

**Oracle® Application Express**

API Reference

Release 3.2

**E13369-01**

February 2009

Oracle Application Express API Reference, Release 3.2

E13369-01

Copyright © 2003, 2009, Oracle and/or its affiliates. All rights reserved.

Primary Author: Drue Baker

Contributors: Marco Adelfio, Drue Baker, Carl Backstrom, Christina Cho, Steve Fogel, Michael Hichwa, Terri Jennings, Christopher Jones, Joel Kallman, Sharon Kennedy, Syme Kutz, Sergio Leunissen, Anne Romano, Kris Rice, Marc Sewtz, Scott Spadafore, Scott Spendolini, Jason Straub, and Simon Watt.

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

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software 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 which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on 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. 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.

---

---

# Contents

<b>Preface</b> .....	xi
Topic Overview .....	xi
Audience .....	xii
Documentation Accessibility .....	xii
Related Documents .....	xiii
Conventions .....	xiv
<b>1 APEX_UTIL</b>	
CACHE_GET_DATE_OF_PAGE_CACHE Function .....	1-4
CACHE_GET_DATE_OF_REGION_CACHE Function .....	1-5
CACHE_PURGE_BY_APPLICATION Procedure .....	1-6
CACHE_PURGE_BY_PAGE Procedure .....	1-7
CACHE_PURGE_STALE Procedure .....	1-8
CHANGE_CURRENT_USER_PW Procedure .....	1-9
CHANGE_PASSWORD_ON_FIRST_USE Function .....	1-10
CLEAR_APP_CACHE Procedure .....	1-11
CLEAR_PAGE_CACHE Procedure .....	1-12
CLEAR_USER_CACHE Procedure .....	1-13
COUNT_CLICK Procedure .....	1-14
CREATE_USER Procedure .....	1-15
CREATE_USER_GROUP Procedure .....	1-18
CURRENT_USER_IN_GROUP Function .....	1-19
DOWNLOAD_PRINT_DOCUMENT Procedure Signature 1 .....	1-20
DOWNLOAD_PRINT_DOCUMENT Procedure Signature 2 .....	1-21
DOWNLOAD_PRINT_DOCUMENT Procedure Signature 3 .....	1-23
DOWNLOAD_PRINT_DOCUMENT Procedure Signature 4 .....	1-25
EDIT_USER Procedure .....	1-26
END_USER_ACCOUNT_DAYS_LEFT Function .....	1-30
EXPIRE_END_USER_ACCOUNT Procedure .....	1-31
EXPIRE_WORKSPACE_ACCOUNT Procedure .....	1-32
EXPORT_USERS Procedure .....	1-33
FETCH_APP_ITEM Function .....	1-34

FETCH_USER Procedure Signature 1.....	1-35
FETCH_USER Procedure Signature 2.....	1-38
FETCH_USER Procedure Signature 3.....	1-40
FIND_SECURITY_GROUP_ID Function .....	1-43
FIND_WORKSPACE Function .....	1-44
GET_ACCOUNT_LOCKED_STATUS Function.....	1-45
GET_ATTRIBUTE Function .....	1-46
GET_AUTHENTICATION_RESULT Function.....	1-47
GET_BLOB_FILE_SRC Function .....	1-48
GET_CURRENT_USER_ID Function.....	1-50
GET_DEFAULT_SCHEMA Function .....	1-51
GET_EMAIL Function.....	1-52
GET_FILE Procedure.....	1-53
GET_FILE_ID Function .....	1-55
GET_FIRST_NAME Function .....	1-56
GET_GROUPS_USER_BELONGS_TO Function .....	1-57
GET_GROUP_ID Function .....	1-58
GET_GROUP_NAME Function.....	1-59
GET_LAST_NAME Function .....	1-60
GET_NUMERIC_SESSION_STATE Function .....	1-61
GET_PREFERENCE Function.....	1-62
GET_PRINT_DOCUMENT Function Signature 1 .....	1-63
GET_PRINT_DOCUMENT Function Signature 2 .....	1-64
GET_PRINT_DOCUMENT Function Signature 3 .....	1-65
GET_PRINT_DOCUMENT Function Signature 4 .....	1-66
GET_SESSION_STATE Function.....	1-68
GET_USER_ID Function .....	1-69
GET_USER_ROLES Function.....	1-70
GET_USERNAME Function.....	1-71
IS_LOGIN_PASSWORD_VALID Function .....	1-72
IS_USERNAME_UNIQUE Function.....	1-73
KEYVAL_NUM Function .....	1-74
KEYVAL_VC2 Function.....	1-75
LOCK_ACCOUNT Procedure .....	1-76
PASSWORD_FIRST_USE_OCCURRED Function.....	1-77
PREPARE_URL Function .....	1-78
PUBLIC_CHECK_AUTHORIZATION Function.....	1-80
PURGE_REGIONS_BY_APP Procedure .....	1-81
PURGE_REGIONS_BY_NAME Procedure.....	1-82
PURGE_REGIONS_BY_PAGE Procedure .....	1-83
REMOVE_PREFERENCE Procedure.....	1-84
REMOVE_SORT_PREFERENCES Procedure .....	1-85

REMOVE_USER Procedure.....	1-86
RESET_AUTHORIZATIONS Procedure.....	1-87
RESET_PW Procedure.....	1-88
SAVEKEY_NUM Function.....	1-89
SAVEKEY_VC2 Function.....	1-90
SET_ATTRIBUTE Procedure.....	1-91
SET_AUTHENTICATION_RESULT Procedure.....	1-92
SET_CUSTOM_AUTH_STATUS Procedure.....	1-93
SET_EMAIL Procedure.....	1-94
SET_FIRST_NAME Procedure.....	1-95
SET_LAST_NAME Procedure.....	1-96
SET_PREFERENCE Procedure.....	1-97
SET_SESSION_LIFETIME_SECONDS Procedure.....	1-98
SET_SESSION_MAX_IDLE_SECONDS Procedure.....	1-100
SET_SESSION_STATE Procedure.....	1-102
SET_USERNAME Procedure.....	1-103
STRONG_PASSWORD_CHECK Procedure.....	1-104
STRONG_PASSWORD_VALIDATION Function.....	1-108
STRING_TO_TABLE Function.....	1-109
TABLE_TO_STRING Function.....	1-110
UNEXPIRE_END_USER_ACCOUNT Procedure.....	1-111
UNEXPIRE_WORKSPACE_ACCOUNT Procedure.....	1-112
UNLOCK_ACCOUNT Procedure.....	1-113
URL_ENCODE Function.....	1-114
WORKSPACE_ACCOUNT_DAYS_LEFT Function.....	1-116

## 2 APEX\_MAIL

ADD_ATTACHMENT Procedure.....	2-2
PUSH_QUEUE Procedure.....	2-3
SEND Procedure.....	2-4

## 3 APEX\_ITEM

CHECKBOX Function.....	3-2
DATE_POPUP Function.....	3-4
DISPLAY_AND_SAVE Function.....	3-6
HIDDEN Function.....	3-7
MD5_CHECKSUM Function.....	3-9
MD5_HIDDEN Function.....	3-10
POPUP_FROM_LOV Function.....	3-11
POPUP_FROM_QUERY Function.....	3-13
POPUPKEY_FROM_LOV Function.....	3-15

POPUPKEY_FROM_QUERY Function .....	3-17
RADIOGROUP Function .....	3-19
SELECT_LIST Function.....	3-20
SELECT_LIST_FROM_LOV Function .....	3-22
SELECT_LIST_FROM_LOV_XL Function .....	3-23
SELECT_LIST_FROM_QUERY Function.....	3-25
SELECT_LIST_FROM_QUERY_XL Function.....	3-27
TEXT Function.....	3-29
TEXTAREA Function .....	3-30
TEXT_FROM_LOV Function .....	3-31
TEXT_FROM_LOV_QUERY Function .....	3-32

#### **4 APEX\_APPLICATION**

Referencing Arrays .....	4-3
Referencing Values Within an On Submit Process .....	4-4
Converting an Array to a Single Value.....	4-5
HELP Procedure.....	4-6

#### **5 APEX\_CUSTOM\_AUTH**

APPLICATION_PAGE_ITEM_EXISTS Function.....	5-2
CURRENT_PAGE_IS_PUBLIC Function .....	5-3
DEFINE_USER_SESSION Procedure .....	5-4
GET_COOKIE_PROPS Procedure.....	5-5
GET_LDAP_PROPS Procedure .....	5-6
GET_NEXT_SESSION_ID Function.....	5-7
GET_SECURITY_GROUP_ID Function .....	5-8
GET_SESSION_ID Function.....	5-9
GET_SESSION_ID_FROM_COOKIE Function .....	5-10
GET_USER Function .....	5-11
GET_USERNAME Function.....	5-12
IS_SESSION_VALID Function.....	5-13
LOGIN Procedure.....	5-14
LOGOUT Procedure .....	5-15
POST_LOGIN Procedure.....	5-16
SESSION_ID_EXISTS Function.....	5-17
SET_SESSION_ID Procedure .....	5-18
SET_SESSION_ID_TO_NEXT_VALUE Procedure.....	5-19
SET_USER Procedure .....	5-20

#### **6 APEX\_LDAP**

AUTHENTICATE Function .....	6-2
GET_ALL_USER_ATTRIBUTES Procedure .....	6-3

GET_USER_ATTRIBUTES Procedure .....	6-4
IS_MEMBER Function.....	6-5
MEMBER_OF Function.....	6-6
MEMBER_OF2 Function.....	6-7

## 7 APEX\_INSTANCE\_ADMIN

ADD_SCHEMA Procedure .....	7-2
ADD_WORKSPACE Procedure .....	7-3
GET_PARAMETER Function.....	7-4
GET_SCHEMAS Function .....	7-5
REMOVE_SAVED_REPORTS Procedure .....	7-6
REMOVE_SCHEMA Procedure .....	7-7
REMOVE_WORKSPACE Procedure .....	7-8
SET_PARAMETER Procedure .....	7-9
Available Parameter Values .....	7-10

## 8 APEX\_UI\_DEFAULT\_UPDATE

UPD_DISPLAY_IN_FORM Procedure .....	8-2
UPD_DISPLAY_IN_REPORT Procedure .....	8-3
UPD_FORM_REGION_TITLE Procedure.....	8-4
UPD_ITEM_DISPLAY_HEIGHT Procedure .....	8-5
UPD_ITEM_DISPLAY_WIDTH Procedure .....	8-6
UPD_ITEM_FORMAT_MASK Procedure .....	8-7
UPD_ITEM_HELP Procedure.....	8-8
UPD_LABEL Procedure.....	8-9
UPD_REPORT_ALIGNMENT Procedure.....	8-10
UPD_REPORT_FORMAT_MASK Procedure.....	8-11
UPD_REPORT_REGION_TITLE Procedure.....	8-12

## 9 JavaScript APIs

\$x(pNd) .....	9-4
\$v(pNd) .....	9-5
\$s(pNd, pValue) .....	9-6
\$u_Carray(pNd) .....	9-7
\$u_Narray(pNd) .....	9-8
\$nvl(pTest, pDefault) .....	9-9
doSubmit(pRequest).....	9-10
confirmDelete(pMessage, pRequest) .....	9-11
\$x_Style(pNd, pStyle, pString).....	9-12
\$x_Hide(pNd).....	9-13
\$x_Show(pNd).....	9-14

<code>\$x_Toggle(pNd)</code> .....	9-15
<code>\$x_Remove(pNd)</code> .....	9-16
<code>\$x_Value(pNd,pValue)</code> .....	9-17
<code>\$x_UpTill(pNd, pToTag)</code> .....	9-18
<code>\$x_ItemRow(pNd,pFunc)</code> .....	9-19
<code>\$x_HideItemRow(pNd)</code> .....	9-20
<code>\$x_ShowItemRow(pNd)</code> .....	9-21
<code>\$x_ToggleItemRow(pNd)</code> .....	9-22
<code>\$x_HideAllExcept(pNd,pNdArray)</code> .....	9-23
<code>\$x_HideSiblings(pNd)</code> .....	9-24
<code>\$x_ShowSiblings(pNd)</code> .....	9-25
<code>\$x_Class(pNd,pClass)</code> .....	9-26
<code>\$x_SetSiblingsClass(pNd, pClass, pNdClass)</code> .....	9-27
<code>\$x_ByClass(pClass, pNd, pTag)</code> .....	9-28
<code>\$x_ShowAllByClass(pNd, pClass, pTag)</code> .....	9-29
<code>\$x_ShowChildren(pNd)</code> .....	9-30
<code>\$x_HideChildren(pNd)</code> .....	9-31
<code>\$x_disableItem(pNd, pTest)</code> .....	9-32
<code>\$f_get_emptyys(pNd, pClassFail, pClass)</code> .....	9-33
<code>\$v_Array(pNd)</code> .....	9-34
<code>\$f_ReturnChecked(pNd)</code> .....	9-35
<code>\$d_ClearAndHide(pNd)</code> .....	9-36
<code>\$f_SelectedOptions(pNd)</code> .....	9-37
<code>\$f_SelectValue(pNd)</code> .....	9-38
<code>\$u_ArrayToString(pArray, pDelim)</code> .....	9-39
<code>\$x_CheckImageSrc(pId,pSearch)</code> .....	9-40
<code>\$v_CheckValueAgainst(pThis, pValue)</code> .....	9-41
<code>\$f_Hide_On_Value_Item(pThis, pThat, pValue)</code> .....	9-42
<code>\$f_Show_On_Value_Item(pThis, pThat, pValue)</code> .....	9-43
<code>\$f_Hide_On_Value_Item_Row(pThis, pThat, pValue)</code> .....	9-44
<code>\$f_Show_On_Value_Item_Row(pThis, pThat, pValue)</code> .....	9-45
<code>\$f_DisableOnValue(pThis, pValue, pThat)</code> .....	9-46
<code>\$x_ClassByClass(pNd, pClass, pTag, pClass2)</code> .....	9-47
<code>\$f_ValuesToArray(pThis, pClass, pTag)</code> .....	9-48
<code>\$x_FormItems(pNd, pType)</code> .....	9-49
<code>\$f_CheckAll(pThis, pCheck, pArray)</code> .....	9-50
<code>\$f_CheckFirstColumn(pNd)</code> .....	9-51
<code>\$v_PopupReturn(pValue, pThat)</code> .....	9-52
<code>\$x_ToggleWithImage(pThis,pNd)</code> .....	9-53
<code>\$x_SwitchImageSrc(pNd, pSearch, pReplace)</code> .....	9-54
<code>\$x_CheckImageSrc(pNd, pSearch)</code> .....	9-55
<code>\$u_SubString(pText,pMatch)</code> .....	9-56



