOUNT Parameters

Parameter	Description
p_user_name	The user name of the user account

### Example

The following example shows how to use the LOCK\_ACCOUNT procedure. Use this procedure to lock an Application Express account (workspace administrator, developer, or end user) in the current workspace. This action locks the account for use by administrators, developers, and end users.

```
BEGIN  
    FOR c1 IN (SELECT user_name from wwv_flow_users) LOOP  
        APEX_UTIL.LOCK_ACCOUNT(p_user_name => c1.user_name);  
        http.p('End User Account: '||c1.user_name||' is now locked.');
```

```
    END LOOP;  
END;
```

**See Also:** ["UNLOCK\\_ACCOUNT Procedure"](#) on page 25-163 and ["GET\\_ACCOUNT\\_LOCKED\\_STATUS Function"](#) on page 25-51

## PASSWORD\_FIRST\_USE\_OCCURRED Function

Returns TRUE if the account's password has changed since the account was created, an Oracle Application Express administrator performs a password reset operation that results in a new password being emailed to the account holder, or a user has initiated password reset operation. This function returns FALSE if the account's password has not been changed since either of the events just described.

This function may be run in a page request context by any authenticated user.

### Syntax

```
APEX_UTIL.PASSWORD_FIRST_USE_OCCURRED (
    p_user_name IN VARCHAR2)
RETURN BOOLEAN;
```

### Parameters

**Table 25–70** *PASSWORD\_FIRST\_USE\_OCCURRED Parameters*

Parameter	Description
p_user_name	The user name of the user account

### Example

The following example shows how to use the PASSWORD\_FIRST\_USE\_OCCURRED function. Use this function to check if the password for an Application Express user account (workspace administrator, developer, or end user) in the current workspace has been changed by the user the first time the user logged in after the password was initially set during account creation, or was changed by one of the password reset operations described above.

This is meaningful only with accounts for which the CHANGE\_PASSWORD\_ON\_FIRST\_USE attribute is set to **Yes**.

```
BEGIN
    FOR c1 IN (SELECT user_name from wwv_flow_users) LOOP
        IF APEX_UTIL.PASSWORD_FIRST_USE_OCCURRED(p_user_name => c1.user_name) THEN
            http.p('User: ' || c1.user_name || ' has logged in and updated the
password. ');
        END IF;
    END LOOP;
END;
```

**See Also:** ["CHANGE\\_PASSWORD\\_ON\\_FIRST\\_USE Function"](#) on page 25-12

## PREPARE\_URL Function

The PREPARE\_URL function serves two purposes:

1. To return an f?p URL with the Session State Protection checksum argument (&cs=) if one is required.
2. To return an f?p URL with the session ID component replaced with zero (0) if the zero session ID feature is in use and other criteria are met.

---

**Note:** The PREPARE\_URL functions returns the f?p URL with &cs=<large hex value> appended. If you use this returned value, for example in JavaScript, it may be necessary to escape the ampersand in the URL to conform with syntax rules of the particular context. One place you may encounter this is in SVG chart SQL queries which might include PREPARE\_URL calls.

---

### Syntax

```
APEX_UTIL.PREPARE_URL (
    p_url          IN VARCHAR2,
    p_url_charset  IN VARCHAR2 default null,
    p_checksum_type IN VARCHAR2 default null,
    p_triggering_element IN VARCHAR2 default 'this')
RETURN VARCHAR2;
```

### Parameters

**Table 25–71** PREPARE\_URL Parameters

Parameter	Description
p_url	An f?p relative URL with all substitutions resolved
p_url_charset	The character set name (for example, UTF-8) to use when escaping special characters contained within argument values
p_checksum_type	Null or any of the following six values, SESSION or 3, PRIVATE_BOOKMARK or 2, or PUBLIC_BOOKMARK or 1
p_triggering_element	A jQuery selector (for example, \$('#my_button') , where my_button is the static ID for a button element), to identify which element to use to trigger the dialog. This is required for Modal Dialog support.

### Example 1

The following example shows how to use the PREPARE\_URL function to return a URL with a valid 'SESSION' level checksum argument. This URL sets the value of P1\_ITEM page item to xyz.

```
DECLARE
    l_url varchar2(2000);
    l_app number := v('APP_ID');
    l_session number := v('APP_SESSION');
BEGIN
    l_url := APEX_UTIL.PREPARE_URL(
        p_url => 'f?p=' || l_app || ':1:' || l_session || '::NO::P1_ITEM:xyz',
        p_checksum_type => 'SESSION');
END;
```

## Example 2

The following example shows how to use the `PREPARE_URL` function to return a URL with a zero session ID. In a PL/SQL Dynamic Content region that generates f?p URLs (anchors), call `PREPARE_URL` to ensure that the session ID is set to zero when the zero session ID feature is in use, when the user is a public user (not authenticated), and when the target page is a public page in the current application:

```
http.p(APEX_UTIL.PREPARE_URL(p_url => 'f?p=' || :APP_ID || ':10:' || :APP_SESSION  
|| '::NO::P10_ITEM:ABC');
```

When using `PREPARE_URL` for this purpose, the `p_url_charset` and `p_checksum_type` arguments can be omitted. However, it is permissible to use them when both the Session State Protection and Zero Session ID features are applicable.

**See Also:** "Facilitating Bookmarks by Using Zero as the Session ID"

## PUBLIC\_CHECK\_AUTHORIZATION Function [DEPRECATED]

---

---

---

**Note:** Use the ["IS\\_AUTHORIZED Function"](#) on page 4-3 instead of this deprecated function.

---

---

Given the name of a authorization scheme, this function determines if the current user passes the security check.

### Syntax

```
APEX_UTIL.PUBLIC_CHECK_AUTHORIZATION (  
    p_security_scheme IN VARCHAR2)  
RETURN BOOLEAN;
```

### Parameters

**Table 25–72** PUBLIC\_CHECK\_AUTHORIZATION Parameters

Parameter	Description
p_security_name	The name of the authorization scheme that determines if the user passes the security check

### Example

The following example shows how to use the PUBLIC\_CHECK\_AUTHORIZATION function to check if the current user passes the check defined in the my\_auth\_scheme authorization scheme.

```
DECLARE  
    l_check_security BOOLEAN;  
BEGIN  
    l_check_security := APEX_UTIL.PUBLIC_CHECK_AUTHORIZATION('my_auth_scheme');  
END;
```



---

## PURGE\_REGIONS\_BY\_APP Procedure

Deletes all cached regions for an application.

### Syntax

```
APEX_UTIL.PURGE_REGIONS_BY_APP (  
    p_application IN NUMBER);
```

### Parameters

**Table 25–73** *PURGE\_REGIONS\_BY\_APP Parameters*

Parameter	Description
p_application	The identification number (ID) of the application.

### Example

The following example show how to use APEX\_UTIL.PURGE\_REGIONS\_BY\_APP to delete all cached regions for application #123.

```
BEGIN  
    APEX_UTIL.PURGE_REGIONS_BY_APP(p_application=>123);  
END;
```

## PURGE\_REGIONS\_BY\_NAME Procedure

Deletes all cached values for a region identified by the application ID, page number and region name.

### Syntax

```
APEX_UTIL.PURGE_REGIONS_BY_NAME (  
    p_application IN NUMBER,  
    p_page        IN NUMBER,  
    p_region_name IN VARCHAR2);
```

### Parameters

**Table 25–74** *PURGE\_REGIONS\_BY\_NAME Parameters*

Parameter	Description
p_application	The identification number (ID) of the application.
p_page	The number of the page containing the region to be deleted.
p_region_name	The region name to be deleted.

### Example

The following example shows how to use the `PURGE_REGIONS_BY_NAME` procedure to delete all the cached values for the region 'my\_cached\_region' on page 1 of the current application.

```
BEGIN  
    APEX_UTIL.PURGE_REGIONS_BY_NAME(  
        p_application => :APP_ID,  
        p_page => 1,  
        p_region_name => 'my_cached_region');  
END;
```

---

## PURGE\_REGIONS\_BY\_PAGE Procedure

Deletes all cached regions by application and page.

### Syntax

```
APEX_UTIL.PURGE_REGIONS_BY_PAGE (  
    p_application IN NUMBER,  
    p_page       IN NUMBER);
```

### Parameters

**Table 25–75** *PURGE\_REGIONS\_BY\_PAGE Parameters*

Parameter	Description
p_application	The identification number (ID) of the application.
p_page	The identification number of page containing the region.

### Example

The following example shows how to use the PURGE\_REGIONS\_BY\_PAGE procedure to delete all the cached values for regions on page 1 of the current application.

```
BEGIN  
    APEX_UTIL.PURGE_REGIONS_BY_PAGE(  
        p_application => :APP_ID,  
        p_page => 1);  
END;
```

---

## REDIRECT\_URL Procedure

This procedure calls `owa_util.redirect_url` to tell the browser to redirect to a new URL. Afterwards, it automatically calls `apex_application.stop_apex_engine` to abort further processing of the Application Express application.

### Syntax

```
APEX_UTIL.REDIRECT_URL (  
    p_url          in varchar2,  
    p_reset_http_buffer in boolean default true );
```

### Parameters

**Table 25–76** REDIRECT\_URL Parameters

Parameter	Description
<code>p_url</code>	The URL the browser requests.
<code>p_reset_http_buffer</code>	Set to <code>TRUE</code> to reset the HTP buffer to make sure the browser understands the redirect to the new URL and is not confused by data that is already written to the HTP buffer. Set to <code>FALSE</code> if the application has its own cookie to use in the response.

### Example

The following example tells the browser to redirect to `http://www.oracle.com` and immediately stops further processing.

```
apex_util.redirect_url (  
    p_url => 'http://www.oracle.com/' );
```

---

## REMOVE\_PREFERENCE Procedure

This procedure removes the preference for the supplied user.

### Syntax

```
APEX_UTIL.REMOVE_PREFERENCE(  
    p_preference    IN    VARCHAR2 DEFAULT NULL,  
    p_user          IN    VARCHAR2 DEFAULT V('USER'));
```

### Parameters

**Table 25–77 REMOVE\_PREFERENCE Parameters**

Parameter	Description
p_preference	Name of the preference to remove
p_user	User for whom the preference is defined

### Example

The following example shows how to use the REMOVE\_PREFERENCE procedure to remove the preference default\_view for the currently authenticated user.

```
BEGIN  
    APEX_UTIL.REMOVE_PREFERENCE(  
        p_preference => 'default_view',  
        p_user       => :APP_USER);  
END;
```

**See Also:** ["GET\\_PREFERENCE Function"](#) on page 25-74, ["SET\\_PREFERENCE Procedure"](#) on page 25-135 and ["Managing User Preferences"](#) in *Oracle Application Express Administration Guide*.

## REMOVE\_SORT\_PREFERENCES Procedure

This procedure removes the user's column heading sorting preference value.

### Syntax

```
APEX_UTIL.REMOVE_SORT_PREFERENCES (  
    p_user IN VARCHAR2 DEFAULT V('USER'));
```

### Parameters

**Table 25–78 REMOVE\_SORT\_PREFERENCES Parameters**

Parameter	Description
p_user	Identifies the user for whom sorting preferences are removed

### Example

The following example shows how to use the REMOVE\_SORT\_PREFERENCES procedure to remove the currently authenticated user's column heading sorting preferences.

```
BEGIN  
    APEX_UTIL.REMOVE_SORT_PREFERENCES (:APP_USER);  
END;
```

---

## REMOVE\_USER Procedure

This procedure removes the user account identified by the primary key or a user name. To execute this procedure, the current user must have administrative privilege in the workspace.

### Syntax

```
APEX_UTIL.REMOVE_USER(  
    p_user_id    IN NUMBER,  
    p_user_name  IN VARCHAR2);
```

### Parameters

**Table 25–79 REMOVE\_USER Parameters**

Parameter	Description
p_user_id	The numeric primary key of the user account record
p_user_name	The user name of the user account

### Example

The following examples show how to use the REMOVE\_USER procedure to remove a user account. Firstly, by the primary key (using the p\_user\_id parameter) and secondly by user name (using the p\_user\_name parameter).

```
BEGIN  
    APEX_UTIL.REMOVE_USER(p_user_id=> 99997);  
END;
```

```
BEGIN  
    APEX_UTIL.REMOVE_USER(p_user_name => 'FRANK');  
END;
```

---

## RESET\_AUTHORIZATIONS Procedure [DEPRECATED]

---

**Note:** Use the ["RESET\\_CACHE Procedure"](#) on page 4-4 instead of this deprecated procedure.

---

To increase performance, Oracle Application Express caches the results of authorization schemes after they have been evaluated. You can use this procedure to undo caching, requiring each authorization scheme be revalidated when it is next encountered during page show or accept processing. You can use this procedure if you want users to have the ability to change their responsibilities (their authorization profile) within your application.

### Syntax

```
APEX_UTIL.RESET_AUTHORIZATIONS;
```

### Parameters

None.

### Example

The following example shows how to use the RESET\_AUTHORIZATIONS procedure to clear the authorization scheme cache.

```
BEGIN
    APEX_UTIL.RESET_AUTHORIZATIONS;
END;
```



## RESET\_PASSWORD Procedure

This procedure is used to change the password of a given user name for the current workspace. This procedure changes the password of `p_user_name` in the current workspace to `p_new_password`. If `p_change_password_on_first_use` is `TRUE`, then the user has to change the password on the next login.

### Syntax

```
APEX_UTIL.RESET_PASSWORD (
  p_user_name      IN VARCHAR2 DEFAULT WWW_FLOW_SECURITY.G_USER,
  p_old_password   IN VARCHAR2 DEFAULT NULL,
  p_new_password   IN VARCHAR2,
  p_change_password_on_first_use IN BOOLEAN DEFAULT TRUE );
```

### Parameters

**Table 25–80** RESET\_PASSWORD Parameters

Parameter	Description
<code>p_user_name</code>	The user whose password should be changed. The default is the currently logged in Application Express user name,
<code>p_old_password</code>	The current password of the user. The call succeeds if the given value matches the current password or it is null and the owner of the calling PL/SQL code has <code>APEX_ADMINISTRATOR_ROLE</code> . If the value is not the user's password, an error occurs.
<code>p_new_password</code>	The new password.
<code>p_change_password_on_first_use</code>	If <code>TRUE</code> (default), the user must change the password on the next login.

### Error Returns

**Table 25–81** RESET\_PASSWORD Parameters

Error	Description
<code>INVALID_CREDENTIALS</code>	Occurs if <code>p_user_name</code> does not match <code>p_old_password</code> ,
<code>APEX.AUTHENTICATION.LOGIN_THROTTLE.COUNTER</code>	Indicates authentication prevented by login throttle.
internal error	Occurs if <code>p_old_password</code> is <code>NULL</code> and caller does not have <code>APEX_ADMINISTRATOR_ROLE</code> .
internal error	Indicates caller is not a valid workspace schema.

### Example

This example demonstrates changing the password of the currently logged in user to a new password.

```
apex_util.reset_password (
  p_old_password => :P111_OLD_PASSWORD,
  p_new_password => :P111_NEW_PASSWORD );
```

---

## RESET\_PW Procedure

This procedure resets the password for a named user and emails it in a message to the email address located for the named account in the current workspace. To execute this procedure, the current user must have administrative privilege in the workspace.

### Syntax

```
APEX_UTIL.RESET_PW(  
    p_user IN VARCHAR2,  
    p_msg  IN VARCHAR2);
```

### Parameters

**Table 25–82** RESET\_PW Parameters

Parameter	Description
p_user	The user name of the user account
p_msg	Message text to be mailed to a user

### Example

The following example shows how to use the RESET\_PW procedure to reset the password for the user 'FRANK'.

```
BEGIN  
    APEX_UTIL.RESET_PW(  
        p_user => 'FRANK',  
        p_msg => 'Contact help desk at 555-1212 with questions');  
END;
```

**See Also:** ["CHANGE\\_CURRENT\\_USER\\_PW Procedure"](#) on page 25-11

---

## SAVEKEY\_NUM Function

This function sets a package variable (`wwv_flow_utilities.g_val_num`) so that it can be retrieved using the function `KEYVAL_NUM`.

### Syntax

```
APEX_UTIL.SAVEKEY_NUM(  
    p_val IN NUMBER)  
RETURN NUMBER;
```

### Parameters

**Table 25–83** *SAVEKEY\_NUM Parameters*

Parameter	Description
<code>p_val</code>	The numeric value to be saved

### Example

The following example shows how to use the `SAVEKEY_NUM` function to set the `wwv_flow_utilities.g_val_num` package variable to the value of 10.

```
DECLARE  
    VAL NUMBER;  
BEGIN  
    VAL := APEX_UTIL.SAVEKEY_NUM(p_val => 10);  
END;
```

**See Also:** ["KEYVAL\\_NUM Function"](#) on page 25-104

## SAVEKEY\_VC2 Function

This function sets a package variable (`wwv_flow_utilities.g_val_vc2`) so that it can be retrieved using the function `KEYVAL_VC2`.

### Syntax

```
APEX_UTIL.SAVEKEY_VC2 (  
    p_val IN VARCHAR2)  
RETURN VARCHAR2;
```

### Parameters

**Table 25–84** *SAVEKEY\_VC2 Parameters*

Parameter	Description
<code>p_val</code>	The is the VARCHAR2 value to be saved

### Example

The following example shows how to use the `SAVEKEY_VC2` function to set the `wwv_flow_utilities.g_val_vc2` package variable to the value of 'XXX'.

```
DECLARE  
    VAL VARCHAR2(4000);  
BEGIN  
    VAL := APEX_UTIL.SAVEKEY_VC2(p_val => 'XXX');  
END;
```

**See Also:** ["KEYVAL\\_VC2 Function"](#) on page 25-105

---

## SET\_ATTRIBUTE Procedure

This procedure sets the value of one of the attribute values (1 through 10) of a user in the Application Express accounts table.

### Syntax

```
APEX_UTIL.SET_ATTRIBUTE(
    p_userid          IN NUMBER,
    p_attribute_number IN NUMBER,
    p_attribute_value IN VARCHAR2);
```

### Parameters

**Table 25–85** SET\_ATTRIBUTE Parameters

Parameter	Description
p_userid	The numeric ID of the user account
p_attribute_number	Attribute number in the user record (1 through 10)
p_attribute_value	Value of the attribute located by p_attribute_number to be set in the user record

### Example

The following example shows how to use the SET\_ATTRIBUTE procedure to set the number 1 attribute for user 'FRANK' with the value 'foo'.

```
DECLARE
    VAL VARCHAR2(4000);
BEGIN
    APEX_UTIL.SET_ATTRIBUTE (
        p_userid => apex_util.get_user_id(p_username => 'FRANK'),
        p_attribute_number => 1,
        p_attribute_value => 'foo');
END;
```

**See Also:** ["GET\\_ATTRIBUTE Function"](#) on page 25-52

## SET\_AUTHENTICATION\_RESULT Procedure

This procedure can be called from an application's custom authentication function (that is, credentials verification function). The status passed to this procedure is logged in the Login Access Log.

**See Also:** "Monitoring Activity within a Workspace" in *Oracle Application Express Administration Guide*

### Syntax

```
APEX_UTIL.SET_AUTHENTICATION_RESULT(
    p_code IN NUMBER);
```

### Parameters

**Table 25–86** SET\_AUTHENTICATION\_RESULT Parameters

Parameter	Description
p_code	Any numeric value the developer chooses. After this value is set in the session using this procedure, it can be retrieved using the APEX_UTIL.GET_AUTHENTICATION_RESULT function.

### Example

One way to use this procedure is to include it in the application authentication scheme. This example demonstrates how text and numeric status values can be registered for logging. In this example, no credentials verification is performed, it just demonstrates how text and numeric status values can be registered for logging.

Note that the status set using this procedure is visible in the apex\_user\_access\_log view and in the reports on this view available to workspace and site administrators.

```
CREATE OR REPLACE FUNCTION MY_AUTH(
    p_username IN VARCHAR2,
    p_password IN VARCHAR2)
RETURN BOOLEAN
IS
BEGIN
    APEX_UTIL.SET_CUSTOM_AUTH_STATUS(p_status=>'User:'||p_username||' is back. ');
    IF UPPER(p_username) = 'GOOD' THEN
        APEX_UTIL.SET_AUTHENTICATION_RESULT(24567);
        RETURN TRUE;
    ELSE
        APEX_UTIL.SET_AUTHENTICATION_RESULT(-666);
        RETURN FALSE;
    END IF;
END;
```

**See Also:** "[GET\\_AUTHENTICATION\\_RESULT Function](#)" on page 25-53 and "[SET\\_CUSTOM\\_AUTH\\_STATUS Procedure](#)" on page 25-128

---

## SET\_BUILD\_OPTION\_STATUS Procedure

Use this procedure to change the build option status of a specified application.

### Syntax

```
apex_util.set_build_option_status(p_application_id IN NUMBER,
                                p_id IN NUMBER,
                                p_build_status IN VARCHAR2);
```

### Parameters

**Table 25–87 SET\_BUILD\_OPTION\_STATUS Parameters**

Parameter	Description
p_application_id	The ID of the application that owns the build option under shared components.
p_id	The ID of the build option in the application.
p_build_status	The new status of the build option. Possible values are INCLUDE, EXCLUDE both upper case.

### Example

The following example demonstrates how to use the SET\_BUILD\_OPTION\_STATUS procedure to change the current status of build option.

```
BEGIN
APEX_UTIL.SET_BUILD_OPTION_STATUS (
  P_APPLICATION_ID => 101,
  P_ID => 245935500311121039, P_BUILD_STATUS=>'INCLUDE');

END;
```

## SET\_CURRENT\_THEME\_STYLE Procedure

This procedure sets the user interface theme style for an application. For example, if there are more than one theme styles available for the current theme, you can use this procedure to change the application theme style.

### Syntax

```
APEX_UTIL.SET_CURRENT_THEME_STYLE (
    p_theme_number IN NUMBER,
    p_theme_style_id IN NUMBER
);
```

### Parameters

**Table 25–88 SET\_CURRENT\_THEME\_STYLE Parameters**

Parameter	Description
p_theme_number	The current theme number of the application. This can be retrieved from APEX_APPLICATION_THEMES view.
p_theme_style_id	The numeric ID of theme style. You can get available theme styles for an application from APEX_APPLICATION_THEME_STYLES view.

### Example

The following example shows how to use the SET\_CURRENT\_THEME\_STYLE procedure to set the current application desktop theme style to Blue.

```
DECLARE
    l_current_theme_number number;
    l_theme_style_id      number;

BEGIN
    select theme_number
    into l_current_theme_number
    from apex_application_themes
    where application_id = :app_id
    and ui_type_name = 'DESKTOP'
    and is_current = 'Yes';

    select s.theme_style_id
    into l_new_theme_style_id
    from apex_application_theme_styles s, apex_application_themes t
    where s.application_id = t.application_id
    and s.theme_number = t.theme_number
    and s.application_id = :app_id
    and t.ui_type_name = 'DESKTOP'
    and t.is_current = 'Yes'
    and s.name = 'Blue';

    if l_current_theme_number is not null and l_new_theme_style_id is not null
    then
        APEX_UTIL.SET_CURRENT_THEME_STYLE (
            p_theme_number => l_current_theme_number,
            p_theme_style_id => l_new_theme_style_id
        );
    end if;
```





## SET\_CUSTOM\_AUTH\_STATUS Procedure

This procedure can be called from an application's custom authentication function (that is, credentials verification function). The status passed to this procedure is logged in the Login Access Log.

**See Also:** "Monitoring Activity within a Workspace" in *Oracle Application Express Administration Guide*

### Syntax

```
APEX_UTIL.SET_CUSTOM_AUTH_STATUS (
    p_status IN VARCHAR2);
```

### Parameters

**Table 25–89** SET\_CUSTOM\_AUTH\_STATUS Parameters

Parameter	Description
p_status	Any text the developer chooses to denote the result of the authentication attempt (up to 4000 characters).

### Example

One way to use the SET\_CUSTOM\_AUTH\_STATUS procedure is to include it in the application authentication scheme. This example demonstrates how text and numeric status values can be registered for logging. Note that no credentials verification is performed.

The status set using this procedure is visible in the apex\_user\_access\_log view and in the reports on this view available to workspace and site administrators.

```
CREATE OR REPLACE FUNCTION MY_AUTH(
    p_username IN VARCHAR2,
    p_password IN VARCHAR2)
RETURN BOOLEAN
IS
BEGIN
    APEX_UTIL.SET_CUSTOM_AUTH_STATUS(p_status=>'User: '||p_username||' is back.');
```

```
IF UPPER(p_username) = 'GOOD' THEN
    APEX_UTIL.SET_AUTHENTICATION_RESULT(24567);
    RETURN TRUE;
ELSE
    APEX_UTIL.SET_AUTHENTICATION_RESULT(-666);
    RETURN FALSE;
END IF;
END;
```

**See Also:** ["SET\\_AUTHENTICATION\\_RESULT Procedure"](#) on page 25-124 and ["GET\\_AUTHENTICATION\\_RESULT Function"](#) on page 25-53

---

## SET\_EDITION Procedure

This procedure sets the name of the edition to be used in all application SQL parsed in the current page view or page submission.

### Syntax

```
APEX_UTIL.SET_EDITION(  
    p_edition IN VARCHAR2);
```

### Parameters

**Table 25–90** SET\_EDITION Parameters

Parameter	Description
p_edition	Edition name.

### Example

The following example shows how to use the SET\_EDITION procedure. It sets the edition name for the database session of the current page view.

```
BEGIN  
    APEX_UTIL.SET_EDITION( P_EDITION => 'Edition1' );  
END;
```

---

---

**Note:** Support for Edition-Based Redefinition is only available in database version 11.2.0.1 or higher.

---

---

## SET\_EMAIL Procedure

This procedure updates a user account with a new email address. To execute this procedure, the current user must have administrative privileges in the workspace.

### Syntax

```
APEX_UTIL.SET_EMAIL(  
    p_userid IN NUMBER,  
    p_email  IN VARCHAR2);
```

### Parameters

**Table 25–91** SET\_EMAIL Parameters

Parameter	Description
p_userid	The numeric ID of the user account
p_email	The email address to be saved in user account

### Example

The following example shows how to use the SET\_EMAIL procedure to set the value of EMAIL to 'frank.scott@somewhere.com' for the user 'FRANK'.

```
BEGIN  
    APEX_UTIL.SET_EMAIL(  
        p_userid => APEX_UTIL.GET_USER_ID('FRANK'),  
        p_email  => 'frank.scott@somewhere.com');  
END;
```

**See Also:** ["GET\\_EMAIL Function"](#) on page 25-61 and ["GET\\_USER\\_ID Function"](#) on page 25-86

---

## SET\_FIRST\_NAME Procedure

This procedure updates a user account with a new `FIRST_NAME` value. To execute this procedure, the current user must have administrative privileges in the workspace.

### Syntax

```
APEX_UTIL.SET_FIRST_NAME(  
    p_userid      IN NUMBER,  
    p_first_name  IN VARCHAR2);
```

### Parameters

**Table 25–92** *SET\_FIRST\_NAME Parameters*

Parameter	Description
<code>p_userid</code>	The numeric ID of the user account
<code>p_first_name</code>	<code>FIRST_NAME</code> value to be saved in user account

### Example

The following example shows how to use the `SET_FIRST_NAME` procedure to set the value of `FIRST_NAME` to 'FRANK' for the user 'FRANK'.

```
BEGIN  
    APEX_UTIL.SET_FIRST_NAME(  
        p_userid      => APEX_UTIL.GET_USER_ID('FRANK'),  
        p_first_name  => 'FRANK');  
END;
```

**See Also:** ["GET\\_FIRST\\_NAME Function"](#) on page 25-66 and ["GET\\_USER\\_ID Function"](#) on page 25-86

---

## SET\_GROUP\_GROUP\_GRANTS Procedure

This procedure modifies the group grants for a given group.

### Syntax

```
APEX_UTIL.SET_GROUP_GROUP_GRANTS (  
    p_group_name IN VARCHAR2,  
    p_granted_group_names IN wwv_flow_t_varchar2 );
```

### Parameters

**Table 25–93 SET\_GROUP\_GROUP\_GRANTS Procedure Parameters**

Parameter	Description
p_group_name	The target group name.
p_granted_group_names	The names of groups to grant to p_group_name.

### Example

This example creates three groups (ACCTS\_PAY, ACCTS\_REC, MANAGER) and then grants ACCTS\_PAY and ACCTS\_REC to MANAGER.

```
apex_util.create_user_group (  
    p_group_name => 'ACCTS_PAY' );  
apex_util.create_user_group (  
    p_group_name => 'ACCTS_REC' );  
apex_util.create_user_group (  
    p_group_name => 'MANAGER' );  
apex_util.set_group_group_grants (  
    p_group_name => 'MANAGER',  
    p_granted_group_names => apex_t_varchar2('ACCTS_PAY', 'ACCTS_REC') );
```

---

## SET\_GROUP\_USER\_GRANTS Procedure

This procedure modifies the group grants for a given user.

### Syntax

```
APEX_UTIL.SET_GROUP_USER_GRANTS (
    p_user_name IN VARCHAR2,
    p_granted_group_names IN wwv_flow_t_varchar2 );
```

### Parameters

**Table 25–94 SET\_GROUP\_USER\_GRANTS Procedure Parameters**

Parameter	Description
p_user_name	The target user name.
p_granted_group_names	The names of groups to grant to p_user_name

### Example

This example creates a user group (MANAGER) and a user (Example User) and then grants MANAGER to Example User.

```
apex_util.create_user_group (
    p_group_name => 'MANAGER' );
apex_util.create_user (
    p_user_name => 'Example User',
    p_web_password => 1_random_password );
-- grant MANAGER to Example User
apex_util.set_group_user_grants (
    p_user_name => 'Example User',
    p_granted_group_names => apex_t_varchar2('MANAGER') );
```

## SET\_LAST\_NAME Procedure

This procedure updates a user account with a new `LAST_NAME` value. To execute this procedure, the current user must have administrative privileges in the workspace.

