# Contents

## Preface
- Topic Overview ................................................................. xi
- Audience ........................................................................... xii
- Documentation Accessibility ........................................... xii
- Related Documents .......................................................... xiii
- Conventions ....................................................................... xiv

## 1 APEX_UTIL
- 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

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........................................................................................................... 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
SETSESSION_LIFETIME_SECONDS Procedure........................................................................... 1-98
SETSESSION_MAX_IDLE_SECONDS Procedure........................................................................... 1-100
SETSESSION_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
$y(pNd) .......................................................................................................... 9-5
$z(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
x_Toggle(pNd) ................................................................. 9-15
x_Remove(pNd) ............................................................. 9-16
x_Value(pNd,pValue) .................................................. 9-17
x_UpTill(pNd, pToTag) ............................................... 9-18
x_ItemRow(pNd,pFunc) ............................................... 9-19
x_HideItemRow(pNd) .................................................... 9-20
x_ShowItemRow(pNd) ................................................... 9-21
x_ToggleItemRow(pNd) ................................................ 9-22
x_HideAllExcept(pNd,pNdArray) ................................ 9-23
x_HideSiblings(pNd) ................................................... 9-24
x_ShowSiblings(pNd) .................................................. 9-25
x_Class(pNd,pClass) ................................................... 9-26
x_SetSiblingsClass(pNd, pClass, pNdClass) ............... 9-27
x_ByClass(pClass, pNd, pTag) ..................................... 9-28
x_ShowAllByClass(pNd, pClass, pTag) ....................... 9-29
x_ShowChildren(pNd) .................................................. 9-30
x_HideChildren(pNd) .................................................. 9-31
x_disableItem(pNd, pTest) ........................................... 9-32
f_get_emptys(pNd, pClassFail, pClass) .................... 9-33
$v_Array(pNd) ............................................................ 9-34
f_ReturnChecked(pNd) .............................................. 9-35
d_ClearAndHide(pNd) .................................................. 9-36
f_SelectedOptions(pNd) ............................................. 9-37
f_SelectValue(pNd) ..................................................... 9-38
u_ArrayToString(pArray, pDelim) .............................. 9-39
x_CheckImageSrc(pId,pSearch) ................................. 9-40
$v_CheckValueAgainst(pThis, pValue) ....................... 9-41
f_Hide_On_Value_Item(pThis, pThat, pValue) ............. 9-42
f_Show_On_Value_Item(pThis, pThat, pValue) .......... 9-43
f_Hide_On_Value_Item_Row(pThis, pThat, pValue) ...... 9-44
f_Show_On_Value_Item_Row(pThis, pThat, pValue) .... 9-45
f_DisableOnValue(pThis, pValue, pThat) .................. 9-46
x_ClassByClass(pNd, pClass, pTag, pClass2) .......... 9-47
f_SelectedOptions(pNd) ............................................. 9-48
f_FormItems(pNd, pType) ......................................... 9-49
f_CheckAll(pThis, pCheck, pArray) .......................... 9-50
f_CheckFirstColumn(pNd) ........................................... 9-51
v_PopupReturn(pValue, pThat) ................................. 9-52
x_ToggleWithImage(pThis,pNd) ................................. 9-53
x_SwitchImageSrc(pNd, pSearch, pReplace) .............. 9-54
x_CheckImageSrc(pNd, pSearch) ............................... 9-55
u_SubString(pText,pMatch) ...................................... 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
$str_AddTD(pThis,pText) ................................................................................................................ 9-65
$str_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
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:

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APEX_UTIL</td>
<td>Use the APEX_UTIL 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.</td>
</tr>
<tr>
<td>APEX_MAIL</td>
<td>Use the APEX_MAIL package to send an email from an Oracle Application Express application.</td>
</tr>
<tr>
<td>APEX_ITEM</td>
<td>Use the APEX_ITEM package to create form elements dynamically based on a SQL query instead of creating individual items page by page.</td>
</tr>
<tr>
<td>APEX_APPLICATION</td>
<td>Use the APEX_APPLICATION package to take advantage of a number of global variables.</td>
</tr>
<tr>
<td>APEX_CUSTOM_AUTH</td>
<td>Use the APEX_CUSTOM_AUTH package to perform various operations related to authentication and session management.</td>
</tr>
<tr>
<td>APEX_LDAP</td>
<td>Use APEX_LDAP to perform various operations related to Lightweight Directory Access Protocol (LDAP) authentication.</td>
</tr>
<tr>
<td>APEX_INSTANCE_ADMIN</td>
<td>Use the APEX_INSTANCE_ADMIN package to get and set email settings, wallet settings, report printing settings and to manage schema to workspace mappings.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APEX_UI_DEFAULT_UPDATE</td>
<td>You can use the APEX_UI_DEFAULT_UPDATE 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.</td>
</tr>
<tr>
<td>JavaScript APIs</td>
<td>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.</td>
</tr>
<tr>
<td>APEX_PLSQL_JOB</td>
<td>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.</td>
</tr>
<tr>
<td>APEX_LANG</td>
<td>You can use APEX_LANG API to translate messages.</td>
</tr>
</tbody>
</table>

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
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](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
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.

Printed documentation is available for sale in the Oracle Store at

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

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

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:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>boldface</td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td>italic</td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td>monospace</td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
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
- 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application</td>
<td>The identification number (ID) of the application.</td>
</tr>
<tr>
<td>p_page</td>
<td>The page number (ID).</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application</td>
<td>The identification number (ID) of the application</td>
</tr>
<tr>
<td>p_page</td>
<td>The page number (ID)</td>
</tr>
<tr>
<td>p_region_name</td>
<td>The region name</td>
</tr>
</tbody>
</table>

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
p_application      The identification number (ID) of the application.

Example
The following example demonstrates how to use the CACHE_PURGE_BY_APPLICATION procedure to purge all the cached pages and regions for the application currently executing.
BEGIN
    APEX_UTIL.CACHE_PURGE_BY_APPLICATION(p_application => :APP_ID);
END;
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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application</td>
<td>The identification number (ID) of the application.</td>
</tr>
<tr>
<td>p_page</td>
<td>The page number (ID).</td>
</tr>
<tr>
<td>p_user_name</td>
<td>The user associated with cached pages and regions.</td>
</tr>
</tbody>
</table>

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 user's cache (only relevant if the cache is set to be by user).

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application</td>
<td>The identification number (ID) of the application.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_new_password</td>
<td>The new password value in clear text</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_name</td>
<td>The user name of the user account</td>
</tr>
</tbody>
</table>

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.');
        END IF;
    END LOOP;
END;

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
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_app_id</td>
<td>The ID of the application for which session state will be cleared for current session</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_page</td>
<td>The ID of the page in the current application for which session state will be cleared for current session.</td>
</tr>
</tbody>
</table>

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\textsubscript{APEX\_UTIL\_COUNT\_CLICK}.  

Syntax

\begin{verbatim}
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);
\end{verbatim}

Parameters

Table 1–10 describes the parameters available in the COUNT_CLICK procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_url</td>
<td>The URL to which to redirect</td>
</tr>
<tr>
<td>p_cat</td>
<td>A category to classify the click</td>
</tr>
<tr>
<td>p_id</td>
<td>Secondary ID to associate with the click (optional)</td>
</tr>
<tr>
<td>p_user</td>
<td>The application user ID (optional)</td>
</tr>
<tr>
<td>p_workspace</td>
<td>The workspace associated with the application (optional)</td>
</tr>
</tbody>
</table>

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.

\begin{verbatim}
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;
\end{verbatim}

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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_id</td>
<td>Numeric primary key of user account</td>
</tr>
<tr>
<td>p_user_name</td>
<td>Alphanumeric name used for login</td>
</tr>
<tr>
<td>p_first_name</td>
<td>Informational</td>
</tr>
<tr>
<td>p_last_name</td>
<td>Informational</td>
</tr>
<tr>
<td>p_description</td>
<td>Informational</td>
</tr>
<tr>
<td>p_email_address</td>
<td>Email address</td>
</tr>
<tr>
<td>p_web_password</td>
<td>Clear text password</td>
</tr>
<tr>
<td>p_web_password_format</td>
<td>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.</td>
</tr>
<tr>
<td>p_group_ids</td>
<td>Colon separated list of numeric group IDs</td>
</tr>
</tbody>
</table>
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').

Table 1–11 (Cont.) CREATE_USER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_developer_privs</td>
<td>Colon separated list of developer privileges. The following are acceptable values for this parameter:</td>
</tr>
<tr>
<td></td>
<td>null - To create an end user (a user who can only authenticate to developed applications).</td>
</tr>
<tr>
<td></td>
<td>CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL - To create a user with developer privilege.</td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>p_default_schema</td>
<td>A database schema assigned to the user’s workspace, used by default for browsing.</td>
</tr>
<tr>
<td>p_allow_access_to_schemas</td>
<td>Colon separated list of schemas assigned to the user’s workspace to which the user is restricted (leave null for all).</td>
</tr>
<tr>
<td>p_account_expiry</td>
<td>Date password was last updated, which will default to today’s date on creation.</td>
</tr>
<tr>
<td>p_account_locked</td>
<td>'Y' or 'N' indicating if account is locked or unlocked.</td>
</tr>
<tr>
<td>p_failed_access_attempts</td>
<td>Number of consecutive login failures that have occurred, defaults to 0 on creation.</td>
</tr>
<tr>
<td>p_change_password_on_first_use</td>
<td>'Y' or 'N' to indicate whether password must be changed on first use, defaults to 'Y' on creation.</td>
</tr>
<tr>
<td>p_first_password_use_occurred</td>
<td>'Y' or 'N' to indicate whether login has occurred since password change, defaults to 'N' on creation.</td>
</tr>
<tr>
<td>p_attribute_01</td>
<td>Arbitrary text accessible with an API</td>
</tr>
</tbody>
</table>
CREATE_USER Procedure