html_RemoveAllChildren(pNd).....	9-57
\$v_IsEmpty(pThis).....	9-58
html_SetSelectValue(pId,pValue).....	9-59
addLoadEvent(pFunction) .....	9-60
\$f_Swap(pThis,pThat) .....	9-61
submitEnter(pNd,e).....	9-62
\$f_SetValueSequence(pArray,pMultiple).....	9-63
\$dom_AddTag(pThis, pTag, pText).....	9-64
\$tr_AddTD(pThis,pText) .....	9-65
\$tr_AddTH(pThis,pText) .....	9-66
\$dom_AddInput(pThis,pType,pId,pName,pValue) .....	9-67
\$dom_MakeParent(p_Node,p_Parent) .....	9-68
\$x_RowHighlight(pThis, pColor) .....	9-69
\$x_RowHighlightOff(pThis) .....	9-70
\$v_Upper(pNd).....	9-71
\$d_Find(pThis,pString,pTags,pClass) .....	9-72
returnInput(p_R, p_D) .....	9-73
setReturn(p_R,p_D) .....	9-74
\$f_First_field(pNd) .....	9-75
GetCookie (pName).....	9-76
SetCookie (pName,pValue) .....	9-77

## 10 APEX\_PLSQL\_JOB

JOBS_ARE_ENABLED Function .....	10-2
PURGE_PROCESS Procedure .....	10-3
SUBMIT_PROCESS Function.....	10-4
TIME_ELAPSED Function.....	10-5
UPDATE_JOB_STATUS Procedure .....	10-6

## 11 APEX\_LANG

LANG Function.....	11-2
MESSAGE Function.....	11-3

## 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.

This preface contains these topics:

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

## Topic Overview

This document contains the following chapters:

Title	Description
<a href="#">APEX_UTIL</a>	Use the <code>APEX_UTIL</code> package to get and set session state, get files, check authorizations for users, reset different states for users, and also to get and set preferences for users.
<a href="#">APEX_MAIL</a>	Use the <code>APEX_MAIL</code> package to send an email from an Oracle Application Express application.
<a href="#">APEX_ITEM</a>	Use the <code>APEX_ITEM</code> package to create form elements dynamically based on a SQL query instead of creating individual items page by page.
<a href="#">APEX_APPLICATION</a>	Use the <code>APEX_APPLICATION</code> package to take advantage of a number of global variables.
<a href="#">APEX_CUSTOM_AUTH</a>	Use the <code>APEX_CUSTOM_AUTH</code> package to perform various operations related to authentication and session management.
<a href="#">APEX_LDAP</a>	Use <code>APEX_LDAP</code> to perform various operations related to Lightweight Directory Access Protocol (LDAP) authentication.
<a href="#">APEX_INSTANCE_ADMIN</a>	Use the <code>APEX_INSTANCE_ADMIN</code> package to get and set email settings, wallet settings, report printing settings and to manage schema to workspace mappings.

Title	Description
<a href="#">APEX_UI_DEFAULT_UPDATE</a>	You can use the <code>APEX_UI_DEFAULT_UPDATE</code> 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.
<a href="#">JavaScript APIs</a>	Use these JavaScript 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.
<a href="#">APEX_PLSQL_JOB</a>	You can use <code>APEX_PLSQL_JOB</code> package to run PL/SQL code in the background of your application. This is an effective approach for managing long running operations that do not need to complete for a user to continue working with your application.
<a href="#">APEX_LANG</a>	You can use <code>APEX_LANG</code> API to translate messages.

---

**Note:** In release 2.2, Oracle Application Express APIs were renamed using the prefix `APEX_`. Note that API's using the previous prefix `HTMLDB_` are still supported to provide backward compatibility. As a best practice, however, use the new API names for new applications unless you plan to run them in an earlier version of Oracle Application Express.

---

## 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 as well as an understanding of the operating system environment under which you are running Oracle Application Express.

**See Also:** *Oracle 2 Day + Application Express Developer's Guide*

## Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at <http://www.oracle.com/accessibility/>.

### Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an

otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

### **Accessibility of Links to External Web Sites in Documentation**

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

### **TTY Access to Oracle Support Services**

To reach AT&T Customer Assistants, dial 711 or 1.800.855.2880. An AT&T Customer Assistant will relay information between the customer and Oracle Support Services at 1.800.223.1711. Complete instructions for using the AT&T relay services are available at <http://www.consumer.att.com/relay/tty/standard2.html>. After the AT&T Customer Assistant contacts Oracle Support Services, an Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process.

## **Related Documents**

For more information, see these Oracle resources:

- *Oracle Application Express Release Notes*
- *Oracle Application Express Installation Guide*
- *Oracle 2 Day + Application Express Developer's Guide*
- *Oracle Application Express Advanced Tutorials*
- *Oracle Application Express Administration Guide*
- *Oracle Application Express Migration Guide*
- *Oracle Application Express SQL Workshop and Utilities 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*

For information about Oracle error messages, see *Oracle Database Error Messages*. Oracle error message documentation is available only in HTML. If you have access to the Oracle Database Documentation Library, you can browse the error messages by range. Once you find the specific range, use your browser's "find in page" feature to locate the specific message. When connected to the Internet, you can search for a specific error message using the error message search feature of the Oracle online documentation.

Many books in the documentation set use the sample schemas of the seed database, which is installed by default when you install Oracle. Refer to *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them yourself.

For additional documentation available on Oracle's Technology Network, please visit the Oracle Application Express web site located at

[http://www.oracle.com/technology/products/database/application\\_express/](http://www.oracle.com/technology/products/database/application_express/)

For additional application examples, please visit the Oracle by Examples (OBES) Application Express page, located on Oracle's Technology Network. The OBES provide step-by-step examples with screenshots on how to perform various tasks within Application Express.

[http://www.oracle.com/technology/products/database/application\\_express/html/obes.html](http://www.oracle.com/technology/products/database/application_express/html/obes.html)

Printed documentation is available for sale in the Oracle Store at

<http://oraclestore.oracle.com/>

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

<http://www.oracle.com/technology/membership/>

If you already have a user name and password for OTN, then you can go directly to the documentation section of the OTN Web site at

<http://www.oracle.com/technology/documentation/>

## 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:

Convention	Meaning
<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.

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.

Topics in this section include:

- [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](#)
- [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](#)
- [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\\_CURRENT\\_USER\\_ID Function](#)
  - [GET\\_DEFAULT\\_SCHEMA Function](#)
  - [GET\\_EMAIL 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\\_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\\_SESSION\\_STATE Function](#)
  - [GET\\_USER\\_ID Function](#)
  - [GET\\_USER\\_ROLES Function](#)
  - [GET\\_USERNAME Function](#)
  - [IS\\_LOGIN\\_PASSWORD\\_VALID 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](#)



- 
- PURGE\_REGIONS\_BY\_APP Procedure
  - PURGE\_REGIONS\_BY\_NAME Procedure
  - PURGE\_REGIONS\_BY\_PAGE Procedure
  - REMOVE\_PREFERENCE Procedure
  - REMOVE\_SORT\_PREFERENCES Procedure
  - REMOVE\_USER Procedure
  - RESET\_AUTHORIZATIONS Procedure
  - RESET\_PW Procedure
  - SAVEKEY\_NUM Function
  - SAVEKEY\_VC2 Function
  - SET\_ATTRIBUTE Procedure
  - SET\_AUTHENTICATION\_RESULT Procedure
  - SET\_CUSTOM\_AUTH\_STATUS Procedure
  - SET\_EMAIL Procedure
  - SET\_FIRST\_NAME Procedure
  - SET\_LAST\_NAME Procedure
  - SET\_PREFERENCE Procedure
  - SET\_SESSION\_LIFETIME\_SECONDS Procedure
  - SET\_SESSION\_MAX\_IDLE\_SECONDS Procedure
  - SET\_SESSION\_STATE Procedure
  - SET\_USERNAME Procedure
  - STRONG\_PASSWORD\_CHECK Procedure
  - STRONG\_PASSWORD\_VALIDATION Function
  - STRING\_TO\_TABLE Function
  - 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 1–1 describes the parameters available in the `CACHE_GET_DATE_OF_PAGE_CACHE` procedure.

**Table 1–1** *CACHE\_GET\_DATE\_OF\_PAGE\_CACHE Parameters*

Parameter	Description
<code>p_application</code>	The identification number (ID) of the application.
<code>p_page</code>	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 1–2 describes the parameters available in the CACHE\_GET\_DATE\_OF\_REGION\_CACHE function.

**Table 1–2** 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 1–3](#) describes the parameters available in the `CACHE_PURGE_BY_APPLICATION` procedure.

**Table 1–3** *CACHE\_PURGE\_BY\_APPLICATION Parameters*

Parameter	Description
<code>p_application</code>	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 will also be 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 1–4](#) describes the parameters available in the `CACHE_PURGE_BY_PAGE` procedure.

**Table 1–4** *CACHE\_PURGE\_BY\_PAGE Parameters*

Parameter	Description
<code>p_application</code>	The identification number (ID) of the application.
<code>p_page</code>	The page number (ID).
<code>p_user_name</code>	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 will no longer be used, thus removing those unusable pages or regions from the cache.

### Syntax

```
APEX_UTIL.CACHE_PURGE_STALE (  
    p_application IN    NUMBER);
```

### Parameters

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

**Table 1–5** *CACHE\_PURGE\_STALE Parameters*

Parameter	Description
<code>p_application</code>	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 1–6](#) describes the parameters available in the CHANGE\_CURRENT\_USER\_PW procedure.

**Table 1–6** 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 1-88

---

## 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 1-7](#) describes the parameters available in the CHANGE\_PASSWORD\_ON\_FIRST\_USE function.

**Table 1-7** 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
            htp.p('User: ' || c1.user_name || ' requires password to be changed the
            first time it is used.');
```

**See Also:** ["PASSWORD\\_FIRST\\_USE\\_OCCURRED Function"](#) on page 1-77



---

## 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 1–8](#) describes the parameters available in the CLEAR\_APP\_CACHE procedure.

**Table 1–8** CLEAR\_APP\_CACHE Parameters

Parameter	Description
p_app_id	The ID of the application for which session state will be 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 IN NUMBER DEFAULT NULL);
```

### Parameters

[Table 1–9](#) describes the parameters available in the CLEAR\_PAGE\_CACHE procedure.

**Table 1–9** CLEAR\_PAGE\_CACHE Parameters

Parameter	Description
p_page	The ID of the page in the current application for which session state will be 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 1–10 describes the parameters available in the COUNT\_CLICK procedure.

**Table 1–10** COUNT\_CLICK Parameters

Parameter	Description
p_url	The URL to which to redirect
p_cat	A category to classify the click
p_id	Secondary ID to associate with the click (optional)
p_user	The application user ID (optional)
p_workspace	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 via 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 1-43 in this document and "Purging the External Click Count Log" 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    DEFAULT NULL,
  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);
```

### Parameters

[Table 1–11](#) describes the parameters available in the CREATE\_USER procedure.

**Table 1–11** 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
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

**Table 1–11 (Cont.) CREATE\_USER Procedure Parameters**

Parameter	Description
p_developer_privs	Colon separated list of developer privileges. The following are acceptable values for this parameter:  <b>null</b> - To create an end user (a user who can only authenticate to developed applications).  <b>CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL</b> - To create a user with developer privilege.  <b>ADMIN:CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL</b> - To create 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.
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 will default 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	

**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 1-40, ["EDIT\\_USER Procedure"](#) on page 1-26, and ["GET\\_GROUP\\_ID Function"](#) on page 1-58

## 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,
    p_group_name        IN                VARCHAR2,
    p_security_group_id IN                NUMBER,
    p_group_desc        IN                VARCHAR2);
```

### Parameter

[Table 1–12](#) describes the parameters available in the CREATE\_USER\_GROUP procedure.

**Table 1–12 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 will assign 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 or not 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 1–13](#) describes the parameters available in the `CURRENT_USER_IN_GROUP` function.