### Syntax

```
APEX_UTIL.SET_LAST_NAME(  
    p_userid      IN NUMBER,  
    p_last_name   IN VARCHAR2);
```

### Parameters

**Table 25–95** *SET\_LAST\_NAME Parameters*

Parameter	Description
<code>p_userid</code>	The numeric ID of the user account
<code>p_last_name</code>	<code>LAST_NAME</code> value to be saved in the user account

### Example

The following example shows how to use the `SET_LAST_NAME` procedure to set the value of `LAST_NAME` to 'SMITH' for the user 'FRANK'.

```
BEGIN  
    APEX_UTIL.SET_LAST_NAME(  
        p_userid      => APEX_UTIL.GET_USER_ID('FRANK'),  
        p_last_name   => 'SMITH');  
END;
```

**See Also:** ["GET\\_LAST\\_NAME Function"](#) on page 25-72 and ["GET\\_USER\\_ID Function"](#) on page 25-86



---

## SET\_PREFERENCE Procedure

This procedure sets a preference that persists beyond the user's current session.

### Syntax

```
APEX_UTIL.SET_PREFERENCE (
    p_preference IN VARCHAR2 DEFAULT NULL,
    p_value      IN VARCHAR2 DEFAULT NULL,
    p_user       IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 25–96** SET\_PREFERENCE Parameters

Parameter	Description
p_preference	Name of the preference (case-sensitive)
p_value	Value of the preference
p_user	User for whom the preference is being set

### Example

The following example shows how to use the SET\_PREFERENCE procedure to set a preference called 'default\_view' to the value 'WEEKLY' that persists beyond session for the currently authenticated user.

```
BEGIN
    APEX_UTIL.SET_PREFERENCE(
        p_preference => 'default_view',
        p_value      => 'WEEKLY',
        p_user       => :APP_USER);
END;
```

**See Also:** ["GET\\_PREFERENCE Function"](#) on page 25-74 and ["REMOVE\\_PREFERENCE Procedure"](#) on page 25-115

## SET\_SECURITY\_GROUP\_ID Procedure

Use this procedure with `apex_util.find_security_group_id` to ease the use of the mail package in batch mode. This procedure is especially useful when a schema is associated with more than one workspace. For example, you might want to create a procedure that is run by a nightly job to email all outstanding tasks.

### Syntax

```
APEX_UTIL.SET_SECURITY_GROUP_ID (
    p_security_group_id IN NUMBER);
```

### Parameters

**Table 25–97 SET\_SECURITY\_GROUP\_ID Parameters**

Parameter	Description
<code>p_security_group_id</code>	This is the security group id of the workspace you are working in.

### Example

The following example sends an alert to each user that has had a task assigned within the last day.

```
create or replace procedure new_tasks
is
    l_workspace_id    number;
    l_subject         varchar2(2000);
    l_body            clob;
    l_body_html       clob;
begin
    l_workspace_id := apex_util.find_security_group_id (p_workspace =>
'PROJECTS');
    apex_util.set_security_group_id (p_security_group_id => l_workspace_id);

    l_body := ' ';
    l_subject := 'You have new tasks';
    for c1 in (select distinct(p.email_address) email_address, p.user_id
               from teamsp_user_profile p, teamsp_tasks t
               where p.user_id = t.assigned_to_user_id
                   and t.created_on > sysdate - 1
                   and p.email_address is not null ) loop
        l_body_html := '<p />The following tasks have been added.';
        for c2 in (select task_name, due_date
                   from teamsp_tasks
                   where assigned_to_user_id = c1.user_id
                       and created_on > sysdate - 1 ) loop
            l_body_html := l_body_html || '<p />Task: ' || c2.task_name || ', due
' || c2.due_date;
        end loop;
    apex_mail.send (
        p_to         => c1.email_address,
        p_from       => c1.email_address,
        p_body       => l_body,
        p_body_html  => l_body_html,
        p_subj       => l_subject );
    end loop;
```

end;

---

## SET\_SESSION\_HIGH\_CONTRAST\_OFF Procedure

This procedure switches off high contrast mode for the current session.

### Syntax

```
APEX_UTIL.SET_SESSION_HIGH_CONTRAST_OFF;
```

### Parameters

None.

### Example

In this example, high contrast mode is switched off for the current session.

```
BEGIN
    apex_util.set_session_high_contrast_off;
END;
```

## SET\_SESSION\_HIGH\_CONTRAST\_ON Procedure

This procedure switches on high contrast mode for the current session.

### Syntax

```
APEX_UTIL.SET_SESSION_HIGH_CONTRAST_ON;
```

### Parameters

None.

### Example

In this example, the current session is put into high contrast mode.

```
BEGIN
    apex_util.set_session_high_contrast_on;
END;
```

## SET\_SESSION\_LANG Procedure

This procedure sets the language to be used for the current user in the current Application Express session. The language must be a valid IANA language name.

### Syntax

```
APEX_UTIL.SET_SESSION_LANG(  
    p_lang IN VARCHAR2);
```

### Parameters

**Table 25–98** SET\_SESSION\_LANG Parameters

Parameter	Description
p_lang	This is an IANA language code. Some examples include: en, de, de-at, zh-cn, and pt-br.

### Example

The following example shows how to use the SET\_SESSION\_LANG procedure. It sets the language for the current user for the duration of the Application Express session.

```
BEGIN  
    APEX_UTIL.SET_SESSION_LANG( P_LANG => 'en' );  
END;
```

## SET\_SESSION\_LIFETIME\_SECONDS Procedure

This procedure sets the current session's Maximum Session Length in Seconds value, overriding the corresponding application attribute. This allows developers to dynamically shorten or lengthen the session life based on criteria determined after the user authenticates.

### Syntax

```
APEX_UTIL.SET_SESSION_LIFETIME_SECONDS (
    p_seconds IN NUMBER,
    p_scope IN VARCHAR2 DEFAULT 'SESSION');
```

### Parameters

**Table 25–99 SET\_SESSION\_LIFETIME\_SECONDS Parameters**

Parameter	Description
p_seconds	A positive integer indicating the number of seconds the session used by this application is allowed to exist.
p_scope	This parameter is obsolete. The procedure always sets the lifetime for the whole session.

### Example 1

The following example shows how to use the SET\_SESSION\_LIFETIME\_SECONDS procedure to set the current application's Maximum Session Length in Seconds attribute to 7200 seconds (two hours).

By allowing the p\_scope input parameter to use the default value of 'SESSION', the following example would actually apply to all applications using the current session. This would be the most common use case when multiple Application Express applications use a common authentication scheme and are designed to operate as a suite in a common session.

```
BEGIN
    APEX_UTIL.SET_SESSION_LIFETIME_SECONDS(p_seconds => 7200);
END;
```

### Example 2

The following example shows how to use the SET\_SESSION\_LIFETIME\_SECONDS procedure to set the current application's Maximum Session Length in Seconds attribute to 3600 seconds (one hour).

```
BEGIN
    APEX_UTIL.SET_SESSION_LIFETIME_SECONDS(p_seconds => 3600);
END;
```

## SET\_SESSION\_MAX\_IDLE\_SECONDS Procedure

Sets the current application's Maximum Session Idle Time in Seconds value for the current session, overriding the corresponding application attribute. This allows developers to dynamically shorten or lengthen the maximum idle time allowed between page requests based on criteria determined after the user authenticates.

### Syntax

```
APEX_UTIL.SET_SESSION_MAX_IDLE_SECONDS (
    p_seconds IN    NUMBER,
    p_scope   IN    VARCHAR2 DEFAULT 'SESSION');
```

### Parameters

**Table 25–100 SET\_SESSION\_MAX\_IDLE\_SECONDS Parameters**

Parameter	Description
p_seconds	A positive integer indicating the number of seconds allowed between page requests.
p_scope	This parameter is obsolete. The procedure always sets the lifetime for the whole session

### Example 1

The following example shows how to use the SET\_SESSION\_MAX\_IDLE\_SECONDS procedure to set the current application's Maximum Session Idle Time in Seconds attribute to 1200 seconds (twenty minutes). The following example applies to all applications using the current session.

```
BEGIN
    APEX_UTIL.SET_SESSION_MAX_IDLE_SECONDS(p_seconds => 1200);
END;
```

### Example 2

The following example shows how to use the SET\_SESSION\_MAX\_IDLE\_SECONDS procedure to set the current application's Maximum Session Idle Time in Seconds attribute to 600 seconds (ten minutes). This example applies to all applications using the current session.

```
BEGIN
    APEX_UTIL.SET_SESSION_MAX_IDLE_SECONDS(p_seconds => 600);
END;
```



## SET\_SESSION\_SCREEN\_READER\_OFF Procedure

This procedure switches off screen reader mode for the current session.

### Syntax

```
APEX_UTIL.SET_SESSION_SCREEN_READER_OFF;
```

### Parameters

None

### Example

In this example, the current session is put into standard mode.

```
BEGIN
    apex_util.set_session_screen_reader_off;
END;
```

---

## SET\_SESSION\_SCREEN\_READER\_ON Procedure

This procedure puts the current session into screen reader mode.

### Syntax

```
APEX_UTIL.SET_SESSION_SCREEN_READER_ON;
```

### Parameters

None

### Example

In this example, the current session is put into screen reader mode.

```
BEGIN
    apex_util.set_session_screen_reader_on;
END;
```

---

## SET\_SESSION\_STATE Procedure

This procedure sets session state for a current Oracle Application Express session.

### Syntax

```
APEX_UTIL.SET_SESSION_STATE (
    p_name      IN      VARCHAR2 DEFAULT NULL,
    p_value     IN      VARCHAR2 DEFAULT NULL);
    p_commit    IN      BOOLEAN  DEFAULT TRUE);
```

### Parameters

**Table 25–101** SET\_SESSION\_STATE Parameters

Parameter	Description
p_name	Name of the application-level or page-level item for which you are setting sessions state
p_value	Value of session state to set
p_commit	If true (the default), commit after modifying session state. If false or if the existing value in session state equals p_value, no commit is issued.

### Example

The following example shows how to use the SET\_SESSION\_STATE procedure to set the value of the item 'my\_item' to 'myvalue' in the current session.

```
BEGIN
    APEX_UTIL.SET_SESSION_STATE('my_item', 'myvalue');
END;
```

**See Also:** ["GET\\_SESSION\\_STATE Function"](#) on page 25-82, ["GET\\_NUMERIC\\_SESSION\\_STATE Function"](#) on page 25-73, and ["Understanding Session State Management"](#) in *Oracle Application Express Application Builder User's Guide*

## SET\_SESSION\_TERRITORY Procedure

This procedure sets the territory to be used for the current user in the current Application Express session. The territory name must be a valid Oracle territory.

### Syntax

```
APEX_UTIL.SET_SESSION_TERRITORY(  
    p_territory IN VARCHAR2);
```

### Parameters

**Table 25–102** SET\_SESSION\_TERRITORY Parameters

Parameter	Description
p_territory	A valid Oracle territory name. Examples include: AMERICA, UNITED KINGDOM, ISRAEL, AUSTRIA, and UNITED ARAB EMIRATES.

### Example

The following example shows how to use the SET\_SESSION\_TERRITORY procedure. It sets the territory for the current user for the duration of the Application Express session.

```
BEGIN  
    APEX_UTIL.SET_SESSION_TERRITORY( P_TERRITORY => 'UNITED KINGDOM' );  
END;
```

---

## SET\_SESSION\_TIME\_ZONE Procedure

This procedure sets the time zone to be used for the current user in the current Application Express session.

### Syntax

```
APEX_UTIL.SET_SESSION_TIME_ZONE(  
    p_time_zone IN VARCHAR2);
```

### Parameters

**Table 25–103** SET\_SESSION\_TIME\_ZONE Parameters

Parameter	Description
p_timezone	A time zone value in the form of hours and minutes. Examples include: +09:00, 04:00, -05:00.

### Example

The following example shows how to use the SET\_SESSION\_TIME\_ZONE procedure. It sets the time zone for the current user for the duration of the Application Express session.

```
BEGIN  
    APEX_UTIL.SET_SESSION_TIME_ZONE( P_TIME_ZONE => '-05:00');  
END;
```

## SET\_USERNAME Procedure

This procedure updates a user account with a new USER\_NAME value. To execute this procedure, the current user must have administrative privileges in the workspace.

### Syntax

```
APEX_UTIL.SET_USERNAME(  
    p_userid    IN NUMBER,  
    p_username  IN VARCHAR2);
```

### Parameters

**Table 25–104** SET\_USERNAME Parameters

Parameter	Description
p_userid	The numeric ID of the user account
p_username	USER_NAME value to be saved in the user account

### Example

The following example shows how to use the SET\_USERNAME procedure to set the value of USERNAME to 'USER-XRAY' for the user 'FRANK'.

```
BEGIN  
    APEX_UTIL.SET_USERNAME(  
        p_userid    => APEX_UTIL.GET_USER_ID('FRANK'),  
        p_username  => 'USER-XRAY');  
END;
```

**See Also:** ["GET\\_USERNAME Function"](#) on page 25-88 and ["GET\\_USER\\_ID Function"](#) on page 25-86

---

## SHOW\_HIGH\_CONTRAST\_MODE\_TOGGLE Procedure

This procedure displays a link to the current page to turn on or off, toggle, the mode. For example, if you are in standard mode, this function displays a link that when clicked switches the high contrast mode on.

### Syntax

```
APEX_UTIL.SHOW_HIGH_CONTRAST_MODE_TOGGLE (
    p_on_message in varchar2 default null,
    p_off_message in varchar2 default null);
```

### Parameters

**Table 25–105** *SHOW\_HIGH\_CONTRAST\_MODE\_TOGGLE Parameters*

Parameters	Description
p_on_message	Optional text used for the link to switch to high contrast mode, when you are in standard mode. If this parameter is not passed, the default 'Set High Contrast Mode On' text is displayed.
p_off_message	Optional text used for the link to switch to standard mode, when you are in high contrast mode. If this parameter is not passed, the default 'Set High Contrast Mode Off' text is displayed.

### Example

When running in standard mode, this procedure displays a link, Set High Contrast Mode On, that when clicked refreshes the current page and switches on high contrast mode. When running in high contrast mode, a link, Set High Contrast Mode Off, is displayed, that refreshes the current page and switches back to standard mode when clicked.

```
BEGIN
    apex_util.show_high_contrast_mode_toggle;
END;
```

**See Also:** ["GET\\_HIGH\\_CONTRAST\\_MODE\\_TOGGLE Function"](#) on page 25-71

---

**Note:** There are also 2 translatable system messages that can be overridden at application level to change the default link text that is returned for this toggle. They include:

- APEX.SET\_HIGH\_CONTRAST\_MODE\_OFF - Default text = Set High Contrast Mode Off
  - APEX.SET\_HIGH\_CONTRAST\_MODE\_ON - Default text = Set High Contrast Mode On
-

---

## SHOW\_SCREEN\_READER\_MODE\_TOGGLE Procedure

This procedure displays a link to the current page to turn on or off, toggle, the mode. For example, if you are in standard mode, this function displays a link that when clicked switches the screen reader mode on.

### Syntax

```
APEX_UTIL.SHOW_SCREEN_READER_MODE_TOGGLE (  
    p_on_message IN VARCHAR2 DEFAULT NULL,  
    p_off_message IN VARCHAR2 DEFAULT NULL)
```

### Parameters

**Table 25–106** *SHOW\_SCREEN\_READER\_MODE\_TOGGLE Parameters*

Parameter	Description
p_on_message	Optional text used for the link to switch to screen reader mode, when you are in standard mode. If this parameter is not passed, the default 'Set Screen Reader Mode On' text is displayed.
p_off_message	Optional text used for the link to switch to standard mode, when you are in screen reader mode. If this parameter is not passed, the default 'Set Screen Reader Mode Off' text is displayed.

### Example

When running in standard mode, this procedure displays a link 'Set Screen Reader Mode On', that when clicked refreshes the current page and switches on screen reader mode. When running in screen reader mode, a link 'Set Screen Reader Mode Off' is displayed, that when clicked refreshes the current page and switches back to standard mode.

```
BEGIN  
    apex_util.show_screen_reader_mode_toggle;  
END;
```



## STRING\_TO\_TABLE Function

Given a string, this function returns a PL/SQL array of type `APEX_APPLICATION_GLOBAL.VC_ARR2`. This array is a `VARCHAR2 (32767)` table.

### Syntax

```
APEX_UTIL.STRING_TO_TABLE (
    p_string      IN VARCHAR2,
    p_separator   IN VARCHAR2 DEFAULT ':')
RETURN APEX_APPLICATION_GLOBAL.VC_ARR2;
```

### Parameters

**Table 25–107** *STRING\_TO\_TABLE Parameters*

Parameter	Description
<code>p_string</code>	String to be converted into a PL/SQL table of type <code>APEX_APPLICATION_GLOBAL.VC_ARR2</code>
<code>p_separator</code>	String separator. The default is a colon

### Example

The following example shows how to use the `STRING_TO_TABLE` function. The function is passed the string 'One:Two:Three' in the `p_string` parameter and it returns a PL/SQL array of type `APEX_APPLICATION_GLOBAL.VC_ARR2` containing 3 elements, the element at position 1 contains the value 'One', position 2 contains the value 'Two' and position 3 contains the value 'Three'. This is then output using the `HTP.P` function call.

```
DECLARE
    l_vc_arr2    APEX_APPLICATION_GLOBAL.VC_ARR2;
BEGIN
    l_vc_arr2 := APEX_UTIL.STRING_TO_TABLE('One:Two:Three');
    FOR z IN 1..l_vc_arr2.count LOOP
        htp.p(l_vc_arr2(z));
    END LOOP;
END;
```

**See Also:** ["TABLE\\_TO\\_STRING Function"](#) on page 25-160

## STRONG\_PASSWORD\_CHECK Procedure

This procedure returns Boolean OUT values based on whether a proposed password meets the password strength requirements as defined by the Oracle Application Express site administrator.

### Syntax

```
APEX_UTIL.STRONG_PASSWORD_CHECK (
    p_username          IN  VARCHAR2,
    p_password          IN  VARCHAR2,
    p_old_password      IN  VARCHAR2,
    p_workspace_name    IN  VARCHAR2,
    p_use_strong_rules  IN  BOOLEAN,
    p_min_length_err    OUT BOOLEAN,
    p_new_differs_by_err OUT BOOLEAN,
    p_one_alpha_err     OUT BOOLEAN,
    p_one_numeric_err   OUT BOOLEAN,
    p_one_punctuation_err OUT BOOLEAN,
    p_one_upper_err     OUT BOOLEAN,
    p_one_lower_err     OUT BOOLEAN,
    p_not_like_username_err OUT BOOLEAN,
    p_not_like_workspace_name_err OUT BOOLEAN,
    p_not_like_words_err OUT BOOLEAN,
    p_not_reusable_err  OUT BOOLEAN);
```

### Parameters

**Table 25–108** *STRONG\_PASSWORD\_CHECK Parameters*

Parameter	Description
p_username	Username that identifies the account in the current workspace
p_password	Password to be checked against password strength rules
p_old_password	Current password for the account. Used only to enforce "new password must differ from old" rule
p_workspace_name	Current workspace name, used only to enforce "password must not contain workspace name" rule
p_use_strong_rules	Pass FALSE when calling this API
p_min_length_err	Result returns TRUE or FALSE depending upon whether the password meets minimum length requirement
p_new_differs_by_err	Result returns TRUE or FALSE depending upon whether the password meets "new password must differ from old" requirements
p_one_alpha_err	Result returns TRUE or FALSE depending upon whether the password meets requirement to contain at least one alphabetic character
p_one_numeric_err	Result returns TRUE or FALSE depending upon whether the password meets requirements to contain at least one numeric character
p_one_punctuation_err	Result returns TRUE or FALSE depending upon whether the password meets requirements to contain at least one punctuation character

**Table 25–108 (Cont.) STRONG\_PASSWORD\_CHECK Parameters**

Parameter	Description
p_one_upper_err	Result returns TRUE or FALSE depending upon whether the password meets requirements to contain at least one upper-case character
p_one_lower_err	Result returns TRUE or FALSE depending upon whether the password meets requirements to contain at least one lower-case character
p_not_like_username_err	Result returns TRUE or FALSE depending upon whether the password meets requirements that it not contain the username
p_not_like_workspace_name_err	Result returns TRUE or FALSE whether upon whether the password meets requirements that it not contain the workspace name
p_not_like_words_err	Result returns TRUE or FALSE whether the password meets requirements that it not contain specified simple words
p_not_reusable_err	Result returns TRUE or FALSE whether the password can be reused based on password history rules

**Example**

The following example shows how to use the STRONG\_PASSWORD\_CHECK procedure. It checks the new password 'foo' for the user 'SOMEBODY' meets all the password strength requirements defined by the Oracle Application Express site administrator. If any of the checks fail (the associated OUT parameter returns TRUE), then the example outputs a relevant message. For example, if the Oracle Application Express site administrator has defined that passwords must have at least one numeric character and the password 'foo' was checked, then the p\_one\_numeric\_err OUT parameter would return TRUE and the message 'Password must contain at least one numeric character' would be output.

```

DECLARE
    l_username          varchar2(30);
    l_password          varchar2(30);
    l_old_password      varchar2(30);
    l_workspace_name    varchar2(30);
    l_min_length_err    boolean;
    l_new_differs_by_err boolean;
    l_one_alpha_err     boolean;
    l_one_numeric_err   boolean;
    l_one_punctuation_err boolean;
    l_one_upper_err     boolean;
    l_one_lower_err     boolean;
    l_not_like_username_err boolean;
    l_not_like_workspace_name_err boolean;
    l_not_like_words_err boolean;
    l_not_reusable_err  boolean;
    l_password_history_days pls_integer;
BEGIN
    l_username := 'SOMEBODY';
    l_password := 'foo';
    l_old_password := 'foo';
    l_workspace_name := 'XYX_WS';
    l_password_history_days :=
        apex_instance_admin.get_parameter ('PASSWORD_HISTORY_DAYS');

    APEX_UTIL.STRONG_PASSWORD_CHECK(
        p_username          => l_username,

```

```
p_password           => l_password,
p_old_password       => l_old_password,
p_workspace_name     => l_workspace_name,
p_use_strong_rules   => false,
p_min_length_err     => l_min_length_err,
p_new_differs_by_err => l_new_differs_by_err,
p_one_alpha_err      => l_one_alpha_err,
p_one_numeric_err    => l_one_numeric_err,
p_one_punctuation_err => l_one_punctuation_err,
p_one_upper_err      => l_one_upper_err,
p_one_lower_err      => l_one_lower_err,
p_not_like_username_err => l_not_like_username_err,
p_not_like_workspace_name_err => l_not_like_workspace_name_err,
p_not_like_words_err => l_not_like_words_err,
p_not_reusable_err   => l_not_reusable_err);

IF l_min_length_err THEN
    http.p('Password is too short');
END IF;

IF l_new_differs_by_err THEN
    http.p('Password is too similar to the old password');
END IF;

IF l_one_alpha_err THEN
    http.p('Password must contain at least one alphabetic character');
END IF;

IF l_one_numeric_err THEN
    http.p('Password must contain at least one numeric character');
END IF;

IF l_one_punctuation_err THEN
    http.p('Password must contain at least one punctuation character');
END IF;

IF l_one_upper_err THEN
    http.p('Password must contain at least one upper-case character');
END IF;

IF l_one_lower_err THEN
    http.p('Password must contain at least one lower-case character');
END IF;

IF l_not_like_username_err THEN
    http.p('Password may not contain the username');
END IF;

IF l_not_like_workspace_name_err THEN
    http.p('Password may not contain the workspace name');
END IF;

IF l_not_like_words_err THEN
    http.p('Password contains one or more prohibited common words');
END IF;

IF l_not_reusable_err THEN
    http.p('Password cannot be used because it has been used for the account
within the last '||l_password_history_days||' days. ');
END IF;
```

END;

**See Also:** "Creating Strong Password Policies" in *Oracle Application Express Administration Guide*

## STRONG\_PASSWORD\_VALIDATION Function

This function returns formatted HTML in a VARCHAR2 result based on whether a proposed password meets the password strength requirements as defined by the Oracle Application Express site administrator.

### Syntax

```
FUNCTION STRONG_PASSWORD_VALIDATION(
    p_username          IN  VARCHAR2,
    p_password          IN  VARCHAR2,
    P_OLD_PASSWORD      IN  VARCHAR2 DEFAULT NULL,
    P_WORKSPACE_NAME    IN  VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 25–109** *STRONG\_PASSWORD\_VALIDATION Parameters*

Parameter	Description
p_username	Username that identifies the account in the current workspace
p_password	Password to be checked against password strength rules
p_old_password	Current password for the account. Used only to enforce "new password must differ from old" rule
p_workspace_name	Current workspace name, used only to enforce "password must not contain workspace name" rule

### Example

The following example shows how to use the STRONG\_PASSWORD\_VALIDATION procedure. It checks the new password 'foo' for the user 'SOMEBODY' meets all the password strength requirements defined by the Oracle Application Express site administrator. If any of the checks fail, then the example outputs formatted HTML showing details of where the new password fails to meet requirements.

```
DECLARE
    l_username          varchar2(30);
    l_password          varchar2(30);
    l_old_password      varchar2(30);
    l_workspace_name    varchar2(30);
BEGIN
    l_username := 'SOMEBODY';
    l_password := 'foo';
    l_old_password := 'foo';
    l_workspace_name := 'XYX_WS';

    HTP.P(APEX_UTIL.STRONG_PASSWORD_VALIDATION(
        p_username          => l_username,
        p_password          => l_password,
        p_old_password      => l_old_password,
        p_workspace_name    => l_workspace_name));
END;
```

## SUBMIT\_FEEDBACK Procedure

This procedure enables you to write a procedure to submit feedback, rather than using the page that can be generated by create page of type feedback.

### Syntax

```
APEX_UTIL.SUBMIT_FEEDBACK (
    p_comment          IN VARCHAR2 DEFAULT NULL,
    p_type             IN NUMBER   DEFAULT '1',
    p_application_id   IN VARCHAR2 DEFAULT NULL,
    p_page_id         IN VARCHAR2 DEFAULT NULL,
    p_email            IN VARCHAR2 DEFAULT NULL,
    p_screen_width     IN VARCHAR2 DEFAULT NULL,
    p_screen_height    IN VARCHAR2 DEFAULT NULL,
    p_attribute_01     IN VARCHAR2 DEFAULT NULL,
    p_attribute_02     IN VARCHAR2 DEFAULT NULL,
    p_attribute_03     IN VARCHAR2 DEFAULT NULL,
    p_attribute_04     IN VARCHAR2 DEFAULT NULL,
    p_attribute_05     IN VARCHAR2 DEFAULT NULL,
    p_attribute_06     IN VARCHAR2 DEFAULT NULL,
    p_attribute_07     IN VARCHAR2 DEFAULT NULL,
    p_attribute_08     IN VARCHAR2 DEFAULT NULL,
    p_label_01        IN VARCHAR2 DEFAULT NULL,
    p_label_02        IN VARCHAR2 DEFAULT NULL,
    p_label_03        IN VARCHAR2 DEFAULT NULL,
    p_label_04        IN VARCHAR2 DEFAULT NULL,
    p_label_05        IN VARCHAR2 DEFAULT NULL,
    p_label_06        IN VARCHAR2 DEFAULT NULL,
    p_label_07        IN VARCHAR2 DEFAULT NULL,
    p_label_08        IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 25–110** SUBMIT\_FEEDBACK Parameters

Parameter	Description
p_comment	Comment to be submitted
p_type	Type of feedback (1 is General Comment, 2 is Enhancement Request, 3 is Bug)
p_application_id	ID of application related to the feedback
p_page_id	ID of page related to the feedback
p_email	Email of the user providing the feedback
p_screen_width	Width of screen at time feedback was provided
p_screen_height	Height of screen at time feedback was provided
p_attribute_01	Custom attribute for collecting feedback
p_attribute_02	Custom attribute for collecting feedback
p_attribute_03	Custom attribute for collecting feedback
p_attribute_04	Custom attribute for collecting feedback
p_attribute_05	Custom attribute for collecting feedback
p_attribute_06	Custom attribute for collecting feedback

**Table 25–110 (Cont.) SUBMIT\_FEEDBACK Parameters**

<b>Parameter</b>	<b>Description</b>
p_attribute_07	Custom attribute for collecting feedback
p_attribute_08	Custom attribute for collecting feedback
p_label_01	Label for corresponding custom attribute
p_label_02	Label for corresponding custom attribute
p_label_03	Label for corresponding custom attribute
p_label_04	Label for corresponding custom attribute
p_label_05	Label for corresponding custom attribute
p_label_06	Label for corresponding custom attribute
p_label_07	Label for corresponding custom attribute
p_label_08	Label for corresponding custom attribute

**Example**

The following example submits a bug about page 22 within application 283.

```
begin
  apex_util.submit_feedback (
    p_comment      => 'This page does not render properly for me',
    p_type         => 3,
    p_application_id => 283,
    p_page_id      => 22,
    p_email        => 'user@xyz.corp',
    p_attribute_01 => 'Charting',
    p_label_01     => 'Component' );
end;
```



---

## SUBMIT\_FEEDBACK\_FOLLOWUP Procedure

This procedure enables you to submit follow up to a feedback.

### Syntax

```
APEX_UTIL.SUBMIT_FEEDBACK_FOLLOWUP (
  p_feedback_id    IN NUMBER,
  p_follow_up      IN VARCHAR2 DEFAULT NULL,
  p_email          IN VARCHAR2 DEFAULT NULL);
```

### Parameters

**Table 25–111** *SUBMIT\_FEEDBACK\_FOLLOWUP Parameters*

Parameter	Description
p_feedback_followup	ID of feedback that this is a follow up to
p_follow_up	Text of follow up
p_email	Email of user providing the follow up

### Example

The following example submits follow up to a previously filed feedback.