- 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_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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_id</td>
<td>Primary key of group</td>
</tr>
<tr>
<td>p_group_name</td>
<td>Name of group</td>
</tr>
<tr>
<td>p_security_group_id</td>
<td>Workspace ID</td>
</tr>
<tr>
<td>p_group_desc</td>
<td>Descriptive text</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_group_name</td>
<td>Identifies the name of an existing group in the workspace</td>
</tr>
<tr>
<td>p_group_id</td>
<td>Identifies the numeric ID of an existing group in the workspace</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_file_name</td>
<td>Defines the filename of the print document</td>
</tr>
<tr>
<td>p_content_disposition</td>
<td>Specifies whether to download the print document or display inline (&quot;attachment&quot;, &quot;inline&quot;)</td>
</tr>
<tr>
<td>p_report_data</td>
<td>XML based report data</td>
</tr>
<tr>
<td>p_report_layout</td>
<td>Report layout in XSL-FO or RTF format</td>
</tr>
<tr>
<td>p_report_layout_type</td>
<td>Defines the report layout type, that is &quot;xsl-fo&quot; or &quot;rtf&quot;</td>
</tr>
<tr>
<td>p_document_format</td>
<td>Defines the document format, that is &quot;pdf&quot;, &quot;rtf&quot;, &quot;xls&quot;, &quot;htm&quot;, or &quot;xml&quot;</td>
</tr>
<tr>
<td>p_print_server</td>
<td>URL of the print server. If not specified, the print server will be derived from preferences.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_file_name</td>
<td>Defines the filename of the print document</td>
</tr>
<tr>
<td>p_content_disposition</td>
<td>Specifies whether to download the print document or display inline (&quot;attachment&quot;, &quot;inline&quot;)</td>
</tr>
<tr>
<td>p_application_id</td>
<td>Defines the application ID of the report query</td>
</tr>
<tr>
<td>p_report_query_name</td>
<td>Name of the report query (stored under application’s Shared Components)</td>
</tr>
<tr>
<td>p_report_layout</td>
<td>Report layout in XSL-FO or RTF format</td>
</tr>
<tr>
<td>p_report_layout_type</td>
<td>Defines the report layout type, that is &quot;xsl-fo&quot; or &quot;rtf&quot;</td>
</tr>
<tr>
<td>p_document_format</td>
<td>Defines the document format, that is &quot;pdf&quot;, &quot;rtf&quot;, &quot;xls&quot;, &quot;htm&quot;, or &quot;xml&quot;</td>
</tr>
<tr>
<td>p_print_server</td>
<td>URL of the print server. If not specified, the print server will be derived from preferences.</td>
</tr>
</tbody>
</table>

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;

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_file_name</td>
<td>Defines the filename of the print document</td>
</tr>
<tr>
<td>p_content_disposition</td>
<td>Specifies whether to download the print document or display inline (&quot;attachment&quot;, &quot;inline&quot;)</td>
</tr>
<tr>
<td>p_application_id</td>
<td>Defines the application ID of the report query</td>
</tr>
<tr>
<td>p_report_query_name</td>
<td>Name of the report query (stored under application’s Shared Components)</td>
</tr>
<tr>
<td>p_report_layout_name</td>
<td>Name of the report layout (stored under application’s Shared Components)</td>
</tr>
<tr>
<td>p_report_layout_type</td>
<td>Defines the report layout type, that is &quot;xsl-fo&quot; or &quot;rtf&quot;</td>
</tr>
<tr>
<td>p_document_format</td>
<td>Defines the document format, that is &quot;pdf&quot;, &quot;rtf&quot;, &quot;xls&quot;, &quot;htm&quot;, or &quot;xml&quot;</td>
</tr>
<tr>
<td>p_print_server</td>
<td>URL of the print server. If not specified, the print server will be derived from preferences.</td>
</tr>
</tbody>
</table>

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;

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_file_name</td>
<td>Defines the filename of the print document</td>
</tr>
<tr>
<td>p_content_disposition</td>
<td>Specifies whether to download the print document or display inline (&quot;attachment&quot;, &quot;inline&quot;)</td>
</tr>
<tr>
<td>p_report_data</td>
<td>XML based report data, must be encoded in UTF-8</td>
</tr>
<tr>
<td>p_report_layout</td>
<td>Report layout in XSL-FO or RTF format</td>
</tr>
<tr>
<td>p_report_layout_type</td>
<td>Defines the report layout type, that is &quot;xsl-fo&quot; or &quot;rtf&quot;</td>
</tr>
<tr>
<td>p_document_format</td>
<td>Defines the document format, that is &quot;pdf&quot;, &quot;rtf&quot;, &quot;xls&quot;, &quot;htm&quot;, or &quot;xml&quot;</td>
</tr>
<tr>
<td>p_print_server</td>
<td>URL of the print server. If not specified, the print server will be derived from preferences.</td>
</tr>
</tbody>
</table>

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;

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_id</td>
<td>Numeric primary key of the user account</td>
</tr>
<tr>
<td>p_user_name</td>
<td>Alphanumeric name used for login.</td>
</tr>
<tr>
<td>p_first_name</td>
<td>Informational.</td>
</tr>
<tr>
<td>p_last_name</td>
<td>Informational.</td>
</tr>
<tr>
<td>p_web_password</td>
<td>Clear text password. If using this procedure to update the password for the user, values for both p_web_password and p_new_password must not be null and must be identical.</td>
</tr>
<tr>
<td>p_new_password</td>
<td>Clear text new password. If using this procedure to update the password for the user, values for both p_web_password and p_new_password must not be null and must be identical.</td>
</tr>
<tr>
<td>p_email_address</td>
<td>Informational.</td>
</tr>
</tbody>
</table>

See Also: "SET_USERNAME Procedure" on page 1-103

See Also: "SET_FIRST_NAME Procedure" on page 1-95

See Also: "SET_LAST_NAME Procedure" on page 1-96

See Also: "SET_EMAIL Procedure" on page 1-94
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_start_date</td>
<td>Unused</td>
</tr>
<tr>
<td>p_end_date</td>
<td>Unused</td>
</tr>
<tr>
<td>p_employee_id</td>
<td>Unused</td>
</tr>
<tr>
<td>p_allow_access_to_schemas</td>
<td>A list of schemas assigned to the user's workspace to which the user is restricted</td>
</tr>
<tr>
<td>p_person_type</td>
<td>Unused</td>
</tr>
<tr>
<td>p_default_schema</td>
<td>A database schema assigned to the user's workspace, used by default for browsing</td>
</tr>
<tr>
<td>p_group_ids</td>
<td>Colon-separated list of numeric group IDs</td>
</tr>
<tr>
<td>p_developer_roles</td>
<td>Colon-separated list of developer privileges. The following are acceptable values for this parameter:</td>
</tr>
<tr>
<td></td>
<td>· <code>null</code> - To update the user to be an end user (a user who can only authenticate to developed applications)</td>
</tr>
<tr>
<td></td>
<td>· <code>CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL</code> - To update the user to have developer privilege</td>
</tr>
<tr>
<td></td>
<td>· <code>ADMIN:CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL</code> - To update the user to have full workspace administrator and developer privilege</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>p_description</td>
<td>Informational</td>
</tr>
<tr>
<td>p_account_expiry</td>
<td>Date password was last updated.</td>
</tr>
<tr>
<td>p_account_locked</td>
<td>'Y' or 'N' indicating if account is locked or unlocked.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;LOCK_ACCOUNT Procedure&quot; on page 1-76, &quot;UNLOCK_ACCOUNT Procedure&quot; on page 1-113</td>
</tr>
<tr>
<td>p_failed_access_attempts</td>
<td>Number of consecutive login failures that have occurred.</td>
</tr>
<tr>
<td>p_change_password_on_first_use</td>
<td>'Y' or 'N' to indicate whether password must be changed on first use.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;CHANGE_PASSWORD_ON_FIRST_USE Function&quot; on page 1-10</td>
</tr>
<tr>
<td>p_first_password_use_occurred</td>
<td>'Y' or 'N' to indicate whether login has occurred since password change.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;PASSWORD_FIRST_USE_OCCURRED Function&quot; on page 1-77</td>
</tr>
</tbody>
</table>
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'.

```sql
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,
  );
END;
```

```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');
END;
```
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_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_USER_ACCOUNT_DAYS_LEFT Function

Returns the number of days remaining before an 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_name</td>
<td>The user name of the user account</td>
</tr>
</tbody>
</table>

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);
      htp.p('End User Account:'||c1.user_name||' will expire in '||l_days_left||' days.');
   END LOOP;
END;

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_name</td>
<td>The user name of the user account</td>
</tr>
</tbody>
</table>

**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);  
      htp.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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_name</td>
<td>The user name of the user account</td>
</tr>
</tbody>
</table>

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);
        htp.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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_export_format</td>
<td>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</td>
</tr>
</tbody>
</table>

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**

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_item</td>
<td>The name of an application-level item (not a page item) whose current value is to be fetched</td>
</tr>
<tr>
<td>p_app</td>
<td>The ID of the application that owns the item (leave null for the current application)</td>
</tr>
<tr>
<td>p_session</td>
<td>The session ID from which to obtain the value (leave null for the current session)</td>
</tr>
</tbody>
</table>

**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.

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_id</td>
<td>Numeric primary key of the user account</td>
</tr>
<tr>
<td>p_workspace</td>
<td>The name of the workspace</td>
</tr>
<tr>
<td>p_user_name</td>
<td>Alphanumeric name used for login.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;GET_USERNAME Function&quot; on page 1-71</td>
</tr>
<tr>
<td>p_first_name</td>
<td>Informational.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;GET_FIRST_NAME Function&quot; on page 1-56</td>
</tr>
<tr>
<td>p_last_name</td>
<td>Informational.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;GET_LAST_NAME Function&quot; on page 1-60</td>
</tr>
<tr>
<td>p_web_password</td>
<td>Obfuscated account password</td>
</tr>
<tr>
<td>p_email_address</td>
<td>Email address.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;GET_EMAIL Function&quot; on page 1-52</td>
</tr>
<tr>
<td>p_start_date</td>
<td>Unused</td>
</tr>
<tr>
<td>p_end_date</td>
<td>Unused</td>
</tr>
<tr>
<td>p_employee_id</td>
<td>Unused</td>
</tr>
<tr>
<td>p_allow_access_to_schemas</td>
<td>A list of schemas assigned to the user's workspace to which user is restricted</td>
</tr>
<tr>
<td>p_person_type</td>
<td>Unused</td>
</tr>
</tbody>
</table>