**Table 1–13** *CURRENT\_USER\_IN\_GROUP Parameters*

Parameter	Description
<code>p_group_name</code>	Identifies the name of an existing group in the workspace
<code>p_group_id</code>	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;
```

## 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 1–14](#) describes the parameters available in the `DOWNLOAD_PRINT_DOCUMENT` procedure for Signature 1.

**Table 1–14** *DOWNLOAD\_PRINT\_DOCUMENT Parameters*

Parameter	Description
<code>p_file_name</code>	Defines the filename of the print document
<code>p_content_disposition</code>	Specifies whether to download the print document or display inline ("attachment", "inline")
<code>p_report_data</code>	XML based report data
<code>p_report_layout</code>	Report layout in XSL-FO or RTF format
<code>p_report_layout_type</code>	Defines the report layout type, that is "xsl-fo" or "rtf"
<code>p_document_format</code>	Defines the document format, that is "pdf", "rtf", "xls", "htm", or "xml"
<code>p_print_server</code>	URL of the print server. If not specified, the print server will be 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 1–15 describes the parameters available in the DOWNLOAD\_PRINT\_DOCUMENT function.

**Table 1–15** 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 will be 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 1–16 describes the parameters available in the `DOWNLOAD_PRINT_DOCUMENT` procedure for Signature 3.

**Table 1–16** *DOWNLOAD\_PRINT\_DOCUMENT Parameters*

Parameter	Description
<code>p_file_name</code>	Defines the filename of the print document
<code>p_content_disposition</code>	Specifies whether to download the print document or display inline ("attachment", "inline")
<code>p_application_id</code>	Defines the application ID of the report query
<code>p_report_query_name</code>	Name of the report query (stored under application's Shared Components)
<code>p_report_layout_name</code>	Name of the report layout (stored under application's Shared Components)
<code>p_report_layout_type</code>	Defines the report layout type, that is "xsl-fo" or "rtf"
<code>p_document_format</code>	Defines the document format, that is "pdf", "rtf", "xls", "htm", or "xml"
<code>p_print_server</code>	URL of the print server. If not specified, the print server will be 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 wish 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 wish 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 1–16 describes the parameters available in the DOWNLOAD\_PRINT\_DOCUMENT procedure for Signature 4.

**Table 1–17** 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, 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 will be 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 1–18](#) describes the parameters available in the `EDIT_USER` procedure.

**Table 1–18** *EDIT\_USER Parameters*

Parameter	Description
<code>p_user_id</code>	Numeric primary key of the user account
<code>p_user_name</code>	Alphanumeric name used for login. <b>See Also:</b> <a href="#">"SET_USERNAME Procedure"</a> on page 1-103
<code>p_first_name</code>	Informational. <b>See Also:</b> <a href="#">"SET_FIRST_NAME Procedure"</a> on page 1-95
<code>p_last_name</code>	Informational. <b>See Also:</b> <a href="#">"SET_LAST_NAME Procedure"</a> on page 1-96
<code>p_web_password</code>	Clear text password. If using this procedure to update the password for the user, values for both <code>p_web_password</code> and <code>p_new_password</code> must not be null and must be identical.
<code>p_new_password</code>	Clear text new password. If using this procedure to update the password for the user, values for both <code>p_web_password</code> and <code>p_new_password</code> must not be null and must be identical.
<code>p_email_address</code>	Informational. <b>See Also:</b> <a href="#">"SET_EMAIL Procedure"</a> on page 1-94



**Table 1–18 (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 1-70</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 1-31, <a href="#">"EXPIRE_WORKSPACE_ACCOUNT Procedure"</a> on page 1-32, <a href="#">"UNEXPIRE_END_USER_ACCOUNT Procedure"</a> on page 1-111, <a href="#">"UNEXPIRE_WORKSPACE_ACCOUNT Procedure"</a> on page 1-112</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 1-76, <a href="#">"UNLOCK_ACCOUNT Procedure"</a> on page 1-113</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 1-10</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 1-77</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 1-40

## 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 1–19 describes the parameters available in the END\_USER\_ACCOUNT\_DAYS\_LEFT function.

**Table 1–19** 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 will expire.

```
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);
        http.p('End User Account: '||c1.user_name||' will expire in '||l_days_
left||' days.');
```

**See Also:** ["EXPIRE\\_END\\_USER\\_ACCOUNT Procedure"](#) on page 1-31 and ["UNEXPIRE\\_END\\_USER\\_ACCOUNT Procedure"](#) on page 1-111

---

## 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 1–21](#) describes the parameters available in the `EXPIRE_END_USER_ACCOUNT` procedure.

**Table 1–20** *EXPIRE\_END\_USER\_ACCOUNT Parameters*

Parameter	Description
<code>p_user_name</code>	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 with respect to its use by end users to authenticate to developed applications, but it may also expire the account with respect to 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 1-111

---

## 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 1–21 describes the parameters available in the EXPIRE\_WORKSPACE\_ACCOUNT procedure.

**Table 1–21** 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 with respect to its use by developers or administrators to log in to a workspace, but it may also expire the account with respect to 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 1-112

---

## 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 1–22](#) describes the parameters available in the EXPORT\_USERS procedure.

**Table 1–22** EXPORT\_USERS Parameters

Parameter	Description
p_export_format	Indicates how rows in the export file will be 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 will be 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 1–23](#) describes the parameters available in the `FETCH_APP_ITEM` function.

**Table 1–23** *FETCH\_APP\_ITEM Parameters*

Parameter	Description
<code>p_item</code>	The name of an application-level item (not a page item) whose current value is to be fetched
<code>p_app</code>	The ID of the application that owns the item (leave null for the current application)
<code>p_session</code>	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 1–24 describes the parameters available in the `FETCH_USER` procedure for signature 1.

**Table 1–24** *Fetch\_User Parameters Signature 1*

Parameter	Description
<code>p_user_id</code>	Numeric primary key of the user account
<code>p_workspace</code>	The name of the workspace
<code>p_user_name</code>	Alphanumeric name used for login. <b>See Also:</b> " <a href="#">GET_USERNAME Function</a> " on page 1-71
<code>p_first_name</code>	Informational. <b>See Also:</b> " <a href="#">GET_FIRST_NAME Function</a> " on page 1-56
<code>p_last_name</code>	Informational. <b>See Also:</b> " <a href="#">GET_LAST_NAME Function</a> " on page 1-60
<code>p_web_password</code>	Obfuscated account password
<code>p_email_address</code>	Email address. <b>See Also:</b> " <a href="#">GET_EMAIL Function</a> " on page 1-52
<code>p_start_date</code>	Unused
<code>p_end_date</code>	Unused
<code>p_employee_id</code>	Unused
<code>p_allow_access_to_schemas</code>	A list of schemas assigned to the user's workspace to which user is restricted
<code>p_person_type</code>	Unused

**Table 1–24 (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 1-51
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 1-57 and " <a href="#">CURRENT_USER_IN_GROUP Function</a> " on page 1-19
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 1-70
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 1-26 and ["GET\\_CURRENT\\_USER\\_ID Function"](#) on page 1-50

## 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 1–25](#) describes the parameters available in the `FETCH_USER` procedure for signature 2.

**Table 1–25 Fetch\_User Parameters Signature 2**

Parameter	Description
<code>p_user_id</code>	Numeric primary key of the user account
<code>p_user_name</code>	Alphanumeric name used for login. <b>See Also:</b> " <a href="#">GET_USERNAME Function</a> " on page 1-71
<code>p_first_name</code>	Informational. <b>See Also:</b> " <a href="#">GET_FIRST_NAME Function</a> " on page 1-56
<code>p_last_name</code>	Informational. <b>See Also:</b> " <a href="#">GET_LAST_NAME Function</a> " on page 1-60
<code>p_email_address</code>	Email address. <b>See Also:</b> " <a href="#">GET_EMAIL Function</a> " on page 1-52
<code>p_groups</code>	List of groups of which user is a member. <b>See Also:</b> " <a href="#">GET_GROUPS_USER_BELONGS_TO Function</a> " on page 1-57 and " <a href="#">CURRENT_USER_IN_GROUP Function</a> " on page 1-19

**Table 1–25 (Cont.) Fetch\_User Parameters Signature 2**

Parameter	Description
p_developer_role	<p>Colon-separated list of developer roles. The following are acceptable values for this parameter:</p> <p>null - Indicates an end user (a user who can only authenticate to developed applications).</p> <p>CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL - Indicates a user with developer privilege.</p> <p>ADMIN:CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL - Indicates a user with full workspace administrator and developer privilege.</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><b>See Also:</b> "GET_USER_ROLES Function" on page 1-70</p>
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 1-26 and "GET\_CURRENT\_USER\_ID Function" on page 1-50

## 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 1–26 describes the parameters available in the `FETCH_USER` procedure.

**Table 1–26 Fetch\_User Parameters Signature 3**

Parameter	Description
<code>p_user_id</code>	Numeric primary key of the user account
<code>p_workspace</code>	The name of the workspace
<code>p_user_name</code>	Alphanumeric name used for login. <b>See Also:</b> <a href="#">"GET_USERNAME Function"</a> on page 1-71
<code>p_first_name</code>	Informational. <b>See Also:</b> <a href="#">"GET_FIRST_NAME Function"</a> on page 1-56
<code>p_last_name</code>	Informational. <b>See Also:</b> <a href="#">"GET_LAST_NAME Function"</a> on page 1-60
<code>p_web_password</code>	Obfuscated account password
<code>p_email_address</code>	Email address. <b>See Also:</b> <a href="#">"GET_EMAIL Function"</a> on page 1-52
<code>p_start_date</code>	Unused
<code>p_end_date</code>	Unused

**Table 1–26 (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 1-51
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 1-57 and <a href="#">"CURRENT_USER_IN_GROUP Function"</a> on page 1-19
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 1-70
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 1-30 and <a href="#">"WORKSPACE_ACCOUNT_DAYS_LEFT Function"</a> on page 1-116
p_account_locked	Locked/Unlocked indicator Y or N. <b>See Also:</b> <a href="#">"GET_ACCOUNT_LOCKED_STATUS Function"</a> on page 1-45
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 1-26 and ["GET\\_CURRENT\\_USER\\_ID Function"](#) on page 1-50



---

## 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 1-27](#) describes the parameters available in the `FIND_SECURITY_GROUP_ID` function.

**Table 1-27** *FIND\_SECURITY\_GROUP\_ID Parameters*

Parameter	Description
<code>p_workspace</code>	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 1–28](#) describes the parameters available in the `FIND_WORKSPACE` function.

**Table 1–28** *FIND\_WORKSPACE Parameters*

Parameter	Description
<code>p_security_group_id</code>	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 1–29](#) describes the parameters available in the GET\_ACCOUNT\_LOCKED\_STATUS function.

**Table 1–29** 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 1-76 and [UNLOCK\\_ACCOUNT Procedure](#) on page 1-113.

## 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 via the APIs.

### Syntax

```
APEX_UTIL.GET_ATTRIBUTE (  
    p_username                IN VARCHAR2,  
    p_attribute_number        IN NUMBER)  
RETURN VARCHAR2;
```

### Parameters

[Table 1–30](#) describes the parameters available in the GET\_ATTRIBUTE function.

**Table 1–30** 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 1-91

## 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 1-92 and ["SET\\_CUSTOM\\_AUTH\\_STATUS Procedure"](#) on page 1-93

## 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 1–31 describes the parameters available in `GET_BLOB_FILE_SRC` function.

**Table 1–31** *GET\_BLOB\_FILE\_SRC Parameters*

Parameter	Description
<code>p_item_name</code>	Name of valid application page ITEM that with type FILE that contains the source type of DB column.
<code>p_v1</code>	Value of primary key column 1.
<code>p_v2</code>	Value of primary key column 2.
<code>p_content_disposition</code>	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\_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\_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 1–32](#) describes the parameters available in GET\_EMAIL function.

**Table 1–32** 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 1-94

## 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 will be invoked under the same condition, as it will interfere 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 1–33 describes the parameters available in GET\_FILE procedure.

**Table 1–33** 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 will be 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 1-55

---

## 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 1–34](#) describes the parameters available in GET\_FILE\_ID function.

**Table 1–34** 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 1–35](#) describes the parameters available in `GET_FIRST_NAME` function.

**Table 1–35** *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 1-95

---

## 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 1–36](#) describes the parameters available in GET\_GROUPS\_USER\_BELONGS\_TO function.

**Table 1–36** 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 1-26

## 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 1–37](#) describes the parameters available in GET\_GROUP\_ID function.

**Table 1–37** GET\_GROUP\_ID Parameters

Parameter	Description
p_group_name	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 NUMBER;
```

### Parameters

[Table 1–38](#) describes the parameters available in GET\_GROUP\_NAME function.

**Table 1–38** 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\_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 1–39](#) describes the parameters available in `GET_LAST_NAME` function.

**Table 1–39** *GET\_LAST\_NAME Parameters*

Parameter	Description
<code>p_username</code>	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 1-96

---

## 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 1–40](#) describes the parameters available in GET\_NUMERIC\_SESSION\_STATE function.

**Table 1–40** 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 1-68 and ["SET\\_SESSION\\_STATE Procedure"](#) on page 1-102

---

## 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 1–41](#) describes the parameters available in the GET\_PREFERENCE function.

**Table 1–41** 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 1-97, ["REMOVE\\_PREFERENCE Procedure"](#) on page 1-84 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 1–42](#) describes the parameters available in the GET\_PRINT\_DOCUMENT function.

**Table 1–42** GET\_PRINT\_DOCUMENT 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 will be 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 1–43](#) describes the parameters available in the GET\_PRINT\_DOCUMENT function.

**Table 1–43** GET\_PRINT\_DOCUMENT 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 will be 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 1–44](#) describes the parameters available in the GET\_PRINT\_DOCUMENT function.

**Table 1–44** GET\_PRINT\_DOCUMENT 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 will be 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 1–45 describes the parameters available in the GET\_PRINT\_DOCUMENT function. for Signature 4

**Table 1–45 GET\_PRINT\_DOCUMENT 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 will be 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 in conjunction 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 via 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\_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 1-46](#) describes the parameters available in `GET_SESSION_STATE` function.

**Table 1-46** *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 1-61 and ["SET\\_SESSION\\_STATE Procedure"](#) on page 1-102

---

## 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 1-47](#) describes the parameters available in GET\_USER\_ID function.

**Table 1-47** 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 1–48](#) describes the parameters available in GET\_USER\_ROLES function.