```
begin
  apex_util.submit_feedback_followup (
    p_feedback_id    => 12345,
    p_follow_up      => 'I tried this on another instance and it does not work
there either',
    p_email          => 'user@xyz.corp' );
end;
/
```

---

## TABLE\_TO\_STRING Function

Given a PL/SQL table of type `APEX_APPLICATION_GLOBAL.VC_ARR2`, this function returns a delimited string separated by the supplied separator, or by the default separator, a colon (:).

### Syntax

```
APEX_UTIL.TABLE_TO_STRING (
    p_table      IN      APEX_APPLICATION_GLOBAL.VC_ARR2,
    p_string     IN      VARCHAR2 DEFAULT ':' )
RETURN VARCHAR2;
```

### Parameters

**Table 25–112** TABLE\_TO\_STRING Parameters

Parameter	Description
<code>p_string</code>	String separator. Default separator is a colon (:)
<code>p_table</code>	PL/SQL table that is to be converted into a delimited string

### Example

The following function returns a comma delimited string of contact names that are associated with the provided `cust_id`.

```
create or replace function get_contacts (
    p_cust_id in number )
    return varchar2
is
    l_vc_arr2 apex_application_global.vc_arr2;
    l_contacts varchar2(32000);
begin

    select contact_name
        bulk collect
        into l_vc_arr2
        from contacts
    where cust_id = p_cust_id
        order by contact_name;

    l_contacts := apex_util.table_to_string (
        p_table => l_vc_arr2,
        p_string => ', ');

    return l_contacts;

end get_contacts;
```

**See Also:** ["STRING\\_TO\\_TABLE Function"](#) on page 25-151

## UNEXPIRE\_END\_USER\_ACCOUNT Procedure

Makes expired end users accounts and the associated passwords usable, enabling a end user to log in to developed applications.

### Syntax

```
APEX_UTIL.UNEXPIRE_END_USER_ACCOUNT (
    p_user_name IN VARCHAR2);
```

### Parameters

**Table 25–113 UNEXPIRE\_END\_USER\_ACCOUNT Parameters**

Parameter	Description
p_user_name	The user name of the user account

### Example

The following example shows how to use the UNEXPIRE\_END\_USER\_ACCOUNT procedure. Use this procedure to renew (unexpire) an Application Express end user account in the current workspace. This action specifically renews the account for use by end users to authenticate to developed applications and may also renew the account for use by developers or administrators to log in to a workspace.

This procedure must be run by a user having administration privileges in the current workspace.

```
BEGIN
    FOR c1 IN (SELECT user_name from wwv_flow_users) LOOP
        APEX_UTIL.UNEXPIRE_END_USER_ACCOUNT(p_user_name => c1.user_name);
        http.p('End User Account: ' || c1.user_name || ' is now valid.');
```

```
END LOOP;
END;
```

**See Also:** " [EXPIRE\\_END\\_USER\\_ACCOUNT Parameters](#)" on page 25-37 and " [END\\_USER\\_ACCOUNT\\_DAYS\\_LEFT Function](#)" on page 25-36

## UNEXPIRE\_WORKSPACE\_ACCOUNT Procedure

Unexpires developer and workspace administrator accounts and the associated passwords, enabling the developer or administrator to log in to a workspace.

### Syntax

```
APEX_UTIL.UNEXPIRE_WORKSPACE_ACCOUNT (
    p_user_name IN VARCHAR2);
```

### Parameters

**Table 25–114 UNEXPIRE\_WORKSPACE\_ACCOUNT Parameters**

Parameter	Description
p_user_name	The user name of the user account

### Example

The following example shows how to use the UNEXPIRE\_WORKSPACE\_ACCOUNT procedure. Use this procedure to renew (unexpire) an Application Express workspace administrator account in the current workspace. This action specifically renews the account for use by developers or administrators to login to a workspace and may also renew the account for its use by end users to authenticate to developed applications.

This procedure must be run by a user having administration privileges in the current workspace.

```
BEGIN
    FOR c1 IN (select user_name from wwv_flow_users) loop
        APEX_UTIL.UNEXPIRE_WORKSPACE_ACCOUNT(p_user_name => c1.user_name);
        http.p('Workspace Account: '||c1.user_name||' is now valid.');
```

```
END LOOP;
END;
```

**See Also:** ["EXPIRE\\_WORKSPACE\\_ACCOUNT Procedure"](#) on page 25-38 and ["WORKSPACE\\_ACCOUNT\\_DAYS\\_LEFT Function"](#) on page 25-166

## UNLOCK\_ACCOUNT Procedure

Sets a user account status to unlocked. Must be run by an authenticated workspace administrator in a page request context.

### Syntax

```
APEX_UTIL.UNLOCK_ACCOUNT (
    p_user_name IN VARCHAR2);
```

### Parameters

**Table 25–115 UNLOCK\_ACCOUNT Parameters**

Parameter	Description
p_user_name	The user name of the user account

### Example

The following example shows how to use the UNLOCK\_ACCOUNT procedure. Use this procedure to unlock an Application Express account in the current workspace. This action unlocks the account for use by administrators, developers, and end users.

This procedure must be run by a user who has administration privileges in the current workspace

```
BEGIN
    FOR c1 IN (SELECT user_name from wwv_flow_users) LOOP
        APEX_UTIL.UNLOCK_ACCOUNT(p_user_name => c1.user_name);
        http.p('End User Account: ' || c1.user_name || ' is now unlocked. ');
    END LOOP;
END;
```

**See Also:** ["LOCK\\_ACCOUNT Procedure"](#) on page 25-106 and ["GET\\_ACCOUNT\\_LOCKED\\_STATUS Function"](#) on page 25-51

## URL\_ENCODE Function

The following special characters are encoded as follows:

Special Characters	After Encoding
%	%25
+	%2B
space	+
.	%2E
*	%2A
?	%3F
\	%5C
/	%2F
>	%3E
<	%3C
}	%7B
{	%7D
~	%7E
[	%5B
]	%5D
'	%60
;	%3B
?	%3F
@	%40
&	%26
#	%23
	%7C
^	%5E
:	%3A
=	%3D
\$	%24

### Syntax

```
APEX_UTIL.URL_ENCODE (
  p_url IN VARCHAR2)
RETURN VARCHAR2;
```

### Parameters

**Table 25–116 URL\_ENCODE Parameters**

Parameter	Description
p_url	The string to be encoded

### Example

The following example shows how to use the URL\_ENCODE function.

```
DECLARE
  l_url VARCHAR2(255);
BEGIN
  l_url := APEX_UTIL.URL_ENCODE('http://www.myurl.com?id=1&cat=foo');
END;
```

In this example, the following URL:

```
http://www.myurl.com?id=1&cat=foo
```

Would be returned as:

```
http%3A%2F%2Fwww%2Emyurl%2Ecom%3Fid%3D1%26cat%3Dfoo
```

## WORKSPACE\_ACCOUNT\_DAYS\_LEFT Function

Returns the number of days remaining before the developer or workspace administrator account password expires. This function may be run in a page request context by any authenticated user.

### Syntax

```
APEX_UTIL.WORKSPACE_ACCOUNT_DAYS_LEFT (
    p_user_name IN VARCHAR2)
RETURN NUMBER;
```

### Parameters

**Table 25–117** *WORKSPACE\_ACCOUNT\_DAYS\_LEFT Parameters*

Parameter	Description
p_user_name	The user name of the user account

### Example

The following example shows how to use the WORKSPACE\_ACCOUNT\_DAYS\_LEFT function. It can be used in to find the number of days remaining before an Application Express administrator or developer account in the current workspace expires.

```
DECLARE
    l_days_left NUMBER;
BEGIN
    FOR c1 IN (SELECT user_name from wwv_flow_users) LOOP
        l_days_left := APEX_UTIL.WORKSPACE_ACCOUNT_DAYS_LEFT(p_user_name =>
c1.user_name);
        htp.p('Workspace Account:'||c1.user_name||' expires in '||l_days_left||'
days. ');
    END LOOP;
END;
```

**See Also:** ["EXPIRE\\_WORKSPACE\\_ACCOUNT Procedure"](#) on page 25-38 and ["UNEXPIRE\\_WORKSPACE\\_ACCOUNT Procedure"](#) on page 25-162



---

---

## APEX\_WEB\_SERVICE

The APEX\_WEB\_SERVICE API enables you to integrate other systems with Application Express by allowing you to interact with Web services anywhere you can use PL/SQL in your application. The API contains procedures and functions to call both SOAP and RESTful style Web services. It contains functions to parse the responses from Web services and to encode/decode into SOAP friendly base64 encoding.

This API also contains package globals for managing cookies and HTTP headers when calling Web services whether from the API or by using standard processes of type Web service. Cookies and HTTP headers can be set before invoking a call to a Web service by populating the globals and the cookies and HTTP headers returned from the Web service response can be read from other globals.

- [About the APEX\\_WEB\\_SERVICE API](#)
- [Invoking a SOAP Style Web Service](#)
- [Invoking a RESTful Style Web Service](#)
- [Retrieving Cookies and HTTP Headers](#)
- [Setting Cookies and HTTP Headers](#)
- [BLOB2CLOBBASE64 Function](#)
- [CLOBBASE642BLOB Function](#)
- [MAKE\\_REQUEST Procedure](#)
- [MAKE\\_REQUEST Function](#)
- [MAKE\\_REST\\_REQUEST Function](#)
- [MAKE\\_REST\\_REQUEST\\_B Function](#)
- [PARSE\\_RESPONSE Function](#)
- [PARSE\\_RESPONSE\\_CLOB Function](#)
- [PARSE\\_XML Function](#)
- [PARSE\\_XML\\_CLOB Function](#)

## About the APEX\_WEB\_SERVICE API

Use the APEX\_WEB\_SERVICE API to invoke a Web service and examine the response anywhere you can use PL/SQL in Application Express.

The following are examples of when you might use the APEX\_WEB\_SERVICE API:

- When you want to invoke a Web service by using an On Demand Process using Ajax.
- When you want to invoke a Web service as part of an Authentication Scheme.
- When you need to pass a large binary parameter to a Web service that is base64 encoded.
- When you want to invoke a Web service as part of a validation.

## Invoking a SOAP Style Web Service

There is a procedure and a function to invoke a SOAP style Web service. The procedure stores the response in the collection specified by the parameter `p_collection_name`. The function returns the results as an XMLTYPE. To retrieve a specific value from the response, you use either the `PARSE_RESPONSE` function if the result is stored in a collection or the `PARSE_XML` function if the response is returned as an XMLTYPE.

To pass a binary parameter to the Web service as base64 encoded character data, use the function `BLOB2CLOBBASE64`. Conversely, to transform a response that contains a binary parameter that is base64 encoded use the function `CLOBBASE642BLOB`.

The following is an example of using the `BLOB2CLOBBASE64` function to encode a parameter, `MAKE_REQUEST` procedure to call a Web service, and the `PARSE_RESPONSE` function to extract a specific value from the response.

```

declare
  l_filename varchar2(255);
  l_BLOB BLOB;
  l_CLOB CLOB;
  l_envelope CLOB;
  l_response_msg varchar2(32767);
BEGIN
  IF :P1_FILE IS NOT NULL THEN
    SELECT filename, BLOB_CONTENT
       INTO l_filename, l_BLOB
    FROM APEX_APPLICATION_FILES
    WHERE name = :P1_FILE;

    l_CLOB := apex_web_service.blob2clobbase64(l_BLOB);

    l_envelope := q'!<?xml version='1.0' encoding='UTF-8'?>!';
    l_envelope := l_envelope || '<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:chec="http://www.stellent.com/CheckIn/">
<soapenv:Header/>
<soapenv:Body>
  <chec:CheckInUniversal>
    <chec:dDocName>' || l_filename || '</chec:dDocName>
    <chec:dDocTitle>' || l_filename || '</chec:dDocTitle>
    <chec:dDocType>Document</chec:dDocType>
    <chec:dDocAuthor>GM</chec:dDocAuthor>
    <chec:dSecurityGroup>Public</chec:dSecurityGroup>
    <chec:dDocAccount></chec:dDocAccount>
    <chec:CustomDocMetaData>
      <chec:property>
        <chec:name></chec:name>
        <chec:value></chec:value>
      </chec:property>
    </chec:CustomDocMetaData>
    <chec:primaryFile>
      <chec:fileName>' || l_filename || '</chec:fileName>
      <chec:fileContent>' || l_CLOB || '</chec:fileContent>
    </chec:primaryFile>
    <chec:alternateFile>
      <chec:fileName></chec:fileName>
      <chec:fileContent></chec:fileContent>
    </chec:alternateFile>
  </chec:CheckInUniversal>
</soapenv:Body>
</soapenv:Envelope>';
  
```

```

        <chec:extraProps>
            <chec:property>
                <chec:name></chec:name>
                <chec:value></chec:value>
            </chec:property>
        </chec:extraProps>
    </chec:CheckInUniversal>
</soapenv:Body>
</soapenv:Envelope>';

apex_web_service.make_request(
    p_url          => 'http://127.0.0.1/idc/idcplg',
    p_action       => 'http://www.stellent.com/CheckIn/',
    p_collection_name => 'STELLENT_CHECKIN',
    p_envelope     => l_envelope,
    p_username     => 'sysadmin',
    p_password     => 'welcome1' );

l_response_msg := apex_web_service.parse_response(
    p_collection_name=>'STELLENT_CHECKIN',
    p_
xpath=>'//idc:CheckInUniversalResponse/idc:CheckInUniversalResult/idc:StatusInfo/i
dc:statusMessage/text()',
    p_ns=>'xmlns:idc="http://www.stellent.com/CheckIn/"');

:P1_RES_MSG := l_response_msg;

END IF;
END;
```

---

## Invoking a RESTful Style Web Service

RESTful style Web services use a simpler architecture than SOAP. Typically the input to a RESTful style Web service is a collection of name/value pairs. The response can be an XML document or simply text such as a comma separated response or JSON.

The following is an example of MAKE\_REST\_REQUEST being used in an application process that is callable by Ajax.

```
declare
  l_clob clob;
  l_buffer      varchar2(32767);
  l_amount      number;
  l_offset      number;
begin
  l_clob := apex_web_service.make_rest_request(
    p_url => 'http://us.music.yahooapis.com/
video/v1/list/published/popular',
    p_http_method => 'GET',
    p_parm_name => apex_util.string_to_table('appid:format'),
    p_parm_value => apex_util.string_to_table(apex_application.g_
x01||':'||apex_application.g_x02));

  l_amount := 32000;
  l_offset := 1;
  begin
    loop
      dbms_lob.read( l_clob, l_amount, l_offset, l_buffer );
      http.p(l_buffer);
      l_offset := l_offset + l_amount;
      l_amount := 32000;
    end loop;
  exception
    when no_data_found then
      null;
  end;
end;
```

## Retrieving Cookies and HTTP Headers

When you invoke a Web service using any of the supported methods in Application Express, the `g_response_cookies` and `g_headers` globals are populated if the Web service response included any cookies or HTTP headers. You can interrogate these globals and store the information in collections.

The following are examples of interrogating the `APEX_WEB_SERVICE` globals to store cookie and HTTP header responses in collections.

```
declare
  i number;
  secure varchar2(1);
begin
  apex_collection.create_or_truncate_collection('P31_RESP_COOKIES');
  for i in 1.. apex_web_service.g_response_cookies.count loop
    IF (apex_web_service.g_response_cookies(i).secure) THEN
      secure := 'Y';
    ELSE
      secure := 'N';
    END IF;
    apex_collection.add_member(p_collection_name => 'P31_RESP_COOKIES',
      p_c001 => apex_web_service.g_response_cookies(i).name,
      p_c002 => apex_web_service.g_response_cookies(i).value,
      p_c003 => apex_web_service.g_response_cookies(i).domain,
      p_c004 => apex_web_service.g_response_cookies(i).expire,
      p_c005 => apex_web_service.g_response_cookies(i).path,
      p_c006 => secure,
      p_c007 => apex_web_service.g_response_cookies(i).version );
  end loop;
end;

declare
  i number;
begin
  apex_collection.create_or_truncate_collection('P31_RESP_HEADERS');

  for i in 1.. apex_web_service.g_headers.count loop
    apex_collection.add_member(p_collection_name => 'P31_RESP_HEADERS',
      p_c001 => apex_web_service.g_headers(i).name,
      p_c002 => apex_web_service.g_headers(i).value,
      p_c003 => apex_web_service.g_status_code);
  end loop;
end;
```

---

## Setting Cookies and HTTP Headers

You set cookies and HTTP headers that should be sent along with a Web service request by populating the globals `g_request_cookies` and `g_request_headers` before the process that invokes the Web service.

The following examples show populating the globals to send cookies and HTTP headers with a request.

```
for c1 in (select seq_id, c001, c002, c003, c004, c005, c006, c007
          from apex_collections
          where collection_name = 'P31_RESP_COOKIES' ) loop
  apex_web_service.g_request_cookies(c1.seq_id).name := c1.c001;
  apex_web_service.g_request_cookies(c1.seq_id).value := c1.c002;
  apex_web_service.g_request_cookies(c1.seq_id).domain := c1.c003;
  apex_web_service.g_request_cookies(c1.seq_id).expire := c1.c004;
  apex_web_service.g_request_cookies(c1.seq_id).path := c1.c005;
  if c1.c006 = 'Y' then
    apex_web_service.g_request_cookies(c1.seq_id).secure := true;
  else
    apex_web_service.g_request_cookies(c1.seq_id).secure := false;
  end if;
  apex_web_service.g_request_cookies(c1.seq_id).version := c1.c007;
end loop;
```

```
for c1 in (select seq_id, c001, c002
          from apex_collections
          where collection_name = 'P31_RESP_HEADERS' ) loop
  apex_web_service.g_request_headers(c1.seq_id).name := c1.c001;
  apex_web_service.g_request_headers(c1.seq_id).value := c1.c002;
end loop;
```

## BLOB2CLOBBASE64 Function

Use this function to convert a BLOB datatype into a CLOB that is base64 encoded. This is often used when sending a binary as an input to a Web service.

### Syntax

```
APEX_WEB_SERVICE.BLOB2CLOBBASE64 (  
    p_blob IN BLOB)  
RETURN CLOB;
```

### Parameters

[Table 26–1](#) describes the parameters available in the BLOB2CLOBBASE64 function.

**Table 26–1** BLOB2CLOBBASE64 Parameters

Parameter	Description
p_blob	The BLOB to convert into base64 encoded CLOB.

### Example

The following example gets a file that was uploaded from the apex\_application\_files view and converts the BLOB into a CLOB that is base64 encoded.

```
declare  
    l_clobCLOB;  
    l_blobBLOB;  
begin  
    SELECT BLOB_CONTENT  
        INTO l_BLOB  
        FROM APEX_APPLICATION_FILES  
        WHERE name = :P1_FILE;  
  
    l_CLOB := apex_web_service.blob2clobbase64(l_BLOB);  
end;
```



---

## CLOBBASE642BLOB Function

Use this function to convert a CLOB datatype that is base64 encoded into a BLOB. This is often used when receiving output from a Web service that contains a binary parameter.

### Syntax

```
APEX_WEB_SERVICE.CLOBBASE642BLOB (  
    p_clob IN CLOB)  
RETURN BLOB;
```

### Parameters

[Table 26–2](#) describes the parameters available in the CLOBBASE642BLOB function.

**Table 26–2 CLOBBASE642BLOB Parameters**

Parameter	Description
p_clob	The base64 encoded CLOB to convert into a BLOB.

### Example

The following example retrieves a base64 encoded node from an XML document as a CLOB and converts it into a BLOB.

```
declare  
    l_base64CLOB;  
    l_blobBLOB;  
    l_xml XMLTYPE;  
begin  
    l_base64 := apex_web_service.parse_xml_clob(l_xml, '  
//runReportReturn/reportBytes/text()');  
    l_blob := apex_web_service.clobbase642blob(l_base64);  
end;
```

## MAKE\_REQUEST Procedure

Use this procedure to invoke a SOAP style Web service with the supplied SOAP envelope and store the results in a collection.

### Syntax

```
APEX_WEB_SERVICE.MAKE_REQUEST (
    p_url           IN VARCHAR2,
    p_action        IN VARCHAR2 default null,
    p_version       IN VARCHAR2 default '1.1',
    p_collection_name IN VARCHAR2 default null,
    p_envelope      IN CLOB,
    p_username      IN VARCHAR2 default null,
    p_password      IN VARCHAR2 default null,
    p_scheme        IN VARCHAR2 DEFAULT 'Basic',
    p_proxy_override IN VARCHAR2 default null,
    p_transfer_timeout IN NUMBER default 180,
    p_wallet_path   IN VARCHAR2 default null,
    p_wallet_pwd    IN VARCHAR2 default null );
```

### Parameters

[Table 26–3](#) describes the parameters available in the MAKE\_REQUEST procedure.

**Table 26–3** MAKE\_REQUEST Procedure Parameters

Parameter	Description
p_url	The URL endpoint of the Web service.
p_action	The SOAP Action corresponding to the operation to be invoked.
p_version	The SOAP version, 1.1 or 1.2. The default is 1.1.
p_collection_name	The name of the collection to store the response.
p_envelope	The SOAP envelope to post to the service.
p_username	The username if basic authentication is required for this service.
p_password	The password if basic authentication is required for this service.
p_scheme	The authentication scheme, Basic (default) or AWS or Digest if supported by your database release.
p_proxy_override	The proxy to use for the request. The proxy supplied overrides the proxy defined in the application attributes.
p_transfer_timeout	The amount of time in seconds to wait for a response.
p_wallet_path	The file system path to a wallet if the URL endpoint is https. For example, file:/usr/home/oracle/WALLETS. The wallet path provided overrides the wallet defined in the instance settings.
p_wallet_pwd	The password to access the wallet.

### Example

The following example uses the `make_request` procedure to retrieve a list of movies from a SOAP style Web service. The response is stored in an Application Express collection named `MOVIE_LISTINGS`.

```
declare
    l_envelope CLOB;
```

```
BEGIN
l_envelope := '<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tns="http://www.ignyte.com/whatsshowing"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <tns:GetTheatersAndMovies>
      <tns:zipCode>43221</tns:zipCode>
      <tns:radius>5</tns:radius>
    </tns:GetTheatersAndMovies>
  </soap:Body>
</soap:Envelope>';

apex_web_service.make_request(
  p_url          => '
http://www.ignyte.com/webservices/ignyte.whatsshowing.webservice/moviefunctions.as
mx',
  p_action       => '
http://www.ignyte.com/whatsshowing/GetTheatersAndMovies',
  p_collection_name => 'MOVIE_LISTINGS',
  p_envelope     => l_envelope
);
END;
```

## MAKE\_REQUEST Function

Use this function to invoke a SOAP style Web service with the supplied SOAP envelope returning the results in an XMLTYPE.

### Syntax

```
APEX_WEB_SERVICE.MAKE_REQUEST (
    p_url           IN VARCHAR2,
    p_action        IN VARCHAR2 default null,
    p_version       IN VARCHAR2 default '1.1',
    p_envelope      IN CLOB,
    p_username      IN VARCHAR2 default null,
    p_password      IN VARCHAR2 default null,
    p_scheme        IN VARCHAR2 default 'Basic',
    p_proxy_override IN VARCHAR2 default null,
    p_transfer_timeout IN NUMBER default 180,
    p_wallet_path   IN VARCHAR2 default null,
    p_wallet_pwd    IN VARCHAR2 default null )
RETURN XMLTYPE;
```

### Parameters

[Table 26–4](#) describes the parameters available in the MAKE\_REQUEST function.

**Table 26–4 MAKE\_REQUEST Function Parameters**

Parameter	Description
p_url	The URL endpoint of the Web service.
p_action	The SOAP Action corresponding to the operation to be invoked.
p_version	The SOAP version, 1.1 or 1.2. The default is 1.1.
p_envelope	The SOAP envelope to post to the service.
p_username	The username if basic authentication is required for this service.
p_password	The password if basic authentication is required for this service.
p_scheme	The authentication scheme, Basic (default) or AWS or Digest if supported by your database release.
p_proxy_override	The proxy to use for the request. The proxy supplied overrides the proxy defined in the application attributes.
p_transfer_timeout	The amount of time in seconds to wait for a response.
p_wallet_path	The file system path to a wallet if the URL endpoint is https. For example, file:/usr/home/oracle/WALLETS. The wallet path provided overrides the wallet defined in the instance settings.
p_wallet_pwd	The password to access the wallet.

### Example

The following example uses the make\_request function to invoke a SOAP style Web service that returns movie listings. The result is stored in an XMLTYPE.

```
declare
    l_envelope CLOB;
    l_xml XMLTYPE;
BEGIN
```

```
l_envelope := ' <?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tns="http://www.ignite.com/whatsshowing"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <tns:GetTheatersAndMovies>
      <tns:zipCode>43221</tns:zipCode>
      <tns:radius>5</tns:radius>
    </tns:GetTheatersAndMovies>
  </soap:Body>
</soap:Envelope>';

l_xml := apex_web_service.make_request(
  p_url => '
http://www.ignite.com/webservices/ignite.whatsshowing.webservice/moviefunctions.as
mx',
  p_action => ' http://www.ignite.com/whatsshowing/GetTheatersAndMovies',
  p_envelope => l_envelope
);
END
```

## MAKE\_REST\_REQUEST Function

Use this function to invoke a RESTful style Web service supplying either name value pairs, a character based payload or a binary payload and returning the response in a CLOB.

### Syntax

```
APEX_WEB_SERVICE.MAKE_REST_REQUEST(
    p_url           IN VARCHAR2,
    p_http_method  IN VARCHAR2,
    p_username     IN VARCHAR2 default null,
    p_password     IN VARCHAR2 default null,
    p_scheme      IN VARCHAR2 default 'Basic',
    p_proxy_override IN VARCHAR2 default null,
    p_transfer_timeout IN NUMBER default 180,
    p_body         IN CLOB default empty_clob(),
    p_body_blob    IN BLOB default empty_blob(),
    p_parm_name    IN apex_application_global.VC_ARR2 default empty_vc_arr,
    p_parm_value   IN apex_application_global.VC_ARR2 default empty_vc_arr,
    p_wallet_path  IN VARCHAR2 default null,
    p_wallet_pwd   IN VARCHAR2 default null )
RETURN CLOB;
```

### Parameters

Table 26–5 describes the parameters available in the MAKE\_REST\_REQUEST function.

**Table 26–5 MAKE\_REST\_REQUEST Function Parameters**

Parameter	Description
p_url	The URL endpoint of the Web service.
p_http_method	The HTTP method to use, PUT, POST, GET, HEAD, or DELETE.
p_username	The username if basic authentication is required for this service.
p_password	The password if basic authentication is required for this service.
p_scheme	The authentication scheme, Basic (default) or AWS or Digest if supported by your database release.
p_proxy_override	The proxy to use for the request. The proxy supplied overrides the proxy defined in the application attributes.
p_transfer_timeout	The amount of time in seconds to wait for a response.
p_body	The HTTP payload to be sent as CLOB.
p_body_blob	The HTTP payload to be sent as binary BLOB. For example, posting a file.
p_parm_name	The name of the parameters to be used in name/value pairs.
p_parm_value	The value of the parameters to be used in name/value pairs.
p_wallet_path	The file system path to a wallet if the URL endpoint is https. For example, file:/usr/home/oracle/WALLETS. The wallet path provided overrides the wallet defined in the instance settings.
p_wallet_pwd	The password to access the wallet.

**Example**

The following example calls a RESTful style Web service using the `make_rest_request` function passing the parameters to the service as name/value pairs. The response from the service is stored in a locally declared CLOB.

```
declare
    l_clob CLOB;
BEGIN

    l_clob := apex_web_service.make_rest_request(
        p_url => 'http://us.music.yahooapis.com/video/v1/list/published/popular',
        p_http_method => 'GET',
        p_parm_name => apex_util.string_to_table('appid:format'),
        p_parm_value => apex_util.string_to_table('xyz:xml'));

END;
```

## MAKE\_REST\_REQUEST\_B Function

Use this function to invoke a RESTful style Web service supplying either name value pairs, a character based payload or a binary payload and returning the response in a BLOB.

### Syntax

```
APEX_WEB_SERVICE.MAKE_REST_REQUEST_B(
    p_url           IN VARCHAR2,
    p_http_method  IN VARCHAR2,
    p_username     IN VARCHAR2 default null,
    p_password     IN VARCHAR2 default null,
    p_scheme       IN VARCHAR2 default 'Basic',
    p_proxy_override IN VARCHAR2 default null,
    p_transfer_timeout IN NUMBER default 180,
    p_body         IN CLOB default empty_clob(),
    p_body_blob    IN BLOB default empty_blob(),
    p_parm_name    IN apex_application_global.VC_ARR2 default empty_vc_arr,
    p_parm_value   IN apex_application_global.VC_ARR2 default empty_vc_arr,
    p_wallet_path  IN VARCHAR2 default null,
    p_wallet_pwd   IN VARCHAR2 default null )
RETURN BLOB;
```

### Parameters

Table 26–5 describes the parameters available in the MAKE\_REST\_REQUEST\_B function.

**Table 26–6 MAKE\_REST\_REQUEST\_B Function Parameters**

Parameter	Description
p_url	The URL endpoint of the Web service.
p_http_method	The HTTP method to use, PUT, POST, GET, HEAD, or DELETE.
p_username	The username if basic authentication is required for this service.
p_password	The password if basic authentication is required for this service.
p_scheme	The authentication scheme, Basic (default) or AWS or Digest if supported by your database release.
p_proxy_override	The proxy to use for the request. The proxy supplied overrides the proxy defined in the application attributes.
p_transfer_timeout	The amount of time in seconds to wait for a response.
p_body	The HTTP payload to be sent as CLOB.
p_body_blob	The HTTP payload to be sent as binary BLOB. For example, posting a file.
p_parm_name	The name of the parameters to be used in name/value pairs.
p_parm_value	The value of the parameters to be used in name/value pairs.
p_wallet_path	The file system path to a wallet if the URL endpoint is https. For example, file:/usr/home/oracle/WALLETS. The wallet path provided overrides the wallet defined in the instance settings.
p_wallet_pwd	The password to access the wallet.



**Example**

The following example calls a RESTful style Web service using the `make_rest_request` function passing the parameters to the service as name/value pairs. The response from the service is stored in a locally declared BLOB.