APEX_UTIL 1-35
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.

```sql
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,
    ...
  );
END;
```

### Table 1–24 (Cont.) Fetch_User Parameters Signature 1

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>See Also:</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_default_schema</td>
<td>A database schema assigned to the user’s workspace, used by default for browsing.</td>
<td>&quot;GET_DEFAULT_SCHEMA Function&quot; on page 1-51</td>
</tr>
<tr>
<td>p_groups</td>
<td>List of groups of which user is a member.</td>
<td>&quot;GET_GROUPS_USER_BELONGS_TO Function&quot; on page 1-57 and &quot;CURRENT_USER_IN_GROUP Function&quot; on page 1-19</td>
</tr>
<tr>
<td>p_developer_role</td>
<td>Colon-separated list of developer roles. The following are acceptable values for this parameter:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>null - Indicates an end user (a user who can only authenticate to developed applications).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td>&quot;GET_USER_ROLES Function&quot; on page 1-70</td>
</tr>
<tr>
<td>p_description</td>
<td>Informational</td>
<td></td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_id</td>
<td>Numeric primary key of the user account</td>
</tr>
<tr>
<td>p_user_name</td>
<td>Alphanumeric name used for login.</td>
</tr>
<tr>
<td>p_first_name</td>
<td>Informational.</td>
</tr>
<tr>
<td>p_last_name</td>
<td>Informational.</td>
</tr>
<tr>
<td>p_email_address</td>
<td>Email address.</td>
</tr>
<tr>
<td>p_groups</td>
<td>List of groups of which user is a member.</td>
</tr>
<tr>
<td>p_description</td>
<td></td>
</tr>
</tbody>
</table>

See Also: "GET_USERNAME Function" on page 1-71

See Also: "GET_FIRST_NAME Function" on page 1-56

See Also: "GET_LAST_NAME Function" on page 1-60

See Also: "GET_EMAIL Function" on page 1-52

See Also: "GET_GROUPS_USER_BELONGS_TO Function" on page 1-57 and "CURRENT_USER_IN_GROUP Function" on page 1-19
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 \texttt{p\_user\_id}. The code then stores all the other OUT parameter values in local variables.

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_id</td>
<td>Numeric primary key of the user account</td>
</tr>
<tr>
<td>p_workspace</td>
<td>The name of the workspace</td>
</tr>
<tr>
<td>p_user_name</td>
<td>Alphanumeric name used for login</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;GET_USERNAME Function&quot; on page 1-71</td>
</tr>
<tr>
<td>p_first_name</td>
<td>Informational.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;GET_FIRST_NAME Function&quot; on page 1-56</td>
</tr>
<tr>
<td>p_last_name</td>
<td>Informational.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;GET_LAST_NAME Function&quot; on page 1-56</td>
</tr>
<tr>
<td>p_web_password</td>
<td>Obfuscated account password</td>
</tr>
<tr>
<td>p_email_address</td>
<td>Email address.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;GET_EMAIL Function&quot; on page 1-52</td>
</tr>
<tr>
<td>p_start_date</td>
<td>Unused</td>
</tr>
<tr>
<td>p_end_date</td>
<td>Unused</td>
</tr>
</tbody>
</table>
### Table 1–26  (Cont.) Fetch_User Parameters Signature 3

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_employee_id</td>
<td>Unused</td>
</tr>
<tr>
<td>p_allow_access_to_schemas</td>
<td>A list of schemas assigned to the user's workspace to which user is restricted</td>
</tr>
<tr>
<td>p_person_type</td>
<td>Unused</td>
</tr>
<tr>
<td>p_default_schema</td>
<td>A database schema assigned to the user's workspace, used by default for browsing.</td>
</tr>
<tr>
<td>p_groups</td>
<td>List of groups of which user is a member.</td>
</tr>
<tr>
<td>p_developer_role</td>
<td>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.</td>
</tr>
<tr>
<td>p_description</td>
<td>Informational</td>
</tr>
<tr>
<td>p_account_expiry</td>
<td>Date account password was last reset.</td>
</tr>
<tr>
<td>p_account_locked</td>
<td>Locked/Unlocked indicator Y or N.</td>
</tr>
<tr>
<td>p_failed_access_attempts</td>
<td>Counter for consecutive login failures</td>
</tr>
<tr>
<td>p_change_password_on_first_use</td>
<td>Setting to force password change on first use Y or N</td>
</tr>
<tr>
<td>p_first_password_use_occurred</td>
<td>Indicates whether login with password occurred Y or N</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_workspace</td>
<td>The name of the workspace</td>
</tr>
</tbody>
</table>

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'.

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_security_group_id</td>
<td>The security group ID of a workspace</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_name</td>
<td>The user name of the user account</td>
</tr>
</tbody>
</table>

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.');
        END IF;
    END LOOP;
END;

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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>User name in the account.</td>
</tr>
<tr>
<td>p_attribute_number</td>
<td>Number of attributes in the user record (1 through 10)</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use the GET_ATTRIBUTE function to return the value for the 1st attribute for the user 'FRANK'.

```sql
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_item_name</td>
<td>Name of valid application page ITEM that with type FILE that contains the source type of DB column.</td>
</tr>
<tr>
<td>p_v1</td>
<td>Value of primary key column 1.</td>
</tr>
<tr>
<td>p_v2</td>
<td>Value of primary key column 2.</td>
</tr>
<tr>
<td>p_content_disposition</td>
<td>Specify inline or attachment, all other values ignored</td>
</tr>
</tbody>
</table>

Example

As a PLSQL Function Body:

```
RETURN '"<img src="'||APEX_UTIL.GET_BLOB_FILE_SRC('P2_ATTACHMENT',:P2_EMPNO)||"" />
';
```

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 '"<img src="'||apex_util.get_blob_file_src('P4_DOCUMENT',id)||"" />' 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**

```sql
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.

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>The user name in the account</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_file_id</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>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;</td>
</tr>
<tr>
<td>p_inline</td>
<td>Valid values include YES and NO. YES to display inline in a browser. NO to download as attachment</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_name</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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'.

```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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>Identifies the user name in the account</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>Identifies the user name in the account</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_group_name</td>
<td>Identifies the user name in the account</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_group_id</td>
<td>Identifies a numeric ID of a group in the workspace</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>The user name in the user account record</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_item</td>
<td>The case insensitive name of the item for which you want to have the session state fetched</td>
</tr>
</tbody>
</table>

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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_preference</td>
<td>Name of the preference to retrieve the value</td>
</tr>
<tr>
<td>p_value</td>
<td>Value of the preference</td>
</tr>
<tr>
<td>p_user</td>
<td>User for whom the preference is being retrieved</td>
</tr>
</tbody>
</table>

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'.

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_report_data</td>
<td>XML based report data</td>
</tr>
<tr>
<td>p_report_layout</td>
<td>Report layout in XSL-FO or RTF format</td>
</tr>
<tr>
<td>p_report_layout_type</td>
<td>Defines the report layout type, that is &quot;xsl-fo&quot; or &quot;rtf&quot;</td>
</tr>
<tr>
<td>p_document_format</td>
<td>Defines the document format, that is &quot;pdf&quot;, &quot;rtf&quot;, &quot;xls&quot;, &quot;htm&quot;, or &quot;xml&quot;</td>
</tr>
<tr>
<td>p_print_server</td>
<td>URL of the print server. If not specified, the print server will be derived from preferences.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application_id</td>
<td>Defines the application ID of the report query</td>
</tr>
<tr>
<td>p_report_query_name</td>
<td>Name of the report query (stored under application’s shared components)</td>
</tr>
<tr>
<td>p_report_layout_name</td>
<td>Name of the report layout (stored under application’s Shared Components)</td>
</tr>
<tr>
<td>p_report_layout_type</td>
<td>Defines the report layout type, that is &quot;xsl-fo&quot; or &quot;rtf&quot;</td>
</tr>
<tr>
<td>p_document_format</td>
<td>Defines the document format, that is &quot;pdf&quot;, &quot;rtf&quot;, &quot;xls&quot;, &quot;htm&quot;, or &quot;xml&quot;</td>
</tr>
<tr>
<td>p_print_server</td>
<td>URL of the print server. If not specified, the print server will be derived from preferences.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application_id</td>
<td>Defines the application ID of the report query</td>
</tr>
<tr>
<td>p_report_query_name</td>
<td>Name of the report query (stored under application’s shared components)</td>
</tr>
<tr>
<td>p_report_layout</td>
<td>Defines the report layout in XSL-FO or RTF format</td>
</tr>
<tr>
<td>p_report_layout_type</td>
<td>Defines the report layout type, that is &quot;xsl-fo&quot; or &quot;rtf&quot;</td>
</tr>
<tr>
<td>p_document_format</td>
<td>Defines the document format, that is &quot;pdf&quot;, &quot;rtf&quot;, &quot;xls&quot;, &quot;htm&quot;, or &quot;xml&quot;</td>
</tr>
<tr>
<td>p_print_server</td>
<td>URL of the print server. If not specified, the print server will be derived from preferences.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_report_data</td>
<td>XML based report data, must be encoded in UTF-8</td>
</tr>
<tr>
<td>p_report_layout</td>
<td>Report layout in XSL-FO or RTF format</td>
</tr>
<tr>
<td>p_report_layout_type</td>
<td>Defines the report layout type, that is &quot;xsl-fo&quot; or &quot;rtf&quot;</td>
</tr>
<tr>
<td>p_document_format</td>
<td>Defines the document format, that is &quot;pdf&quot;, &quot;rtf&quot;, &quot;xls&quot;, &quot;htm&quot;, or &quot;xml&quot;</td>
</tr>
<tr>
<td>p_print_server</td>
<td>URL of the print server. If not specified, the print server will be derived from preferences</td>
</tr>
</tbody>
</table>

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');
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_item</td>
<td>The case insensitive name of the item for which you want to have the session state fetched</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>Identifies the name of a user in the workspace</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>Identifies a user name in the account</td>
</tr>
</tbody>
</table>

**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
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_userid</td>
<td>Identifies the numeric ID of a user account in the workspace</td>
</tr>
</tbody>
</table>

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
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>User name in account</td>
</tr>
<tr>
<td>p_password</td>
<td>Password to be compared with password stored in the account</td>
</tr>
</tbody>
</table>

**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;
```
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>Identifies the user name to be tested</td>
</tr>
</tbody>
</table>

**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.

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_name</td>
<td>The user name of the user account</td>
</tr>
</tbody>
</table>

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);  
        htp.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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_name</td>
<td>The user name of the user account</td>
</tr>
</tbody>
</table>

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
      htp.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
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.