**Table 1–48** 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 1–49](#) describes the parameters available in GET\_USERNAME function.

**Table 1–49** 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 1-103

## 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 1–50](#) describes the parameters available in the IS\_LOGIN\_PASSWORD\_VALID function.

**Table 1–50 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 will be returned if this is a valid password for 'FRANK', FALSE if not.

```
DECLARE  
    VAL BOOLEAN;  
BEGIN  
    VAL := APEX_UTIL.IS_LOGIN_PASSWORD_VALID (  
        p_username=>'FRANK',  
        p_password=>'tiger');  
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 1–51](#) describes the parameters available in IS\_USERNAME\_UNIQUE function.

**Table 1–51 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 will be 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 1-89



## 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 1-90

---

## 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 1–52](#) describes the parameters available in the LOCK\_ACCOUNT procedure.

**Table 1–52** 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 1-113 and ["GET\\_ACCOUNT\\_LOCKED\\_STATUS Function"](#) on page 1-45

## 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 1-53](#) describes the parameters available in the `PASSWORD_FIRST_USE_OCCURRED` procedure.

**Table 1-53** *PASSWORD\_FIRST\_USE\_OCCURRED Parameters*

Parameter	Description
<code>p_user_name</code>	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 1-10

## 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 in order 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)
RETURN VARCHAR2;
```

### Parameters

Table 1–54 describes the parameters available in the PREPARE\_URL function.

**Table 1–54** 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

### 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 will 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" and "Understanding Session State Protection" in *Oracle Application Express Advanced Tutorials*

## PUBLIC\_CHECK\_AUTHORIZATION Function

Given the name of a security 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 1–55](#) describes the parameters available in PUBLIC\_CHECK\_AUTHORIZATION function.

**Table 1–55 PUBLIC\_CHECK\_AUTHORIZATION Parameters**

Parameter	Description
p_security_name	The name of the security 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 1–56](#) describes the parameters available in `PURGE_REGIONS_BY_APP`.

**Table 1–56** *PURGE\_REGIONS\_BY\_APP Parameters*

Parameter	Description
<code>p_application</code>	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_UTILITIES.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 1–57](#) describes the parameters available in PURGE\_REGIONS\_BY\_NAME.

**Table 1–57** *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 1–58](#) describes the parameters available in PURGE\_REGIONS\_BY\_PAGE.

**Table 1–58** *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;
```

---

## 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 1–59](#) describes the parameters available in the REMOVE\_PREFERENCE procedure.

**Table 1–59 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 1-62, ["SET\\_PREFERENCE Procedure"](#) on page 1-97 and "Managing Session State and 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 1–60](#) describes the parameters available in REMOVE\_SORT\_PREFERENCES function.

**Table 1–60 REMOVE\_SORT\_PREFERENCES Parameters**

Parameter	Description
p_user	Identifies the user for whom sorting preferences will be 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 1–61](#) describes the parameters available in the REMOVE\_USER procedure.

**Table 1–61 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

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\_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 1–62](#) describes the parameters available in the `RESET_PW` procedure.

**Table 1–62** *RESET\_PW Parameters*

Parameter	Description
<code>p_user</code>	The user name of the user account
<code>p_msg</code>	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 1-9

---

## 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 1–63](#) describes the parameters available in the `SAVEKEY_NUM` procedure.

**Table 1–63** *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 1-74

## 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 1–64](#) describes the parameters available in the `SAVEKEY_VC2` function.

**Table 1–64** *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 1-75



---

## 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 1–65](#) describes the parameters available in the SET\_ATTRIBUTE procedure.

**Table 1–65** 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 1-46

## 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 1–21](#) describes the parameters available in the SET\_AUTHENTICATION\_RESULT procedure.

**Table 1–66 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 1-47 and "[SET\\_CUSTOM\\_AUTH\\_STATUS Procedure](#)" on page 1-93

## 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 1-67 describes the parameters available in the SET\_CUSTOM\_AUTH\_STATUS procedure.

**Table 1-67 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 1-92 and "GET\_AUTHENTICATION\_RESULT Function" on page 1-47

## 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 1–68](#) describes the parameters available in the `SET_EMAIL` procedure.

**Table 1–68** *SET\_EMAIL Parameters*

Parameter	Description
<code>p_userid</code>	The numeric ID of the user account
<code>p_email</code>	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@oracle.com' for the user 'FRANK'.

```
BEGIN  
    APEX_UTIL.SET_EMAIL(  
        p_userid => APEX_UTIL.GET_USER_ID('FRANK'),  
        p_email  => 'frank.scott@oracle.com');  
END;
```

**See Also:** ["GET\\_EMAIL Function"](#) on page 1-52 and ["GET\\_USER\\_ID Function"](#) on page 1-69

---

## 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 1-69](#) describes the parameters available in the `SET_FIRST_NAME` procedure.

**Table 1-69** *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 1-56 and ["GET\\_USER\\_ID Function"](#) on page 1-69

## 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 1-70](#) describes the parameters available in the `SET_LAST_NAME` procedure.

**Table 1-70** *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 1-60 and ["GET\\_USER\\_ID Function"](#) on page 1-69

---

## SET\_PREFERENCE Procedure

This procedure sets a preference that will persist 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 1-71](#) describes the parameters available in the SET\_PREFERENCE procedure.

**Table 1-71** 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 will persist 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 1-62 and ["REMOVE\\_PREFERENCE Procedure"](#) on page 1-84

---

## SET\_SESSION\_LIFETIME\_SECONDS Procedure

This procedure sets the current application's Maximum Session Length in Seconds value for the current session, overriding the corresponding application attribute. This allows developers to dynamically shorten or lengthen the session life based on criteria determined after the user authenticates.

---

**Note:** In order for this procedure to have any effect, the application's Maximum Session Length in Seconds attribute must have been set to a non-zero value in the application definition. This procedure will have no effect if that attribute was not set by the developer.

---

### Syntax

```
APEX_UTIL.SET_SESSION_LIFETIME_SECONDS (
    p_seconds IN    NUMBER,
    p_scope   IN    VARCHAR2 DEFAULT 'SESSION');
```

### Parameters

[Table 1–74](#) describes the parameters available in the SET\_SESSION\_LIFETIME\_SECONDS procedure.

**Table 1–72** 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	Defaults to 'SESSION' and may also be set to 'APPLICATION'. If 'SESSION', all applications using this session are affected. If 'APPLICATION', only the current application using the current session is affected.

### 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). This API call will have no effect if the application's Maximum Session Length in Seconds attribute was not set by the developer to a non-zero value in the application definition.

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). This API call will have no effect if the application's Maximum Session Length in Seconds attribute was not set by the developer to a non-zero value in the application definition.

By overriding the p\_scope input parameter's default value and setting it to 'APPLICATION', the following example would actually apply to only to the current application using the current session even if other applications are using the same session.

```
BEGIN
  APEX_UTIL.SET_SESSION_LIFETIME_SECONDS(p_seconds => 3600,
    p_scope => 'APPLICATION');
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.

---

**Note:** In order for this procedure to have any effect, the application's Maximum Session Idle Time in Seconds attribute must have been set to a non-zero value in the application definition. This procedure will have no effect if that attribute was not set by the developer.

---

### Syntax

```
APEX_UTIL.SET_SESSION_MAX_IDLE_SECONDS (
    p_seconds IN    NUMBER,
    p_scope   IN    VARCHAR2 DEFAULT 'SESSION');
```

### Parameters

[Table 1–73](#) describes the parameters available in the SET\_SESSION\_MAX\_IDLE\_SECONDS procedure.

**Table 1–73 SET\_SESSION\_MAX\_IDLE\_SECONDS Parameters**

Parameter	Description
p_seconds	A positive integer indicating the number of seconds allowed between page requests.
p_scope	Defaults to 'SESSION' and may also be set to 'APPLICATION'. If 'SESSION', this idle time applies to all applications using this session. If 'APPLICATION', this idle time only applies to the current application using the current 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). This API call will have no effect if the application's Maximum Session Idle Time in Seconds attribute was not set by the developer to a non-zero value in the application definition.

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_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 API call will have no effect if the application's Maximum Session Idle Time in Seconds attribute was not set by the developer to a non-zero value in the application definition.

By overriding the `p_scope` input parameter's default value and setting it to 'APPLICATION', the following example would actually apply to only to the current application using the current session even if other applications are using the same session.

```
BEGIN
  APEX_UTIL.SET_SESSION_MAX_IDLE_SECONDS(p_seconds => 600,
    p_scope => 'APPLICATION');
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);
```

### Parameters

[Table 1-74](#) describes the parameters available in the SET\_SESSION\_STATE procedure.

**Table 1-74** 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

### 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 1-68, ["GET\\_NUMERIC\\_SESSION\\_STATE Function"](#) on page 1-61, and ["Understanding Session State Management"](#) in *Oracle Application Express Application Builder User's Guide*

---

## 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 1-75](#) describes the parameters available in the `SET_USERNAME` procedure.

**Table 1-75** *SET\_USERNAME Parameters*

Parameter	Description
<code>p_userid</code>	The numeric ID of the user account
<code>p_username</code>	<code>USER_NAME</code> 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 1-71 and ["GET\\_USER\\_ID Function"](#) on page 1-69

## STRONG\_PASSWORD\_CHECK Procedure

This procedure returns `Boolean` `OUT` values based on whether or not 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 1–76 describes the parameters available in the `STRONG_PASSWORD_CHECK` procedure.

**Table 1–76** *STRONG\_PASSWORD\_CHECK Parameters*

Parameter	Description
<code>p_username</code>	Username that identifies the account in the current workspace
<code>p_password</code>	Password to be checked against password strength rules
<code>p_old_password</code>	Current password for the account. Used only to enforce "new password must differ from old" rule
<code>p_workspace_name</code>	Current workspace name, used only to enforce "password must not contain workspace name" rule
<code>p_use_strong_rules</code>	Pass <code>FALSE</code> when calling this API
<code>p_min_length_err</code>	Result returns <code>True</code> or <code>False</code> depending upon whether the password meets minimum length requirement
<code>p_new_differs_by_err</code>	Result returns <code>True</code> or <code>False</code> depending upon whether the password meets "new password must differ from old" requirements
<code>p_one_alpha_err</code>	Result returns <code>True</code> or <code>False</code> depending upon whether the password meets requirement to contain at least one alphabetic character
<code>p_one_numeric_err</code>	Result returns <code>True</code> or <code>False</code> depending upon whether the password meets requirements to contain at least one numeric character

**Table 1–76 (Cont.) STRONG\_PASSWORD\_CHECK Parameters**

Parameter	Description
p_one_punctuation_err	Result returns True or False depending upon whether the password meets requirements to contain at least one punctuation character
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:** "About Password Policies" in *Oracle Application Express Administration Guide*

## STRONG\_PASSWORD\_VALIDATION Function

This function returns formatted HTML in a VARCHAR2 result based on whether or not 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 1-77](#) describes the parameters available in the STRONG\_PASSWORD\_VALIDATION function.

**Table 1-77 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;
```

## 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 1-78](#) describes the parameters available in the `STRING_TO_TABLE` function.

**Table 1-78** *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 `HTTP.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
        http.p(l_vc_arr2(z));
    END LOOP;
END;
```

**See Also:** ["TABLE\\_TO\\_STRING Function"](#) on page 1-110

---

## 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 1-79](#) describes the parameters available in the `TABLE_TO_STRING` function.

**Table 1-79** *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 example shows how to use the `TABLE_TO_STRING` function. The example first calls `STRING_TO_TABLE` which is passed the string 'One:Two:Three' in the `p_string` parameter, and 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 array is then passed in to the `TABLE_TO_STRING` function in the `p_string` parameter, which then returns back the original string 'One:Two:Three'.

```
DECLARE
    l_string      VARCHAR2(255);
    l_vc_arr2     APEX_APPLICATION_GLOBAL.VC_ARR2;
BEGIN
    l_vc_arr2 := APEX_UTIL.STRING_TO_TABLE('One:Two:Three');
    l_string := APEX_UTIL.TABLE_TO_STRING(l_vc_arr2);
END;
```

**See Also:** ["STRING\\_TO\\_TABLE Function"](#) on page 1-109

---

## 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 1–80](#) describes the parameters available in the UNEXPIRE\_END\_USER\_ACCOUNT procedure.

**Table 1–80 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 1-31 and " [END\\_USER\\_ACCOUNT\\_DAYS\\_LEFT Function](#)" on page 1-30

## 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 1–81](#) describes the parameters available in the UNEXPIRE\_WORKSPACE\_ACCOUNT procedure.

**Table 1–81 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 with respect to 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 1-32 and ["WORKSPACE\\_ACCOUNT\\_DAYS\\_LEFT Function"](#) on page 1-116

---

## 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 1–82](#) describes the parameters available in the UNLOCK\_ACCOUNT procedure.

**Table 1–82 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 1-76 and ["GET\\_ACCOUNT\\_LOCKED\\_STATUS Function"](#) on page 1-45

## 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 1–83](#) describes the parameters available in the URL\_ENCODE function.

**Table 1–83 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 1–84](#) describes the parameters available in the WORKSPACE\_ACCOUNT\_DAYS\_LEFT procedure.

**Table 1–84** 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)
        http.p('Workspace Account: '||c1.user_name||' will expire in '||l_days_
left||' days.');
```

```
        END LOOP;
    END;
```

**See Also:** ["EXPIRE\\_WORKSPACE\\_ACCOUNT Procedure"](#) on page 1-32 and ["UNEXPIRE\\_WORKSPACE\\_ACCOUNT Procedure"](#) on page 1-112

---

---

## APEX\_MAIL

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 in order 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.

This section contains the following topics:

- [ADD\\_ATTACHMENT Procedure](#)
- [PUSH\\_QUEUE Procedure](#)
- [SEND Procedure](#)

---

---

**Note:** The most efficient approach to sending email is to create a background job (using a `DBMS_JOB` package) to periodically send 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*

## 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 2–1 describes the parameters available in the `ADD_ATTACHMENT` procedure.

**Table 2–1 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;
```

## 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. Keep in mind, the most efficient approach to sending email is to create a background job (using a `DBMS_JOB` package) to periodically send all mail messages stored in the active mail queue.