```
declare
    l_blob BLOB;
BEGIN

    l_blob := apex_web_service.make_rest_request_b(
        p_url => 'http://us.music.yahooapis.com/ video/v1/list/published/popular',
        p_http_method => 'GET',
        p_parm_name => apex_util.string_to_table('appid:format'),
        p_parm_value => apex_util.string_to_table('xyz:xml'));

END;
```

## PARSE\_RESPONSE Function

Use this function to parse the response from a Web service that is stored in a collection and return the result as a VARCHAR2 type.

### Syntax

```
APEX_WEB_SERVICE.PARSE_RESPONSE (
    p_collection_name  IN VARCHAR2,
    p_xpath            IN VARCHAR2,
    p_ns               IN VARCHAR2 default null )
RETURN VARCHAR2;
```

### Parameters

Table 26–7 describes the parameters available in the PARSE\_RESPONSE function.

**Table 26–7** PARSE\_RESPONSE Function Parameters

Parameter	Description
p_collection_name	The name of the collection where the Web service response is stored.
p_xpath	The XPath expression to the desired node.
p_ns	The namespace to the desired node.

### Example

The following example parses a response stored in a collection called STELLENT\_CHECKIN and stores the value in a locally declared VARCHAR2 variable.

```
declare
    l_response_msg  VARCHAR2(4000);
BEGIN
    l_response_msg := apex_web_service.parse_response(
        p_collection_name=>'STELLENT_CHECKIN',
        p_xpath =>
        '//idc:CheckInUniversalResponse/idc:CheckInUniversalResult/idc:StatusInfo/idc:stat
        usMessage/text()',
        p_ns=>'xmlns:idc="http://www.stellent.com/CheckIn/"');
END;
```

## PARSE\_RESPONSE\_CLOB Function

Use this function to parse the response from a Web service that is stored in a collection and return the result as a CLOB type.

### Syntax

```
APEX_WEB_SERVICE.PARSE_RESPONSE_CLOB (
    p_collection_name  IN VARCHAR2,
    p_xpath            IN VARCHAR2,
    p_ns              IN VARCHAR2 default null )
RETURN CLOB;
```

### Parameters

Table 26–8 describes the parameters available in the PARSE\_RESPONSE\_CLOB function.

**Table 26–8** PARSE\_RESPONSE\_CLOB Function Parameters

Parameter	Description
p_collection_name	The name of the collection where the Web service response is stored.
p_xpath	The XPath expression to the desired node.
p_ns	The namespace to the desired node.

### Example

The following example parses a response stored in a collection called STELLENT\_CHECKIN and stores the value in a locally declared CLOB variable.

```
declare
    l_response_msg CLOB;
BEGIN
    l_response_msg := apex_web_service.parse_response_clob(
        p_collection_name=>'STELLENT_CHECKIN',
        p_xpath=>
        '//idc:CheckInUniversalResponse/idc:CheckInUniversalResult/idc:StatusInfo/idc:stat
        usMessage/text()',
        p_ns=>'xmlns:idc="http://www.stellent.com/CheckIn/"');
END;
```

## PARSE\_XML Function

Use this function to parse the response from a Web service returned as an XMLTYPE and return the value requested as a VARCHAR2.

### Syntax

```
APEX_WEB_SERVICE.PARSE_XML (
    p_xml          IN XMLTYPE,
    p_xpath        IN VARCHAR2,
    p_ns           IN VARCHAR2 default null )
RETURN VARCHAR2;
```

### Parameters

Table 26–9 describes the parameters available in the PARSE\_XML function.

**Table 26–9 PARSE\_XML Function Parameters**

Parameter	Description
p_xml	The XML document as an XMLTYPE to parse.
p_xpath	The XPath expression to the desired node.
p_ns	The namespace to the desired node.

### Example

The following example uses the make\_request function to call a Web service and store the results in a local XMLTYPE variable. The parse\_xml function is then used to pull out a specific node of the XML document stored in the XMLTYPE and stores it in a locally declared VARCHAR2 variable.

```
declare
    l_envelope CLOB;
    l_xml XMLTYPE;
    l_movie VARCHAR2(4000);
BEGIN
    l_envelope := ' <?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tns="http://www.ignyte.com/whatsshowing"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <soap:Body>
        <tns:GetTheatersAndMovies>
            <tns:zipCode>43221</tns:zipCode>
            <tns:radius>5</tns:radius>
        </tns:GetTheatersAndMovies>
    </soap:Body>
</soap:Envelope>';

    l_xml := apex_web_service.make_request(
        p_url => '
http://www.ignyte.com/webservices/ignyte.whatsshowing.webservice/moviefunctions.as
mx',
        p_action => ' http://www.ignyte.com/whatsshowing/GetTheatersAndMovies',
        p_envelope => l_envelope );

    l_movie := apex_web_service.parse_xml(
        p_xml => l_xml,
```

```
    p_xpath => '  
//GetTheatersAndMoviesResponse/GetTheatersAndMoviesResult/Theater/Movies/Movie/Name[1]',  
    p_ns => ' xmlns="http://www.ignyte.com/whatsshowing" ' );  
  
END;
```

## PARSE\_XML\_CLOB Function

Use this function to parse the response from a Web service returned as an XMLTYPE and return the value requested as a CLOB.

### Syntax

```
APEX_WEB_SERVICE.PARSE_XML_CLOB (
    p_xml          IN XMLTYPE,
    p_xpath        IN VARCHAR2,
    p_ns           IN VARCHAR2 default null )
RETURN VARCHAR2;
```

### Parameters

[Table 26–10](#) describes the parameters available in the PARSE\_XML\_CLOB function.

**Table 26–10** PARSE\_XML\_CLOB Function Parameters

Parameter	Description
p_xml	The XML document as an XMLTYPE to parse.
p_xpath	The XPath expression to the desired node.
p_ns	The namespace to the desired node.

### Example

The following example uses the `make_request` function to call a Web service and store the results in a local XMLTYPE variable. The `parse_xml` function is then used to pull out a specific node of the XML document stored in the XMLTYPE and stores it in a locally declared VARCHAR2 variable

```
declare
    l_envelope CLOB;
    l_xml XMLTYPE;
    l_movie CLOB;
BEGIN
    l_envelope := ' <?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tns="http://www.ignyte.com/whatsshowing"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <soap:Body>
        <tns:GetTheatersAndMovies>
            <tns:zipCode>43221</tns:zipCode>
            <tns:radius>5</tns:radius>
        </tns:GetTheatersAndMovies>
    </soap:Body>
</soap:Envelope>';

    l_xml := apex_web_service.make_request(
        p_url => '
http://www.ignyte.com/webservices/ignyte.whatsshowing.webservice/moviefunctions.as
mx',
        p_action => ' http://www.ignyte.com/whatsshowing/GetTheatersAndMovies',
        p_envelope => l_envelope );

    l_movie := apex_web_service.parse_xml_clob(
        p_xml => l_xml,
```

```
    p_xpath => '  
//GetTheatersAndMoviesResponse/GetTheatersAndMoviesResult/Theater/Movies/Movie/Name[1]',  
    p_ns => ' xmlns="http://www.ignyte.com/whatsshowing" ' );  
  
END;
```





This package manages the zipping and unzipping of files.

- [Data Types](#)
- [ADD\\_FILE Procedure](#)
- [FINISH Procedure](#)
- [GET\\_FILE\\_CONTENT Function](#)
- [GET\\_FILES Function](#)

## Data Types

The data types used by the APEX\_ZIP package are described in this section.

### **t\_files**

```
type t_files is table of varchar2(32767) index by binary_integer;
```

---

## ADD\_FILE Procedure

This procedure adds a single file to a zip file.

---

**Note:** After all files are added, you must call `APEX_ZIP.FINISH`.

---

### Syntax

```
APEX_ZIP.ADD_FILE (
    p_zipped_blob IN OUT NOCOPY BLOB,
    p_file_name   IN VARCHAR2,
    p_content     IN BLOB );
```

### Parameters

**Table 27-1** ADD\_FILE Procedure Parameters

Parameter	Description
<code>p_zipped_blob</code>	BLOB containing the zip file.
<code>p_file_name</code>	File name, including path, of the file to be added to the zip file.
<code>p_content</code>	BLOB containing the file.

### Example

This example reads multiple files from a table and puts them into a single zip file.

```
declare
    l_zip_file blob;
begin
    for l_file in ( select file_name,
                        file_content
                    from my_files )
    loop
        apex_zip.add_file (
            p_zipped_blob => l_zip_file,
            p_file_name   => l_file.file_name,
            p_content     => l_file.file_content );
    end loop;

    apex_zip.finish (
        p_zipped_blob => l_zip_file );
end;
```

## FINISH Procedure

This procedure completes the creation of a zip file after adding files with `APEX_ZIP.ADD_FILE`.

### Syntax

```
APEX_ZIP.FINISH (  
    p_zipped_blob IN OUT NOCOPY BLOB );
```

### Parameters

**Table 27–2** *FINISH Procedure Parameters*

Parameter	Description
<code>p_zipped_blob</code>	BLOB containing the zip file.

### Example

See "[ADD\\_FILE Procedure](#)" on page 27-3 for an example.

---

## GET\_FILE\_CONTENT Function

This function returns the BLOB of a file contained in a provided zip file.

### Syntax

```
APEX_ZIP.GET_FILE_CONTENT (
    p_zipped_blob IN BLOB,
    p_file_name   IN VARCHAR2,
    p_encoding    IN VARCHAR2 DEFAULT NULL )
RETURN BLOB;
```

### Parameters

**Table 27-3** GET\_FILE\_CONTENT Function Parameters

Parameter	Description
p_zipped_blob	This is the BLOB containing the zip file.
p_file_name	File name, including path, of a file located in the zip file.
p_encoding	Encoding used to zip the file.

### Returns

**Table 27-4** GET\_FILE\_CONTENT Function Returns

Return	Description
BLOB	BLOB containing the zip file.

### Example

See "[GET\\_FILES Function](#)" on page 27-6 for an example.

## GET\_FILES Function

This function returns an array of file names, including the path, of a provided zip file that contains a BLOB.

### Syntax

```
APEX_ZIP.GET_FILES (
    p_zipped_blob IN BLOB,
    p_only_files  IN BOOLEAN DEFAULT TRUE,
    p_encoding    IN VARCHAR2 DEFAULT NULL )
RETURN t_files;
```

### Parameters

**Table 27–5** GET\_FILES Function Parameters

Parameter	Description
p_zipped_blob	This is the zip file containing the BLOB.
p_only_files	If set to TRUE, empty directory entries are not included in the returned array. Otherwise, set to FALSE to include empty directory entries.
p_encoding	This is the encoding used to zip the file.

### Returns

**Table 27–6** GET\_FILES Function Returns

Return	Description
t_files	A table of file names and path. See <a href="#">"Data Types"</a> for more details.

### Example

This example demonstrates reading a zip file from a table, extracting it and storing all files of the zip file into my\_files.

```
declare
    l_zip_file      blob;
    l_unzipped_file blob;
    l_files         apex_zip.t_files;
begin
    select file_content
       into l_zip_file
      from my_zip_files
     where file_name = 'my_file.zip';

    l_files := apex_zip.get_files (
        p_zipped_blob => l_zip_file );

    for i in 1 .. l_files.count loop
        l_unzipped_file := apex_zip.get_file_content (
            p_zipped_blob => l_zip_file,
            p_file_name   => l_files(i) );

        insert into my_files ( file_name, file_content )
           values ( l_files(i), l_unzipped_file );
    end loop;
```

```
end;
```





---

## JavaScript APIs

This section describes JavaScript functions and objects included with Oracle Application Express and available on every page. You can use these functions and objects to provide client-side functionality, such as showing and hiding page elements, or making XML HTTP Asynchronous JavaScript and XML (Ajax) requests.

- [Using the apex namespace](#)
- [apex.da namespace](#)
- [apex.debug namespace](#)
- [apex.event namespace](#)
- [apex.item](#)
- [apex.lang namespace](#)
- [apex.navigation namespace](#)
- [apex.server namespace](#)
- [apex.storage namespace](#)
- [apex.util namespace](#)
- [apex.widget namespace](#)
- [Non-namespace Javascript APIs](#)
- [Legacy JavaScript APIs](#)

## Using the apex namespace

The `apex.confirm` function displays a confirmation and depending on the user's choice either submits the page, or cancels a page submit. The `apex.submit` function submits the current page.

- [Global Variables](#)
- [About apex.confirm Function](#)
- [apex.confirm\(pMessage, pRequest\)](#)
- [apex.confirm\(pMessage, pOptions\)](#)
- [About apex.submit Function](#)
- [apex.submit\(pOptions\)](#)
- [apex.submit\(pRequest\)](#)

## Global Variables

The apex namespace stores global variables and functions in Application Express.

- `apex.gPageContext$` - Application Express variable that stores the current page context. The current page context is different depending on whether the page is a Desktop, or jQuery Mobile page. For Desktop, this is set to the document level. For jQuery Mobile, where pages are actually represented as DIV elements in the Browser DOM and multiple page DIVs can be loaded in the Browser DOM at one time, this is set to the DIV element representing the current page.

This is used to set the context for your jQuery selectors, to ensure that the selector is executing within the context of the correct page.

For example:

```
jQuery( ".my_class", apex.gPageContext$ );
```

This selects all elements with a CSS class of `my_class`, in the context of the current page.

## About apex.confirm Function

The `apex.confirm` function displays a confirmation and depending on the user's choice either submits the page, or cancels a page submit. This function has 2 signatures.

**See Also:** `"apex.confirm(pMessage, pRequest)"`,  
`"apex.confirm(pMessage, pOptions)"`

## **apex.confirm(pMessage, pRequest)**

Displays a confirmation showing a message, `pMessage`, and depending on user's choice, submits a page setting request value, `pRequest`, or cancels page submit.

### **Parameters**

`pMessage` (string)

`pRequest` (string)

### **Example**

This example shows a confirmation dialog with the text 'Delete Department'. If the user chooses to proceed with the delete, the current page is submitted with a REQUEST value of 'DELETE'

```
apex.confirm('Delete Department', 'DELETE');
```

## apex.confirm(pMessage, pOptions)

Displays a confirmation showing a message (pMessage) and depending on user's choice, submits a page setting request values specified by (pOptions) or cancels page submit.

### Parameters

pMessage (string)

pOptions (Object)

where pOptions contains one to any of the following properties:

**submitIfEnter** - If you only want to confirm when the ENTER key has been pressed, call apex.confirm in the event callback and pass the event object as this parameter.

**request** - The request value to set (defaults to null)

**set** - Object containing name/value pairs of items to be set on the page prior to submission (defaults to null).

**showWait** - Flag to control if a 'Wait Indicator' icon is displayed, which can be useful when running long page operations (Defaults to false).

### Return Values

Boolean - If the submitIfEnter option is specified, a boolean value is returned. True is returned if the ENTER key was not pressed and FALSE if the ENTER key was pressed. If submitIfEnter is not been specified, nothing is returned.

### Example

This example shows a confirmation message with the 'Save Department?' text. If the user chooses to proceed with the save, the page is submitted with a REQUEST value of 'SAVE' and 2 page item values are set, P1\_DEPTNO to 10 and P1\_EMPNO to 5433.

```
apex.confirm("Save Department?", {
  request:"SAVE",
  set:{"P1_DEPTNO":10, "P1_EMPNO":5433}
});
```

## About apex.submit Function

The `apex.submit` function submits the current page. This function has 2 signatures.

**See Also:** [apex.submit\(pOptions\)](#), [apex.submit\(pRequest\)](#)

## apex.submit(pOptions)

This function submits the page using the options specified in pOptions.

### Parameters

pOptions (Object)

where pOptions contains one to any of the following properties:

**submitIfEnter** - If you only want to submit when the ENTER key has been pressed, call apex.submit in the event callback and pass the event object as this parameter.

**request** - The request value to set (defaults to null)

**set** - Object containing name/value pairs of items to be set on the page prior to submission (defaults to null).

**showWait** - Flag to control if a 'Wait Indicator' icon is displayed, which can be useful when running long page operations (Defaults to false).

### Return Values

Boolean - If the submitIfEnter option is specified, a boolean value is returned. True is returned if the ENTER key was not pressed and FALSE if the ENTER key was pressed. If submitIfEnter is not been specified, nothing is returned.

### Example

This example submits the page with a REQUEST value of 'DELETE' and 2 page item values are set, P1\_DEPTNO to 10 and P1\_EMPNO to 5433. During submit a wait icon is displayed as visual indicator for the user as well.

```
apex.submit({
  request: "DELETE",
  set: {"P1_DEPTNO":10, "P1_EMPNO":5433}});
```



## **apex.submit(pRequest)**

This function submits the page setting the Application Express Request value pRequest.

### **Parameters**

pRequest (String)

### **Example**

Submits the current page with a REQUEST value of 'DELETE'.

```
apex.submit( 'DELETE' );
```

---

## **apex.da namespace**

This namespace holds all Dynamic Action functions in Oracle Application Express.

## apex.da.resume (pCallback, pErrorOccurred)

This function resumes execution of a Dynamic Action. Execution of a Dynamic Action can be paused, if the action's `Wait for Result` attribute is checked. The `Wait for Result` is a Dynamic Action plug-in standard attribute designed for use with Ajax based Dynamic Actions. If a plug-in exposes this attribute, it needs to resume execution by calling this function in the relevant place in the plug-in JavaScript code, otherwise, your action breaks execution of Dynamic Actions.

### Parameters

`pCallback` (function) - This is a required parameter that references a callback function available from the `this.resumeCallback` property.

`pErrorOccurred` (boolean) - This is a required parameter that indicates to the framework whether an error has occurred. If an error has occurred and the action's `Stop Execution on Error` attribute is checked, execution of the Dynamic Action is stopped.

### Return Values

None

### Example 1

Resume execution of the actions indicating that no error has occurred, for example from a success callback of an Ajax based action.

```
apex.da.resume( lResumeCallback, false );
```

### Example 2

Resume execution of the actions indicating that an error has occurred, for example from an error callback of an Ajax based action. If the action's

`Stop Execution on Error` attribute is checked, execution of the dynamic action is stopped.

```
apex.da.resume( lResumeCallback, true );
```

## apex.debug namespace

This namespace stores all debug functions of Oracle Application Express.

- [Log Level Constants](#)
- [apex.debug.error\(...\\*\)](#)
- [apex.debug.getLevel\(\)](#)
- [apex.debug.info\(...\\*\)](#)
- [apex.debug.log\(...\\*\)](#)
- [apex.debug.message\(pLevel,...\\*\)](#)
- [apex.debug.setLevel\(pLevel\)](#)
- [apex.debug.trace\(...\\*\)](#)
- [apex.debug.warn\(...\\*\)](#)

## Log Level Constants

### LOG\_LEVEL

```
apex.debug.LOG_LEVEL = {  
    OFF: 0,  
    ERROR: 1,  
    WARN: 2,  
    INFO: 4,  
    APP_TRACE: 6,  
    ENGINE_TRACE: 9  
};
```

**Table 28–1** LOG\_LEVEL Descriptions

Value	Description
OFF: 0	Logging is off.
ERROR: 1	Error logging level
WARN: 2	Warning logging level.
INFO: 4	Information logging level.
APP_TRACE: 6	Application tracing logging level.
ENGINE_TRACE: 9	Engine tracing logging level.

## apex.debug.error(...\*)

Log an error message. The error function always writes the error regardless of the log level from the server or set with `apex.debug.setLevel`. Messages are written using the browsers built-in console logging if available. If supported `console.trace` is called. Older browsers may not support the console object or all of its features.

### Parameters

**Table 28–2** *Parameters for `debug.error(...*)`*

Name	Type	Optional/Required	Default	Description
...*	arguments	Required		Any number of parameters logged to the console.

### Example 1

This example writes the message "Update Failed" to the console.

```
apex.debug.error("Update Failed");
```

### Example 2

This example writes an exception message to the console.

```
apex.debug.error("Exception: ", ex);
```

## apex.debug.getLevel()

Method that returns the debug log level. The debug log level is synchronized with hidden item "#pdebug"

### Return Values

Returns logging level as an integer 1 to 9 or 0 to indicate debug logging is turned off. See [Log Level Constants](#) on page 28-13 for return value meanings.

### Parameters

None

### Example

This example retrieves the logging level, prepends "Level" and logs to the console.

```
apex.debug.log("Level=", apex.debug.getLevel());
```

## apex.debug.info(...\*)

Log an informational message. Similar to `apex.debug.message` with the level set to INFO.

### Parameters

**Table 28–3** *Parameters for debug.info( ...\*)*

Name	Type	Optional/Required	Default	Description
...*	arguments	Required		Any number of parameters logged to the console.

### Example 1

This example prints an informational message to the console if the log level is INFO or greater.

```
apex.debug.info("Command successful");
```



## apex.debug.log(...\*)

Log a message. Similar to `apex.debug.message` with the level set to the highest level.

### Parameters

**Table 28–4** Parameters for `debug.log(...*)`

Name	Type	Optional/Required	Default	Description
...*	arguments	Required		Any number of parameters logged to the console.

### Example 1

This example gets the logging level and writes it to the console, regardless of the current logging level.

```
apex.debug.log("Level=", apex.debug.getLevel());
```

## apex.debug.message(pLevel,...\*)

Log a message at the given debug log level. The log level set from the server or with `apex.debug.setLevel` controls if the message is actually written. If the set log level is  $\geq$  `pLevel` then the message is written. Messages are written using the browsers built-in console logging if available. Older browsers may not support the console object or all of its features.

### Parameters

**Table 28–5 Parameters for `debug.message( pLevel )`**

Name	Type	Optional/Required	Default	Description
<code>pLevel</code>	NUMBER	Required		A number from 1 to 9 where level 1 is most important and level 9 is least important. Can be one of the <code>LOG_LEVEL</code> constants. Any other value such as 0 will turn off debug logging.
<code>...*</code>	arguments	Required		Any number of parameters logged to the console.

### Example

This example writes the message "Testing" to the console if the logging level is greater than or equal to 7.

```
apex.debug.message(7, "Testing");
```

## apex.debug.setLevel(pLevel)

Method that sets the debug log level. Log messages at or below the specified level are written to the console log. It is rarely necessary to call this function because the debug log level is synchronized with the hidden item #pdebug that comes from the server.

### Parameters

**Table 28–6** Parameters for `debug.setLevel( pLevel )`

Name	Type	Optional/Required	Default	Description
pLevel	NUMBER	Required		A number from 1 to 9 where level 1 is most important and level 9 is least important. Can be one of the LOG_LEVEL constants. Any other value such as 0 will turn off debug logging.

### Example

This example sets the logging level to application tracing.

```
apex.debug.setLevel( apex.debug.LOG_LEVEL.APP_TRACE );
```

## apex.debug.trace(...\*)

Log a trace message. Similar to `apex.debug.message` with the level set to `APP_TRACE`.

### Parameters

**Table 28–7** *Parameters for `debug.trace(...*)`*

Name	Type	Optional/Required	Default	Description
...*	arguments	Required		Any number of parameters logged to the console.

### Example 1

This example writes a log message to the console if the debug log level is `APP_TRACE` or greater.

```
apex.debug.trace("Got click event: ", event);
```

## apex.debug.warn(...\*)

Log a warning message. Similar to `apex.debug.message` with the level set to `WARN`.

### Parameters

**Table 28–8** *Parameters for `debug.warn(...*)`*

Name	Type	Optional/Required	Default	Description
...*	arguments	Required		Any number of parameters logged to the console.

### Example 1

This example writes a warning message to the console if the debug log level is `WARN` or greater.

```
apex.debug.warn("Empty string ignored");
```

## **apex.event namespace**

The `apex.event` namespace stores all event related functions of Oracle Application Express.

- `apex.event.trigger(pSelector,pEvent,pData)`

## **apex.event.trigger(pSelector,pEvent,pData)**

Given a jQuery selector, jQuery object or DOM Node the specified pEvent is triggered. pEvent can be a browser event like "click" or "change" but also a custom event like "slidechange". This function should only be used to trigger events that are handled by the dynamic action framework. Otherwise, custom events registered by plug-ins installed in your application or any event that is already exposed in dynamic actions can be compromised.

### **Parameters**

pSelector (jQuery selector | jQuery object | DOM Node)

pEvent (String)

pData (Object)

### **Return Value**

Boolean

## apex.item

The `apex.item` API provides a single interface for item related functionality of Application Express. This API returns an Application Express item object, which can then be used to access item related functions and properties.

- `apex.item( pNd )`
- `apex.item( pNd ).addValue( pValue )`
- `apex.item( pNd ).disable()`
- `apex.item( pNd ).enable()`
- `apex.item( pNd ).getValue()`
- `apex.item( pNd ).hide( pHideRow )`
- `apex.item( pNd ).isEmpty()`
- `apex.item( pNd ).setFocus()`
- `apex.item( pNd ).setStyle( pPropertyName, pPropertyValue )`
- `apex.item( pNd ).setValue( pValue, pDisplayValue, pSuppressChangeEvent )`
- `apex.item( pNd ).show( pShowRow )`



## apex.item( pNd )

This API returns an Application Express item object, which can then be used to access item related functions and properties. The API returns an Application Express item object, which can then be used to access item related functions and properties.

Plug-in developers can override much of the behavior defined in the `apex.item` namespace, by calling `apex.widget.initPageItem` with their overrides. See the documentation on "[apex.widget.initPageItem\( pName, pOptions\)](#)" on page 28-75, for more details.

### Parameters

**Table 28–9 Parameters for apex.item( pNd )**

Name	Type	Optional/Required	Default	Description
pNd	(DOM Node   String)	Required		Application Express item name or DOM node.

### Return Values

[Table 28–10, "Return Values for apex.item\( pNd \)"](#) describes the return values for this function.

**Table 28–10 Return Values for apex.item( pNd )**

Type	Description
(Object)	Returns the Application Express item object, which is used to access item specific functions. For example <code>getValue</code> , <code>setValue</code> , and so on.

### Examples

This will not be used by itself, rather it is used to access item specific functions and properties, as documented in the following APIs

## apex.item( pNd ).addValue( pValue )

Adds a value to an Application Express item that supports multiple values.

### Parameters

**Table 28–11** Parameters for *apex.item( pNd ).addValue( pValue )*

Name	Type	Optional/Required	Default	Description
pValue	(String)	Required		The value to be set.

### Return Values

None.

### Examples

In this example, the page item called 'P1\_ITEM' will have the value '100' added to the values currently selected.

```
apex.item( "P1_ITEM" ).addValue('100') ;
```

## **apex.item( pNd ).disable()**

Disables the Application Express item value, taking into account the item type, making it unavailable for edit.

### **Parameters**

None.

### **Return Values**

None.

### **Examples**

In this example, the page item called 'P1\_ITEM' will be disabled and unavailable for edit.

```
apex.item( "P1_ITEM" ).disable() ;
```

## **apex.item( pNd ).enable()**

Enables the Application Express item value, taking into account the item type, making it available for edit.

### **Parameters**

None.

### **Return Values**

None.

### **Examples**

In this example, the page item called 'P1\_ITEM' will be enabled and available for edit.

```
apex.item( "P1_ITEM" ).enable() ;
```

## apex.item( pNd ).getValue()

Returns the current value of an Application Express item on a page, taking into account the current item type. This does not return the item's current value from session state (although that could be the same), rather it will return the value as it is on the current page.

There are 2 related functions to `.getValue()`. `$v( pNd )` which returns an item's value, but in the format it will be posted. This will either be a single value, or if the item supports multiple values, will be a ':' colon separated list of values. There is also the `$v2( pNd )` function, which is just a shortcut to `.getValue()` and returns either a single value, or array of values.

### Parameters

None.

### Return Values

**Table 28–12** *Return Values for apex.item( pNd ).getValue()*

Name	Description
(String   Array)	Returns either a single string value or array of string values if the item supports multiple values (for example the 'Select List' with attribute 'Allow Multi Selection' set to 'Yes' or 'Shuttle' native item types).

### Examples

In this example, the current value of the page item called 'P1\_ITEM' will be shown in an alert.