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_url</td>
<td>An f?p relative URL with all substitutions resolved</td>
</tr>
<tr>
<td>p_url_charset</td>
<td>The character set name (for example, UTF-8) to use when escaping special characters contained within argument values</td>
</tr>
<tr>
<td>p_checksum_type</td>
<td>Null or any of the following six values, SESSION or 3, PRIVATE_BOOKMARK or 2, or PUBLIC_BOOKMARK or 1</td>
</tr>
</tbody>
</table>

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.

```sql
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

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.
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:

```
htp.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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_security_name</td>
<td>The name of the security scheme that determines if the user passes the security check</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application</td>
<td>The identification number (ID) of the application.</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application</td>
<td>The identification number (ID) of the application.</td>
</tr>
<tr>
<td>p_page</td>
<td>The number of the page containing the region to be deleted.</td>
</tr>
<tr>
<td>p_region_name</td>
<td>The region name to be deleted.</td>
</tr>
</tbody>
</table>

**Example**

The following example shows how to use the PURGE_REGIONS_BY_NAME procedure to delete all the cached values for the region 'myCached_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 => 'myCached_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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application</td>
<td>The identification number (ID) of the application.</td>
</tr>
<tr>
<td>p_page</td>
<td>The identification number of page containing the region.</td>
</tr>
</tbody>
</table>

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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_preference</td>
<td>Name of the preference to remove</td>
</tr>
<tr>
<td>p_user</td>
<td>User for whom the preference is defined</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use the REMOVE_PREFERENCE procedure to remove the preference default_view for the currently authenticated user.

```sql
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**

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user</td>
<td>Identifies the user for whom sorting preferences will be removed</td>
</tr>
</tbody>
</table>

**Example**

The following example shows how to use the `REMOVE_SORT_PREFERENCES` procedure to remove the currently authenticated user’s column heading sorting preferences.

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_id</td>
<td>The numeric primary key of the user account record</td>
</tr>
<tr>
<td>p_user_name</td>
<td>The user name of the user account</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user</td>
<td>The user name of the user account</td>
</tr>
<tr>
<td>p_msg</td>
<td>Message text to be mailed to a user</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_val</td>
<td>The numeric value to be saved</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_val</td>
<td>The is the VARCHAR2 value to be saved</td>
</tr>
</tbody>
</table>

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'.

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_userid</td>
<td>The numeric ID of the user account</td>
</tr>
<tr>
<td>p_attribute_number</td>
<td>Attribute number in the user record (1 through 10)</td>
</tr>
<tr>
<td>p_attribute_value</td>
<td>Value of the attribute located by p_attribute_number to be set in the user record</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_code</td>
<td>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.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_status</td>
<td>Any text the developer chooses to denote the result of the authentication attempt (up to 4000 characters).</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_userid</td>
<td>The numeric ID of the user account</td>
</tr>
<tr>
<td>p_email</td>
<td>The email address to be saved in user account</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_userid</td>
<td>The numeric ID of the user account</td>
</tr>
<tr>
<td>p_first_name</td>
<td>FIRST_NAME value to be saved in user account</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_userid</td>
<td>The numeric ID of the user account</td>
</tr>
<tr>
<td>p_last_name</td>
<td>LAST_NAME value to be saved in the user account</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_preference</td>
<td>Name of the preference (case-sensitive)</td>
</tr>
<tr>
<td>p_value</td>
<td>Value of the preference</td>
</tr>
<tr>
<td>p_user</td>
<td>User for whom the preference is being set</td>
</tr>
</tbody>
</table>

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
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**

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_seconds</td>
<td>A positive integer indicating the number of seconds the session used by this application is allowed to exist.</td>
</tr>
<tr>
<td>p_scope</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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.

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_seconds</td>
<td>A positive integer indicating the number of seconds allowed between page requests.</td>
</tr>
<tr>
<td>p_scope</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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.

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_name</td>
<td>Name of the application-level or page-level item for which you are setting session state</td>
</tr>
<tr>
<td>p_value</td>
<td>Value of session state to set</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_userid</td>
<td>The numeric ID of the user account</td>
</tr>
<tr>
<td>p_username</td>
<td>USER_NAME value to be saved in the user account</td>
</tr>
</tbody>
</table>

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

```sql
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
    htp.p('Password is too short');
END IF;

IF l_new_differs_by_err THEN
    htp.p('Password is too similar to the old password');
END IF;

IF l_one_alpha_err THEN
    htp.p('Password must contain at least one alphabetic character');
END IF;

IF l_one_numeric_err THEN
    htp.p('Password must contain at least one numeric character');
END IF;

IF l_one_punctuation_err THEN
    htp.p('Password must contain at least one punctuation character');
END IF;

IF l_one_upper_err THEN
    htp.p('Password must contain at least one upper-case character');
END IF;

IF l_one_lower_err THEN
    htp.p('Password must contain at least one lower-case character');
END IF;

IF l_not_like_username_err THEN
    htp.p('Password may not contain the username');
END IF;

IF l_not_like_workspace_name_err THEN
    htp.p('Password may not contain the workspace name');
END IF;

IF l_not_like_words_err THEN
    htp.p('Password contains one or more prohibited common words');
END IF;

IF l_not_reusable_err THEN
htp.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

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>Username that identifies the account in the current workspace</td>
</tr>
<tr>
<td>p_password</td>
<td>Password to be checked against password strength rules</td>
</tr>
<tr>
<td>p_old_password</td>
<td>Current password for the account. Used only to enforce &quot;new password must differ from old&quot; rule</td>
</tr>
<tr>
<td>p_workspace_name</td>
<td>Current workspace name, used only to enforce &quot;password must not contain workspace name&quot; rule</td>
</tr>
</tbody>
</table>

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.

```sql
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_APPLICATIONGLOBAL.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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_string</td>
<td>String to be converted into a PL/SQL table of type APEX_APPLICATIONGLOBAL.VC_ARR2</td>
</tr>
<tr>
<td>p_separator</td>
<td>String separator. The default is a colon</td>
</tr>
</tbody>
</table>

Example
The following example shows how to use the STRING_TO_TABLE function. The function is passed the string 'One:Two:Three' in the p_string parameter and it returns a PL/SQL array of type APEX_APPLICATION_GLOBAL.VC_ARR2 containing 3 elements, the element at position 1 contains the value 'One', position 2 contains the value 'Two' and position 3 contains the value 'Three'. This is then output using the HTP.P function call.

```declare
    l_vc_arr2    APEX_APPLICATION_GLOBAL.VC_ARR2;
begin
    l_vc_arr2 := APEX_UTIL.STRING_TO_TABLE('One:Two:Three');
    for z in 1..l_vc_arr2.count loop
      htp.p(l_vc_arr2(z));
    end loop;
end;
```

See Also: "TABLE_TO_STRING Function" on page 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
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_string</td>
<td>String separator. Default separator is a colon (:)</td>
</tr>
<tr>
<td>p_table</td>
<td>PL/SQL table that is to be converted into a delimited string</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_name</td>
<td>The user name of the user account</td>
</tr>
</tbody>
</table>

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);
        htp.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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_name</td>
<td>The user name of the user account</td>
</tr>
</tbody>
</table>

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);
    htp.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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_name</td>
<td>The user name of the user account</td>
</tr>
</tbody>
</table>

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);
   htp.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:

<table>
<thead>
<tr>
<th>Special Characters</th>
<th>After Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>%25</td>
</tr>
<tr>
<td>+</td>
<td>%2B</td>
</tr>
<tr>
<td>space</td>
<td>+</td>
</tr>
<tr>
<td>.</td>
<td>%2E</td>
</tr>
<tr>
<td>*</td>
<td>%2A</td>
</tr>
<tr>
<td>?</td>
<td>%3F</td>
</tr>
<tr>
<td>\</td>
<td>%5C</td>
</tr>
<tr>
<td>/</td>
<td>%2F</td>
</tr>
<tr>
<td>&gt;</td>
<td>%3E</td>
</tr>
<tr>
<td>&lt;</td>
<td>%3C</td>
</tr>
<tr>
<td>}</td>
<td>%7B</td>
</tr>
<tr>
<td>{</td>
<td>%7D</td>
</tr>
<tr>
<td>~</td>
<td>%7E</td>
</tr>
<tr>
<td>[</td>
<td>%5B</td>
</tr>
<tr>
<td>]</td>
<td>%5D</td>
</tr>
<tr>
<td>'</td>
<td>%60</td>
</tr>
<tr>
<td>;</td>
<td>%3B</td>
</tr>
<tr>
<td>?</td>
<td>%3F</td>
</tr>
<tr>
<td>@</td>
<td>%40</td>
</tr>
<tr>
<td>&amp;</td>
<td>%26</td>
</tr>
<tr>
<td>#</td>
<td>%23</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>%5E</td>
</tr>
<tr>
<td>:</td>
<td>%3A</td>
</tr>
<tr>
<td>=</td>
<td>%3D</td>
</tr>
<tr>
<td>$</td>
<td>%24</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_url</td>
<td>The string to be encoded</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user_name</td>
<td>The user name of the user account</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use the WORKSPACE_ACCOUNT_DAYS_LEFT function. It can be used in to find the number of days remaining before an Application Express administrator or developer account in the current workspace expires.

DECLARE
    l_days_left NUMBER;
BEGIN
    FOR c1 IN (SELECT user_name from wwv_flow_users) LOOP
        l_days_left := APEX_UTIL.WORKSPACE_ACCOUNT_DAYS_LEFT(p_user_name => c1.user_name);
        htp.p('Workspace Account:'||c1.user_name||' 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
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_mail_id</td>
<td>The numeric ID associated with the email. This is the numeric identifier returned from the call to APEX_MAIL.SEND to compose the email body.</td>
</tr>
<tr>
<td>p_attachment</td>
<td>A BLOB variable containing the binary content to be attached to the email message.</td>
</tr>
<tr>
<td>p_filename</td>
<td>The filename associated with the email attachment.</td>
</tr>
<tr>
<td>p_mime_type</td>
<td>A valid MIME type (or Internet media type) to associate with the email attachment.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_smtp_hostname</td>
<td>SMTP gateway host name</td>
</tr>
<tr>
<td>p_smtp_portno</td>
<td>SMTP gateway port number</td>
</tr>
</tbody>
</table>

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:

```html
<img src="http://someserver.com/hello.gif" alt="Hello" />
```

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.
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;
/

Table 2–3  SEND Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_to</td>
<td>Valid email address to which the email will be sent (required). For multiple email addresses, use a comma-separated list</td>
</tr>
<tr>
<td>p_from</td>
<td>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</td>
</tr>
<tr>
<td>p_body</td>
<td>Body of the email in plain text, not HTML (required). If a value is passed to p_body_html, then this is the only text the recipient sees. If a value is not passed to p_body_html, 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.</td>
</tr>
<tr>
<td>p_body_html</td>
<td>Body of the email in HTML format. This must be a full HTML document including the &lt;html&gt; and &lt;body&gt; tags. A single line cannot exceed 1000 characters without a carriage return or line feed (CRLF)</td>
</tr>
<tr>
<td>p_subj</td>
<td>Subject of the email</td>
</tr>
<tr>
<td>p_cc</td>
<td>Valid email addresses to which the email is copied. For multiple email addresses, use a comma-separated list</td>
</tr>
<tr>
<td>p_bcc</td>
<td>Valid email addresses to which the email is blind copied. For multiple email addresses, use a comma-separated list</td>
</tr>
</tbody>
</table>
| p_replyto     | Address of the Reply-To mail header. You can use this parameter as follows:
  - If you omit the p_replyto parameter, the Reply-To mail header is set to the value specified in the p_from parameter
  - If you include the p_replyto parameter, but provide a NULL value, the Reply-To mail header is set to NULL. This results in the suppression of automatic email replies
  - If you include p_replyto 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) |
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

```sql
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_delimitor  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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>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</td>
</tr>
<tr>
<td>p_value</td>
<td>Value of a check box, hidden field, or input form item</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Controls HTML tag attributes (such as disabled)</td>
</tr>
<tr>
<td>p_checked_values</td>
<td>Values to be checked by default</td>
</tr>
<tr>
<td>p_checked_values_delimitor</td>
<td>Delimits the values in the previous parameter, p_checked_values</td>
</tr>
<tr>
<td>p_item_id</td>
<td>HTML attribute ID for the &lt;input&gt; tag</td>
</tr>
<tr>
<td>p_item_label</td>
<td>Invisible label created for the item</td>
</tr>
</tbody>
</table>

Examples of Default Check Box Behavior

The following example demonstrates how to create a selected check box for each employee in the emp table.

```sql
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.

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>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.</td>
</tr>
<tr>
<td>p_row</td>
<td>This parameter is deprecated. Anything specified for this value will be ignored.</td>
</tr>
<tr>
<td>p_value</td>
<td>Value of a field item</td>
</tr>
<tr>
<td>p_date_format</td>
<td>Valid database date format</td>
</tr>
<tr>
<td>p_size</td>
<td>Controls HTML tag attributes (such as disabled)</td>
</tr>
<tr>
<td>p_maxlength</td>
<td>Determines the maximum number of enterable characters. Becomes the maxlength attribute of the &lt;input&gt; HTML tag</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Extra HTML parameters you want to add</td>
</tr>
<tr>
<td>p_item_id</td>
<td>HTML attribute ID for the &lt;input&gt; tag</td>
</tr>
<tr>
<td>p_item_label</td>
<td>Invisible label created for the item</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>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</td>
</tr>
<tr>
<td>p_value</td>
<td>Current value</td>
</tr>
<tr>
<td>p_item_id</td>
<td>HTML attribute ID for the &lt;span&gt; tag</td>
</tr>
<tr>
<td>p_item_label</td>
<td>Invisible label created for the item</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>Number to identify the item you want to generate. The number will determine</td>
</tr>
<tr>
<td></td>
<td>which g_FXX global is populated</td>
</tr>
<tr>
<td>p_value</td>
<td>Value of the hidden input form item</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Extra HTML parameters you want to add</td>
</tr>
<tr>
<td>p_item_id</td>
<td>HTML attribute ID for the &lt;input&gt; tag</td>
</tr>
<tr>
<td>p_item_label</td>
<td>Invisible label created for the item</td>
</tr>
</tbody>
</table>

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:

\[
\text{UTL_RAW.CAST\_TO\_RAW(DBMS\_OBFUSCATION\_TOOLKIT.MD5())}
\]

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_value01</td>
<td>Fifty available inputs. If no parameters are supplied, the default to NULL</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>p_value50</td>
<td></td>
</tr>
<tr>
<td>p_col_sep</td>
<td>String used to separate p_value inputs. Defaults to the pipe symbol (</td>
</tr>
</tbody>
</table>

**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
```
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>Indicates the form element to be generated. For example, 1 equals F01 and 2 equals F02. Typically the p_idx parameter is constant for a given column.</td>
</tr>
<tr>
<td>p_value01</td>
<td>Fifty available inputs. Parameters not supplied default to NULL</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>p_value50</td>
<td></td>
</tr>
<tr>
<td>p_col_sep</td>
<td>String used to separate p_value inputs. Defaults to the pipe symbol (</td>
</tr>
</tbody>
</table>

### 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>Form element name. For example, 1 equals F01 and 2 equals F02. Typically, p_idx is a constant for a given column.</td>
</tr>
<tr>
<td>p_value</td>
<td>Form element current value. This value should be one of the values in the p_lov_name parameter.</td>
</tr>
<tr>
<td>p_lov_name</td>
<td>Named LOV used for this popup.</td>
</tr>
<tr>
<td>p_width</td>
<td>Width of the text box</td>
</tr>
<tr>
<td>p_max_length</td>
<td>Maximum number of characters that can be entered in the text box.</td>
</tr>
<tr>
<td>p_form_index</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Table 3–7  (Cont.) POPUP_FROM_LOV Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_escape_html</td>
<td>Replacements for special characters that require an escaped equivalent:</td>
</tr>
<tr>
<td></td>
<td>- &lt; for &lt;</td>
</tr>
<tr>
<td></td>
<td>- &gt; for &gt;</td>
</tr>
<tr>
<td></td>
<td>- &amp; for &amp;</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>p_max_elements</td>
<td>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.</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Additional HTML attributes to use for the form item.</td>
</tr>
<tr>
<td>p_ok_to_query</td>
<td>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.</td>
</tr>
<tr>
<td>p_item_id</td>
<td>ID attribute of the form element.</td>
</tr>
<tr>
<td>p_item_label</td>
<td>Invisible label created for the item.</td>
</tr>
</tbody>
</table>

### Example

The following example demonstrates a sample query the generates a popup from an LOV named DEPT_LOV.

```sql
SELECT APEX_ITEM.POPUP_FROM_LOV (1, deptno, 'DEPT_LOV') dt
FROM emp
```
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>Form element name. For example, 1 equals F01 and 2 equals F02. Typically, p_idx is a constant for a given column.</td>
</tr>
<tr>
<td>p_value</td>
<td>Form element current value. This value should be one of the values in the p_lov_query parameter.</td>
</tr>
<tr>
<td>p_lov_query</td>
<td>SQL query that is expected to select two columns (a display column and a return column). For example:</td>
</tr>
<tr>
<td>p_width</td>
<td>Width of the text box.</td>
</tr>
<tr>
<td>p_max_length</td>
<td>Maximum number of characters that can be entered in the text box.</td>
</tr>
<tr>
<td>p_form_index</td>
<td>HTML form on the page in which an item is contained. Defaults to 0 and rarely used.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
### PopUp_From_Query Function

The **PopUp_From_Query** function generates a popup select list from the specified table. The function takes a query as input and returns a select list of results.