**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 2–2](#) describes the parameters available in the `PUSH_QUEUE` procedure.

**Table 2–2** *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 will result 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 will not display. 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 2-3](#) describes the parameters available in the `SEND` procedure.

**Table 2–3 SEND Parameters**

Parameter	Description
<code>p_to</code>	Valid email address to which the email will be sent (required). For multiple email addresses, use a comma-separated list
<code>p_from</code>	Email address from which the email will be sent (required). This email address must be a valid address. Otherwise, the message will not be 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 will send these messages, but the automatic replies will 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 APEX 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 APEX Dev Team</span><br
/>'||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_body_html => l_body_html,
        p_subj    => 'APEX_MAIL Package - HTML formatted message');
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.

This section contains the following topics:

- [CHECKBOX Function](#)
- [DATE\\_POPUP 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](#)

## CHECKBOX Function

This function creates check boxes.

### Syntax

```
APEX_ITEM.CHECKBOX (
    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 3–1 describes the parameters available in the CHECKBOX function.

**Table 3–1** CHECKBOX Parameters

Parameter	Description
p_idx	Number that determines which APEX_APPLICATION global variable will be 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.CHECKBOX(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.CHECKBOX(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.CHECKBOX(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.CHECKBOX(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 utilizes the following logic:

```
SELECT APEX_ITEM.CHECKBOX(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.CHECKBOX(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 3–2 describes the parameters available in the DATE\_POPUP function.

**Table 3–2 DATE\_POPUP Parameters**

Parameter	Description
p_idx	Number that determines which APEX_APPLICATION global variable will be 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 will be 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
```

---

## 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 3–3 describes the parameters available in the DISPLAY\_AND\_SAVE function.

**Table 3–3** DISPLAY\_AND\_SAVE Parameters

Parameter	Description
p_idx	Number that determines which APEX_APPLICATION global variable will be 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 3–4 describes the parameters available in the HIDDEN function.

**Table 3–4 HIDDEN Parameters**

Parameter	Description
p_idx	Number to identify the item you want to generate. The number will determine which G_FXX global is populated  <b>See Also:</b> "APEX_APPLICATION" on page 4-1
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

This function is used 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 includes 50 inputs. APEX\_ITEM.MD5\_CHECKSUM also produces an MD5 checksum using the Oracle database DBMS\_OBFUSCATION\_TOOLKIT:

```
UTL_RAW.CAST_TO_RAW(DBMS_OBFUSCATION_TOOLKIT.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 '|')
RETURN VARCHAR2;
```

### Parameters

[Table 3–5](#) describes the parameters available in the MD5\_CHECKSUM function.

**Table 3–5 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 ( )

### Example

This function generates hidden form elements with the name 'fcs'. The values can subsequently be accessed via the APEX\_APPLICATION.G\_FCS array.

```
SELECT APEX_ITEM.MD5_CHECKSUM(ename, job, sal) md5_cks,
       ename, job, sal
FROM emp
```

## MD5\_HIDDEN Function

This function is used 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 and includes 50 inputs. `APEX_ITEM.MD5_HIDDEN` also produces an MD5 checksum using the Oracle database `DBMS_OBFUSCATION_TOOLKIT`:

```
UTL_RAW.CAST_TO_RAW(DBMS_OBFUSCATION_TOOLKIT.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 '|' )
RETURN VARCHAR2;
```

### Parameters

[Table 3–6](#) describes the parameters available in the `MD5_HIDDEN` function.

**Table 3–6 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 ( )

### 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 will be 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 3–7](#) describes the some parameters in the `POPUP_FROM_LOV` function.

**Table 3–7** *POPUP\_FROM\_LOV Parameters*

Parameter	Description
<code>p_idx</code>	Form element name. For example, 1 equals F01 and 2 equals F02. Typically, <code>p_idx</code> is a constant for a given column
<code>p_value</code>	Form element current value. This value should be one of the values in the <code>p_lov_name</code> parameter
<code>p_lov_name</code>	Named LOV used for this popup
<code>p_width</code>	Width of the text box
<code>p_max_length</code>	Maximum number of characters that can be entered in the text box
<code>p_form_index</code>	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 Web site). If this form comes before the <code>#FORM_OPEN#</code> 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 3–7 (Cont.) POPUP\_FROM\_LOV 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 will be escaped. This parameter is useful if you know your query will return 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 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 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 3–8](#) describes the parameters in the `POPUP_FROM_QUERY` function.

**Table 3–8** *POPUP\_FROM\_QUERY Parameters*

Parameter	Description
<code>p_idx</code>	Form element name. For example, 1 equals F01 and 2 equals F02. Typically, <code>p_idx</code> is a constant for a given column.
<code>p_value</code>	Form element current value. This value should be one of the values in the <code>p_lov_query</code> parameter.
<code>p_lov_query</code>	SQL query that is expected to select two columns (a display column and a return column). For example: <code>SELECT dname, deptno FROM dept</code>
<code>p_width</code>	Width of the text box.
<code>p_max_length</code>	Maximum number of characters that can be entered in the text box.
<code>p_form_index</code>	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 Web site). If this form comes before the <code>#FORM_OPEN#</code> 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 3–8 (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 will be escaped. This parameter is useful if you know your query will return 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 3–9 describes the some parameters in the POPUPKEY\_FROM\_LOV function.

**Table 3–9 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 Web site). 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 3–9 (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 will return 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 3–10 describes the some parameters in the POPUPKEY\_FROM\_QUERY function.

**Table 3–10 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 Web site). 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 3–10 (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 will return 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 3–11](#) describes the parameters available in the RADIOGROUP function.

**Table 3–11** RADIOGROUP Parameters

Parameter	Description
p_idx	Number that determines which APEX_APPLICATION global variable will be 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 3–12 describes the parameters available in the `SELECT_LIST` function.

**Table 3–12** *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 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;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 will be generated (`p_idx` parameter).
- The initial value for each element will be equal to the value for `deptno` for the row from `emp` (`p_value` parameter).
- The select list will contain 4 options (`p_list_values` parameter).
- The text within the select list will display in red (`p_attributes` parameter).
- A null option will be displayed (`p_show_null`) and this option will display `-Select-` as the text (`p_null_text` parameter).
- An HTML ID attribute will be generated for each row, where `#ROWNUM#` will be substituted for the current row `rownum` (`p_item_id` parameter). (So an ID of `'f03_4'` will be generated for row 4.)
- A HTML label element will be generated for each row (`p_item_label` parameter).
- The current value for `deptno` will be 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 3–13](#) describes the parameters available in the `SELECT_LIST_FROM_LOV` function.

**Table 3–13** *SELECT\_LIST\_FROM\_LOV 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 an application 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 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. This enables you to use it 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 3–14](#) describes the parameters available in the `SELECT_LIST_FROM_LOV_XL` function.

**Table 3–14** *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 3–15](#) describes the parameters available in the SELECT\_LIST\_FROM\_QUERY function.

**Table 3–15** 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 will be 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 3–16](#) describes the parameters available in the `SELECT_LIST_FROM_QUERY_XL` function.

**Table 3–16** *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:  SELECT dname, deptno FROM dept  Note that this is used only by the <code>SELECT_LIST_FROM_QUERY_XL</code> function.  Also note, if only one column is specified in the select clause of this query, the value for this column will be used for both display and return purposes.
<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 <select> tag.
<code>p_item_label</code>	Invisible label created for the item.

**Table 3–16 (Cont.) SELECT\_LIST\_FROM\_QUERY\_XL Parameters**

Parameter	Description
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_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 3–17 describes the parameters available in the TEXT function.

**Table 3–17** TEXT Parameters

Parameter	Description
p_idx	Number to identify the item you want to generate. The number will determine which G_FXX global is populated. <b>See Also:</b> "APEX_APPLICATION" on page 4-1
p_value	Value of a text field item.
p_size	Controls HTML tag attributes (such as disabled).
p_maxlength	Maximum number of characters that can be entered in the text box.
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

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 DEAFULT 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 3–18](#) describes the parameters available in the TEXTAREA function.

**Table 3–18** TEXTAREA Parameters

Parameter	Description
p_idx	Number to identify the item you want to generate. The number will determine which G_FXX global is populated. <b>See Also:</b> "APEX_APPLICATION" on page 4-1
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 3–19](#) describes the parameters available in the TEXT\_FROM\_LOV function.

**Table 3–19** 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 3–20](#) describes the parameters available in the `TEXT_FROM_LOV_QUERY` function.

**Table 3–20** *TEXT\_FROM\_LOV\_QUERY Parameters*

Parameter	Description
<code>p_value</code>	Value of a field item.
<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> Note if only one column is specified in the select clause of this query, the value for this column will be used for both display and return purposes.
<code>p_null_text</code>	Value to be displayed when the value of the field item is NULL or a corresponding entry is not located for the value <code>p_value</code> 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\_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 a number of global variables. [Table 4–1](#) describes the global variables available in the APEX\_APPLICATION package.

**Table 4–1 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 will redirect to this location if no page is given and if no alternative page is dictated by the authentication scheme's logic.
G_LOGIN_URL	Can be 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 or not 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.

---

Topics in this section include:

- [Referencing Arrays](#)
- [Referencing Values Within an On Submit Process](#)
- [Converting an Array to a Single Value](#)
- [HELP Procedure](#)

## 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.

If you need 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` will 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 will only have 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 and can be used 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>:&nbsp; ',
  p_before_item_html IN VARCHAR2 DEFAULT NULL,
  p_after_item_html  IN VARCHAR2 DEFAULT NULL);
```

### Parameters

[Table 4–2](#) describes the parameters available in the HELP procedure.

**Table 4–2 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 will be '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 will be '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 4–2 (Cont.) HELP Parameters**

Parameter	Description
<code>p_after_prompt_html</code>	Use this parameter to include HTML after every item label for item level help. Note this parameter is ignored if <code>p_show_item_help</code> is set to 'NO'.
<code>p_before_item_html</code>	Use this parameter to include HTML before every item help text for item level help. Note this parameter is ignored if <code>p_show_item_help</code> is set to 'NO'.
<code>p_after_item_html</code>	Use this parameter to include HTML after every item help text for item level help. Note this parameter is ignored if <code>p_show_item_help</code> 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 will be used to control 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;');
```

In order 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`.





---

---

## APEX\_CUSTOM\_AUTH

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

Topics in this section include:

- 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
- 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 5–1](#) describes the parameters available in the `APPLICATION_PAGE_ITEM_EXISTS` function.

**Table 5–1** *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  
        htp.p('Item Exists');  
    ELSE  
        htp.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 5–2](#) describes the parameters available in the `DEFINE_USER_SESSION` procedure.

**Table 5–2** *DEFINE\_USER\_SESSION Parameters*

Parameter	Description
<code>p_user</code>	Login name of the user.
<code>p_session_id</code>	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 5-20 and ["SET\\_SESSION\\_ID Procedure"](#) on page 5-18.

---

## 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 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);
```

### Parameters

[Table 5–3](#) describes the parameters available in the GET\_COOKIE\_PROPS procedure.

**Table 5–3 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.

### Example

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

```
DECLARE
  l_cookie_name  varchar2(256);
  l_cookie_path  varchar2(256);
  l_cookie_domain varchar2(256);
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);
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_ldap_dn            OUT VARCHAR2,
    p_ldap_edit_function OUT VARCHAR2);
```

### Parameters

Table 5–4 describes the parameters available in the GET\_LDAP\_PROPS procedure.

**Table 5–4 GET\_LDAP\_PROPS Parameters**

Parameter	Description
p_ldap_host	LDAP host name.
p_ldap_port	LDAP port number.
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_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_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 the context of 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  
        htp.p('Valid');  
    ELSE  
        htp.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 5–5 describes the parameters available in the LOGIN procedure.

**Table 5–5** 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

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 5–6](#) describes the parameters available in the LOGOUT procedure.

**Table 5–6 LOGOUT Parameters**

Parameter	Description
p_this_app	Current application ID.
p_next_app_page_sess	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).
p_next_url	URL to redirect to (use this instead of p_next_app_page_sess).

### 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 5–7 describes the parameters available in the POST\_LOGIN procedure.

**Table 5–7 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  
        htp.p('Exists');  
    ELSE  
        htp.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 5–8](#) describes the parameters available in the SET\_SESSION\_ID procedure.

**Table 5–8** 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 5–9](#) describes the parameters available in the SET\_USER procedure.

**Table 5–9 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;
```

You can use APEX\_LDAP to perform various operations related to Lightweight Directory Access Protocol (LDAP) authentication.

Topics in this section include:

- [AUTHENTICATE Function](#)
- [GET\\_ALL\\_USER\\_ATTRIBUTES Procedure](#)
- [GET\\_USER\\_ATTRIBUTES Procedure](#)
- [IS\\_MEMBER Function](#)
- [MEMBER\\_OF Function](#)
- [MEMBER\\_OF2 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)
RETURN BOOLEAN;
```

### Parameters

Table 6–1 describes the parameters available in the `AUTHENTICATE` function.