```
alert( "P1_ITEM value = " + apex.item( "P1_ITEM" ).getValue() );
```

## apex.item( pNd ).hide( pHideRow)

Hides the Application Express item value, taking into account the item type. When using the .hide() function, it is important to understand the following:

- If the item being hidden is rendered on a page using table layout (meaning the page references a page template with Grid Layout Type set to 'HTML Table'), and the call to hide has specified to hide the entire table row (pHideRow = true), then it is assumed that everything pertaining to the item is contained in that row, and the entire row will be hidden.
- If the item being hidden is rendered on a page using table layout, and the call to hide has specified not to hide the entire table row (pHideRow = false, or not passed), then the function will attempt to hide the item's label, where the FOR attribute matches the ID of the item.
- If the item being hidden is rendered on a page using grid layout (meaning the page references a page template with Grid Layout Type set to either 'Fixed Number of Columns', or 'Variable Number of Columns'), and the item references a Label template that includes a Field Container element with a known ID (so where the Field Container > Before Label and Item attribute includes an HTML element with id="#CURRENT\_ITEM\_CONTAINER\_ID#"), then it is assumed that everything pertaining to the item is contained in the Field Container, and this will be hidden.

### Parameters

**Table 28–13 Parameters for apex.item( pDN ).hide( pHideRow )**

Name	Type	Optional	Required	Default	Description
pHideRow	(String   Array)	Optional		false	If TRUE, hides the nearest containing table row (TR). Only applicable when item is on a page using table layout (meaning the page references a page template with Grid Layout Type set to 'HTML Table').

### Return Values

None.

### Examples

In this example, the page item called P1\_ITEM will be hidden. If P1\_ITEM is on a page using grid layout and the item references a Label template that includes a Field Container element with a known ID (as detailed above), then that container element will be hidden. Otherwise just the item and its corresponding label will be hidden.

```
apex.item( "P1_ITEM" ).hide();
```

In this example, the page item called P1\_ITEM's nearest containing table row (TR) will be hidden (as pHideRow = true). Hiding the entire table row should only be used on a page using table layout. If P1\_ITEM is on a page using grid layout, then passing pHideRow = true will not work and could result in adverse consequence for the page layout, where an incorrect table row is wrongly hidden.

```
apex.item( "P1_ITEM" ).hide(true);
```

## apex.item( pNd ).isEmpty()

Returns TRUE or FALSE if an Application Express item is empty and considers any item value consisting of only whitespace including space, tab, or form-feed, as empty. This also respects if the item type uses a List of Values, and a 'Null Return Value' has defined in the List of Values. In that case, the 'Null Return Value' is used to assert if the item is empty.

### Parameters

None.

### Return Values

**Table 28–14** Return Values for apex.item( pNd ).isEmpty()

Type	Description
(Boolean)	Returns TRUE or FALSE if an Application Express item is empty.

### Examples

In this example, the call to .isEmpty() determines if the page item called 'P1\_ITEM' is empty, and if so displays an alert.

```
if( apex.item( "P1_ITEM" ).isEmpty() ) {  
    alert( "P1_ITEM empty!" );  
}
```

## **apex.item( pNd ).setFocus()**

Places user focus on the Application Express item, taking into account how specific items are designed to receive focus.

### **Parameters**

None.

### **Return Values**

None.

### **Examples**

In this example, user focus is set to the page item called 'P1\_ITEM'.

```
apex.item( "P1_ITEM" ).setFocus();
```



## apex.item( pNd ).setStyle( pPropertyName, pPropertyValue )

Sets a style for the Application Express item, taking into account how specific items are designed to be styled.

### Parameters

**Table 28–15** Parameters for apex.item( pNd ).setStyle( pPropertyName, pPropertyValue )

Name	Type	Optional/Required	Default	Description
pPropertyName	(CSS Property Name)	Required		The CSS property name that will be set.
pPropertyValue	(CSS Property Value)	Required		The value used to set the CSS property.

### Return Values

None.

### Examples

In this example, the CSS property 'color' will be set to 'red' for the page item called 'P1\_ITEM'.

```
apex.item( "P1_ITEM" ).setStyle( "color", "red" );
```

## apex.item( pNd ).setValue(pValue, pDisplayValue, pSuppressChangeEvent)

Sets the Application Express item value, taking into account the item type. This function sets the current value of an Application Express item on the page, not the item's current value in session state. It also allows for the caller to suppress the 'change' event for the item being set, if desired.

See the `$$s( pNd, pValue, pDisplayValue, pSuppressChangeEvent )` function for a shortcut to `.setValue()`.

### Parameters

**Table 28–16 Parameters for apex.item( pNd ).setValue( pValue, pDisplayValue, pSuppressChangeEvent)**

Name	Type	Optional/Required	Default	Description
pValue	(String   Array)	Required		The value to be set. For items that support multiple values (for example a 'Shuttle'), an array of string values can be passed to set multiple values at once.
pDisplayValue	(String)	Optional		Optional parameter used to set the page item's display value, in the case where the return value is different. For example for the item type "Popup LOV", with the attribute "Input Field" = "Not Enterable, Show Display Value and Store Return Value", this value sets the "Input Field". The value of pValue is then used to set the item's hidden return field.
pSuppressChangeEvent	(Boolean)	Optional	false	Pass TRUE to prevent the 'change' event from being triggered, for the item being set.

### Return Values

None.

### Examples

In this example, the value of the page item called P1\_ITEM will be set to "10". As pSuppressChangeEvent has not been passed, the default behavior of the 'change' event triggering for P1\_ITEM will occur.

```
apex.item( "P1_ITEM" ).setValue( "10" );
```

In this example P1\_ITEM is a "Popup LOV" page item with the attribute "Input Field" = "Not Enterable, Show Display Value and Store Return Value", set to "Input Field". The display value of P1\_ITEM will be set to "SALES" and the hidden return value will

be set to "10". As 'true' has been passed for the pSuppressChangeEvent parameter, the 'change' event will not trigger for the P1\_ITEM item.

```
apex.item( "P1_ITEM" ).setValue( "10", "SALES", true );
```

## apex.item( pNd ).show( pShowRow )

Shows the Application Express item value, taking into account the item type. When using the .show() function, it is important to understand the following:

- If the item being shown is rendered on a page using table layout (meaning the page references a page template with Grid Layout Type set to 'HTML Table'), and the call to show has specified to show the entire table row (pShowRow = true), then it is assumed that everything pertaining to the item is contained in that row, and the entire row will be shown.
- If the item being shown is rendered on a page using table layout, and the call to show has specified not to show the entire table row (pShowRow = false, or not passed), then the function will attempt to show the item's label, where the FOR attribute matches the ID of the item.
- If the item being shown is rendered on a page using grid layout (meaning the page references a page template with Grid Layout Type set to either 'Fixed Number of Columns', or 'Variable Number of Columns'), and the item references a Label template that includes a Field Container element with a known ID (so where the Field Container > Before Label and Item attribute includes an HTML element with id="#CURRENT\_ITEM\_CONTAINER\_ID#"), then it is assumed that everything pertaining to the item is contained in the Field Container, and this will be shown.

### Parameters

**Table 28–17 Parameters for apex.item ( pNd ).show( pShowRow )**

Name	Type	Optional/Required	Default	Description
pShowRow	(String   Array)	Optional	false	If TRUE, shows the nearest containing table row (TR). Only applicable when item is on a page using table layout (meaning the page references a page template with Grid Layout Type set to 'HTML Table').

### Return Values

None.

### Examples

In this example, the page item called P1\_ITEM will be shown. If P1\_ITEM is on a page using grid layout and the item references a Label template that includes a Field Container element with a known ID (as detailed above), then that container element will be shown. Otherwise just the item and its corresponding label will be shown.

```
apex.item( "P1_ITEM" ).show();
```

In this example, the page item called P1\_ITEM's nearest containing table row (TR) will be shown (as pShowRow = true). Showing the entire table row should only be used on a page using table layout. If P1\_ITEM is on a page using grid layout, then passing pShowRow = true will not work and could result in adverse consequence for the page layout, where an incorrect table row is wrongly shown.

```
apex.item( "P1_ITEM" ).show(true);
```

## apex.lang namespace

This namespace is used for localization related functions of Oracle Application Express.

- `apex.lang.addMessages ( pMessages )`
- `apex.lang.clearMessages ( pMessages )`
- `apex.lang.format ( pPattern, ...* )`
- `apex.lang.formatMessage ( pKey, ...* )`
- `apex.lang.formatMessageNoEscape ( pKey, ...* )`
- `apex.lang.formatNoEscape ( pPattern, ...* )`
- `apex.lang.getMessage ( pKey )`

## apex.lang.addMessages ( pMessages )

Add messages for use by `getMessage` and the format functions. Can be called multiple times. Additional messages are merged. It is generally not necessary to call this function, because it is automatically called with all the application text messages that have Used in JavaScript set to Yes.

### Parameters

**Table 28–18** Parameters for `apex.lang.addMessages( ...* )`

Name	Type	Optional/Required	Default	Description
<code>pMessages</code>	{Object}	Required		An object whose properties are message keys and the values are localized message text.

### Example

This example adds a message.

```
apex.lang.addMessages({  
    APPLY_BUTTON_LABEL: "Apply"  
});
```

## **apex.lang.clearMessages ( pMessages )**

Remove all messages.

### **Parameters**

None.

### **Example**

This example removes all messages.

```
apex.lang.clearMessages()
```

## apex.lang.format ( pPattern, ...\* )

Same as `formatMessage` except the message pattern is given directly (already localized or isn't supposed to be). It is not a key. See "[apex.lang.formatMessage \( pKey, ...\\* \)](#)" on page 28-44.

### Parameters

**Table 28–19 Parameters for `apex.lang.format( pPattern, ...* )`**

Name	Type	Optional/Required	Default	Description
pPattern	{String}	Required		The message pattern that contains one or more parameters %0 to %9.
	{...*}	Optional		Optional replacement values one for each message parameter %0 to %9. Non string arguments are converted to strings.

### Return Values

**Table 28–20 Returns for `apex.lang.format( pPattern, ...* )`**

Type	Description
{String}	The localized and formatted message text.

### Example

This example returns **Total cost: \$34.00** assuming the `totalCost` variable equals **34.00**.

```
apex.lang.format("Total cost: $%0", orderTotal);
```



## apex.lang.formatMessage ( pKey, ...\* )

Format a message. Parameters in the message %0 to %9 are replaced with the corresponding function argument. Use %% to include a single %. The replacement arguments are HTML escaped.

### Parameters

**Table 28–21** Parameters for *apex.lang.formatMessage( pKey, ...\* )*

Name	Type	Optional/Required	Default	Description
pKey	{String}	Required		The key is used to lookup the localized message text as if with getMessage.
	{...*}	Optional		Optional replacement values one for each message parameter %0 to %9.

### Return Values

**Table 28–22** Returns for *apex.lang.formatMessage( pKey, ...\* )*

Type	Description
{String}	The localized and formatted message text. If the key is not found then the key is returned.

### Example

This example returns "Process 60% complete" when the PROCESS\_STATUS message text is "Process %0%% complete" and the progress variable value is 60.

```
apex.lang.formatMessage("PROCESS_STATUS", progress);
```

## apex.lang.formatMessageNoEscape ( pKey, ...\* )

Same as `formatMessage` except the replacement arguments are not HTML escaped. They must be known to be safe or are used in a context that is safe. See "[apex.lang.formatMessage \( pKey, ...\\* \)](#)" on page 28-44.

### Parameters

**Table 28–23 Parameters for `apex.lang.formatMessageNoEscape( pKey, ...* )`**

Name	Type	Optional/Required	Default	Description
<code>pKey</code>	{String}	Required		The key is used to lookup the localized message text as if with <code>getMessage</code> .
	{...*}	Optional		Optional replacement values one for each message parameter %0 to %9.

### Return Values

**Table 28–24 Returns for `apex.lang.formatMessageNoEscape( pKey, ...* )`**

Type	Description
{String}	The localized and formatted message text. If the key is not found then the key is returned.

### Example

This example returns "You entered <ok>" when the CONFIRM message text is "You entered %0" and the `inputValue` variable value is "<ok>". Note this string must be used in a context where HTML escaping is done to avoid XSS vulnerabilities.

```
apex.lang.formatMessageNoEscape("CONFIRM", inputValue);
```

## apex.lang.formatNoEscape ( pPattern, ...\* )

Same as `format`, except the replacement arguments are not HTML escaped. They must be known to be safe or are used in a context that is safe. See "[apex.lang.format \( pPattern, ...\\* \)](#)" on page 28-40.

### Parameters

**Table 28–25** Parameters for `apex.lang.formatNoEscape( pPattern, ...* )`

Name	Type	Optional/Required	Default	Description
pPattern	{String}	Required		The message pattern that contains one or more parameters %0 to %9.
	{...*}	Optional		Optional replacement values one for each message parameter %0 to %9.

### Return Values

**Table 28–26** Returns for `apex.lang.formatNoEscape( pPattern, ...* )`

Type	Description
{String}	The localized and formatted message text. If the key is not found then the key is returned.

### Example

This example returns "You entered <ok>" when the `inputValue` variable value is "<ok>". Note this string must be used in a context where HTML escaping is done to avoid XSS vulnerabilities.

```
apex.lang.formatNoEscape("You entered %0", inputValue);
```

## apex.lang.getMessage ( pKey )

Return the message associated with the given key. The key is looked up in the messages added with addMessages.

### Parameters

**Table 28–27 Parameters for apex.lang.getMessage( pKey)**

Name	Type	Optional/Required	Default	Description
pKey	{String}	Required		The message key.

### Return Values

**Table 28–28 Returns for apex.lang.getMessage( pKey)**

Type	Description
{String}	The localized message text. If the key is not found then the key is returned.

### Example

This example returns "OK" when the localized text for key OK\_BTN\_LABEL is "OK".

```
apex.lang.getMessage ( "OK_BTN_LABEL" );
```

## apex.navigation namespace

The `apex.navigation` namespace contains popup and redirect related functions of Oracle Application Express.

- `apex.navigation.dialog(pUrl,pOptions,pCssClasses,pTriggeringElement)`
- `apex.navigation.dialog.cancel(plsModal)`
- `apex.navigation.dialog.close(plsModal,pAction)`
- `apex.navigation.dialog.fireCloseHandler(pHandler$,pAction)`
- `apex.navigation.dialog.registerCloseHandler(pOptions)`
- `apex.navigation.openInNewWindow(pUrl,pWindowName,pOptions)`
- `apex.navigation.popup(pOptions)`
- `apex.navigation.popup.close(pItem,pValue)`
- `apex.navigation.redirect(pUrl)`

## apex.navigation.dialog(pUrl,pOptions,pCssClasses,pTriggeringElement)

Opens the specified page in a dialog. For mobile UI, the page is loaded using a role 'dialog' in a mobile.changePage call. For desktop UI, a modal page is loaded in an iframe using jQuery UI dialog widget. For desktop UI, a non-modal page is loaded in a popup browser window. The names `_self`, `_parent` and `_top` should not be used. The window name is made unique so that it cannot be shared with other apps. Every effort is made to then focus the window.

### Parameters

`pUrl` {String} the url of the page to load.

`pOptions` {Object} object to identify the attributes of the dialog, with the following properties:

`title` - the title of the dialog. The default is the name of the page.

`height` - height of dialog content area, in pixels, default is 500.

`width` - width of window content area, in pixels, default 500.

`maxWidth` - maximum width of window content area, in pixels, default 1500.

`modal` - true or false. Default is true.

`dialog_attributes` - optional attribute, to allow the setting of any additional options supported by the underlying dialog implementation.

For example, to define jQuery UI Dialog attribute `resizable`:

```
resizable:false
```

See Also: See jQuery UI documentation of Dialog widget for all other available options for a modal dialog in a desktop user interface.

<http://api.jqueryui.com/>

See jQuery Mobile documentation of Dialog widget for all other available options for a modal dialog in a mobile user interface.

<http://jquerymobile.com/>

`pCssClasses` {String} to identify the CSS classes, if any, to be applied to the dialog, and appended on to the `dialogClass` attribute

`pTriggeringElement` {String} jQuery selector to identify APEX page element opening the dialog

### Return Value

Not applicable.

### Example

```
apex.navigation.dialog(url, {
  title:'About',
  height:'480',
  width:'800',
  maxWidth:'1200',
  modal:true,
  resizable:false },
'a-Dialog--uiDialog',
$('#myregion_static_id'));
```

## **apex.navigation.dialog.cancel(pIsModal)**

Closes the dialog window.

### **Parameters**

pIsModal {Boolean} to identify whether the dialog is modal.

### **Return Value**

Not applicable.

## apex.navigation.dialog.close(pIsModal,pAction)

Executes an action and then closes the dialog window.

### Parameters

pIsModal {Boolean} to identify whether the dialog is modal.

pAction {String, Function, Object} can be

- ) a URL which will trigger a redirect in the parent page
- ) a function to redirect to a different dialog page
- ) false to cancel the dialog
- ) an object of page items and values which will be exposed in the 'Dialog Closed dynamic action event

### Return Value

Not applicable.

### Example

To handle chaining from one modal dialog page to another:

```
apex.navigation.dialog.close(true, function( pDialog ) {
    apex.navigation.dialog(url, {
        title:'About',
        height:'480',
        width:'800',
        maxWidth:'1200',
        modal:true,
        dialog:pDialog,
        resizable:false },
        'a-Dialog--uiDialog',
        $('#myregion_static_id'));
});
```



## **apex.navigation.dialog.fireCloseHandler(pHandler\$,pAction)**

Fires the internal "close" event of a dialog which was registered with the registerCloseHandler when the dialog was opened.

### **Parameters**

pOptions {Object} pOptions has to contain the following attributes

- "handler\$" jQuery object where the event will be registered for.
- "dialog" DOM/jQuery/... object of the current dialog instance which will be passed into the open dialog call if the existing dialog should be re-used.
- "closeFunction" Function which is used to close the dialog.

### **Return Value**

Not applicable.

## apex.navigation.dialog.registerCloseHandler(pOptions)

Registers the internal "close" event of a dialog. The event will be triggered by `fireCloseEvent` and depending on the passed in `pAction`, it will:

- Re-use the existing dialog and navigate to a different dialog page
- Navigate to a different page in the caller
- Cancel the dialog
- Close the dialog and trigger the "apexafterclosedialog" event

### Parameters

`pOptions` {Object} `pOptions` has to contain the following attributes

"handler\$" jQuery object where the event will be registered for.

"dialog" DOM/jQuery/... object of the current dialog instance which will be passed into the open dialog call if the existing dialog should be re-used.

"closeFunction" Function which is used to close the dialog.

### Return Value

Not applicable.

## apex.navigation.openInNewWindow(pUrl,pWindowName,pOptions)

Opens the given url in a new named window or tab (the browser / browser user preference settings may control if a window or tab is used). If a window with that name already exists it is reused. The names `_self`, `_parent` and `_top` should not be used. The window name is made unique so that it cannot be shared with other apps. Every effort is made to then focus the window.

---

---

**Note:** Firefox and IE will not focus a tab if that tab is not the currently active tab in its browser window. This is why, unless `favorTabbedBrowsing` is `TRUE`, this API forces the url to open in a new window so that it has a better chance of being focused.

---

---

---

---

**Note:** For Opera, the Advanced/content > JavaScript Options: "Allow raising of windows" must be checked in order for focus to work.

---

---

---

---

**Note:** To avoid being suppressed by a popup blocker, call this from a click event handler on a link or button.

---

---

### Parameters

`pUrl` {String} the url of the page to load.

`pWindowName` {String} the name of the window (optional) The default is `"_blank"`

`pOptions` {Object} optional object with the following properties:

`favorTabbedBrowsing` - {Boolean} if true don't try to force a new window for the benefit of being able to focus it. This option only affects Firefox and IE.

### Return Value

Returns the window object of the named window or null if for some reason the window isn't opened.

### Example

```
apex.navigation.openInNewWindow(url, "MyWindow");
```

## apex.navigation.popup(pOptions)

Opens the given url in a new typically named popup window. If a window with that name already exists it is reused. If no name is given or the name is "\_blank" then a new unnamed popup window is opened. The names `_self`, `_parent` and `_top` should not be used. The window name is made unique so that it cannot be shared with other applications.

---

---

**Note:** To avoid being suppressed by a popup blocker, call this from a click event handler on a link or button.

---

---

### Parameters

pOptions {Object} an object with the following optional properties:

- url - the page url to open in the window. The default is "about:blank"
- name - the name of the window. The default is "\_blank", which opens a new unnamed window.
- height - height of window content area in pixels. Default 600
- width - width of window content area in pixels. Default 600
- scroll - "yes" or "no" Default is "yes"
- resizable - "yes" or "no" Default is "yes"
- toolbar - "yes" or "no" Default is "no"
- location - "yes" or "no" Default is "no"
- statusbar - "yes" or "no" Default is "no" This controls the status feature
- menubar - "yes" or "no" Default is "no"

### Return Value

Not applicable.

### Example

```
apex.navigation.popup ({
  url: "about:blank",
  name: "_blank",
  width: 400,
  height: 400,
  scroll: "no",
  resizable: "no",
  toolbar: "yes" });
```

## apex.navigation.popup.close(pItem,pValue)

Sets the value of the item (pItem) in the parent window, with (pValue) and then closes the popup window.

### Parameters

pItem (DOM node | string ID)  
pValue (string)

### Return Value

Not applicable.

## apex.navigation.redirect(pUrl)

Opens the specified page (pUrl) in the current window.

### Parameters

pWhere {String} the url of the page to open in the current window

### Return Value

Not applicable.

## apex.server namespace

The `apex.server` namespace stores all Ajax functions to communicate with the server part of Oracle Application Express.

- `apex.server.plugin(pAjaxIdentifier,pData,pOptions)`
- `apex.server.pluginUrl( pAjaxIdentifier, pData )`
- `apex.server.process( pName, pData, pOptions )`
- `apex.server.url( pData ) (pPage)`

## **apex.server.plugin(pAjaxIdentifier,pData,pOptions)**

This function calls the PL/SQL Ajax function which has been defined for a plug-in. This function is a wrapper of the `jQuery.ajax` function and supports all the settings the `jQuery` function provides, with additional Application Express specific features.



## Parameters



**Table 28–29** *apex.server.plugin(pAjaxIdentifier,pData,pOptions) Parameters*

Parameter	Type	Optional/Required	Description
pAjaxIdentifier	(String)	Required	Use the value returned by the PL/SQL package apex_plugin.get_ajax_identifier to identify your plug-in.
pData	{Object}	Optional	Object which can optionally be used to send additional values to be sent with the Ajax request. The special attribute pageItems which can be of type jQuery selector, jQuery or DOM object or array of item names identifies the page items which should be included in the URL. But you can also set additional parameters that the wwv_flow.show procedure provides (for example you can set the scalar parameters x01 - x10 and the arrays f01 - f20).

**Table 28–29 (Cont.) apex.server.plugin(pAjaxIdentifier,pData,pOptions) Parameters**

Parameter	Type	Optional/Required	Description
pOptions	{Object}	Optional	<p>Object which can optionally be used to set additional options used by the Ajax. The following optional Application Express specific attributes are supported:</p> <p><b>refreshObject</b> - jQuery selector, jQuery- or DOM object which identifies the DOM element for which the apexbeforerefresh and apexafterrefresh events are fired.</p> <p><b>refreshObjectData</b> - Specify data that is internally passed by the apexbeforerefresh and apexafterrefresh event triggering code, so that any handlers defined for these events can access this data. In Dynamic Actions defined for the Before Refresh or After Refresh events, this can be accessed from JavaScript using the <code>this.data</code> property. For custom jQuery event handlers, this can be accessed through the <code>pData</code> parameter of the event handler.</p> <p><b>clear</b> - JavaScript function used to clear the DOM after the apexbeforerefresh event has fired and before the actual Ajax call is triggered.</p> <p><b>loadingIndicator</b> - jQuery selector, jQuery- or DOM object which identifies the DOM element where the loading indicator should be displayed next to it. <b>loadingIndicator</b> can also be a function which gets the loading Indicator as jQuery object and has to return the jQuery reference to the created loading indicator. For example:</p> <pre>function( pLoadingIndicator ) {     return pLoadingIndicator.prependTo ( apex.jQuery( "td.shuttleControl", gShuttle ) ) }</pre> <p><b>loadingIndicatorPosition</b> - Six options to define the position of the loading indicator displayed. Only considered if the value passed to <b>loadingIndicator</b> is not a function.</p> <ul style="list-style-type: none"> <li>■ <b>before</b>: Displays before the DOM element(s) defined by <b>loadingIndicator</b>.</li> <li>■ <b>after</b>: Displays after the DOM element(s) defined by <b>loadingIndicator</b>.</li> <li>■ <b>prepend</b>: Displays inside at the beginning of the DOM element(s) defined by <b>loadingIndicator</b>.</li> <li>■ <b>append</b>: Displays inside at the end of the DOM element(s) defined by <b>loadingIndicator</b>.</li> <li>■ <b>centered</b>: Displays in the center of the DOM element defined by <b>loadingIndicator</b>.</li> <li>■ <b>page</b>: Displays in the center of the page.</li> </ul> <p><b>See Also:</b> See jQuery documentation of <code>jQuery.ajax</code> for all other available attributes. The attribute <code>dataType</code> is defaulted to <code>json</code>.</p> <p><a href="http://docs.jquery.com/">http://docs.jquery.com/</a></p>

## Return Values

**Table 28–30** Return Value

Type	Description
{Object}	<p>Returns a jqXHR object.</p> <p><b>See Also:</b> See the jQuery documentation for more details on this object:</p> <p><a href="http://docs.jquery.com/">http://docs.jquery.com/</a></p>

### Example

This call to `apex.server.plugin` sets the scalar value `x01` to `test` (which can be accessed from PL/SQL using `apex_application.g_x01`) and sets the page item's `P1_DEPTNO` and `P1_EMPNO` values in session state (using jQuery selector syntax). The `P1_MY_LIST` item is used as the element for which the `apexbeforerefresh` and `apexafterrefresh` events are fired. `P1_MY_LIST` is used as the element for which to display the loading indicator next to. The success callback is stubbed out and is used for developers to add their own code that fires when the call successfully returns. The value for `lAjaxIdentifier` must be set to the value returned by the server PL/SQL API `apex_plugin.get_ajax_identifier`.

The `pData` parameter to the success callback will contain any response sent from the call.

```
apex.server.plugin ( lAjaxIdentifier, {
    x01: "test",
    pageItems: "#P1_DEPTNO,#P1_EMPNO"
}, {
    refreshObject: "#P1_MY_LIST",
    loadingIndicator: "#P1_MY_LIST",
    success: function( pData ) { ... do something here ... }
} );
```

## apex.server.pluginUrl( pAjaxIdentifier, pData )

This function returns the URL to issue a GET request to the PL/SQL Ajax function which has been defined for a plug-in.

### Parameters

**Table 28–31** *apex.server.pluginUrl( pAjaxIdentifier, pData) Parameters*

Name	Type	Optional/Required	Default	Description
pAjaxIdentifier	(String)	Required		Use the value returned by the PL/SQL package apex_plugin.get_ajax_identifier to identify your plug-in.
pData	{Object}	Optional		Object which can optionally be used to set additional values which are included into the URL. The special attribute pageItems which can be of type jQuery selector, jQuery or DOM object or array of item names identifies the page items which are included in the URL. You can also set additional parameters that the wwv_flow.show procedure provides (for example you can set the scalar parameters x01 - x10 and the arrays f01 - f20).

### Return Value

**Table 28–32** *Return Value*

Type	Description
(String)	The URL to issue the GET request.

### Example

This call to apex.server.pluginUrl returns a URL to issue a GET request to the PL/SQL Ajax function which has been defined for a plug-in, where the URL sets the scalar value x01 to test (which can be accessed from PL/SQL using apex\_application.g\_x01) and will also set the page item's P1\_DEPTNO and P1\_EMPNO values in session state (using jQuery selector syntax). The value for lAjaxIdentifier must be set to the value returned by the server PL/SQL API apex\_plugin.get\_ajax\_identifier.

```
var lUrl = apex.server.pluginUrl ( lAjaxIdentifier, {
  x01: "test",
  pageItems: "#P1_DEPTNO,#P1_EMPNO" } );
```

## apex.server.process( pName, pData, pOptions )

This function calls a PL/SQL on-demand (Ajax callback) process defined on page or application level. This function is a wrapper of the `jQuery.ajax` function and supports all the setting the jQuery function provides but provides additional Application Express features.

### Parameters

**Table 28–33** *apex.server.process Parameters*

Name	Type	Optional/Required	Description
pName	(String)	Required	The name of the PL/SQL on-demand page or application process to call.
pData	{Object}	Optional	Object which can optionally be used to send additional values to be sent with the Ajax request. The special attribute <code>pageItems</code> which can be of type jQuery selector, jQuery or DOM object or array of item names identifies the page items which are included in the URL. You can also set additional parameters that the <code>wwv_flow.show</code> procedure provides (for example you can set the scalar parameters <code>x01 - x10</code> and the arrays <code>f01 - f20</code> ).

**Table 28–33 (Cont.) apex.server.process Parameters**

Name	Type	Optional/Required	Description
pOptions	{Object}	Optional	<p>Object which can optionally be used to set additional options used by the Ajax.</p> <p>It supports the following optional Application Express specific attributes:</p> <p><code>refreshObject</code> - jQuery selector, jQuery- or DOM object which identifies the DOM element for which the <code>apexbeforerefresh</code> and <code>apexafterrefresh</code> events are fired.</p> <p><code>refreshObjectData</code> - Specify data that is internally passed by the <code>apexbeforerefresh</code> and <code>apexafterrefresh</code> event triggering code, so that any handlers defined for these events can access this data. In Dynamic Actions defined on the <code>Before Refresh</code> or <code>After Refresh</code> events, this can be accessed from JavaScript through the <code>this.data</code> property. For custom jQuery event handlers, this can be accessed through the <code>pData</code> parameter of the event handler.</p> <p><code>clear</code> - JavaScript function used to clear the DOM after the <code>apexbeforerefresh</code> event has fired and before the actual Ajax call is triggered.</p> <p><code>loadingIndicator</code> - jQuery selector, jQuery- or DOM object which identifies the DOM element where the loading indicator should be displayed next to it. <code>loadingIndicator</code> can also be a function which gets the loading Indicator as jQuery object and has to return the jQuery reference to the created loading indicator. For example:</p> <pre>function( pLoadingIndicator ) {     return lLoadingIndicator.prependTo ( apex.jQuery( "td.shuttleControl", gShuttle )) }</pre> <p><code>loadingIndicatorPosition</code> - Six options to define the position of the loading indicator displayed. Only considered if the value passed to <code>loadingIndicator</code> is not a function.</p> <p><code>before</code>: Displays before the DOM element(s) defined by <code>loadingIndicator</code></p> <p><code>after</code>: Displays after the DOM element(s) defined by <code>loadingIndicator</code></p> <p><code>prepend</code>: Displays inside at the beginning of the DOM element(s) defined by <code>loadingIndicator</code></p> <p><code>append</code>: Displays inside at the end of the DOM element(s) defined by <code>loadingIndicator</code></p> <p><code>centered</code>: Displays in the center of the DOM element defined by <code>loadingIndicator</code>.</p> <p><code>page</code>: Displays in the center of the page.</p> <p><b>See Also:</b> See jQuery documentation of <code>jQuery.ajax</code> for all other available attributes. The attribute <code>dataType</code> is defaulted to <code>json</code>.</p> <p>See the jQuery documentation for more details on this object:</p> <p><a href="http://docs.jquery.com/">http://docs.jquery.com/</a></p>



## Return Values

**Table 28–34** Return Value

Type	Description
{Object}	Returns a jqXHR object. See the jQuery documentation for more details on this object: <a href="http://docs.jquery.com/">http://docs.jquery.com/</a>

### Example

This call to `apex.server.process` calls an on-demand process called `MY_PROCESS` and sets the scalar value `x01` to `test` (which can be accessed from PL/SQL using `apex_application.g_x01`) and sets the page item's `P1_DEPTNO` and `P1_EMPNO` values in session state (using jQuery selector syntax). The success callback is stubbed out so that developers can add their own code that fires when the call successfully returns.

Note: The `pData` parameter to the success callback contains any response sent from the call.