#### Example

The following example demonstrates a sample query that generates a popup select list from the `emp` table.

```sql
SELECT APEX_ITEM.POPUP_FROM_QUERY (1, deptno, 'SELECT dname, deptno FROM dept') dt FROM emp
```

### Table 3-8 (Cont.) PopUp_From_Query Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_escape_html</td>
<td>Replacements for special characters that require an escaped equivalent.</td>
</tr>
<tr>
<td></td>
<td>- &lt; for <code>&lt;</code></td>
</tr>
<tr>
<td></td>
<td>- &gt; for <code>&gt;</code></td>
</tr>
<tr>
<td></td>
<td>- &amp; for <code>&amp;</code></td>
</tr>
<tr>
<td></td>
<td>Range of values is YES and NO. If YES, special characters will be escaped.</td>
</tr>
<tr>
<td></td>
<td>This parameter is useful if you know your query will return illegal HTML.</td>
</tr>
<tr>
<td>p_max_elements</td>
<td>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.</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Additional HTML attributes to use for the form item.</td>
</tr>
<tr>
<td>p_ok_to_query</td>
<td>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.</td>
</tr>
<tr>
<td>p_item_id</td>
<td>ID attribute of the form element.</td>
</tr>
<tr>
<td>p_item_label</td>
<td>Invisible label created for the item.</td>
</tr>
</tbody>
</table>
**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**

```sql
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>SELECT APEX_ITEM.POPUPKEY_FROM_LOV (1,deptno,'DEPT') dt, APEX_ITEM.HIDDEN(3,empno) eno</td>
</tr>
<tr>
<td>p_value</td>
<td>Indicates the current value. This value should be one of the values in the P_LOV_NAME parameter.</td>
</tr>
<tr>
<td>p_lov_name</td>
<td>Identifies a named LOV used for this popup.</td>
</tr>
<tr>
<td>p_width</td>
<td>Width of the text box.</td>
</tr>
<tr>
<td>p_max_length</td>
<td>Maximum number of characters that can be entered in the text box.</td>
</tr>
<tr>
<td>p_form_index</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 3-9 (Cont.) POPUPKEY_FROM_LOV Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_escape_html</td>
<td>Replacements for special characters that require an escaped equivalent.</td>
</tr>
<tr>
<td></td>
<td>- &lt; for &lt;</td>
</tr>
<tr>
<td></td>
<td>- &gt; for &gt;</td>
</tr>
<tr>
<td></td>
<td>- &amp; for &amp;</td>
</tr>
<tr>
<td></td>
<td>This parameter is useful if you know your query will return illegal HTML.</td>
</tr>
<tr>
<td>p_max_elements</td>
<td>Limit on the number of rows that can be returned by your query.</td>
</tr>
<tr>
<td></td>
<td>Limits the performance impact of user searches. By entering a value in this</td>
</tr>
<tr>
<td></td>
<td>parameter, you force the user to search for a narrower set of results.</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Additional HTML attributes to use for the form item.</td>
</tr>
<tr>
<td>p_ok_to_query</td>
<td>Range of values is YES and NO. If YES, a popup returns the first set of</td>
</tr>
<tr>
<td></td>
<td>rows for the LOV. If NO, a search is initiated to return rows.</td>
</tr>
<tr>
<td>p_item_id</td>
<td>HTML attribute ID for the &lt;input&gt; tag</td>
</tr>
<tr>
<td>p_item_label</td>
<td>Invisible label created for the item</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>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:</td>
</tr>
<tr>
<td>p_value</td>
<td>Form element current value. This value should be one of the values in the P_LOV_QUERY parameter.</td>
</tr>
<tr>
<td>p_lov_query</td>
<td>LOV query used for this popup.</td>
</tr>
<tr>
<td>p_width</td>
<td>Width of the text box.</td>
</tr>
<tr>
<td>p_max_length</td>
<td>Maximum number of characters that can be entered in the text box.</td>
</tr>
<tr>
<td>p_form_index</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 3–10  POPUPKEY_FROM_QUERY Parameters
The following example demonstrates how to generate a popup select list from a SQL query.

```sql
SELECT APEX_ITEM.POPUPKEY_FROM_QUERY (1,deptno,'SELECT dname, deptno FROM dept')
FROM emp
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_escape_html</td>
<td>Replacements for special characters that require an escaped equivalent.</td>
</tr>
<tr>
<td></td>
<td>- &lt; for &lt;</td>
</tr>
<tr>
<td></td>
<td>- &gt; for &gt;</td>
</tr>
<tr>
<td></td>
<td>- &amp; for &amp;</td>
</tr>
<tr>
<td></td>
<td>This parameter is useful if you know your query will return illegal HTML.</td>
</tr>
<tr>
<td>p_max_elements</td>
<td>Limit on the number of rows that can be returned by your query. Limit the</td>
</tr>
<tr>
<td></td>
<td>performance impact of user searches. By entering a value in this parameter,</td>
</tr>
<tr>
<td></td>
<td>you force the user to search for a narrower set of results.</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Additional HTML attributes to use for the form item.</td>
</tr>
<tr>
<td>p_ok_to_query</td>
<td>Range of values is YES and NO. If YES, a popup returns first set of rows for</td>
</tr>
<tr>
<td></td>
<td>the LOV. If NO, a search is initiated to return rows.</td>
</tr>
<tr>
<td>p_item_id</td>
<td>ID attribute of the form element.</td>
</tr>
<tr>
<td>p_item_label</td>
<td>Invisible label created for the item.</td>
</tr>
</tbody>
</table>
RADIOGROUP Function

This function generates a radio group from a SQL query.

Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>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.</td>
</tr>
<tr>
<td>p_value</td>
<td>Value of the radio group.</td>
</tr>
<tr>
<td>p_selected_value</td>
<td>Value that should be selected.</td>
</tr>
<tr>
<td>p_display</td>
<td>Text to display next to the radio option.</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Extra HTML parameters you want to add.</td>
</tr>
<tr>
<td>p_onblur</td>
<td>JavaScript to execute in the onBlur event.</td>
</tr>
<tr>
<td>p_onchange</td>
<td>JavaScript to execute in the onChange event.</td>
</tr>
<tr>
<td>p_onfocus</td>
<td>JavaScript to execute in the onFocus event.</td>
</tr>
<tr>
<td>p_item_id</td>
<td>HTML attribute ID for the &lt;input&gt; tag.</td>
</tr>
<tr>
<td>p_item_label</td>
<td>Invisible label created for the item.</td>
</tr>
</tbody>
</table>

Example

The following example demonstrates how to select department 20 from the emp table as a default in a radio group.

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>Form element name. For example, 1 equals F01 and 2 equals F02. Typically the p_IDX parameter is constant for a given column.</td>
</tr>
<tr>
<td>p_value</td>
<td>Current value. This value should be a value in the P_LIST_VALUES parameter.</td>
</tr>
<tr>
<td>p_list_values</td>
<td>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 SELECT_LIST function.</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Extra HTML parameters you want to add.</td>
</tr>
<tr>
<td>p_show_null</td>
<td>Extra select option to enable the NULL selection. Range of values is YES and NO.</td>
</tr>
<tr>
<td>p_null_value</td>
<td>Value to be returned when a user selects the NULL option. Only relevant when p_show_null equals YES.</td>
</tr>
<tr>
<td>p_null_text</td>
<td>Value to be displayed when a user selects the NULL option. Only relevant when p_show_null equals YES.</td>
</tr>
<tr>
<td>p_item_id</td>
<td>HTML attribute ID for the &lt;input&gt; tag.</td>
</tr>
<tr>
<td>p_item_label</td>
<td>Invisible label created for the item.</td>
</tr>
<tr>
<td>p_show_extra</td>
<td>Shows the current value even if the value of p_value is not located in the select list.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>Form element name. For example, 1 equals F01 and 2 equals F02. Typically, the p_idx parameter is constant for a given column.</td>
</tr>
<tr>
<td>p_value</td>
<td>Current value. This value should be a value in the p_lov parameter.</td>
</tr>
<tr>
<td>p_lov</td>
<td>Text name of an application list of values. This list of values must be defined in your application. This parameter is used only by the select_list_from_lov function.</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Extra HTML parameters you want to add.</td>
</tr>
<tr>
<td>p_show_null</td>
<td>Extra select option to enable the NULL selection. Range of values is YES and NO.</td>
</tr>
<tr>
<td>p_null_value</td>
<td>Value to be returned when a user selects the NULL option. Only relevant when p_show_null equals YES.</td>
</tr>
<tr>
<td>p_null_text</td>
<td>Value to be displayed when a user selects the NULL option. Only relevant when p_show_null equals YES.</td>
</tr>
<tr>
<td>p_item_id</td>
<td>HTML attribute ID for the &lt;select&gt; tag.</td>
</tr>
<tr>
<td>p_item_label</td>
<td>Invisible label created for the item.</td>
</tr>
<tr>
<td>p_show_extra</td>
<td>Shows the current value even if the value of p_value is not located in the select list.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>Form element name. For example, 1 equals F01 and 2 equals F02. Typically, the p_idx parameter is constant for a given column.</td>
</tr>
<tr>
<td>p_value</td>
<td>Current value. This value should be a value in the p_lov parameter.</td>
</tr>
<tr>
<td>p_lov</td>
<td>Text name of a list of values. This list of values must be defined in your application. This parameter is used only by the select_list_from_lov function.</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Extra HTML parameters you want to add.</td>
</tr>
<tr>
<td>p_show_null</td>
<td>Extra select option to enable the NULL selection. Range of values is YES and NO.</td>
</tr>
<tr>
<td>p_null_value</td>
<td>Value to be returned when a user selects the NULL option. Only relevant when p_show_null equals YES.</td>
</tr>
<tr>
<td>p_null_text</td>
<td>Value to be displayed when a user selects the NULL option. Only relevant when p_show_null equals YES.</td>
</tr>
<tr>
<td>p_item_id</td>
<td>HTML attribute ID for the &lt;select&gt; tag.</td>
</tr>
<tr>
<td>p_item_label</td>
<td>Invisible label created for the item.</td>
</tr>
<tr>
<td>p_show_extra</td>
<td>Shows the current value even if the value of p_value is not located in the select list.</td>
</tr>
</tbody>
</table>
Example
The following example demonstrates how to create a select list based on an LOV defined in the application.