**Table 6–1 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.

### 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_attributes    OUT wwv_flow_global.vc_arr2,
  p_attribute_values OUT wwv_flow_global.vc_arr2);
```

### Parameters

Table 6–2 describes the parameters for the `GET_ALL_USER_ATTRIBUTES` procedure.

**Table 6–2** *GET\_ALL\_USER\_ATTRIBUTES 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_attributes</code>	An array of attribute names returned.
<code>p_attribute_values</code>	An array of values returned for each corresponding attribute name returned in <code>p_attributes</code> .

### 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      wwv_flow_global.vc_arr2;
  L_ATTRIBUTE_VALUES wwv_flow_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
    htp.p('attribute name: ' || L_ATTRIBUTES(i));
    htp.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_attributes    IN wwv_flow_global.vc_arr2,
    p_attribute_values OUT wwv_flow_global.vc_arr2);
```

### Parameters

Table 6–3 describes the parameters available in the `GET_USER_ATTRIBUTES` procedure.

**Table 6–3** *GET\_USER\_ATTRIBUTES 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_attributes</code>	An array of attribute names for which values are to be returned.
<code>p_attribute_values</code>	An array of values returned for each corresponding attribute name in <code>p_attributes</code> .

### 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 wwv_flow_global.vc_arr2;
    L_ATTRIBUTE_VALUES wwv_flow_global.vc_arr2;
BEGIN
    L_ATTRIBUTES(1) := 'xxxxxxxxx'; /* 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_group         IN VARCHAR2,
  p_group_base    IN VARCHAR2)
RETURN BOOLEAN;
```

### Parameters

[Table 6–4](#) describes the parameters available in the `IS_MEMBER` function.

**Table 6–4 IS\_MEMBER 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_group</code>	Name of the group to be search for membership.
<code>p_group_base</code>	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
    http.p('Is a member.');
```

```
ELSE
  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)
RETURN wwv_flow_global.vc_arr2;
```

### Parameters

Table 6–5 describes the parameters available in the `MEMBER_OF` function.

**Table 6–5 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.

### 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    wwv_flow_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)
RETURN VARCHAR2;
```

### Parameters

Table 6–6 describes the parameters available in the `MEMBER_OF2` function.

**Table 6–6 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.

### 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);
    htp.p('Is Member of:' || L_VAL);
END;
```



---

---

## 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, wallet settings, report printing settings and to manage schema to workspace mappings. APEX\_INSTANCE\_ADMIN can be executed by the SYS, SYSTEM, and APEX\_030200 database users as well as any database user granted the role APEX\_ADMINISTRATOR\_ROLE.

Topics in this section include:

- [ADD\\_SCHEMA Procedure](#)
- [ADD\\_WORKSPACE Procedure](#)
- [GET\\_PARAMETER Function](#)
- [GET\\_SCHEMAS Function](#)
- [REMOVE\\_SAVED\\_REPORTS Procedure](#)
- [REMOVE\\_SCHEMA Procedure](#)
- [REMOVE\\_WORKSPACE Procedure](#)
- [SET\\_PARAMETER Procedure](#)
- [Available Parameter Values](#)

## 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 7-1](#) describes the parameters available in the ADD\_SCHEMA procedure.

**Table 7-1** ADD\_SCHEMA Parameters

Parameter	Description
p_workspace	The name of the workspace to which the schema mapping will be 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_primary_schema    IN VARCHAR2,
  p_additional_schemas IN VARCHAR2 );
```

### Parameters

[Table 7–2](#) describes the parameters available in the ADD\_WORKSPACE procedure.

**Table 7–2 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 will be assigned.
p_workspace	The name of the workspace to be added.
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.

### 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 (8675309, 'MY_WORKSPACE', 'SCOTT', 'HR:OE');
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 7-3](#) describes the parameters available in the GET\_PARAMETER function.

**Table 7-3** GET\_PARAMETER Parameters

Parameter	Description
p_parameter	The instance parameter to be retrieved. See " <a href="#">Available Parameter Values</a> " on page 7-10.

### 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 7-4](#) describes the parameters available in the GET\_SCHEMAS function.

**Table 7-4** 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;
```

---

## 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 7-5](#) describes the parameters available in the REMOVE\_SAVED\_REPORTS procedure.

**Table 7-5 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 will be 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 7-6](#) describes the parameters available in the REMOVE\_SCHEMA procedure.

**Table 7-6 REMOVE\_SCHEMA Parameters**

Parameter	Description
p_workspace	The name of the workspace from which the schema mapping will be 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\_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 7-7 describes the parameters available in the REMOVE\_WORKSPACE procedure.

**Table 7-7 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;
```

---

## SET\_PARAMETER Procedure

The SET\_PARAMETER procedure sets a parameter used in administering a runtime environment.

### Syntax

```
APEX_INSTANCE_ADMIN.SET_PARAMETER(  
    p_parameter    IN VARCHAR2,  
    p_value        IN VARCHAR2 DEFAULT 'N');
```

### Parameters

[Table 7-8](#) describes the parameters available in the SET\_PARAMETER procedure.

**Table 7-8** 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 7-10.

### 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.mycompany.com');  
END;
```

## Available Parameter Values

Table 7–9 lists all the available parameter values you can set within the `APEX_INSTANCE_ADMIN` package, including parameters for email, wallet, and reporting printing.

**Table 7–9 Available Parameters**

Parameter Name	Description
<code>SMTP_FROM</code>	<p>Defines the "from" address for administrative tasks that generate email, such as approving a provision request or resetting a password.</p> <p>Enter a valid email address, for example:</p> <p><code>someone@somewhere.com</code></p>
<code>SMTP_HOST_ADDRESS</code>	<p>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.</p> <p>Default setting:</p> <p><code>localhost</code></p>
<code>SMTP_HOST_PORT</code>	<p>Defines the port the SMTP server listens to for mail requests.</p> <p>Default setting:</p> <p><code>25</code></p>
<code>WALLET_PATH</code>	<p>The path to the wallet on the file system, for example:</p> <p><code>file:/home/&lt;username&gt;/wallets</code></p>
<code>WALLET_PWD</code>	The password associated with the wallet.
<code>PRINT_BIB_LICENSED</code>	<p>Specify either standard support or advanced support. Advanced support requires an Oracle BI Publisher license. Valid values include:</p> <ul style="list-style-type: none"> <li>■ <code>STANDARD</code></li> <li>■ <code>ADVANCED</code></li> </ul>
<code>PRINT_SVR_PROTOCOL</code>	<p>Valid values include:</p> <ul style="list-style-type: none"> <li>■ <code>http</code></li> <li>■ <code>https</code></li> </ul>
<code>PRINT_SVR_HOST</code>	<p>Specifies the host address of the print server converting engine, for example, <code>localhost</code>. Enter the appropriate host address if the print server is installed at another location.</p>
<code>PRINT_SVR_PORT</code>	<p>Defines the port of the print server engine, for example <code>8888</code>. Value must be a positive integer.</p>
<code>PRINT_SVR_SCRIPT</code>	<p>Defines the script that is the print server engine, for example:</p> <p><code>/xmlpsrver/convert</code></p>

**See Also:** "Configuring Email in a Runtime Environment", "Configuring a Wallet in a Runtime Environment", "Configuring Report Printing Settings in a Runtime Environment" in *Oracle Application Express Administration Guide*.





---

---

## 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.

Topics in this section include:

- [UPD\\_DISPLAY\\_IN\\_FORM Procedure](#)
- [UPD\\_DISPLAY\\_IN\\_REPORT Procedure](#)
- [UPD\\_FORM\\_REGION\\_TITLE 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](#)

**See Also:** "Managing User Interface Defaults" in *Oracle Application Express Application Builder User's Guide*

---

## UPD\_DISPLAY\_IN\_FORM Procedure

The `UPD_DISPLAY_IN_FORM` procedure sets the display in form user interface defaults. This user interface default will be used by wizards when you select to create a form based upon the table. It controls whether the column will be 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 8–1 describes the parameters available in the `UPD_DISPLAY_IN_FORM` procedure.

**Table 8–1** *UPD\_DISPLAY\_IN\_FORM Parameters*

Parameter	Description
<code>p_table_name</code>	Table name
<code>p_column_name</code>	Column name
<code>p_display_in_form</code>	Determines whether or not 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 would default 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 will be used by wizards when you select to create a report based upon the table and controls whether the column will be 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 8–2](#) describes the parameters available in the `UPD_DISPLAY_IN_REPORT` procedure.

**Table 8–2** *UPD\_DISPLAY\_IN\_REPORT Parameters*

Parameter	Description
<code>p_table_name</code>	Table name
<code>p_column_name</code>	Column name
<code>p_display_in_report</code>	Determines whether or not 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 would default 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 8–3 describes the parameters available in the UPD\_FORM\_REGION\_TITLE procedure.

**Table 8–3** APEX\_UI\_DEFAULT\_UPDATE 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\_ITEM\_DISPLAY\_HEIGHT Procedure

The `UPD_ITEM_DISPLAY_HEIGHT` procedure sets the item display height user interface default. This user interface default will be 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 will be 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 8–4 describes the parameters available in the `UPD_ITEM_DISPLAY_HEIGHT` procedure.

**Table 8–4** *UPD\_ITEM\_DISPLAY\_HEIGHT Parameters*

Parameter	Description
<code>p_table_name</code>	Table name
<code>p_column_name</code>	Column name
<code>p_display_height</code>	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 will be used by wizards when you select to create a form based upon the table and include the specified column.n.

### 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 8–5 describes the parameters available in the `UPD_ITEM_DISPLAY_WIDTH` procedure.

**Table 8–5** *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 will be 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 8–6](#) describes the parameters available in the `UPD_ITEM_FORMAT_MASK` procedure.

**Table 8–6** *UPD\_ITEM\_FORMAT\_MASK Parameters*

Parameter	Description
<code>p_table_name</code>	Table name
<code>p_column_name</code>	Column name
<code>p_format_mask</code>	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 will be 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 8–7](#) describes the parameters available in the `UPD_ITEM_HELP` procedure.

**Table 8–7** *UPD\_ITEM\_HELP Parameters*

Parameter	Description
<code>p_table_name</code>	Table name
<code>p_column_name</code>	Column name
<code>p_help_text</code>	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 will be used when you create a form or report based on the specified table and include a specific column.

### Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_ITEM_LABEL (
  p_table_name      IN VARCHAR2,
  p_column_name     IN VARCHAR2,
  p_label           IN VARCHAR2 DEFAULT NULL);
```

### Parameters

[Table 8–8](#) describes the parameters available in the UPD\_LABEL procedure.

**Table 8–8 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 will be 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 8–9 describes the parameters available in the UPD\_REPORT\_ALIGNMENT procedure.

**Table 8–9 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 will be 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 8–10 describes the parameters available in the UPD\_REPORT\_FORMAT\_MASK procedure.

**Table 8–10 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 8–11 describes the parameters available in the UPD\_REPORT\_REGION\_TITLE procedure.

**Table 8–11** 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');
```

---

---

## 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.

Topics in this section include:

- `$x(pNd)`
- `$v(pNd)`
- `$$s(pNd, pValue)`
- `$u_Carray(pNd)`
- `$u_Narray(pNd)`
- `$nvl(pTest, pDefault)`
- `doSubmit(pRequest)`
- `confirmDelete(pMessage, pRequest)`
- `$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)`
  - `$v_PopupReturn(pValue, pThat)`
  - `$x_ToggleWithImage(pThis,pNd)`
  - `$x_SwitchImageSrc(pNd, pSearch, pReplace)`
  - `$x_CheckImageSrc(pNd, pSearch)`
  - `$u_SubString(pText,pMatch)`
  - `html_RemoveAllChildren(pNd)`
  - `$v_IsEmpty(pThis)`
  - `html_SetSelectValue(pId,pValue)`
  - `addLoadEvent(pFunction)`
  - `$f_Swap(pThis,pThat)`
  - `submitEnter(pNd,e)`
  - `$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)`
  - `$v_Upper(pNd)`
  - `$d_Find(pThis,pString,pTags,pClass)`
  - `returnInput(p_R, p_D)`
  - `setReturn(p_R,p_D)`
  - `$f_First_field(pNd)`
  - `GetCookie (pName)`
  - `SetCookie (pName,pValue)`

## **\$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)

## **`$s(pNd, pValue)`**

Given a DOM node or string ID (pNd), this function sets the Application Express item value taking into account what type of item it is.

### **Parameters**

pNd (DOM Node | string ID)  
pValue (String | Array)

## \$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

## **\$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 will be returned otherwise the array will be 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

## **\$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)

---

## doSubmit(pRequest)

Submits the page setting the Application Express Request value (pRequest).

### Parameters

pRequest (String)

## confirmDelete(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)

## **\$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).

### **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).

### **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.

### **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.

### **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 will be the item and its label.

### **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 will be the item and its label.

### **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 will be the item and its label.

### **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 will be all be 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` (Array)

`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 an 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 an 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\_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 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\_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 checkbox (pNd), useful for tabular forms.

### Return Value

DOM node Array

### Parameters

pNd (DOM node | String)

## **`$v_PopupReturn(pValue, pThat)`**

Sets the value of the item in the parent window (`pThat`), with (`pValue`) and then closes the popup window.

### **Return Value**

Not applicable.

### **Parameters**

`pValue` (string)

`pThat` (DOM node | string ID)



## **\$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`) will be rewritten. The image `src` will have any plus substrings replaced with minus substrings or minus substrings will be 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 | false)

### **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)

---

## **\$v\_IsEmpty(pThis)**

Returns true or false if a form element is empty, this will consider any whitespace including a space, a tab, a form-feed, as empty.

### **Return Value**

[true | false]

### **Parameters**

pThis (DOM Node | String)

## html\_SetSelectValue(pId,pValue)

Sets the value (`pValue`) of a select item (`pId`). If the value is not found, this function 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)

---

## submitEnter(pNd,e)

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)"`

### Return Value

Not applicable.