```
apex.server.process ( "MY_PROCESS", {
  x01: "test",
  pageItems: "#P1_DEPTNO,#P1_EMPNO"
}, {
  success: function( pData ) { ... do something here ... }
} );
```

## apex.server.url( pData ) (pPage)

This function returns the URL to issue a GET request to the current page.

### Parameters

**Table 28–35** *apex.server.url( pData, pPage) Parameters*

Name	Type	Optional/Required	Default	Description
pData	{Object}	Optional		Object which can optionally be used to set additional values which are included into the URL. The special attribute <code>pageItems</code> which can be of type jQuery selector, jQuery or DOM object or array of item names identifies the page items which are included in the URL. You can also set additional parameters that the <code>wwv_flow.show</code> procedure provides (for example you can set the scalar parameters <code>x01 - x10</code> and the arrays <code>f01 - f20</code> ).
pPage	{String}	Optional		Page ID of the current page, which can be optionally used to set the page ID in the URL being returned.

### Return Value

**Table 28–36** *Return Value*

Type	Description
(String)	The URL to issue the GET request.

### Example

This call to `apex.server.url` returns a URL to issue a GET request to the DELETE function which has been defined for this page, where the URL sets the scalar value `x01` to test (which can be accessed from PL/SQL using `apex_application.g_x01`) and will also set the page item's `P1_DEPTNO` and `P1_EMPNO` values in session state (using jQuery selector syntax).

```
apex.server.url ( {
  p_request: "DELETE",
  x01: "test",
  pageItems: "#P1_DEPTNO,#P1_EMPNO" } );
```

## apex.storage namespace

Use the `apex.storage` namespace to store storage related functions of Oracle Application Express.

- `apex.storage.getCookie(pName)`
- `apex.storage.setCookie(pName,pValue)`

## **apex.storage.getCookie(pName)**

Returns the value of cookie name (pName).

### **Return Value**

Not applicable.

### **Parameters**

pName (String)

## **apex.storage.setCookie(pName,pValue)**

Sets a cookie (pName) to a specified value (pValue).

### **Return Value**

Not applicable.

### **Parameters**

pName (String)

pValue (String)

## apex.util namespace

apex.util namespace contains general utility functions of Oracle Application Express.

- [apex.util.showSpinner](#)
- [apex.util.delayLinger](#)

## apex.util.showSpinner

Function that renders a spinning alert to show the user processing is taking place. Note that the alert is defined as an ARIA alert so that assistive technologies such as screen readers are alerted to the processing status.

### Parameters

**Table 28–37** *util.showSpinner Parameters*

Name	Type	Optional/Required	Default	Description
pContainer	{Object}	Optional	\$( "body" )	Optional jQuery selector, jQuery, or DOM object identifying the container within which you want to center the spinner. If not passed, the spinner will be centered on the whole page.
pOptions	{Object}	Optional	Processing	Optional object with the following options: - "alert" Alert text visually hidden, but available to assistive technologies. Defaults to Processing.

### Return Value

**Table 28–38** *Return Value*

Type	Description
{Object}	jQuery object for the spinner.

### Example

To show the spinner:

```
var lSpinner$ = apex.util.showSpinner( $( "#container_id" ) );
```

To remove the spinner:

```
lSpinner$.remove();
```

## apex.util.delayLinger

The `delayLinger` singleton solves the problem of flashing progress indicators (such as spinners). For processes such as an ajax request (and subsequent user interface update) that may take a while it is important to let the user know that something is happening. The problem is that if an async process is quick there is no need for a progress indicator. The user experiences the user interface update as instantaneous. Showing and hiding a progress indicator around an async process that lasts a very short time causes a flash of content that the user may not have time to fully perceive. At best this can be a distraction and at worse the user wonders if something is wrong or if they missed something important. Simply delaying the progress indicator doesn't solve the problem because the process could finish a short time after the indicator is shown. The indicator must be shown for at least a short but perceivable amount of time even if the request is already finished.

You can use this object to help manage the duration of a progress indication such `apex.util.delayLinger` or with any other progress implementation. Many of the Oracle Application Express built-in progress indicators such as in the `apex.server` namespace functions already use `delayLinger` internally so you only need this API for your own custom long running asynchronous processing.

### apex.util.delayLinger.start

Call this function when a potentially long running async process starts. For each call to start with `start` with a given `pScopeName`, you must make a corresponding call to finish with the same `pScopeName`. Calls with different `pScopeName` arguments will not interfere with each other.

Multiple calls to start for the same `pScopeName` before any calls to finish is allowed but only the `pAction` from the first call is called at most once.

#### Parameters

**Table 28–39** *apex.util.delayLinger.start Function Parameters*

Name	Type	Optional/Required	Default	Description
<code>pScopeName</code>	String			Unique name for each unique progress indicator.
<code>pAction</code>	Function			Function to call to display the progress indicator.

### apex.util.delayLinger.finish

Call this function when a potentially long running async process finishes. For each call to start with a given `pScopeName`, you must make a corresponding call to finish with the same `pScopeName`. The `pAction` is called exactly once if and only if the corresponding start `pAction` was called. If there are multiple calls to finish, the `pAction` from the last one is called.

#### Parameters

**Table 28–40** *apex.util.delayLinger.finish Function Parameters*

Name	Type	Optional/Required	Default	Description
<code>pScopeName</code>	String			Unique name for each unique progress indicator
<code>pAction</code>	Function			function to call to display the progress indicator



**Example**

```
var lSpinner$, lPromise;
lPromise = doLongProcess();
apex.util.delayLinger.start( "main", function() {
    lSpinner$ = apex.util.showSpinnter( $( "#container_id" ) );
} );
lPromise.always( function() {
    apex.util.delayLinger.finish( "main", function() {
        lSpinner$.remove();
    } );
} );
```

## apex.widget namespace

The `apex.widget` namespace stores all the general purpose widget related functions of Oracle Application Express.

- [apex.widget.initPageItem\( pName, pOptions\)](#)

## apex.widget.initPageItem( pName, pOptions)

Given the Application Express page item name or the DOM node, different callbacks and properties can be registered for a page item. This is necessary to seamlessly integrate a plug-in item type with the built-in page item related client-side functionality of Application Express.

For more information about implementing plug-ins, see "Implementing Plug-ins" in *Oracle Application Express Application Builder User's Guide*:

### Return Values

None.

### Parameters

**Table 28–41 Parameters for apex.widget.initPageItem( pName, pOptions )**

Name	Type	Optional/Required	Default	Description
pName	(DOM Node   String)	Required		Application Express page item name or DOM node.
pOptions	(Object)	Required (individual properties are optional)		<p>Supports many properties to specify callbacks and certain item-specific values. Specifying any of these properties will override the default behavior of Application Express for that particular property.</p> <p>See <a href="#">Table 28–42, "Properties for the pOptions parameter"</a> for pOption property details.</p> <p>pOptions can contain one of the following properties:</p> <ul style="list-style-type: none"> <li>▪ <code>getValue()</code></li> <li>▪ <code>setValue( pValue, pDisplayValue )</code></li> <li>▪ <code>enable()</code></li> <li>▪ <code>disable()</code></li> <li>▪ <code>show()</code></li> <li>▪ <code>hide()</code></li> <li>▪ <code>addValue()</code></li> <li>▪ <code>nullValue()</code></li> <li>▪ <code>setFocusTo</code></li> <li>▪ <code>setStyleTo</code></li> <li>▪ <code>afterModify()</code></li> <li>▪ <code>loadingIndicator( pLoadingIndicator\$ )</code></li> </ul>

**Table 28–42 Properties for the pOptions parameter**

Name	Description
getValue()	<p>Specify a function for getting the item's value, which overrides the default page item handling. Ensuring the item returns its value correctly means certain item related client-side functionality of Application Express still works, for example in Dynamic Actions to evaluate a When condition on the item, or when calling the JavaScript function \$v to get the item's value.</p> <p>See "<a href="#">apex.item( pNd ).getValue()</a>" on page 28-29, for details on how to define this function.</p> <p>Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check by calling <code>apex.item( pNd ).getValue()</code>; to see if that returns the item value correctly.</p>
setValue( pValue, pDisplayValue )	<p>Specify a function for setting the item's value, which overrides the default page item handling. Ensuring the item can set its value correctly means certain item related client-side functionality of Application Express still works, for example when using the Set Value action of a Dynamic Action to set the item's value, or when calling the JavaScript function \$s to set the item's value.</p> <p>Note: Even if this function is defined, the default handling always handles the logic associated with the <code>.afterModify()</code> function and the <code>pSuppressChangeEvent</code> parameter, so that is outside the scope of what a plug-in developer is concerned with.</p> <p>See the "<a href="#">apex.item( pNd ).setValue(pValue, pDisplayValue, pSuppressChangeEvent)</a>" on page 28-34, for details on how to define this function.</p> <p>Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check by calling <code>apex.item( pNd ).setValue( pValue )</code>; to see if that sets the item value correctly.</p>
enable()	<p>Specify a function for enabling the item, which overrides the default page item handling. This could be useful for example where the item consists of compound elements which also need enabling, or if the item is based on a widget that already has its own enable method that you want to reuse. Ensuring the item can enable correctly means certain item related client-side functionality of Application Express still works, for example when using the Enable action of a Dynamic Actions, to enable the item.</p> <p>Note: Even if this function is defined, the default handling always handles the logic associated with the <code>.afterModify()</code> function, so that is outside the scope of what a plug-in developer is concerned with.</p> <p>See the "<a href="#">apex.item( pNd ).enable()</a>" on page 28-28, for details on how to define this function.</p> <p>Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check by calling <code>apex.item( pNd ).enable()</code>; to see if that enables the item satisfactorily.</p>

Table 28–42 (Cont.) Properties for the `pOptions` parameter

Name	Description
<code>disable()</code>	<p>Specify a function for disabling the item, which overrides the default page item handling. This could be useful for example where the item consists of compound elements which also need disabling, or if the item is based on a widget that already has its own disable method that you want to reuse. Ensuring the item can disable correctly means certain item related client-side functionality of Application Express still works, for example when using the Disable action of a Dynamic Action to disable the item.</p> <p>Note: Even if this function is defined, the default handling always handles the logic associated with the <code>.afterModify()</code> function, so that is outside the scope of what a plug-in developer is concerned with.</p> <p>See the "<a href="#">apex.item( pNd ).disable()</a>" on page 28-27, for details on how to define this function.</p> <p>Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check by calling <code>apex.item( pNd ).disable()</code>; to see if that disables the item satisfactorily.</p>
<code>show()</code>	<p>Specify a function for showing the item, which overrides the default page item handling. This is useful for example where the item consists of compound elements which also need showing, or if the item is based on a widget that already has its own show method that you want to reuse. Ensuring the item can show correctly means certain item related client-side functionality of Application Express still works, for example when using the Show action of a Dynamic Action, to show the item.</p> <p>See the "<a href="#">apex.item( pNd ).show( pShowRow )</a>" on page 28-36, for details on how to define this function.</p> <p>Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check by calling <code>apex.item( pNd ).show()</code>; to see if that shows the item satisfactorily.</p>
<code>hide()</code>	<p>Specify a function for hiding the item, which overrides the default page item handling. This could be useful for example where the item consists of compound elements which also needs hiding, or if the item is based on a widget that already has its own hide method that you want to reuse. Ensuring the item can hide correctly means certain item related client-side functionality of Application Express still works, for example when using the Hide action of a Dynamic Action, to hide the item.</p> <p>See the "<a href="#">apex.item( pNd ).hide( pHideRow )</a>" on page 28-30, for details on how this function should be defined.</p> <p>Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check by calling <code>apex.item( pNd ).hide()</code>; to see if that hides the item satisfactorily.</p>
<code>addValue()</code>	<p>Specify a function for adding a value to the item, where the item supports multiple values. Currently there is no client-side functionality of Application Express dependent on this. There is also no default page item handling.</p> <p>Note: Even if this function is defined, the default handling always handles the logic associated with the <code>.afterModify()</code> function, so that is outside the scope of what a plug-in developer is concerned with.</p> <p>See the "<a href="#">apex.item( pNd ).addValue( pValue )</a>" on page 28-26, for details on how this function should be defined.</p>

**Table 28–42 (Cont.) Properties for the pOptions parameter**

Name	Description
nullValue	<p>Specify a value that to be used to determine if the item is null. This is used when the item supports definition of a List of Values, where a developer can define a Null Return Value for the item and where the default item handling needs to know this in order to assert if the item is null or empty. This can be done by following these steps:</p> <ol style="list-style-type: none"> <li data-bbox="581 411 1338 898"> <p>1. From the Render function in the plug-in definition, emit the value stored in <code>p_item.lov_null_value</code> as part of the item initialization JavaScript code that fires when the page loads. For example:</p> <pre data-bbox="630 506 1338 898"> /* Assumes that you have some JavaScript function called 'com_your_company_your_item' that accepts 2 parameters, the first being the name of the item and the second being an object storing properties (say pOptions) required by the item's client side code. */ apex_javascript.add_onload_code (     p_code =&gt; 'com_your_company_your_item'                   apex_javascript.add_value(                     apex_plugin_util.page_item_names_to_ jquery(p_item.name)  ', {'                       apex_javascript.add_ attribute('lovNullValue', p_item.lov_null_value, false, false)                   '});' ); </pre> </li> <li data-bbox="581 919 1338 1283"> <p>2. Then, in the implementation of <code>com_your_company_your_item( pName, pOptions )</code> you have the value defined for the specific item's Null Return Value in the <code>pOptions.lovNullValue</code> property. This can then be used in your call to <code>apex.widget.initPageItem</code>, to set the <code>nullValue</code> property.</p> <p>Ensuring the <code>nullValue</code> property is set means certain item related client-side functionality of Application Express still works, for example, in Dynamic Actions to correctly evaluate an <code>is null</code> or <code>is not null</code> when condition on the item, or when calling the JavaScript function <code>apex.item( pNd ).isEmpty()</code> to determine if the item is null.</p> <p>See the "<a href="#">apex.item( pNd ).isEmpty()</a>" on page 28-31, for further details of this API.</p> </li> </ol>

**Table 28–42 (Cont.) Properties for the pOptions parameter**

Name	Description
setFocusTo	<p>Specify the element to receive focus, when focus is set to the item using the <code>apex.item( pNd ).setFocus()</code> API. This can be defined as either a jQuery selector, jQuery or DOM object which identifies the DOM element, or a function that returns a jQuery object referencing the element. This can be useful when the item consists of compound elements, and you do not want focus to go to the element that has an ID matching the item name, which is the default behavior. For example, the native item type <code>Popup LOV</code> when the attribute <code>Input Field</code> is set to <code>Not enterable</code>, <code>Show Display Value</code> and <code>Store Return Value</code> renders a disabled input field as the main element with an ID matching the item name and a popup selection icon next to the input. In this case, because you do not want focus to go to the disabled input, use the <code>setFocusTo</code> item property and set that to the popup selection icon.</p> <p>Ensuring the item sets focus correctly means certain item related client-side functionality of Application Express still works, for example when using the <code>Set Focus</code> action of a Dynamic Action to set focus to the item, when users follow the <code>Go to Error</code> link that displays in a validation error message to go straight to the associated item, or when the item is the first item on a page and the developer has the page level attribute <code>Cursor Focus</code> set to <code>First item on page</code>.</p> <p>See the "<a href="#">apex.item( pNd ).setFocus()</a>" on page 28-32, for further details of this API.</p> <p>Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check this by adding the item as the first item on a page, where the page has the page attribute <code>Cursor Focus</code> set to <code>First item on page</code>, and then running the page. The item receives focus.</p>

**Table 28–42 (Cont.) Properties for the pOptions parameter**

Name	Description
setStyleTo	<p>Specify the element to receive style, when style is set to the item using the <code>apex.item( pNd ).setStyle()</code> API. This can be defined as either a jQuery selector, jQuery- or DOM object which identifies the DOM element(s), or a function that returns a jQuery object referencing the element(s). This is useful when the item consists of compound elements, and you do not want style to be set to the element or just the element, that has an ID matching the item name which is the default behavior. Ensuring the item sets style correctly means certain item related client-side functionality of Application Express still works, for example when using the <code>Set Style</code> action of a Dynamic Action to add style to the item.</p> <p>Note: Even if this property is defined, the default handling still always handles the logic associated with the <code>.afterModify()</code> function, so that is outside the scope of what a plug-in developer is concerned with.</p> <p>See the <code>apex.item( pNd ).setStyle()</code> documentation, for further details of this API.</p> <p>Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check by calling <code>apex.item( pNd ).setStyle( pPropertyName, pPropertyValue )</code>; to see if the item correctly sets the style.</p>
afterModify()	<p>Specify a function that is called after an item is modified. This is useful, for example as some frameworks such as jQuery Mobile need to be notified if widgets are modified, for example their value has been set, or they have been disabled in order to keep both the native and enhanced controls in sync. This callback provides the hook to do so.</p>
loadingIndicator( pLoadingIndicator\$ )	<p>Specify a function that normalizes how the item's loading indicator is displayed during a partial page refresh of the item. This function must pass the <code>pLoadingIndicator\$</code> parameter as the first parameter, which contains a jQuery object with a reference to the DOM element for the loading indicator. The function then adds this loading indicator to the appropriate DOM element on the page for the item, and also returns the jQuery object reference to the loading indicator, such that the framework has a reference to it, so it can remove it once the call is complete.</p> <p>This is used, for example, if the item is a Cascading LOV and the Cascading LOV Parent Item changes, or when setting the item's value by using one of the server-side Dynamic Actions such as <code>Set Value - SQL Statement</code>.</p>

### Examples

The following example shows a call to `apex.widget.initPageItem` with all the available callbacks and properties passed.

```
apex.widget.initPageItem( "P100_COMPANY_NAME", {
    getValue: function() {
        var lValue;
        // code to determine lValue based on the item type.
        return lValue;
    },
    setValue: function( pValue, pDisplayValue ) {
        // code that sets pValue and pDisplayValue (if required), for the item
type
    },
    enable: function() {
        // code that enables the item type
    },
    disable: function() {
```