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>Form element name. For example, 1 equals F01 and 2 equals F02. Typically, the p_idx parameter is constant for a given column.</td>
</tr>
<tr>
<td>p_value</td>
<td>Current value. This value should be a value in the p_query parameter.</td>
</tr>
</tbody>
</table>
| 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.

```sql
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_FROMQUERY_XL Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>Form element name. For example, 1 equals F01 and 2 equals F02. Typically the p_idx parameter is constant for a given column.</td>
</tr>
<tr>
<td>p_value</td>
<td>Current value. This value should be a value in the p_query parameter.</td>
</tr>
<tr>
<td>p_query</td>
<td>SQL query that is expected to select two columns, a display column, and a return column. For example: SELECT dname, deptno FROM dept</td>
</tr>
<tr>
<td></td>
<td>Note that this is used only by the SELECT_LIST_FROM_QUERY_XL function.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Extra HTML parameters you want to add.</td>
</tr>
<tr>
<td>p_show_null</td>
<td>Extra select option to enable the NULL selection. Range of values is YES and NO.</td>
</tr>
<tr>
<td>p_null_value</td>
<td>Value to be returned when a user selects the NULL option. Only relevant when p_show_null equals YES.</td>
</tr>
<tr>
<td>p_null_text</td>
<td>Value to be displayed when a user selects the NULL option. Only relevant when p_show_null equals YES.</td>
</tr>
<tr>
<td>p_item_id</td>
<td>HTML attribute ID for the &lt;select&gt; tag.</td>
</tr>
<tr>
<td>p_item_label</td>
<td>Invisible label created for the item.</td>
</tr>
</tbody>
</table>
Table 3–16  (Cont.) SELECT_LIST_FROM_QUERY_XL Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_show_extra</td>
<td>Show the current value even if the value of p_value is not located in the select list.</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>Number to identify the item you want to generate. The number will determine</td>
</tr>
<tr>
<td></td>
<td>which G_FXX global is populated.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;APEX_APPLICATION&quot; on page 4-1</td>
</tr>
<tr>
<td>p_value</td>
<td>Value of a text field item.</td>
</tr>
<tr>
<td>p_size</td>
<td>Controls HTML tag attributes (such as disabled).</td>
</tr>
<tr>
<td>p_maxlength</td>
<td>Maximum number of characters that can be entered in the text box.</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Extra HTML parameters you want to add.</td>
</tr>
<tr>
<td>p_item_id</td>
<td>HTML attribute ID for the <code>&lt;input&gt;</code> tag.</td>
</tr>
<tr>
<td>p_item_label</td>
<td>Invisible label created for the item.</td>
</tr>
</tbody>
</table>

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
```
This function creates text areas.

**Syntax**

```sql
APEX_ITEM.TEXTAREA(
  p_idx IN NUMBER,
  p_value IN VARCHAR2 DEFAULT NULL,
  p_rows IN NUMBER DEFAULT 40,
  p_cols IN NUMBER DEFAULT 4,
  p_attributes IN VARCHAR2 DEFAULT NULL,
  p_item_id IN VARCHAR2 DEFAULT NULL,
  p_item_label IN VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

**Parameters**

Table 3–18 describes the parameters available in the TEXTAREA function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_idx</td>
<td>Number to identify the item you want to generate. The number will determine which G_FXX global is populated. See Also: &quot;APEX_APPLICATION&quot; on page 4-1</td>
</tr>
<tr>
<td>p_value</td>
<td>Value of the text area item.</td>
</tr>
<tr>
<td>p_rows</td>
<td>Height of the text area (HTML rows attribute)</td>
</tr>
<tr>
<td>p_cols</td>
<td>Width of the text area (HTML column attribute).</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Extra HTML parameters you want to add.</td>
</tr>
<tr>
<td>p_item_id</td>
<td>HTML attribute ID for the <code>&lt;textarea&gt;</code> tag.</td>
</tr>
<tr>
<td>p_item_label</td>
<td>Invisible label created for the item.</td>
</tr>
</tbody>
</table>

**Example**

The following example demonstrates how to create a text area based on a SQL query.

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_value</td>
<td>Value of a field item. Note that if p_value is not located in the list of</td>
</tr>
<tr>
<td></td>
<td>values, p_null_text is value displayed.</td>
</tr>
<tr>
<td>p_lov</td>
<td>Text name of a shared list of values. This list of values must be defined in</td>
</tr>
<tr>
<td></td>
<td>your application.</td>
</tr>
<tr>
<td>p_null_text</td>
<td>Value displayed when the value of the field item is NULL.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_value</td>
<td>Value of a field item.</td>
</tr>
<tr>
<td>p_query</td>
<td>SQL query that is expected to select two columns, a display column and a return column. For example: SELECT dname, deptno FROM dept. Note if only one column is specified in the select clause of this query, the value for this column will be used for both display and return purposes.</td>
</tr>
<tr>
<td>p_null_text</td>
<td>Value to be displayed when the value of the field item is NULL or a corresponding entry is not located for the value p_value in the list of values query.</td>
</tr>
</tbody>
</table>

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
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>
<thead>
<tr>
<th>Global Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G_USER</td>
<td>Specifies the currently logged in user.</td>
</tr>
<tr>
<td>G_FLOW_ID</td>
<td>Specifies the ID of the currently running application.</td>
</tr>
<tr>
<td>G_FLOW_STEP_ID</td>
<td>Specifies the ID of the currently running page.</td>
</tr>
<tr>
<td>G_FLOW_OWNER</td>
<td>Specifies the schema to parse for the currently running application.</td>
</tr>
<tr>
<td>G_REQUEST</td>
<td>Specifies the value of the request variable most recently passed to or set within the show or accept modules.</td>
</tr>
<tr>
<td>G_BROWSER_LANGUAGE</td>
<td>Refers to the Web browser's current language preference.</td>
</tr>
<tr>
<td>G_DEBUG</td>
<td>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.</td>
</tr>
<tr>
<td>G_HOME_LINK</td>
<td>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.</td>
</tr>
<tr>
<td>G_LOGIN_URL</td>
<td>Can be used to display a link to a login page for users that are not currently logged in.</td>
</tr>
<tr>
<td>G_IMAGE_PREFIX</td>
<td>Refers to the virtual path the web server uses to point to the images directory distributed with Oracle Application Express.</td>
</tr>
<tr>
<td>G_FLOW_SCHEMA_OWNER</td>
<td>Refers to the owner of the Application Express schema.</td>
</tr>
<tr>
<td>G_PRINTER_FRIENDLY</td>
<td>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.</td>
</tr>
<tr>
<td>G_PROXY_SERVER</td>
<td>Refers to the application attribute 'Proxy Server'.</td>
</tr>
<tr>
<td>G_SYSDATE</td>
<td>Refers to the current date on the database server. This uses the DATE DATATYPE.</td>
</tr>
<tr>
<td>G_PUBLIC_USER</td>
<td>Refers to the Oracle schema used to connect to the database through the database access descriptor (DAD).</td>
</tr>
<tr>
<td>G_GLOBAL_NOTIFICATION</td>
<td>Specifies the application's global notification attribute.</td>
</tr>
</tbody>
</table>
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:

```html
<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>
  <OPTION VALUE="123">123</OPTION>
</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:

```html
<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.

```plsql
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;
```
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:

```plsql
FOR i IN 1..APEX_APPLICATION.G_F01.COUNT LOOP
    htp.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.
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:

```plaintext
htp.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:

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_request</td>
<td>Not used.</td>
</tr>
<tr>
<td>p_flow_id</td>
<td>The application ID that contains the page or item level help you want to output.</td>
</tr>
<tr>
<td>p_flow_step_id</td>
<td>The page ID that contains the page or item level help you want to display.</td>
</tr>
<tr>
<td>p_show_item_help</td>
<td>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'.</td>
</tr>
<tr>
<td>p_show_regions</td>
<td>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'.</td>
</tr>
<tr>
<td>p_before_page_html</td>
<td>Use this parameter to include HTML between the page level help text and item level help text.</td>
</tr>
<tr>
<td>p_after_page_html</td>
<td>Use this parameter to include HTML at the bottom of the output, after all other help.</td>
</tr>
<tr>
<td>p_before_region_html</td>
<td>Use this parameter to include HTML before every region section. Note this parameter is ignored if p_show_regions is set to 'NO'.</td>
</tr>
<tr>
<td>p_after_region_html</td>
<td>Use this parameter to include HTML after every region section. Note this parameter is ignored if p_show_regions is set to 'NO'.</td>
</tr>
<tr>
<td>p_before_prompt_html</td>
<td>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'.</td>
</tr>
</tbody>
</table>
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_region_html  => '</b></p>&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.

### Table 4–2 (Cont.) HELP Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_after_prompt_html</td>
<td>Use this parameter to include HTML after every item label for item level help. Note this parameter is ignored if p_show_item_help is set to 'NO'.</td>
</tr>
<tr>
<td>p_before_item_html</td>
<td>Use this parameter to include HTML before every item help text for item level help. Note this parameter is ignored if p_show_item_help is set to 'NO'.</td>
</tr>
<tr>
<td>p_after_item_html</td>
<td>Use this parameter to include HTML after every item help text for item level help. Note this parameter is ignored if p_show_item_help is set to 'NO'.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p_item_name</code></td>
<td>The name of the page-level item.</td>
</tr>
</tbody>
</table>

**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
        htp.p('Page is Public');
    ELSE
        htp.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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user</td>
<td>Login name of the user.</td>
</tr>
<tr>
<td>p_session_id</td>
<td>The session ID.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_app_id</td>
<td>An application ID in the current workspace.</td>
</tr>
<tr>
<td>p_cookie_name</td>
<td>The cookie name.</td>
</tr>
<tr>
<td>p_cookie_path</td>
<td>The cookie path.</td>
</tr>
<tr>
<td>p_cookie_domain</td>
<td>The cookie domain.</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_ldap_host</td>
<td>LDAP host name.</td>
</tr>
<tr>
<td>p_ldap_port</td>
<td>LDAP port number.</td>
</tr>
<tr>
<td>p_ldap_dn</td>
<td>LDAP DN string.</td>
</tr>
<tr>
<td>p_ldap_edit_function</td>
<td>LDAP edit function name.</td>
</tr>
</tbody>
</table>

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_uname                    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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_uname</td>
<td>Login name of the user.</td>
</tr>
<tr>
<td>p_password</td>
<td>Clear text user password.</td>
</tr>
<tr>
<td>p_session_id</td>
<td>Current Oracle Application Express session ID.</td>
</tr>
<tr>
<td>p_app_page</td>
<td>Current application ID. After login page separated by a colon (:).</td>
</tr>
<tr>
<td>p_entry_point</td>
<td>Internal use only.</td>
</tr>
<tr>
<td>p_preserve_case</td>
<td>If true, do not upper p_uname during session registration</td>
</tr>
</tbody>
</table>

Example

The following example performs the user authentication and session registration.