### Parameters

pNd (DOM node | String | Array)

## **\$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 will be 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)

## returnInput(p\_R, p\_D)

Sets DOM node in the global variables `returnInput` (`p_R`) and `returnDisplay` (`p_D`) for use in populating items from popups.

### Return Value

Not applicable.

### Parameters

`p_R` (DOM node | String)

`p_D` (DOM node | String)

---

## setReturn(p\_R,p\_D)

Sets DOM items in the global variables `returnInput` (p\_R) and `returnDisplay` (p\_D) for use in populating items from popups.

### Return Value

Not applicable.

### Parameters

p\_R

p\_D

## **\$f\_First\_field(pNd)**

Places the user focus on the 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

---

## GetCookie (pName)

Returns the value of cookie name (pName).

### Return Value

Not applicable.

### Parameters

pName (String)



## SetCookie (pName,pValue)

Sets a cookie (pName) to a specified value (pValue).

### Return Value

Not applicable.

### Parameters

pName (String)

pValue (String)



# 10

---

---

## APEX\_PLSQL\_JOB

You can use APEX\_PLSQL\_JOB package to run PL/SQL code in the background of your application. This is an effective approach for managing long running operations that do not need to complete for a user to continue working with your application.

Topics in this section include:

- [JOBS\\_ARE\\_ENABLED Function](#)
- [PURGE\\_PROCESS Procedure](#)
- [SUBMIT\\_PROCESS Function](#)
- [TIME\\_ELAPSED Function](#)
- [UPDATE\\_JOB\\_STATUS Procedure](#)

## JOBS\_ARE\_ENABLED Function

Call this function to determine whether or not the database is currently in a mode that supports submitting jobs to the APEX\_PLSQL\_JOB package.

### Syntax

```
APEX_PLSQL_JOB.JOBS_ARE_ENABLED  
RETURN BOOLEAN;
```

### Parameters

None.

### Example

The following example shows how to use the JOBS\_ARE\_ENABLED function. In the example, if the function returns TRUE the message 'Jobs are enabled on this database instance' is displayed, otherwise the message 'Jobs are not enabled on this database instance' is displayed.

```
BEGIN  
  IF APEX_PLSQL_JOB.JOBS_ARE_ENABLED THEN  
    HTP.P('Jobs are enabled on this database instance.');
```

```
  ELSE  
    HTP.P('Jobs are not enabled on this database instance.');
```

```
  END IF;  
END;
```

---

## PURGE\_PROCESS Procedure

Call this procedure to clean up submitted jobs. Submitted jobs stay in the APEX\_PLSQL\_JOBS view until either Oracle Application Express cleans out those records, or you call PURGE\_PROCESS to manually remove them.

### Syntax

```
APEX_PLSQL_JOB.PURGE_PROCESS (  
    p_job IN NUMBER);
```

### Parameters

[Table 10–1](#) describes the parameters available in the PURGE\_PROCESS procedure.

**Table 10–1** PURGE\_PROCESS Parameters

Parameter	Description
p_job	The job number that identifies the submitted job you wish to purge.

### Example

The following example shows how to use the PURGE\_PROCESS procedure to purge the submitted job identified by a job number of 161. You could also choose to purge all or some of the current submitted jobs by referencing the APEX\_PLSQL\_JOBS view.

```
BEGIN  
    APEX_PLSQL_JOB.PURGE_PROCESS (  
        p_job => 161);  
END;
```

## SUBMIT\_PROCESS Function

Use this procedure to submit background PL/SQL. This procedure returns a unique job number. Because you can use this job number as a reference point for other procedures and functions in this package, it may be useful to store it in your own schema.

### Syntax

```
APEX_PLSQL_JOB.SUBMIT_PROCESS (
    p_sql IN VARCHAR2,
    p_when IN DATE DEFAULT SYSDATE,
    p_status IN VARCHAR2 DEFAULT 'PENDING')
RETURN NUMBER;
```

### Parameters

Table 10–2 describes the parameters available in the SUBMIT\_PROCESS function.

**Table 10–2** SUBMIT\_PROCESS Parameters

Parameter	Description
p_sql	The process you wish to run in your job. This can be any valid anonymous block, for example:  'BEGIN <your code> END;' or 'DECLARE <your declaration> BEGIN <your code> END;'
p_when	When you want to run it. The default is SYSDATE which means the job will run as soon as possible. You can also set the job to run in the future, for example:  sysdate + 1 - The job will run in 1 days time. sysdate + (1/24) - The job will run in 1 hours time. sysdate + (10/24/60) - The job will run in 10 minutes time.
p_status	Plain text status information for this job.

### Example

The following example shows how to use the SUBMIT\_PROCESS function to submit a background process that will start as soon as possible.

```
DECLARE
    l_sql VARCHAR2(4000);
    l_job NUMBER;
BEGIN
    l_sql := 'BEGIN MY_PACKAGE.MY_PROCESS; END;';
    l_job := APEX_PLSQL_JOB.SUBMIT_PROCESS(
        p_sql => l_sql,
        p_status => 'Background process submitted');
    --store l_job for later reference
END;
```

---

## TIME\_ELAPSED Function

Use this function to determine how much time has elapsed since the job was submitted.

### Syntax

```
APEX_PLSQL_JOB.TIME_ELAPSED(  
    p_job IN NUMBER)  
RETURN NUMBER;
```

### Parameters

[Table 10–3](#) describes the parameters available in the `TIME_ELAPSED` function.

**Table 10–3** *TIME\_ELAPSED Parameters*

Parameter	Description
<code>p_job</code>	The job ID for the job you wish to see how long since it was submitted.

### Example

The following example shows how to use the `TIME_ELAPSED` function to get the time elapsed for the submitted job identified by the job number 161.

```
DECLARE  
    l_time NUMBER;  
BEGIN  
    l_time := APEX_PLSQL_JOB.TIME_ELAPSED(p_job => 161);  
END;
```

## UPDATE\_JOB\_STATUS Procedure

Call this procedure to update the status of the currently running job. This procedure is most effective when called from the submitted PL/SQL.

### Syntax

```
APEX_PLSQL_JOB.UPDATE_JOB_STATUS (
    p_job IN NUMBER,
    p_status IN VARCHAR2);
```

### Parameters

[Table 10–4](#) describes the parameters available in the UPDATE\_JOB\_STATUS procedure.

**Table 10–4** UPDATE\_JOB\_STATUS Parameters

Parameter	Description
p_job	Passed the reserved word JOB. When this code is executed it will have visibility to the job number via the reserved word JOB.
p_status	Plain text that you want associated with JOB: p_job.

### Example

The following example shows how to use the UPDATE\_JOB\_STATUS procedure. In this example, note that:

- Lines 002 to 010 run a loop that inserts 100 records into the emp table.
- APP\_JOB is referenced as a bind variable inside the VALUES clause of the INSERT, and specified as the p\_job parameter value in the call to UPDATE\_JOB\_STATUS.
- APP\_JOB represents the job number which will be assigned to this process as it is submitted to APEX\_PLSQL\_JOB. By specifying this reserved item inside your process code, it will be replaced for you at execution time with the actual job number.
- Note that this example calls to UPDATE\_JOB\_STATUS every ten records, inside the block of code. Normally, Oracle transaction rules dictate updates made inside code blocks will not be seen until the entire transaction is committed. The APEX\_PLSQL\_JOB.UPDATE\_JOB\_STATUS procedure, however, has been implemented in such a way that the update will happen regardless of whether or not the job succeeds or fails. This last point is important for two reasons:
  1. Even if your status shows "100 rows inserted", it does not mean the entire operation was successful. If an error occurred at the time the block of code tried to commit, the user\_status column of APEX\_PLSQL\_JOBS would not be affected because status updates are committed separately.
  2. Updates are performed autonomously. You can view the job status before the job has completed. This gives you the ability to display status text about ongoing operations in the background as they are happening.

```
BEGIN
    FOR i IN 1 .. 100 LOOP
        INSERT INTO emp(a,b) VALUES (:APP_JOB,i);
```



```
IF MOD(i,10) = 0 THEN
    APEX_PLSQL_JOB.UPDATE_JOB_STATUS(
        P_JOB => :APP_JOB,
        P_STATUS => i || ' rows inserted');
END IF;
APEX_UTIL.PAUSE(2);
END LOOP;
END;
```



You can use APEX\_LANG API to translate messages.

Topics in this section include:

- [LANG Function](#)
- [MESSAGE Function](#)

## LANG Function

This function is used 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 11–1](#) describes the parameters available in the `APEX_LANG.LANG` function.

**Table 11–1 LANG Parameters**

Parameter	Description
<code>p_primary_text_string</code>	Text string of the primary language. This will be the value of the Translate From Text in the dynamic translation.
<code>p_p0</code> through <code>p_p9</code>	Dynamic substitution value: <code>p0</code> corresponds to 0% in the translation string; <code>p1</code> corresponds to 1% in the translation string; <code>p2</code> corresponds to 2% in the translation string, and so on.
<code>p_primary_language</code>	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 11–2](#) describes the parameters available in the `APEX_LANG.MESSAGE` function.

**Table 11–2** MESSAGE Parameters

Parameter	Description
<code>p_name</code>	Name of the message as defined in Shared Components > Text Messages of your application in Oracle Application Express.
<code>p_p0</code> through <code>p_p9</code>	Dynamic substitution value: <code>p0</code> corresponds to 0% in the translation string; <code>p1</code> corresponds to 1% in the translation string; <code>p2</code> corresponds to 2% in the translation string, and so on.
<code>p_lang</code>	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
--
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 will revert 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-us`.

**See also:** Specifying the Primary Language for an Application in the *Oracle Application Express Application Builder User's Guide*.

---

---

# Index

## A

---

- APEX\_APPLICATION
  - global variables, 4-1
  - HELP Procedure, 4-6
  - package, 4-1
  - Referencing Arrays
    - referencing, 4-3
- APEX\_CUSTOM\_AUTH, 5-1
- APPLICATION\_PAGE\_ITEM\_EXISTS
  - function, 5-2
- CURRENT\_PAGE\_IS\_PUBLIC function, 5-3
- DEFINE\_USER\_SESSION procedure, 5-4
- GET\_COOKIE\_PROPS, 5-5
- GET\_LDAP\_PROPS, 5-6
- GET\_NEXT\_SESSION\_ID function, 5-7
- GET\_SECURITY\_GROUP\_ID function, 5-8
- GET\_SESSION\_ID function, 5-9
- GET\_SESSION\_ID\_FROM\_COOKIE, 5-10
- GET\_USER function, 5-11
- GET\_USERNAME, 5-12
- IS\_SESSION\_VALID, 5-13
- LOGIN
  - Login API, 5-14
- LOGOUT, 5-15
- POST\_LOGIN, 5-16
- SESSION\_ID\_EXISTS function, 5-17
- SET\_SESSION\_ID procedure, 5-18
- SET\_SESSION\_ID\_TO\_NEXT\_VALUE
  - procedure, 5-19
- SET\_USER procedure, 5-20
- APEX\_INSTANCE\_ADMIN, 7-1
  - ADD\_SCHEMA procedure, 7-2
  - ADD\_WORKSPACE procedure, 7-3
  - GET\_PARAMETER function, 7-4
  - GET\_SCHEMAS function, 7-5
    - parameter values, 7-10
  - REMOVE\_SAVED\_REPORTS procedure, 7-6
  - REMOVE\_SCHEMA procedure, 7-7
  - REMOVE\_WORKSPACE procedure, 7-8
  - SET\_PARAMETER procedure, 7-9
- APEX\_ITEM, 3-1
  - CHECKBOX function, 3-2
  - DATE\_POPUP function, 3-4
  - DISPLAY\_AND\_SAVE, 3-6
  - HIDDEN function, 3-7
  - MD5\_CHECKSUM function, 3-9
  - MD5\_HIDDEN function, 3-10
  - POPUP\_FROM\_LOV function, 3-11
  - POPUP\_FROM\_QUERY function, 3-13
  - POPUPKEY\_FROM\_LOV function, 3-15
  - POPUPKEY\_FROM\_QUERY function, 3-17
  - RADIOGROUP function, 3-19
  - SELECT\_LIST function, 3-20
  - SELECT\_LIST\_FROM\_LOV function, 3-22
  - SELECT\_LIST\_FROM\_LOV\_XL function, 3-23
  - SELECT\_LIST\_FROM\_QUERY function, 3-25
  - SELECT\_LIST\_FROM\_QUERY\_XL function, 3-27
  - TEXT function, 3-29
  - TEXT\_FROM\_LOV function, 3-31
  - TEXT\_FROM\_LOV\_QUERY function, 3-32
  - TEXTAREA function, 3-30
- APEX\_LANG
  - LANG Function, 11-2, 11-3
- APEX\_LANG APIs
  - APEX\_LANG, 0-xii, 11-1
- APEX\_LDAP
  - AUTHENTICATE, 6-2
  - GET\_ALL\_USER\_ATTRIBUTES, 6-3
  - GET\_USER\_ATTRIBUTES, 6-4
  - IS\_MEMBER, 6-5
  - MEMBER\_OF, 6-6
  - MEMBER\_OF2 Function, 6-7
- APEX\_LDAP APIs
  - APEX\_LDAP, 6-1
- APEX\_MAIL, 2-1
  - ADD\_ATTACHMENT procedure, 2-2
  - PUSH\_QUEUE procedure, 2-3
  - SEND procedure, 2-4
- APEX\_MAIL\_QUEUE
  - sending email in queue, 2-3
- APEX\_PLSQL\_JOB APIs
  - APEX\_PLSQL\_JOB, 0-xii, 10-1
- APEX\_PLSQL\_JOBS
  - JOBS\_ARE\_ENABLED Function, 10-2
  - PURGE\_PROCESS Procedure, 10-3
  - SUBMIT\_PROCESS Function, 10-4
  - TIME\_ELAPSED Function, 10-5
  - UPDATE\_JOB\_STATUS Procedure, 10-6
- APEX\_UI\_DEFAULT\_UPDATE, 8-1
  - UPD\_DISPLAY\_IN\_FORM procedure, 8-2
  - UPD\_DISPLAY\_IN\_REPORT procedure, 8-3