```
        // code that disables the item type
    },
    show:      function() {
        // code that shows the item type
    },
    hide:      function() {
        // code that hides the item type
    },
    addValue:  function( pValue ) {
        // code that adds pValue to the values already in the item type
    },
    nullValue: "<null return value for the item>",
    setFocusTo: $( "<some jQuery selector>" ),
    setStyleTo: $( "<some jQuery selector>" ),
    afterModify: function(){
        // code to always fire after the item has been modified (value set,
        // enabled, etc.)
    },
    loadingIndicator: function( pLoadingIndicator$ ){
        // code to add the loading indicator in the best place for the item
        return pLoadingIndicator$;
    }
});
```

## Non-namespace Javascript APIs

This section contains all the miscellaneous, non-namespace APIs of Oracle Application Express, including shortcuts to highly used functions.

- `$x(pNd)`
- `$v(pNd)`
- `$v2(pNd)`
- `$s(pNd, pValue, pDisplayValue, pSuppressChangeEvent)`
- `$u_Carray(pNd)`
- `$u_Narray(pNd)`
- `$nvl(pTest, pDefault)`
- `$x_Style(pNd, pStyle, pString)`
- `$x_Hide(pNd)`
- `$x_Show(pNd)`
- `$x_Toggle(pNd)`
- `$x_Remove(pNd)`
- `$x_Value(pNd,pValue)`
- `$x_UpTill(pNd, pToTag)`
- `$x_ItemRow(pNd,pFunc)`
- `$x_HideItemRow(pNd)`
- `$x_ShowItemRow(pNd)`
- `$x_ToggleItemRow(pNd)`
- `$x_HideAllExcept(pNd,pNdArray)`
- `$x_HideSiblings(pNd)`
- `$x_ShowSiblings(pNd)`
- `$x_Class(pNd,pClass)`
- `$x_SetSiblingsClass(pNd, pClass, pNdClass)`
- `$x_ByClass(pClass, pNd, pTag)`
- `$x_ShowAllByClass(pNd, pClass, pTag)`
- `$x_ShowChildren(pNd)`
- `$x_HideChildren(pNd)`
- `$x_disableItem(pNd, pTest)`
- `$f_get_emptyys(pNd, pClassFail, pClass)`
- `$v_Array(pNd)`
- `$f_ReturnChecked(pNd)`
- `$d_ClearAndHide(pNd)`
- `$f_SelectedOptions(pNd)`

- `$f_SelectValue(pNd)`
- `$u_ArrayToString(pArray, pDelim)`
- `$x_CheckImageSrc(pId, pSearch)`
- `$v_CheckValueAgainst(pThis, pValue)`
- `$f_Hide_On_Value_Item(pThis, pThat, pValue)`
- `$f_Show_On_Value_Item(pThis, pThat, pValue)`
- `$f_Hide_On_Value_Item_Row(pThis, pThat, pValue)`
- `$f_Show_On_Value_Item_Row(pThis, pThat, pValue)`
- `$f_DisableOnValue(pThis, pValue, pThat)`
- `$x_ClassByClass(pNd, pClass, pTag, pClass2)`
- `$f_ValuesToArray(pThis, pClass, pTag)`
- `$x_FormItems(pNd, pType)`
- `$f_CheckAll(pThis, pCheck, pArray)`
- `$f_CheckFirstColumn(pNd)`
- `$x_ToggleWithImage(pThis, pNd)`
- `$x_SwitchImageSrc(pNd, pSearch, pReplace)`
- `$x_CheckImageSrc(pNd, pSearch)`
- `$u_SubString(pText, pMatch)`
- `html_RemoveAllChildren(pNd)`
- `html_SetSelectValue(pId, pValue)`
- `addLoadEvent(pFunction)`
- `$f_Swap(pThis, pThat)`
- `$f_SetValueSequence(pArray, pMultiple)`
- `$dom_AddTag(pThis, pTag, pText)`
- `$tr_AddTD(pThis, pText)`
- `$dom_AddInput(pThis, pType, pId, pName, pValue)`
- `$dom_MakeParent(p_Node, p_Parent)`
- `$x_RowHighlight(pThis, pColor)`
- `$x_RowHighlightOff(pThis)`
- `$v_Upper(pNd)`
- `$d_Find(pThis, pString, pTags, pClass)`
- `$f_First_field(pNd)`

## **\$x(pNd)**

Given a DOM node or string ID (pNd), this function returns a DOM node if the element is on the page, or returns FALSE if it is not.

### **Return Value**

(DOM Node | false)

### **Parameters**

pNd (DOM Node | string ID)

## **\$v(pNd)**

Given a DOM node or string ID (pNd), this function returns the value of an Application Express item in the same format as it would be posted.

### **Parameters**

pNd (DOM Node | string ID)

## \$v2(pNd)

Given a DOM node or string ID (pNd), this function returns the value of an Application Express item as a string or an array. If the page item type can contain multiple values like a shuttle, checkboxes or a multi select list an array is returned, otherwise a string.

### Return Value

(string|array)

### Parameters

pNd (DOM Node | string ID)

## **\$s(pNd, pValue, pDisplayValue, pSuppressChangeEvent)**

Given a DOM node or string ID (`pNd`), this function sets the Application Express item value taking into account the item type. The `pDisplayValue` is optional. If used for a page item of type "Popup LOV" where the attribute "Input Field" = "Not Enterable, Show Display Value and Store Return Value", it sets the "Input Field". The value of `pValue` is stored in the hidden return field. The `pSuppressChangeEvent` parameter is optional. Passing either `FALSE` or not passing this parameter value results in a change event firing for the item being set. Pass `TRUE` to prevent the change event from firing for the item being set.

### **Parameters**

`pNd` (DOM Node | string ID)  
`pValue` (String | Array)  
`pDisplayValue` (String)  
`pSuppressChangeEvent` (Boolean)

## **\$u\_Narray(pNd)**

Given a DOM node or string ID or an array (pNd), this function returns a single value, if an pNd is an array but only has one element the value of that element is returned otherwise the array is returned. Used for creating DOM based functionality that can accept a single or multiple DOM nodes.

### **Return Value**

Array (DOM Node | string ID | Array)

### **Parameters**

Array or first value



## **\$u\_Carray(pNd)**

Given a DOM node or string ID or an array (pNd), this function returns an array. Used for creating DOM based functionality that can accept a single or multiple DOM nodes.

### **Return Value**

pNd (DOM Node | string ID | Array)

### **Parameters**

Array

## **\$nvl(pTest, pDefault)**

If pTest is empty or FALSE return pDefault otherwise return pTest.

### **Return Value**

(string | Array)

### **Parameters**

pTest (String | Array)  
pDefault (String | Array)

## **\$x\_Style(pNd, pStyle, pString)**

Sets a specific style property (`pStyle`) to given value (`pString`) of a DOM node or DOM node Array (`pNd`).

### **Return Value**

(DOM node | DOM Array)

### **Parameters**

`pNd` (DOM node | string ID | DOM node Array )  
`pStyle` (String)  
`pString` (String)

## **\$x\_Hide(pNd)**

Hides a DOM node or array of DOM nodes (pNd). This also takes into consideration which type of Application Express item is being hidden.

### **Return Value**

(DOM node | Array)

### **Parameters**

pNd (DOM node | string ID | DOM node Array )

## **\$x\_Show(pNd)**

Shows a DOM node or array of DOM nodes (pNd). This also takes into consideration which type of Application Express item is being hidden.

### **Return Value**

(DOM node | Array)

### **Parameters**

pNd (DOM node | string ID | DOM node Array )

## **\$x\_Toggle(pNd)**

Toggles a DOM node or array of DOM nodes (pNd).

### **Return Value**

(DOM node | Array)

### **Parameters**

pNd (DOM node | string ID | Array)

## **\$x\_Remove(pNd)**

Removes a DOM node or array of DOM nodes.

### **Return Value**

(DOM Node | Array)

### **Parameters**

pNd (DOM node | string ID | DOM node Array)

## **\$x\_Value(pNd,pValue)**

Sets the value (pValue) of a DOM node or array of DOM nodes (pNd).

### **Return Value**

Not applicable.

### **Parameters**

pNd (DOM node | string ID | DOM node Array)  
pValue (String)



## **\$x\_UpTill(pNd, pToTag)**

Starting from a DOM node (`pNd`), this function cascades up the DOM tree until the tag of node name (`pToTag`) is found. If the optional `pToClass` is present, the ancestor node must have a node name that equals `pToTag` and the class must equal `pToClass`.

### **Return Value**

(DOM Node | false)

### **Parameters**

`pNd` (DOM Node | string ID)

String (`pToTag`)

String (`pToClass` )

## **\$x\_ItemRow(pNd,pFunc)**

Given DOM node or array of DOM nodes, this function (shows, hides, or toggles) the entire row that contains the DOM node or array of DOM nodes. This is most useful when using Page Items. This function only works in table layouts since it explicitly looks for a containing tr element.

### **Return Value**

Not applicable.

### **Parameters**

pNd (DOM Node | string ID | Dom node Array)  
pFunc ['TOGGLE','SHOW','HIDE'] (String )

## **\$x\_HideItemRow(pNd)**

Given a page item name, this function hides the entire row that holds the item. In most cases, this is the item and its label. This function only works in table layouts since it explicitly looks for a containing `tr` element.

### **Return Value**

Not applicable.

### **Parameters**

`pNd` (DOM Node | string ID | DON node Array)

## **\$x\_ShowItemRow(pNd)**

Given a page item name, this function shows the entire row that holds the item. In most cases, this is the item and its label. This function only works in table layouts since it explicitly looks for a containing `tr` element.

### **Return Value**

Not applicable.

### **Parameters**

pNd (DOM node | string ID | DOM note Array)

## **\$x\_ToggleItemRow(pNd)**

Given a page item name (pNd), this function toggles the entire row that holds the item. In most cases, this is the item and its label. This function only works in table layouts since it explicitly looks for a containing `tr` element.

### **Return Value**

Not applicable.

### **Parameters**

pNd (DOM node | string ID | DOM node ray)

## **\$x\_HideAllExcept(pNd,pNdArray)**

Hides all DOM nodes referenced in `pNdArray` and then shows the DOM node referenced by `pNd`. This is most useful when `pNd` is also a node in `pNdArray`.

### **Return Value**

(DOM node | DOM Array)

### **Parameters**

`pNd` (DOM node | string ID | DOM node Array)

`pNdArray` (DOM node | String | Array)

## **\$x\_HideSiblings(pNd)**

Hides all sibling nodes of given pNd.

### **Return Value**

(DOM node)

### **Parameters**

pNd (DOM node | string ID )

## **\$x\_ShowSiblings(pNd)**

Shows all sibling DOM nodes of given DOM nodes (pNd).

### **Return Value**

(DOM node)

### **Parameters**

pNd (DOM node | string ID )



## **\$x\_Class(pNd,pClass)**

Sets a DOM node or array of DOM nodes to a single class name.

### **Return Value**

Not applicable.

### **Parameters**

pNd (DOM node | string ID | DOM node Array)  
pClass (String)

## **\$x\_SetSiblingsClass(pNd, pClass, pNdClass)**

Sets the class (pClass) of all DOM node siblings of a node (pNd). If pNdClass is not null the class of pNd is set to pNdClass.

### **Return Value**

(DOM node | false)

### **Parameters**

pNd (DOM Nnde | string ID)

pClass (String)

pThisClass (String)

## **\$x\_ByClass(pClass, pNd, pTag)**

Returns an array of DOM nodes by a given class name (`pClass`). If the `pNd` parameter is provided, then the returned elements are all children of that DOM node. Including the `pTag` parameter further narrows the list to just return nodes of that tag type.

### **Return Value**

(Array)

### **Parameters**

`pClass` (String)  
`pNd` (DOM node | string ID)  
`pTag` (String)

## **\$x\_ShowAllByClass(pNd, pClass, pTag)**

Show all the DOM node children of a DOM node (pNd) that have a specific class (pClass) and tag (pTag).

### **Return Value**

Not applicable.

### **Parameters**

pNd (DOM node | string ID)

pClass (String)

pTag (String)

## **\$x\_ShowChildren(pNd)**

Show all DOM node children of a DOM node (pNd).

### **Return Value**

Not applicable.

### **Parameters**

pNd (DOM node | string ID)

## **\$x\_HideChildren(pNd)**

Hide all DOM node children of a DOM node (pNd).

### **Return Value**

Not applicable.

### **Parameters**

pNd (DOM node | string ID)

## **\$x\_disableItem(pNd, pTest)**

Disables or enables an item or array of items based on (pTest).

### **Return Value**

Not applicable.

### **Parameters**

pNd (DOM node | string ID | DOM node array)  
a (true | false)

## **\$f\_get\_emptyys(pNd, pClassFail, pClass)**

Checks an item or an array of items to see if any are empty, set the class of all items that are empty to pClassFail, set the class of all items that are not empty to pClass.

### **Return Value**

false, Array Array of all items that are empty (false | Array)

### **Parameters**

pNd (DOM node | string ID | DOM node Array)  
String (pClassFail)  
String (pClass)



## **\$v\_Array(pNd)**

Returns an item value as an array. Useful for multiselects and checkboxes.

### **Return Value**

(Array)

### **Parameters**

pId (DOM Node | string ID)

## **\$f\_ReturnChecked(pNd)**

Returns an item value as an array. Useful for radio items and check boxes.

### **Return Value**

(Array)

### **Parameters**

pId (DOM node | string ID)

## **\$d\_ClearAndHide(pNd)**

Clears the content of an DOM node or array of DOM nodes and hides them.

### **Return Value**

Not applicable.

### **Parameters**

pNd (DOM node | string ID | DOM node array)

## **\$f\_SelectedOptions(pNd)**

Returns the DOM nodes of the selected options of a select item (pNd).

### **Return Value**

(DOM Array)

### **Parameters**

pNd (DOM node | string ID)

## **\$f\_SelectValue(pNd)**

Returns the values of the selected options of a select item (pNd).

### **Return Value**

(DOM Array | String)

### **Parameters**

pNd (DOM node | string ID)

## **\$u\_ArrayToString(pArray, pDelim)**

Given an array (`pArray`) return a string with the values of the array delimited with a given delimiter character (`pDelim`).

### **Return Value**

Not applicable.

### **Parameters**

`pArray` (`pArray`)  
`pDelim` (`String`)

## **\$x\_CheckImageSrc(pId,pSearch)**

Checks an image (pId) source attribute for a substring (pSearch). The function returns TRUE if a substring (pSearch) is found. It returns FALSE if a substring (pSearch) is not found.

### **Return Value**

(true | false)

### **Parameters**

pId (DOM Node | String)  
pSearch (pSearch)

## **\$v\_CheckValueAgainst(pThis, pValue)**

Checks an page item's (pThis) value against a set of values (pValue). This function returns TRUE if any value matches.

### **Return Value**

(true | false)

### **Parameters**

pThis (DOM node | string ID)

pValue (Number | String | Array)



## **\$f\_Hide\_On\_Value\_Item(pThis, pThat, pValue)**

Checks page item's (pThis) value against a value (pValue). If it matches, a DOM node (pThat) is set to hidden. If it does not match, then the DOM node (pThat) is set to visible.

### **Return Value**

(true | false)

### **Parameters**

pThis (DOM node | string ID)

pThat (DOM node | string ID | DOM node Array )

pValue (Number | String | Array)

## **\$f\_Show\_On\_Value\_Item(pThis, pThat, pValue)**

Checks page item's (pThis) value against a value (pValue). If it matches, a DOM node (pThat) is set to visible. If it does not match, then the DOM node (pThat) is set to hidden.

### **Return Value**

(true | false)

### **Parameters**

pThis (DOM node | string ID)  
pThat (DOM node | string ID | DOM node Array )  
pValue (Number | String | Array)

## **\$f\_Hide\_On\_Value\_Item\_Row(pThis, pThat, pValue)**

Checks the value (pValue) of an item (pThis). If it matches, this function hides the table row that holds (pThat). If it does not match, then the table row is shown.

### **Return Value**

(true | false)

### **Parameters**

pThis (DOM node | string ID)

pThat (DOM node | string ID | DOM node Array )

pValue (Number | String | Array)

## **\$f\_Show\_On\_Value\_Item\_Row(pThis, pThat, pValue)**

Checks the value (pValue) of an item (pThis). If it matches, this function shows the table row that holds (pThat). If it does not match, then the table row is hidden.

### **Return Value**

(true | false)

### **Parameters**

pThis (DOM node | string ID)

pThat (DOM node | string ID | DOM node Array )

pValue (Number | String | Array)

## **\$f\_DisableOnValue(pThis, pValue, pThat)**

Checks the value (`pValue`) of an item (`pThis`). If it matches, this function disables the item or array of items (`pThat`). If it does not match, then the item is enabled.

### **Return Value**

(true | false)

### **Parameters**

`pThis` (DOM node | string ID)

`pValue` (String)

`pThat` (DOM node | string ID | DOM node Array )

## **\$x\_ClassByClass(pNd, pClass, pTag, pClass2)**

Sets a class attribute of an array of nodes that are selected by class.

### **Return Value**

(DOM node | DOM node Array)

### **Parameters**

pNd (DOM node | string ID)

pClass (String)

pTag (String)

pClass2 (String)

## **\$f\_ValuesToArray(pThis, pClass, pTag)**

Collects the values of form items contained within DOM node (`pThis`) of class attribute (`pClass`) and nodeName (`pTag`) and returns an array.

### **Return Value**

No applicable.

### **Parameters**

`pThis` (DOM node | string ID)  
`pClass` (String)  
`pTag` (String)

## **\$x\_FormItems(pNd, pType)**

Returns all form input items contained in a DOM node (`pThis`) of a certain type (`pType`).

### **Return Value**

DOM node Array

### **Parameters**

`pNd` (DOM node | string ID)

`pType` (String)



## **\$f\_CheckAll(pThis, pCheck, pArray)**

Check or uncheck (pCheck) all check boxes contained within a DOM node (pThis). If an array of checkboxes DOM nodes (pArray) is provided, use that array for affected check boxes.

### **Return Value**

Not applicable.

### **Parameters**

pThis (DOM node | string ID)

pCheck (true | false)

pArray (DOM node array)

## **\$f\_CheckFirstColumn(pNd)**

This function sets all checkboxes located in the first column of a table based on the checked state of the calling check box (pNd), useful for tabular forms.

### **Return Value**

DOM node Array

### **Parameters**

pNd (DOM node | String)

## **\$x\_ToggleWithImage(pThis,pNd)**

Given an image element (`pThis`) and a DOM node (`pNd`), this function toggles the display of the DOM node (`pNd`). The `src` attribute of the image element (`pThis`) is rewritten. The image `src` has any plus substrings replaced with minus substrings or minus substrings are replaced with plus substrings.

### **Return Value**

(DOM Node)

### **Parameters**

`pThis` (DOM Node | string ID)

`pNd` (DOM Nnde | string id | DOM node Array)

## **\$x\_SwitchImageSrc(pNd, pSearch, pReplace)**

Checks an image (pId) src attribute for a substring (pSearch). If a substring is found, this function replaces the image entire src attribute with (pReplace).

### **Return Value**

(DOM node | false)

### **Parameters**

pNd (DOM node | string ID)

pSearch (String)

pReplace (String)

## **\$x\_CheckImageSrc(pNd, pSearch)**

Checks an image (pNd) source attribute for a substring (pSearch). The function returns TRUE if a substring (pSearch) is found. It returns FALSE if a substring (pSearch) is not found.

### **Return Value**

(true | fales)

### **Parameters**

pNd (DOM node | string ID)  
pSearch (String)

## **`$u_SubString(pText,pMatch)`**

Returns a TRUE or FALSE if a string (`pText`) contains a substring (`pMatch`).

### **Return Value**

(true | false)

### **Parameters**

`pText` (String)

`pMatch` (String)

## html\_RemoveAllChildren(pNd)

Use DOM methods to remove all DOM children of DOM node (pND).

### Return Value

Not applicable.

### Parameters

pNd (DOM node | string ID)

## html\_SetSelectValue(pId,pValue)

Sets the value (pValue) of a select item (pId). If the value is not found, this functions selects the first option (usually the NULL selection).

### Return Value

Not applicable.

### Parameters

pId (DOM node | String)  
pValue (String)



## **addLoadEvent(pFunction)**

Adds an onload function (func) without overwriting any previously specified onload functions.

### **Return Value**

Not applicable.

### **Parameters**

pFunction (Javascript Function)

## **\$f\_Swap(pThis,pThat)**

Swaps the form values of two form elements (pThis,pThat).

### **Return Value**

Not applicable.

### **Parameters**

pThis (DOM Node | String)

pThat (DOM Node | String)

## **\$f\_SetValueSequence(pArray,pMultiple)**

Sets array of form item (`pArray`) to sequential number in multiples of (`pMultiple`).

### **Return Value**

Not applicable.

### **Parameters**

`pArray` (Array)

`pMultiple` (Number)

## **\$dom\_AddTag(pThis, pTag, pText)**

Inserts the html element (`pTag`) as a child node of a DOM node (`pThis`) with the `innerHTML` set to (`pText`).

### **Return Value**

DOM node

### **Parameters**

`pThis` (DOM node | string ID )

`pTag` (String)

`pText` (String)

## **\$tr\_AddTD(pThis,pText)**

Appends a table cell to a table row (`pThis`). And sets the content to (`pText`).

### **Return Value**

(DOM node)

### **Parameters**

`pThis` (DOM node | string ID)  
`pText` (String)

## **\$tr\_AddTH(pThis,pText)**

Appends a table cell to a table row (`pThis`). And sets the content to (`pText`).

### **Return Value**

DOM node

### **Parameters**

`pThis` (DOM node | string ID)

`pText` (String)

## **\$dom\_AddInput(pThis,pType,pId,pName,pValue)**

Inserts the html form input element (`pType`) as a child node of a DOM node (`pThis`) with an id (`pId`) and name (`pName`) value set to `pValue`.

### **Return Value**

(DOM node)

### **Parameters**

`pThis` (DOM node | string ID)

`pType` (String)

`pId` (String)

`pName` (String)

`pValue` (String)

## **\$dom\_MakeParent(p\_Node,p\_Parent)**

Takes a DOM node (`p_Node`) and makes it a child of DOM node (`p_Parent`) and then returns the DOM node (`pNode`).

### **Return Value**

(DOM node)

### **Parameters**

`p_This` (DOM node | string ID)

`p_Parent` (DOM node | string ID)



## **\$x\_RowHighlight(pThis, pColor)**

Give an table row DOM element (`pThis`), this function sets the background of all table cells to a color (`pColor`). A global variable `gCurrentRow` is set to `pThis`.

### **Return Value**

Not applicable.

### **Parameters**

`pThis` (DOM node | String)  
`pColor`(String)

## **\$x\_RowHighlightOff(pThis)**

Give an table row Dom node (`pThis`), this function sets the background of all table cells to NULL.

### **Return Value**

Not applicable.

### **Parameters**

`pThis` (DOM Element | String)

## **\$v\_Upper(pNd)**

Sets the value of a form item (pNd) to uppercase.

### **Return Value**

Not applicable.

### **Parameters**

pNd (DOM Node | String)

## **\$d\_Find(pThis,pString,pTags,pClass)**

Hides child nodes of a Dom node (`pThis`) where the child node's inner HTML matches any instance of `pString`. To narrow the child nodes searched by specifying a tag name (`pTag`) or a class name (`pClass`). Note that the child node is set to a block level element when set to visible.

### **Return Value**

Not applicable.

### **Parameters**

`pThis` (DOM node | String)  
`pString` (String)  
`pTags` (String)  
`pClass` (String)

## **\$f\_First\_field(pNd)**

Places the user focus on a form item (pNd). If pNd is not found then this function places focus on the first found user editable field.

### **Return Value**

true (if successful)

### **Parameters**

pNd

## Legacy JavaScript APIs

Work has commenced to reduce the overall size of JavaScript that is loaded by Oracle Application Express when rendering a page. JavaScript functions that are no longer served on every page are gradually being moved to a legacy JavaScript file, which can be found in `/i/libraries/apex/legacy.js`.

When developing applications, a developer has the option to either include, or not include the legacy JavaScript functions. This is achieved by using the Include Legacy JavaScript property on the User Interface Attributes page under the application's Shared Components.

Existing applications are migrated with this option enabled, for backward compatibility. To not include this legacy file, you need to go through the functions listed in the legacy file, and search your application and associated JavaScript files for any references to those files. If you are happy that there are no references to these functions, you can switch off including the legacy file and benefit from the slightly smaller library.

When developing new applications, the legacy file is included by default in all applications that use a Desktop User Interface Type. New applications that use a jQuery Mobile Smartphone User Interface Type do not include this file.

For both new and existing application development, Oracle recommends that you do not continue to use any of the functions in `legacy.js`, to reduce your dependency on this legacy JavaScript.

- [\\$\\_PopupReturn\(pValue, pThat\)](#) [Deprecated]
- [\\$\\_IsEmpty\(pThis\)](#) [Deprecated]
- [submitEnter\(pNd,e\)](#) [Deprecated]
- [setReturn\(p\\_R,p\\_D\)](#) [Deprecated]
- [GetCookie \(pName\)](#) [Deprecated]
- [SetCookie \(pName,pValue\)](#) [Deprecated]

## **\$v\_PopupReturn(pValue, pThat) [Deprecated]**

Sets the value of the item in the parent window (`pThat`), with (`pValue`) and then closes the popup window.

---

---

**Note:** This function is deprecated. Instead, use:

[apex.navigation.popup\(pOptions\)](#)

For existing applications, the old function is still available, because of the application including the 'Legacy JavaScript' file (`legacy.js`). For details on how to control the inclusion of this file, see "About Database Applications" in *Oracle Application Express Application Builder User's Guide*.

---

---

### **Return Value**

Not applicable.

### **Parameters**

`pValue` (string)  
`pThat` (DOM node | string ID)

## **\$v\_IsEmpty(pThis) [Deprecated]**

Returns TRUE or FALSE if a form element is empty, this considers any whitespace including a space, a tab, a form-feed, as empty. This also considers any null value that has been specified on the item.

---

---

**Note:** This function is deprecated. Instead, use:

[apex.item\( pNd \).isEmpty\(\)](#)

For existing applications, the old function is still available, because of the application including the 'Legacy JavaScript' file (legacy.js). For details on how to control the inclusion of this file, see "About Database Applications" in *Oracle Application Express Application Builder User's Guide*.

---

---

### **Return Value**

[true | false]

### **Parameters**

pThis (DOM Node | String)



## submitEnter(pNd,e) [Deprecated]

Submits a page when ENTER is pressed in a text field, setting the request value to the ID of a DOM node (pNd).

Usage is `onkeypress="submitEnter(this,event) "`

---

---

**Note:** This function is deprecated. Instead, use:

```
apex.submit( { submitIfEnter : event } )
```

See `apex.submit` for further details on how to use the 'submitIfEnter' `pOptions` property.

For existing applications, the old function is still available, because of the application including the 'Legacy JavaScript' file (`legacy.js`). For details on how to control the inclusion of this file, see "About Database Applications" in *Oracle Application Express Application Builder User's Guide*.

---

---

### Return Value

Not applicable.

### Parameters

pNd (DOM node | String | Array)

## setReturn(p\_R,p\_D) [Deprecated]

Sets DOM items in the global variables returnInput (p\_R) and returnDisplay (p\_D) for use in populating items from popups.

---

---

**Note:** This function is deprecated and due to very limited value there is no alternative.

For existing applications, the old function is still available, because of the application including the 'Legacy JavaScript' file (legacy.js). For details on how to control the inclusion of this file, see "About Database Applications" in *Oracle Application Express Application Builder User's Guide*.

---

---

### Return Value

Not applicable.

### Parameters

p\_R  
p\_D

## GetCookie (pName) [Deprecated]

Returns the value of cookie name (pName).

---

---

**Note:** This function is deprecated. Instead, use:

[apex.storage.getCookie\(pName\)](#)

For existing applications, the old function is still available, because of the application including the 'Legacy JavaScript' file (legacy.js). For details on how to control the inclusion of this file, see "About Database Applications" in *Oracle Application Express Application Builder User's Guide*.

---

---

### Return Value

Not applicable.

### Parameters

pName (String)

## SetCookie (pName,pValue) [Deprecated]

Sets a cookie (pName) to a specified value (pValue).

---

---

**Note:** This function is deprecated. Instead, use:

[apex.storage.setCookie\(pName,pValue\)](#)

For existing applications, the old function is still available, because of the application including the 'Legacy JavaScript' file (legacy.js). For details on how to control the inclusion of this file, see "About Database Applications" in *Oracle Application Express Application Builder User's Guide*.

---

---

### Return Value

Not applicable.

### Parameters

pName (String)  
pValue (String)

## A

- APEX\_APPLICATION
  - global variables, 1-1
  - HELP Procedure, 1-6
  - package, 1-1
  - Referencing Arrays, 1-3
  - STOP\_APEX\_ENGINE Procedure, 1-8
- APEX\_APPLICATION\_INSTALL, 2-1
  - CLEAR\_ALL procedure, 2-7
    - example, 2-4
    - examples, 2-4, 2-5
  - GENERATE\_APPLICATION\_ID procedure, 2-8
  - GENERATE\_OFFSET procedure, 2-9
  - GET\_APPLICATION\_ALIAS function, 2-10
  - GET\_APPLICATION\_ID function, 2-11
  - GET\_APPLICATION\_NAME function, 2-12
  - GET\_IMAGE\_PREFIX function, 2-14
  - GET\_OFFSET function, 2-15
  - GET\_PROXY function, 2-16
  - GET\_SCHEMA function, 2-17
  - GET\_WORKSPACE\_ID function, 2-18
  - import application into different workspaces, 2-4
  - import application with generated app ID, 2-4
  - import application with specified app ID, 2-4
  - import application without modification, 2-4
  - import into training instance, 2-5
  - import script examples, 2-4
  - package overview, 2-2
  - SET\_APPLICATION\_ALIAS procedure, 2-19
  - SET\_APPLICATION\_ID procedure, 2-20
  - SET\_APPLICATION\_NAME procedure, 2-21
  - SET\_IMAGE\_PREFIX procedure, 2-23
  - SET\_OFFSET procedure, 2-24
  - SET\_PROXY procedure, 2-25
  - SET\_SCHEMA procedure, 2-26
  - SET\_WORKSPACE\_ID procedure, 2-27
- APEX\_APPLICATION\_INSTALL.GET\_AUTO\_INSTALL\_SUP\_OBJ, 2-13
- APEX\_APPLICATION\_INSTALL.SET\_AUTO\_INSTALL\_SUP\_OBJ, 2-22
- APEX\_AUTHENTICATION, 3-1
  - CALLBACK Procedure, 3-3
  - GET\_CALLBACK\_URL Procedure, 3-5
  - GET\_LOGIN\_USERNAME\_COOKIE\_LOGIN Function, 3-6
  - IS\_AUTHENTICATED Function, 3-7
  - IS\_PUBLIC\_USER Function, 3-8
  - LOGIN Procedure, 3-9
  - LOGOUT Procedure, 3-10
  - POST\_LOGIN Procedure, 3-11
  - SEND\_LOGIN\_USERNAME\_COOKIE Procedure, 3-12
- APEX\_AUTHORIZATION, 3-xxiii, 4-1
  - ENABLE\_DYNAMIC\_GROUPS procedure, 4-2
  - IS\_AUTHORIZED function, 4-3
  - RESET\_CACHE procedure, 4-4
- APEX\_COLLECTION, 5-3
  - ADD\_MEMBER function, 5-26
  - ADD\_MEMBER procedure, 5-24
  - ADD\_MEMBERS procedure, 5-28
  - COLLECTION\_EXISTS function, 5-30
  - COLLECTION\_HAS\_CHANGED function, 5-31
  - COLLECTION\_MEMBER\_COUNT function, 5-32
  - CREATE\_COLLECTION procedure, 5-33
  - CREATE\_COLLECTION\_FROM\_QUERY, 5-35
  - CREATE\_COLLECTION\_FROM\_QUERY\_B procedure, 5-37
  - CREATE\_COLLECTION\_FROM\_QUERY2 procedure, 5-36
  - CREATE\_COLLECTION\_FROM\_QUERYB2 procedure, 5-39
  - CREATE\_OR\_TRUNCATE\_COLLECTION, 5-22
  - CREATE\_OR\_TRUNCATE\_COLLECTION procedure, 5-34
  - DELETE\_ALL\_COLLECTIONS procedure, 5-41
  - DELETE\_ALL\_COLLECTIONS\_SESSION procedure, 5-42
  - DELETE\_COLLECTION procedure, 5-43
  - DELETE\_MEMBER procedure, 5-44
  - DELETE\_MEMBERS procedure, 5-45
  - GET\_MEMBER\_MD5 function, 5-46
  - MERGE\_MEMBERS procedure, 5-47
  - MOVE\_MEMBER\_DOWN procedure, 5-49
  - MOVE\_MEMBER\_UP procedure, 5-50
  - RESEQUENCE\_COLLECTION procedure, 5-51
  - RESET\_COLLECTION\_CHANGED procedure, 5-52
  - RESET\_COLLECTION\_CHANGED\_ALL procedure, 5-53
  - SORT\_MEMBERS procedure, 5-54

- TRUNCATE\_COLLECTION procedure, 5-55
- UPDATE\_MEMBER procedure, 5-56
- UPDATE\_MEMBER\_ATTRIBUTE procedure
  - signature 1, 5-60
- UPDATE\_MEMBER\_ATTRIBUTE procedure
  - signature 2, 5-62
- UPDATE\_MEMBER\_ATTRIBUTE procedure
  - signature 3, 5-64
- UPDATE\_MEMBER\_ATTRIBUTE procedure
  - signature 4, 5-66
- UPDATE\_MEMBER\_ATTRIBUTE procedure
  - signature 5, 5-68
- UPDATE\_MEMBER\_ATTRIBUTE procedure
  - signature 6, 5-70
- UPDATE\_MEMBERS procedure, 5-58
- APEX\_CSS, 6-1
  - ADD\_3RD\_PARTY\_LIBRARY\_FILE procedure, 6-3
  - ADD\_FILE procedure, 6-4
- APEX\_CUSTOM\_AUTH, 7-1
- APPLICATION\_PAGE\_ITEM\_EXISTS
  - function, 7-2
- CURRENT\_PAGE\_IS\_PUBLIC function, 7-3
- DEFINE\_USER\_SESSION procedure, 7-4
- GET\_COOKIE\_PROPS, 7-5
- GET\_LDAP\_PROPS, 7-6
- GET\_NEXT\_SESSION\_ID function, 7-8
- GET\_SECURITY\_GROUP\_ID function, 7-9
- GET\_SESSION\_ID function, 7-10
- GET\_SESSION\_ID\_FROM\_COOKIE, 7-11
- GET\_USER function, 7-12
- GET\_USERNAME, 7-13
- IS\_SESSION\_VALID, 7-14
- LOGIN
  - Login API, 7-15
- LOGOUT, 7-16
- POST\_LOGIN, 7-17
- SESSION\_ID\_EXISTS function, 7-18
- SET\_SESSION\_ID procedure, 7-19
- SET\_SESSION\_ID\_TO\_NEXT\_VALUE
  - procedure, 7-20
- SET\_USER procedure, 7-21
- APEX\_DEBUG
  - Constants, 8-3
  - DISABLE procedure, 8-4
  - ENABLE procedure, 8-5
  - ENTER procedure, 8-6
  - ERROR procedure, 8-8
  - INFO procedure, 8-9
- APEX\_DEBUG, 8-1
  - LOG\_DBMS\_OUTPUT procedure, 8-10
  - LOG\_LONG\_MESSAGE procedure, 8-11
  - LOG\_MESSAGE procedure, 8-12
  - LOG\_PAGE\_SESSION\_STATE procedure, 8-13
  - MESSAGE procedure, 8-14
  - REMOVE\_DEBUG\_BY\_AGE procedure, 8-16
  - REMOVE\_DEBUG\_BY\_APP procedure, 8-17
  - REMOVE\_DEBUG\_BY\_VIEW procedure, 8-18
  - REMOVE\_SESSION\_MESSAGES procedure, 8-19
  - TOCHAR function, 8-20
  - TRACE procedure, 8-21
  - WARN procedure, 8-22
- APEX\_ERROR, 9-1
  - ADD\_ERROR Procedure Signature 1, 9-6
  - ADD\_ERROR Procedure Signature 2, 9-7
  - ADD\_ERROR Procedure Signature 3, 9-8
  - ADD\_ERROR Procedure Signature 4, 9-10
  - ADD\_ERROR Procedure Signature 5, 9-12
  - AUTO\_SET\_ASSOCIATED\_ITEM
    - Procedure, 9-14
  - example
    - error handling function, 9-4
  - EXTRACT\_CONSTRAINT\_NAME
    - Function, 9-15
  - GET\_ARIA\_ERROR\_ATTRIBUTES
    - Function, 9-16
  - GET\_FIRST\_ORA\_ERROR\_TEXT
    - Procedure, 9-17
  - HTML function, 10-3
  - INT\_ERROR\_RESULT Function, 9-18
- APEX\_ESCAPE, 10-1
  - constants, 10-2
  - HTML\_ATTRIBUTE Function, 10-5
  - HTML\_TRUNC Function, 10-6
  - HTML\_WHITELIST Function, 10-7
  - JS\_LITERAL Function, 10-8
  - JSON Function, 10-9
  - LDAP\_DN Function, 10-10
  - LDAP\_SEARCH\_FILTER Function, 10-11
  - NOOP Function, 10-12
  - REGEXP Function, 10-13
  - SET\_HTML\_ESCAPING\_MODE
    - Procedure, 10-14
- APEX\_INSTANCE\_ADMIN, 11-1
  - ADD\_SCHEMA procedure, 11-8
  - ADD\_WORKSPACE procedure, 11-9
  - CREATE\_SCHEMA\_EXCEPTION
    - Procedure, 11-10
  - FREE\_WORKSPACE\_APP\_IDS procedure, 11-11
  - GET\_PARAMETER function, 11-12
  - GET\_SCHEMAS function, 11-13
    - parameter values, 11-2
  - REMOVE\_APPLICATION procedure, 11-15
  - REMOVE\_SAVED\_REPORT procedure, 11-17
  - REMOVE\_SAVED\_REPORTS procedure, 11-16
  - REMOVE\_SCHEMA procedure, 11-18
  - REMOVE\_SCHEMA\_EXCEPTION
    - Procedure, 11-19
  - REMOVE\_SCHEMA\_EXCEPTIONS
    - Procedure, 11-20
  - REMOVE\_SUBSCRIPTION Procedure, 11-21
  - REMOVE\_WORKSPACE procedure, 11-22
  - REMOVE\_WORKSPACE\_EXCEPTIONS
    - Procedure, 11-23
  - RESERVE\_WORKSPACE\_APP\_IDS
    - procedure, 11-24
  - RESTRICT\_SCHEMA Procedure, 11-26
  - SET\_LOG\_SWITCH\_INTERVAL
    - Procedure, 11-27
  - SET\_PARAMETER procedure, 11-29

- SET\_WORKSPACE\_CONSUMER\_GROUP
  - Procedure, 11-30
- TRUNCATE\_LOG procedure, 11-31
- UNRESTRICT\_SCHEMA Procedure, 11-32
- APEX\_INSTANCE\_ADMIN.GET\_WORKSPACE\_PARAMETER, 11-14
- APEX\_INSTANCE\_ADMIN.SET\_WORKSPACE\_PARAMETER, 11-28
- APEX\_IR, 12-1
  - ADD\_FILTER procedure signature 1, 12-2
  - ADD\_FILTER procedure signature 2, 12-4
  - CHANGE\_REPORT\_OWNER Procedure, 12-7
  - CHANGE\_SUBSCRIPTION\_LANG procedure, 12-9
  - CLEAR\_REPORT procedure signature 1, 12-10
  - CLEAR\_REPORT procedure signature 2, 12-11
  - DELETE\_REPORT procedure, 12-12
  - DELETE\_SUBSCRIPTION procedure, 12-13
  - GET\_LAST\_VIEWED\_REPORT\_ID function, 12-14
  - GET\_REPORT function, 12-15
  - RESET\_REPORT procedure signature 1, 12-16
  - RESET\_REPORT procedure signature 2, 12-17
- APEX\_IR.CHANGE\_SUBSCRIPTION\_EMAIL, 12-8
- APEX\_IR.CHANGE\_SUBSCRIPTION\_EMAIL Procedure, 12-6
- APEX\_ITEM, 13-1
  - CHECKBOX2 function, 13-2
  - DATE\_POPUP function, 13-4, 13-6
  - DISPLAY\_AND\_SAVE, 13-8
  - HIDDEN function, 13-9
  - MD5\_CHECKSUM function, 13-11
  - MD5\_HIDDEN function, 13-12
  - POPUP\_FROM\_LOV function, 13-13
  - POPUP\_FROM\_QUERY function, 13-15
  - POPUPKEY\_FROM\_LOV function, 13-17
  - POPUPKEY\_FROM\_QUERY function, 13-19
  - RADIOGROUP function, 13-21
  - SELECT\_LIST function, 13-22
  - SELECT\_LIST\_FROM\_LOV function, 13-24
  - SELECT\_LIST\_FROM\_LOV\_XL function, 13-25
  - SELECT\_LIST\_FROM\_QUERY function, 13-27
  - SELECT\_LIST\_FROM\_QUERY\_XL function, 13-29
  - TEXT function, 13-31
  - TEXT\_FROM\_LOV function, 13-33
  - TEXT\_FROM\_LOV\_QUERY function, 13-34
  - TEXTAREA function, 13-32
- APEX\_JAVASCRIPT, 14-1
  - ADD\_ATTRIBUTE function signature 1, 14-3
  - ADD\_ATTRIBUTE function signature 2, 14-5
  - ADD\_ATTRIBUTE function signature 3, 14-6
  - ADD\_ATTRIBUTE function signature 4, 14-7
  - ADD\_INLINE\_CODE procedure, 14-8
  - ADD\_LIBRARY procedure, 14-9
  - ADD\_ONLOAD\_CODE procedure, 14-10
  - ADD\_VALUE function signature 1, 14-11
  - ADD\_VALUE function signature 2, 14-12
  - ADD\_VALUE function signature 3, 14-13
  - ADD\_VALUE function signature 4, 14-14
- ESCAPE function, 14-15
- APEX\_JSON, 15-1
  - CLOSE\_ALL procedure, 15-5
  - CLOSE\_ARRAY procedure, 15-6
  - CLOSE\_OBJECT procedure, 15-7
  - constants and datatypes, 15-4
  - DOES\_EXIST function, 15-8
  - FIND\_PATHS\_LIKE function, 15-9
  - FLUSH procedure, 15-11
  - FREE\_OUTPUT procedure, 15-10
  - GET\_BOOLEAN function, 15-12
  - GET\_CLOB\_OUTPUT function, 15-13
  - GET\_COUNT function, 15-14
  - GET\_DATE function, 15-15
  - GET\_MEMBERS function, 15-16
  - GET\_NUMBER function, 15-17
  - GET\_VALUE function, 15-18
  - GET\_VARCHAR2 function, 15-19
  - INITIALIZE\_OUTPUT procedure, 15-20
  - OPEN\_ARRAY procedure, 15-22
  - OPEN\_OBJECT procedure, 15-23
  - package overview, 15-3
  - PARSE Procedure Signature 1, 15-24
  - PARSE Procedure Signature 2, 15-25
  - STRINGIFY Function Signature 1, 15-26
  - STRINGIFY Function Signature 2, 15-27
  - STRINGIFY Function Signature 3, 15-28
  - STRINGIFY Function Signature 4, 15-29
  - TO\_XMLTYPE function, 15-30
  - WRITE Procedure Signature 1, 15-31
  - WRITE Procedure Signature 10, 15-40
  - WRITE Procedure Signature 11, 15-41
  - WRITE Procedure Signature 12, 15-42
  - WRITE Procedure Signature 13, 15-43
  - WRITE Procedure Signature 14, 15-44
  - WRITE Procedure Signature 15, 15-45
  - WRITE Procedure Signature 16, 15-46
  - WRITE Procedure Signature 2, 15-32
  - WRITE Procedure Signature 3, 15-33
  - WRITE Procedure Signature 4, 15-34
  - WRITE Procedure Signature 5, 15-35
  - WRITE Procedure Signature 6, 15-36
  - WRITE Procedure Signature 7, 15-37
  - WRITE Procedure Signature 8, 15-38
  - WRITE Procedure Signature 9, 15-39
- APEX\_JSON.INITIALIZE\_OUTPUT procedure, 15-21
- APEX\_LANG, 16-1
  - CREATE\_LANGUAGE\_MAPPING procedure, 16-2
  - DELETE\_LANGUAGE\_MAPPING procedure, 16-4
  - LANG function, 16-6
  - MESSAGE function, 16-7
  - PUBLISH\_APPLICATION procedure, 16-9
  - SEED\_TRANSLATIONS procedure, 16-10
  - UPDATE\_LANGUAGE\_MAPPING procedure, 16-11
  - UPDATE\_MESSAGE procedure, 16-13
  - UPDATE\_TRANSLATED\_STRING

- procedure, 16-14
- APEX\_LDAP, 17-1
  - AUTHENTICATE, 17-2
  - GET\_ALL\_USER\_ATTRIBUTES, 17-3
  - GET\_USER\_ATTRIBUTES, 17-5
  - IS\_MEMBER, 17-7
  - MEMBER\_OF, 17-9
  - MEMBER\_OF2 Function, 17-10
- APEX\_MAIL, 18-1
  - ADD\_ATTACHMENT procedure, 18-3
  - PUSH\_QUEUE procedure, 18-6
  - SEND procedure, 18-7
- APEX\_MAIL\_QUEUE, sending email in queue, 18-6
- APEX\_PAGE, 19-1
  - GET\_PAGE\_MODE function, 19-8
  - GET\_UI\_TYPE function, 19-6
  - GET\_URL function, 19-10
  - Global Constants, 19-2
  - IS\_DESKTOP\_UI function, 19-3
  - IS\_JQM\_SMARTPHONE\_UI function, 19-4
  - IS\_JQM\_TABLET\_UI function, 19-5
  - IS\_READ\_ONLY function, 19-7
  - PURGE\_CACHE procedure, 19-9
- APEX\_PAGE.PURGE\_CACHE Procedure, 19-9
- APEX\_PLUGIN, 20-1
  - data types, 20-2
  - GET AJAX\_IDENTIFIER function, 20-10
  - GET\_INPUT\_NAME\_FOR\_PAGE\_ITEM function, 20-11
- APEX\_PLUGIN\_UTIL, 21-1
  - DEBUG\_DYNAMIC\_ACTION procedure, 21-3
  - DEBUG\_PAGE\_ITEM procedure signature 1, 21-4
  - DEBUG\_PAGE\_ITEM procedure signature 2, 21-5
  - DEBUG\_PROCESS procedure, 21-6
  - DEBUG\_REGION procedure signature 1, 21-7
  - DEBUG\_REGION procedure signature 2, 21-8
  - ESCAPE function, 21-9
  - EXECUTE\_PLSQL\_CODE procedure, 21-10
  - GET\_ATTRIBUTE\_AS\_NUMBER function, 21-11
  - GET\_DATA function signature 1, 21-12
  - GET\_DATA Function Signature 2, 21-14
  - GET\_DATA2 function signature 1, 21-16
  - GET\_DATA2 function signature 2, 21-18
  - GET\_DISPLAY\_DATA function signature 1, 21-20
  - GET\_DISPLAY\_DATA function signature 2, 21-22
  - GET\_ELEMENT\_ATTRIBUTES function, 21-24
  - GET\_PLSQL\_EXPRESSION\_RESULT function, 21-26
  - GET\_PLSQL\_FUNCTION\_RESULT function, 21-27
  - GET\_POSITION\_IN\_LIST function, 21-28
  - GET\_SEARCH\_STRING function, 21-29
  - IS\_EQUAL function, 21-30
  - PAGE\_ITEM\_NAMES\_TO JQuery function, 21-31
  - PRINT\_DISPLAY\_ONLY procedure, 21-32
  - PRINT\_ESCAPED\_VALUE procedure, 21-33
  - PRINT\_HIDDEN\_IF\_READONLY procedure, 21-34
  - PRINT\_JSON\_HTTP\_HEADER procedure, 21-35
  - PRINT\_LOV\_AS\_JSON procedure, 21-36
  - PRINT\_OPTION procedure, 21-37
  - REPLACE\_SUBSTITUTIONS function, 21-38
- APEX\_REGION, 22-1
  - IS\_READ\_ONLY function, 22-2
  - PURGE\_CACHE procedure, 22-3
- APEX\_SPATIAL, 23-1
  - CHANGE\_GEOM\_METADATA Procedure, 23-3
  - CIRCLE\_POLYGON Function, 23-4
  - Data Types, 23-2
  - DELETE\_GEOM\_METADATA Procedure, 23-5
  - INSERT\_GEOM\_METADATA Procedure, 23-6
  - INSERT\_GEOM\_METADATA\_LONLAT Procedure, 23-7
  - POINT Function, 23-8
  - RECTANGLE Function, 23-9
- APEX\_UI\_DEFAULT\_UPDATE, 24-1
  - ADD\_AD\_COLUMN procedure, 24-3
  - ADD\_AD\_SYNONYM procedure, 24-5
  - DEL\_AD\_COLUMN procedure, 24-6
  - DEL\_AD\_SYNONYM procedure, 24-7
  - DEL\_COLUMN procedure, 24-8
  - DEL\_GROUP procedure, 24-9
  - DEL\_TABLE procedure, 24-10
  - SYNCH\_TABLE procedure, 24-11
  - UPD\_AD\_COLUMN procedure, 24-12
  - UPD\_AD\_SYNONYM procedure, 24-14
  - UPD\_COLUMN procedure, 24-15
  - UPD\_DISPLAY\_IN\_FORM procedure, 24-17
  - UPD\_DISPLAY\_IN\_REPORT procedure, 24-18
  - UPD\_FORM\_REGION\_TITLE procedure, 24-19
  - UPD\_GROUP procedure, 24-20
  - UPD\_ITEM\_DISPLAY\_HEIGHT procedure, 24-21
  - UPD\_ITEM\_DISPLAY\_WIDTH procedure, 24-22
  - UPD\_ITEM\_FORMAT\_MASK procedure, 24-23
  - UPD\_ITEM\_HELP procedure, 24-24
  - UPD\_ITEM\_LABEL procedure, 24-25
  - UPD\_REPORT\_ALIGNMENT procedure, 24-26
  - UPD\_REPORT\_FORMAT\_MASK procedure, 24-27
  - UPD\_REPORT\_REGION\_TITLE procedure, 24-28
  - UPD\_TABLE procedure, 24-29
- APEX\_UTIL, 25-1
  - CACHE\_GET\_DATE\_OF\_PAGE\_CACHE function, 25-6, 25-7
  - CACHE\_PURGE\_BY\_APPLICATION procedure, 25-8
  - CACHE\_PURGE\_BY\_PAGE procedure, 25-9
  - CACHE\_PURGE\_STALE procedure, 25-10
  - CHANGE\_CURRENT\_USER\_PW procedure, 25-11
  - CHANGE\_PASSWORD\_ON\_FIRST\_USE function, 25-12
  - CLEAR\_APP\_CACHE procedure, 25-14



CLEAR\_PAGE\_CACHE procedure, 25-15  
 CLEAR\_USER\_CACHE procedure, 25-16  
 COUNT\_CLICK procedure, 25-17  
 CREATE\_USER procedure, 25-18  
 CREATE\_USER\_GROUP procedure, 25-21  
 CURRENT\_USER\_IN\_GROUP function, 25-22  
 CUSTOM\_CALENDAR procedure, 25-23  
 DELETE\_USER\_GROUP procedure signature  
   1, 25-24  
 DELETE\_USER\_GROUP procedure signature  
   2, 25-25  
 DOWNLOAD\_PRINT\_DOCUMENT procedure  
   signature 1, 25-26  
 DOWNLOAD\_PRINT\_DOCUMENT procedure  
   signature 2, 25-27  
 DOWNLOAD\_PRINT\_DOCUMENT procedure  
   signature 3, 25-29  
 DOWNLOAD\_PRINT\_DOCUMENT procedure  
   signature 4, 25-31  
 EDIT\_USER procedure, 25-32  
 END\_USER\_ACCOUNT\_DAYS\_LEFT  
   function, 25-36  
 EXPIRE\_END\_USER\_ACCOUNT  
   procedure, 25-37  
 EXPIRE\_WORKSPACE\_ACCOUNT  
   procedure, 25-38  
 EXPORT\_USERS procedure, 25-39  
 FETCH\_APP\_ITEM function, 25-40  
 FETCH\_USER procedure signature 1, 25-41  
 FETCH\_USER procedure signature 2, 25-44  
 FETCH\_USER procedure signature 3, 25-46  
 FIND\_SECURITY\_GROUP\_ID function, 25-49  
 FIND\_WORKSPACE function, 25-50  
 GET\_ACCOUNT\_LOCKED\_STATUS  
   function, 25-51  
 GET\_ATTRIBUTE function, 25-52  
 GET\_AUTHENTICATION\_RESULT  
   function, 25-53  
 GET\_BLOB\_FILE\_SRC function, 25-54  
 GET\_BUILD\_OPTION\_STATUS function  
   signature 1, 25-56  
 GET\_BUILD\_OPTION\_STATUS function  
   signature 2, 25-57  
 GET\_CURRENT\_USER\_ID function, 25-58  
 GET\_DEFAULT\_SCHEMA function, 25-59  
 GET\_EDITION function, 25-60  
 GET\_EMAIL function, 25-61  
 GET\_FEEDBACK\_FOLLOW\_UP function, 25-62  
 GET\_FILE procedure, 25-63  
 GET\_FILE\_ID function, 25-65  
 GET\_FIRST\_NAME function, 25-66  
 GET\_GROUP\_ID function, 25-68  
 GET\_GROUP\_NAME function, 25-69  
 GET\_GROUPS\_USER\_BELONGS\_TO  
   function, 25-67  
 GET\_HASH function, 25-70  
 GET\_HIGH\_CONTRAST\_MODE\_TOGGLE  
   function, 25-71  
 GET\_LAST\_NAME function, 25-72  
 GET\_NUMERIC\_SESSION\_STATE  
   function, 25-73  
 GET\_PREFERENCE function, 25-74  
 GET\_PRINT\_DOCUMENT function signature  
   1, 25-75  
 GET\_PRINT\_DOCUMENT function signature  
   2, 25-76  
 GET\_PRINT\_DOCUMENT function signature  
   3, 25-77  
 GET\_PRINT\_DOCUMENT function signature  
   4, 25-78  
 GET\_SCREEN\_READER\_MODE\_TOGGLE  
   function, 25-80  
 GET\_SESSION\_LANG function, 25-81  
 GET\_SESSION\_STATE function, 25-82  
 GET\_SESSION\_TERRITORY function, 25-83  
 GET\_SESSION\_TIME\_ZONE function, 25-84  
 GET\_SINCE function, 25-85  
 GET\_USER\_ID function, 25-86  
 GET\_USER\_ROLES function, 25-87  
 GET\_USERNAME function, 25-88  
 HOST\_URL function, 25-89  
 IS\_HIGH\_CONTRAST\_SESSION function, 25-98  
 IS\_HIGH\_CONTRAST\_SESSION\_YN  
   function, 25-99  
 IS\_LOGIN\_PASSWORD\_VALID function, 25-100  
 IS\_SCREEN\_READER\_SESSION function, 25-101  
 IS\_SCREEN\_READER\_SESSION\_YN  
   function, 25-102  
 IS\_USERNAME\_UNIQUE function, 25-103  
 KEYVAL\_NUM function, 25-104  
 KEYVAL\_VC2 function, 25-105  
 LOCK\_ACCOUNT procedure, 25-106, 25-163  
 PASSWORD\_FIRST\_USE\_OCCURRED  
   function, 25-107  
 PREPARE\_URL function, 25-108  
 PURGE\_REGIONS\_BY\_NAME  
   procedure, 25-112  
 PURGE\_REGIONS\_BY\_PAGE procedure, 25-113  
 REDIRECT\_URL procedure, 25-114  
 REMOVE\_PREFERENCE procedure, 25-115  
 REMOVE\_SORT\_PREFERENCES  
   procedure, 25-116  
 REMOVE\_USER procedure, 25-117  
 RESET\_PASSWORD Procedure, 25-119  
 RESET\_PW procedure, 25-120  
 SAVEKEY\_NUM function, 25-121  
 SAVEKEY\_VC2 function, 25-122  
 SET\_ATTRIBUTE procedure, 25-123  
 SET\_AUTHENTICATION\_RESULT  
   procedure, 25-124  
 SET\_BUILD\_OPTION\_STATUS  
   procedure, 25-125  
 SET\_CURRENT\_THEME\_STYLE  
   procedure, 25-126  
 SET\_CUSTOM\_AUTH\_STATUS  
   procedure, 25-128  
 SET\_EDITION procedure, 25-129  
 SET\_EMAIL procedure, 25-130  
 SET\_FIRST\_NAME procedure, 25-131  
 SET\_GROUP\_GROUP\_GRANTS

- Procedure, 25-132
- SET\_GROUP\_USER\_GRANTS Procedure, 25-133
- SET\_LAST\_NAME procedure, 25-134
- SET\_PREFERENCE procedure, 25-135
- SET\_SECURITY\_GROUP\_ID procedure, 25-136, 25-140
- SET\_SECURITY\_HIGH\_CONTRAST\_OFF procedure, 25-138
- SET\_SECURITY\_HIGH\_CONTRAST\_ON procedure, 25-139
- SET\_SESSION\_MAX\_IDLE\_SECONDS procedure, 25-142
- SET\_SESSION\_SCREEN\_READER\_OFF procedure, 25-143
- SET\_SESSION\_SCREEN\_READER\_ON procedure, 25-144
- SET\_SESSION\_STATE procedure, 25-141, 25-145
- SET\_SESSION\_TERRITORY procedure, 25-146
- SET\_SESSION\_TIME\_ZONE procedure, 25-147
- SET\_USERNAME procedure, 25-148
- SHOW\_HIGH\_CONTRAST\_MODE\_TOGGLE procedure, 25-149
- SHOW\_SCREEN\_READER\_MODE\_TOGGLE procedure, 25-150
- STRING\_TO\_TABLE function, 25-151
- STRONG\_PASSWORD\_CHECK procedure, 25-152
- STRONG\_PASSWORD\_VALIDATION function, 25-156
- SUBMIT\_FEEDBACK procedure, 25-157
- SUBMIT\_FEEDBACK\_FOLLOWUP procedure, 25-159
- TABLE\_TO\_STRING function, 25-160
- UNEXPIRE\_END\_USER\_ACCOUNT procedure, 25-161
- UNEXPIRE\_WORKSPACE\_ACCOUNT procedure, 25-162
- URL\_ENCODE function, 25-164
- WORKSPACE\_ACCOUNT\_DAYS\_LEFT function, 25-166
- APEX\_WEB\_SERVICE, 26-1
  - About the APEX\_WEB\_SERVICE API, 26-2
  - BLOB2CLOBBASE64 function, 26-8
  - CLOBBASE642BLOB function, 26-9
  - Invoking a RESTful Style Web Service, 26-5
  - Invoking a SOAP Style Web Service, 26-3
  - MAKE\_REQUEST function, 26-12
  - MAKE\_REQUEST procedure, 26-10
  - MAKE\_REST\_REQUEST function, 26-14, 26-16
  - PARSE\_RESPONSE function, 26-18
  - PARSE\_RESPONSE\_CLOB function, 26-19
  - PARSE\_XML function, 26-20
  - PARSE\_XML\_CLOB function, 26-22
  - Retrieving Cookies and HTTP HeadersI, 26-6
  - Setting Cookies and HTTP Headers, 26-7
- APEX\_ZIP, 27-1
  - ADD\_FILE procedure, 27-3
  - Data Types, 27-2
  - FINISH procedure, 27-4
  - GET\_FILE\_CONTENT function, 27-5

- GET\_FILES function, 27-6
- APIs
  - APEX\_APPLICATION, 1-1
  - APEX\_APPLICATION\_INSTALL, 2-1
  - APEX\_AUTHORIZATION, 3-xxiii, 4-1
  - APEX\_COLLECTION, 5-3
  - APEX\_CSS, 6-1
  - APEX\_CUSTOM\_AUTH, 7-1
  - APEX\_DEBUG, 8-1
  - APEX\_ESCAPE, 10-1
  - APEX\_INSTANCE\_ADMIN, 11-1
  - APEX\_IR, 12-1
  - APEX\_ITEM, 13-1
  - APEX\_JAVASCRIPT, 14-1
  - APEX\_JSON, 15-1
  - APEX\_LANG, 16-1
  - APEX\_LDAP, 17-1
  - APEX\_MAIL, 18-1
  - APEX\_PAGE, 19-1
  - APEX\_PLUGIN, 20-1
  - APEX\_PLUGIN\_UTIL, 21-1
  - APEX\_REGION, 22-1
  - APEX\_SPATIAL, 23-1
  - APEX\_UI\_DEFAULT\_UPDATE, 24-1
  - APEX\_UTIL, 25-1
  - APEX\_WEB\_SERVICE, 26-1
  - APEX\_ZIP, 27-1
  - JavaScript API, 28-1
- application
  - sending messages in APEX\_MAIL\_QUEUE, 18-6
  - sending outbound email, 18-7
  - sending outbound email as attachment, 18-3
- attribute values
  - setting, 25-123
- authenticated user
  - create user group, 25-21
  - delete user group, 25-24, 25-25
- authentication, scheme session cookies, 7-5

## C

---

- check box, creating, 13-2
- clicks, counting, 25-17
- collections
  - accessing, 5-7
  - APEX\_COLLECTION API, 5-3
  - clearing session state, 5-22
  - creating, 5-5
  - deleting members, 5-16
  - determining status, 5-23
  - merging, 5-8
  - truncating, 5-9
  - updating members, 5-15

## D

---

- APEX\_UTIL
  - IR\_CLEAR procedure, 25-92
  - IR\_DELETE\_REPORT procedure, 25-93
  - IR\_DELETE\_SUBSCRIPTION procedure, 25-94

IR\_FILTER procedure, 25-95  
IR\_RESET procedure, 25-97  
PUBLIC\_CHECK\_AUTHORIZATION  
function, 25-110  
PURGE\_REGIONS\_BY\_APP procedure, 25-111  
RESET\_AUTHORIZATIONS procedure, 25-118

## E

email

sending as an attachment, 18-3  
sending messages in APEX\_MAIL\_QUEUE, 18-6  
sending outbound, 18-7

export file, of workspace, 25-39

## F

F01, 1-3

file repository

downloading files, 25-63  
obtaining primary key, 25-65

Function, 26-22

## J

JavaScript API, 28-1

`$d_ClearAndHide(pNd)`, 28-115  
`$d_Find(pThis,pString,pTags,pClass)`, 28-148  
`$dom_`  
  `AddInput(pThis,pType,pId,pName,pValue)`,  
  28-143  
`$dom_AddTag(pThis, pTag, pText)`, 28-140  
`$dom_MakeParent(p_Node,p_Parent)`, 28-144  
`$f_CheckAll(pThis, pCheck, pArray)`, 28-129  
`$f_CheckFirstColumn(pNd)`, 28-130  
`$f_DisableOnValue(pThis, pValue, pThat)`, 28-125  
`$f_First_field(pNd)`, 28-149  
`$f_get_emptys(pNd, pClassFail, pClass)`, 28-112  
`$f_Hide_On_Value_Item_Row(pThis, pThat,`

`pValue)`, 28-123  
`$f_Hide_On_Value_Item(pThis, pThat,`

`pValue)`, 28-121  
`$f_ReturnChecked(pNd)`, 28-114  
`$f_SelectedOptions(pNd)`, 28-116  
`$f_SelectValue(pNd)`, 28-117  
`$f_SetValueSequence(pArray,pMultiple)`, 28-139  
`$f_Show_On_Value_Item_Row(pThis, pThat,`

`pValue)`, 28-124  
`$f_Show_On_Value_Item(pThis, pThat,`

`pValue)`, 28-122  
`$f_Swap(pThis,pThat)`, 28-138  
`$f_ValuesToArray(pThis, pClass, pTag)`, 28-127  
`$nvl(pTest, pDefault)`, 28-90  
`$s(pNd, pValue)`, 28-87  
`$tr_AddTD(pThis,pText)`, 28-141  
`$tr_AddTH(pThis,pText)`, 28-142  
`$u_ArrayToString(pArray, pDelim)`, 28-118  
`$u_Carray(pNd)`, 28-89  
`$u_Narray(pNd)`, 28-88  
`$u_SubString(pText,pMatch)`, 28-134  
`$v_Array(pNd)`, 28-113

`$v_CheckValueAgainst(pThis, pValue)`, 28-120  
`$v_IsEmpty(pThis)`, 28-152  
`$v_PopupReturn(pValue, pThat)`, 28-151  
`$v_Upper(pNd)`, 28-147  
`$v(pNd)`, 28-85  
`$v2(pNd)`, 28-86  
`$x_ByClass(pClass, pNd, pTag)`, 28-107  
`$x_CheckImageSrc(pId,pSearch)`, 28-119  
`$x_CheckImageSrc(pNd, pSearch)`, 28-133  
`$x_Class(pNd,pClass)`, 28-105  
`$x_ClassByClass(pNd, pClass, pTag,`

`pClass2)`, 28-126  
`$x_disableItem(pNd,a)`, 28-111  
`$x_FormItems(pNd, pType)`, 28-128  
`$x_Hide(pNd)`, 28-92  
`$x_HideAllExcept(pNd,pNdArray)`, 28-102  
`$x_HideItemRow(pNd)`, 28-99  
`$x_HideSiblings(pNd)`, 28-103  
`$x_ItemRow(pNd,pFunc)`, 28-98  
`$x_Remove(pNd)`, 28-95  
`$x_RowHighlight(pThis,pColor)`, 28-145  
`$x_RowHighlightOff(pThis)`, 28-146  
`$x_SetSiblingsClass(pNd, pClass,`

`pNdClass)`, 28-106  
`$x_Show(pNd)`, 28-93  
`$x_ShowAllByClass(pNd, pClass, pTag)`, 28-108  
`$x_ShowChildren(pThis)`, 28-109, 28-110  
`$x_ShowItemRow(pNd)`, 28-100  
`$x_ShowSiblings(pNd)`, 28-104  
`$x_Style(pNd, pStyle, pString)`, 28-91  
`$x_SwitchImageSrc(pNd, pSearch,`

`pReplace)`, 28-132  
`$x_Toggle(pNd)`, 28-94  
`$x_ToggleItemRow(pNd)`, 28-101  
`$x_ToggleWithImage(pThis,pNd)`, 28-131  
`$x_UpTill(pNd, pToTag)`, 28-97  
`$x_Value(pNd,pValue)`, 28-96  
`$x(pNd)`, 28-84  
`addLoadEvent(pFunction)`, 28-137  
apex namespace, 28-2  
apex.da namespace, 28-10  
apex.debug namespace, 28-12  
apex.event namespace, 28-22  
`apex.event.trigger(pSelector,pEvent,dData)`, 28-2  
3  
apex.item, 28-24  
apex.lang namespace, 28-37  
apex.navigation namespace, 28-45  
apex.server namespace, 28-55  
apex.storage namespace, 28-67  
`apex.submit(pRequest | pOptions)`, 28-8, 28-9  
apex.util namespace, 28-70  
`apex.util namespace.apex.util.delayLinger`, 28-72  
apex.util  
  namespace.apex.util.delayLinger.finish, 28-7  
  2  
apex.util  
  namespace.apex.util.delayLinger.start, 28-72  
apex.util  
  namespace.apex.util.showSpinner, 28-71

- apex.widget namespace, 28-74
- apex.widget.initPageItem(pName,pOptions), 28-75
- confirmDelete(pMessage, pRequest), 28-5, 28-6
- GetCookie (pName), 28-155
- html\_RemoveAllChildren(pNd), 28-135
- html\_SetSelectValue(pId,pValue), 28-136
- non-namespace APIs, 28-82
- SetCookie (pName,pValue), 28-156
- setReturn(p\_R,p\_D), 28-154
- submitEnter(pNd,e), 28-153

## L

---

LDAP attributes, obtaining, 7-6

## P

---

password

- changing, 25-11
- resetting and emailing, 25-120

## R

---

radio group, generate, 13-21

## S

---

session cookies, 7-5  
session state

- fetching for current application, 25-40
- removing for current page, 25-15
- removing for current session, 25-14
- setting, 25-141, 25-145

special characters, encoding, 25-164

## U

---

user

- get e-mail address, 25-61
- remove preference, 25-115

user account

- altering, 25-32
- creating new, 25-18
- fetching, 25-41, 25-44, 25-46
- removing, 25-117
- update email address, 25-130
- updating FIRST\_NAME, 25-131
- updating LAST\_NAME value, 25-134
- updating USER\_NAME value, 25-148

## V

---

variables, global, 1-1

## W

---

workspace

- export file, 25-39
- numeric security group ID, 25-49