UPD\_FORM\_REGION\_TITLE procedure, 8-4  
 UPD\_ITEM\_DISPLAY\_HEIGHT procedure, 8-5  
 UPD\_ITEM\_DISPLAY\_WIDTH procedure, 8-6  
 UPD\_ITEM\_FORMAT\_MASK procedure, 8-7  
 UPD\_ITEM\_HELP procedure, 8-8  
 UPD\_ITEM\_LABEL procedure, 8-9  
 UPD\_REPORT\_ALIGNMENT procedure, 8-10  
 UPD\_REPORT\_FORMAT\_MASK  
     procedure, 8-11  
 UPD\_REPORT\_REGION\_TITLE procedure, 8-12  
 APEX\_UTIL, 1-1  
 CACHE\_GET\_DATE\_OF\_PAGE\_CACHE  
     function, 1-4, 1-5  
 CACHE\_PURGE\_BY\_APPLICATION  
     procedure, 1-6  
 CACHE\_PURGE\_BY\_PAGE procedure, 1-7  
 CACHE\_PURGE\_STALE procedure, 1-8  
 CHANGE\_CURRENT\_USER\_PW procedure, 1-9  
 CHANGE\_PASSWORD\_ON\_FIRST\_USE  
     function, 1-10  
 CLEAR\_APP\_CACHE procedure, 1-11  
 CLEAR\_PAGE\_CACHE procedure, 1-12  
 CLEAR\_USER\_CACHE procedure, 1-13  
 COUNT\_CLICK procedure, 1-14  
 CREATE\_USER procedure, 1-15  
 CREATE\_USER\_GROUP procedure, 1-18  
 CURRENT\_USER\_IN\_GROUP function, 1-19  
 DOWNLOAD\_PRINT\_DOCUMENT procedure  
     signature 1, 1-20  
 DOWNLOAD\_PRINT\_DOCUMENT procedure  
     signature 2, 1-21  
 DOWNLOAD\_PRINT\_DOCUMENT procedure  
     signature 3, 1-23  
 DOWNLOAD\_PRINT\_DOCUMENT procedure  
     signature 4, 1-25  
 EDIT\_USER procedure, 1-26  
 END\_USER\_ACCOUNT\_DAYS\_LEFT  
     function, 1-30  
 EXPIRE\_END\_USER\_ACCOUNT  
     procedure, 1-31  
 EXPIRE\_WORKSPACE\_ACCOUNT  
     procedure, 1-32  
 EXPORT\_USERS procedure, 1-33  
 FETCH\_APP\_ITEM function, 1-34  
 FETCH\_USER procedure signature 1, 1-35  
 FETCH\_USER procedure signature 2, 1-38  
 FETCH\_USER procedure signature 3, 1-40  
 FIND\_SECURITY\_GROUP\_ID function, 1-43  
 FIND\_WORKSPACE function, 1-44  
 GET\_ACCOUNT\_LOCKED\_STATUS  
     function, 1-45  
 GET\_ATTRIBUTE function, 1-46  
 GET\_AUTHENTICATION\_RESULT  
     function, 1-47  
 GET\_BLOB\_FILE\_SRC function, 1-48  
 GET\_CURRENT\_USER\_ID function, 1-50  
 GET\_DEFAULT\_SCHEMA function, 1-51  
 GET\_EMAIL function, 1-52  
 GET\_FILE procedure, 1-53  
 GET\_FILE\_ID function, 1-55  
 GET\_FIRST\_NAME function, 1-56  
 GET\_GROUP\_ID function, 1-58  
 GET\_GROUP\_NAME function, 1-59  
 GET\_GROUPS\_USER\_BELONGS\_TO  
     function, 1-57  
 GET\_LAST\_NAME function, 1-60  
 GET\_NUMERIC\_SESSION\_STATE function, 1-61  
 GET\_PREFERENCE function, 1-62  
 GET\_PRINT\_DOCUMENT function signature  
     1, 1-63  
 GET\_PRINT\_DOCUMENT function signature  
     2, 1-64  
 GET\_PRINT\_DOCUMENT function signature  
     3, 1-65  
 GET\_PRINT\_DOCUMENT function signature  
     4, 1-66  
 GET\_SESSION\_STATE function, 1-68  
 GET\_USER\_ID function, 1-69  
 GET\_USER\_ROLES function, 1-70  
 GET\_USERNAME function, 1-71  
 IS\_LOGIN\_PASSWORD\_VALID function, 1-72  
 IS\_USERNAME\_UNIQUE function, 1-73  
 KEYVAL\_NUM function, 1-74  
 KEYVAL\_VC2 function, 1-75  
 LOCK\_ACCOUNT procedure, 1-76, 1-113  
 PASSWORD\_FIRST\_USE\_OCCURRED  
     function, 1-77  
 PREPARE\_URL function, 1-78  
 PUBLIC\_CHECK\_AUTHORIZATION  
     function, 1-80  
 PURGE\_REGIONS\_BY\_APP procedure, 1-81  
 PURGE\_REGIONS\_BY\_NAME procedure, 1-82  
 PURGE\_REGIONS\_BY\_PAGE procedure, 1-83  
 REMOVE\_PREFERENCE procedure, 1-84  
 REMOVE\_SORT\_PREFERENCES  
     procedure, 1-85  
 REMOVE\_USER procedure, 1-86  
 RESET\_AUTHORIZATIONS procedure, 1-87  
 RESET\_PW procedure, 1-88  
 SAVEKEY\_NUM function, 1-89  
 SAVEKEY\_VC2 function, 1-90  
 SET\_ATTRIBUTE procedure, 1-91  
 SET\_AUTHENTICATION\_RESULT  
     procedure, 1-92  
 SET\_CUSTOM\_AUTH\_STATUS procedure, 1-93  
 SET\_EMAIL procedure, 1-94  
 SET\_FIRST\_NAME procedure, 1-95  
 SET\_LAST\_NAME procedure, 1-96  
 SET\_PREFERENCE procedure, 1-97  
 SET\_SESSION\_STATE procedure, 1-98, 1-100,  
     1-102  
 SET\_USERNAME procedure, 1-103  
 STRING\_TO\_TABLE function, 1-109  
 STRONG\_PASSWORD\_CHECK  
     procedure, 1-104  
 STRONG\_PASSWORD\_VALIDATION  
     function, 1-108  
 TABLE\_TO\_STRING function, 1-110  
 UNEXPIRE\_END\_USER\_ACCOUNT  
     procedure, 1-111



UNEXPIRE\_WORKSPACE\_ACCOUNT  
     procedure, 1-112  
 URL\_ENCODE function, 1-114  
 WORKSPACE\_ACCOUNT\_DAYS\_LEFT  
     function, 1-116

**APIs**  
 APEX\_APPLICATION, 4-1  
 APEX\_CUSTOM\_AUTH, 5-1  
 APEX\_ITEM, 3-1  
 APEX\_MAIL, 2-1  
 APEX\_UTIL, 1-1

**application**  
   sending messages in APEX\_MAIL\_QUEUE, 2-3  
   sending outbound email, 2-4  
   sending outbound email as attachment, 2-2

**attribute values**  
   setting, 1-91

**authenticated user**  
   create user group, 1-18

**authentication**  
   scheme session cookies, 5-5

---

**C**

**check box**  
   creating, 3-2

**clicks**  
   counting, 1-14

---

**E**

**email**  
   sending as an attachment, 2-2  
   sending messages in APEX\_MAIL\_QUEUE, 2-3  
   sending outbound, 2-4

**export file**  
   of workspace, 1-33

---

**F**

F01, 4-3

**file repository**  
   downloading files, 1-53  
   obtaining primary key, 1-55

---

**J**

**JavaScript API**  
   \$d\_ClearAndHide(pNd), 9-36  
   \$d\_Find(pThis,pString,pTags,pClass), 9-72  
   \$dom\_  
     AddInput(pThis,pType,pId,pName,pValue),  
       9-67  
   \$dom\_AddTag(pThis, pTag, pText), 9-64  
   \$dom\_MakeParent(p\_Node,p\_Parent), 9-68  
   \$f\_CheckAll(pThis, pCheck, pArray), 9-50  
   \$f\_CheckFirstColumn(pNd), 9-51  
   \$f\_DisableOnValue(pThis, pValue, pThat), 9-46  
   \$f\_First\_field(pNd), 9-75  
   \$f\_get\_emptyys(pNd, pClassFail, pClass), 9-33  
   \$f\_Hide\_On\_Value\_Item(pThis, pThat,  
     pValue), 9-42  
   \$f\_Hide\_On\_Value\_Item\_Row(pThis, pThat,  
     pValue), 9-44  
   \$f\_ReturnChecked(pNd), 9-35  
   \$f\_SelectedOptions(pNd), 9-37  
   \$f\_SelectValue(pNd), 9-38  
   \$f\_SetValueSequence(pArray,pMultiple), 9-63  
   \$f\_Show\_On\_Value\_Item(pThis, pThat,  
     pValue), 9-43  
   \$f\_Show\_On\_Value\_Item\_Row(pThis, pThat,  
     pValue), 9-45  
   \$f\_Swap(pThis,pThat), 9-61  
   \$f\_ValuesToArray(pThis, pClass, pTag), 9-48  
   \$nvI(pTest, pDefault), 9-9  
   \$s(pNd, pValue), 9-6  
   \$str\_AddTD(pThis,pText), 9-65  
   \$str\_AddTH(pThis,pText), 9-66  
   \$u\_ArrayToString(pArray, pDelim), 9-39  
   \$u\_Carray(pNd), 9-7  
   \$u\_Narray(pNd), 9-8  
   \$u\_SubString(pText,pMatch), 9-56  
   \$v(pNd), 9-5  
   \$v\_Array(pNd), 9-34  
   \$v\_CheckValueAgainst(pThis, pValue), 9-41  
   \$v\_IsEmpty(pThis), 9-58  
   \$v\_PopupReturn(pValue, pThat), 9-52  
   \$v\_Upper(pNd), 9-71  
   \$x(pNd), 9-4  
   \$x\_ByClass(pClass, pNd, pTag), 9-28  
   \$x\_CheckImageSrc(pId,pSearch), 9-40  
   \$x\_CheckImageSrc(pNd, pSearch), 9-55  
   \$x\_Class(pNd,pClass), 9-26  
   \$x\_ClassByClass(pNd, pClass, pTag,  
     pClass2), 9-47  
   \$x\_disableItem(pNd,a), 9-32  
   \$x\_FormItems(pNd, pType), 9-49  
   \$x\_Hide(pNd), 9-13  
   \$x\_HideAllExcept(pNd,pNdArray), 9-23  
   \$x\_HideItemRow(pNd), 9-20  
   \$x\_HideSiblings(pNd), 9-24  
   \$x\_ItemRow(pNd,pFunc), 9-19  
   \$x\_Remove(pNd), 9-16  
   \$x\_RowHighlight(pThis,pColor), 9-69  
   \$x\_RowHighlightOff(pThis), 9-70  
   \$x\_SetSiblingsClass(pNd, pClass,  
     pNdClass), 9-27  
   \$x\_Show(pNd), 9-14  
   \$x\_ShowAllByClass(pNd, pClass, pTag), 9-29  
   \$x\_ShowChildren(pThis), 9-30,9-31  
   \$x\_ShowItemRow(pNd), 9-21  
   \$x\_ShowSiblings(pNd), 9-25  
   \$x\_Style(pNd, pStyle, pString), 9-12  
   \$x\_SwitchImageSrc(pNd, pSearch,  
     pReplace), 9-54  
   \$x\_Toggle(pNd), 9-15  
   \$x\_ToggleItemRow(pNd), 9-22  
   \$x\_ToggleWithImage(pThis,pNd), 9-53  
   \$x\_UpTill(pNd, pToTag), 9-18  
   \$x\_Value(pNd,pValue), 9-17  
   addLoadEvent(pFunction), 9-60

- confirmDelete(pMessage, pRequest), 9-11
- doSubmit(pRequest), 9-10
- GetCookie (pName), 9-76
- html\_RemoveAllChildren(pNd), 9-57
- html\_SetSelectValue(pId,pValue), 9-59
- returnInput(p\_R, p\_D), 9-73
- SetCookie (pName,pValue), 9-77
- setReturn(p\_R,p\_D), 9-74
- submitEnter(pNd,e), 9-62

- export file, 1-33
- numeric security group ID, 1-43

## L

---

- LDAP attributes
  - obtaining, 5-6

## P

---

- password
  - changing, 1-9
  - resetting and emailing, 1-88

## R

---

- radio group
  - generate, 3-19

## S

---

- session
  - cookies, 5-5
- session state
  - fetching for current application, 1-34
  - removing for current page, 1-12
  - removing for current session, 1-11
  - setting, 1-98, 1-100, 1-102
- special characters
  - encoding, 1-114

## U

---

- user
  - get e-mail address, 1-52
  - remove preference, 1-84
- user account
  - altering, 1-26
  - creating new, 1-15
  - fetching, 1-35, 1-38, 1-40
  - removing, 1-86
  - update email address, 1-94
  - updating FIRST\_NAME, 1-95
  - updating LAST\_NAME value, 1-96
  - updating USER\_NAME value, 1-103

## V

---

- variables
  - global, 4-1

## W

---

- workspace