```
BEGIN
    APEX_CUSTOM_AUTH.LOGIN ( 
        p_uname => '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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_this_app</td>
<td>Current application ID.</td>
</tr>
<tr>
<td>p_next_app_page_sess</td>
<td>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).</td>
</tr>
<tr>
<td>p_next_url</td>
<td>URL to redirect to (use this instead of p_next_app_page_sess).</td>
</tr>
</tbody>
</table>

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**

```sql
APEX_CUSTOM_AUTH.POST_LOGIN(  
  p_uname                    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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_uname</td>
<td>Login name of user.</td>
</tr>
<tr>
<td>p_session_id</td>
<td>Current Oracle Application Express session ID.</td>
</tr>
<tr>
<td>p_app_page</td>
<td>Current application ID and after login page separated by a colon (:).</td>
</tr>
<tr>
<td>p_preserve_case</td>
<td>If true, do not include p_uname in uppercase during session registration.</td>
</tr>
</tbody>
</table>

**Example**

The following example performs the session registration following a successful authentication.

```sql
BEGIN  
  APEX_CUSTOM_AUTH.POST_LOGIN (  
    p_uname => '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 L_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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_session_id</td>
<td>The session ID to be registered.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_user</td>
<td>The user ID to be registered.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>Login name of the user.</td>
</tr>
<tr>
<td>p_password</td>
<td>Password for p_username.</td>
</tr>
<tr>
<td>p_search_base</td>
<td>LDAP search base, for example, dc=users,dc=my,dc=org.</td>
</tr>
<tr>
<td>p_host</td>
<td>LDAP server host name.</td>
</tr>
<tr>
<td>p_port</td>
<td>LDAP server port number.</td>
</tr>
</tbody>
</table>

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_server.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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>Login name of the user.</td>
</tr>
<tr>
<td>p_pass</td>
<td>Password for p_username.</td>
</tr>
<tr>
<td>p_auth_base</td>
<td>LDAP search base, for example, dc=users,dc=my,dc=org.</td>
</tr>
<tr>
<td>p_host</td>
<td>LDAP server host name.</td>
</tr>
<tr>
<td>p_port</td>
<td>LDAP server port number.</td>
</tr>
<tr>
<td>p_attributes</td>
<td>An array of attribute names returned.</td>
</tr>
<tr>
<td>p_attribute_values</td>
<td>An array of values returned for each corresponding attribute name returned in p_attributes.</td>
</tr>
</tbody>
</table>

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_server.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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>Login name of the user.</td>
</tr>
<tr>
<td>p_pass</td>
<td>Password for p_username.</td>
</tr>
<tr>
<td>p_auth_base</td>
<td>LDAP search base, for example, dc=users, dc=my, dc=org.</td>
</tr>
<tr>
<td>p_host</td>
<td>LDAP server host name.</td>
</tr>
<tr>
<td>p_port</td>
<td>LDAP server port number.</td>
</tr>
<tr>
<td>p_attributes</td>
<td>An array of attribute names for which values are to be returned.</td>
</tr>
<tr>
<td>p_attribute_values</td>
<td>An array of values returned for each corresponding attribute name in p_attributes.</td>
</tr>
</tbody>
</table>

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) := 'xxxxxxxxxx'; /* name of the employee number attribute */
    APEX_LDAP.GET_USER_ATTRIBUTES(
        p_username => 'firstname.lastname',
        p_pass => NULL,
        p_auth_base => 'cn=user,l=amer,dc=my_company,dc=com',
        p_host => 'our_ldap_server.my_company.com',
        p_port => '389',
        p_attributes => L_ATTRIBUTES,
        p_attribute_values => L_ATTRIBUTE_VALUES);
END;
The IS_MEMEber function returns a boolean true if the user named by \texttt{p\_username} (with password if required) is a member of the group specified by the \texttt{p\_group} and \texttt{p\_group\_base} parameters using the provided auth base, host, and port.

### Syntax

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>Login name of the user.</td>
</tr>
<tr>
<td>p_pass</td>
<td>Password for p_username.</td>
</tr>
<tr>
<td>p_auth_base</td>
<td>LDAP search base, for example, \texttt{dc=users,dc=my,dc=org}.</td>
</tr>
<tr>
<td>p_host</td>
<td>LDAP server host name.</td>
</tr>
<tr>
<td>p_port</td>
<td>LDAP server port number.</td>
</tr>
<tr>
<td>p_group</td>
<td>Name of the group to be search for membership.</td>
</tr>
<tr>
<td>p_group_base</td>
<td>The base from which the search should be started.</td>
</tr>
</tbody>
</table>

### 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.

```sql
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_server.my_company.com',
        p_port => 389,
        p_group => 'group_name',
        p_group_base => 'group_base');
    IF L_VAL THEN
        htp.p('Is a member.');
    ELSE
        htp.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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>Login name of the user.</td>
</tr>
<tr>
<td>p_pass</td>
<td>Password for p_username.</td>
</tr>
<tr>
<td>p_auth_base</td>
<td>LDAP search base, for example, dc=users,dc=my,dc=org.</td>
</tr>
<tr>
<td>p_host</td>
<td>LDAP server host name.</td>
</tr>
<tr>
<td>p_port</td>
<td>LDAP server port number.</td>
</tr>
</tbody>
</table>

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_server.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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>Login name of the user.</td>
</tr>
<tr>
<td>p_pass</td>
<td>Password for p_username.</td>
</tr>
<tr>
<td>p_auth_base</td>
<td>LDAP search base, for example, dc=users,dc=my,dc=org.</td>
</tr>
<tr>
<td>p_host</td>
<td>LDAP server host name.</td>
</tr>
<tr>
<td>p_port</td>
<td>LDAP server port number.</td>
</tr>
</tbody>
</table>

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_server.my_company.com',
    p_port => 389);
  htp.p('Is Member of:'||L_VAL);
END;
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
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_workspace</td>
<td>The name of the workspace to which the schema mapping will be added.</td>
</tr>
<tr>
<td>p_schema</td>
<td>The schema to add to the schema to workspace mapping.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_workspace_id</td>
<td>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.</td>
</tr>
<tr>
<td>p_workspace</td>
<td>The name of the workspace to be added.</td>
</tr>
<tr>
<td>p_primary_schema</td>
<td>The primary database schema to associate with the new workspace.</td>
</tr>
<tr>
<td>p_additional_schemas</td>
<td>A colon delimited list of additional schemas to associate with this workspace.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_parameter</td>
<td>The instance parameter to be retrieved.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Available Parameter Values&quot; on page 7-10.</td>
</tr>
</tbody>
</table>

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;
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_workspace</td>
<td>The name of the workspace from which to retrieve the schema list.</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application_id</td>
<td>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.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_workspace</td>
<td>The name of the workspace from which the schema mapping will be removed.</td>
</tr>
<tr>
<td>p_schema</td>
<td>The schema to remove from the schema to workspace mapping.</td>
</tr>
</tbody>
</table>

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;
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_workspace</td>
<td>The name of the workspace to be removed.</td>
</tr>
<tr>
<td>p_drop_users</td>
<td>'Y' to drop the database user associated with the workspace. The default is 'N'.</td>
</tr>
<tr>
<td>p_drop_tablespaces</td>
<td>'Y' to drop the tablespace associated with the database user associated with the workspace. The default is 'N'.</td>
</tr>
</tbody>
</table>

**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
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_parameter</td>
<td>The instance parameter to be set.</td>
</tr>
<tr>
<td>p_value</td>
<td>The value of the parameter.</td>
</tr>
<tr>
<td></td>
<td>See &quot;Available Parameter Values&quot; on page 7-10.</td>
</tr>
</tbody>
</table>

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;
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>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SMTP_FROM</strong></td>
<td>Defines the “from” address for administrative tasks that generate email, such as approving a provision request or resetting a password. Enter a valid email address, for example: <a href="mailto:someone@somewhere.com">someone@somewhere.com</a></td>
</tr>
<tr>
<td><strong>SMTP_HOST_ADDRESS</strong></td>
<td>Defines the server address of the SMTP server. If you are using another server as an SMTP relay, change this parameter to that server's address. Default setting: localhost</td>
</tr>
<tr>
<td><strong>SMTP_HOST_PORT</strong></td>
<td>Defines the port the SMTP server listens to for mail requests. Default setting: 25</td>
</tr>
<tr>
<td><strong>WALLET_PATH</strong></td>
<td>The path to the wallet on the file system, for example: file:/home/&lt;username&gt;/wallets</td>
</tr>
<tr>
<td><strong>WALLET_PWD</strong></td>
<td>The password associated with the wallet.</td>
</tr>
<tr>
<td><strong>PRINT_BIB LICENSED</strong></td>
<td>Specify either standard support or advanced support. Advanced support requires an Oracle BI Publisher license. Valid values include:</td>
</tr>
<tr>
<td></td>
<td>■ STANDARD</td>
</tr>
<tr>
<td></td>
<td>■ ADVANCED</td>
</tr>
<tr>
<td><strong>PRINT_SVR_PROTOCOL</strong></td>
<td>Valid values include: http, https</td>
</tr>
<tr>
<td><strong>PRINT_SVR_HOST</strong></td>
<td>Specifies the host address of the print server converting engine, for example, localhost. Enter the appropriate host address if the print server is installed at another location.</td>
</tr>
<tr>
<td><strong>PRINT_SVR_PORT</strong></td>
<td>Defines the port of the print server engine, for example 8888. Value must be a positive integer.</td>
</tr>
<tr>
<td><strong>PRINT_SVR_SCRIPT</strong></td>
<td>Defines the script that is the print server engine, for example: /xmlpserver/convert</td>
</tr>
</tbody>
</table>
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.
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
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Table name</td>
</tr>
<tr>
<td>p_column_name</td>
<td>Column name</td>
</tr>
<tr>
<td>p_display_in_form</td>
<td>Determines whether or not to display in the form by default, valid values are Y and N</td>
</tr>
</tbody>
</table>

**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');  
```
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Table name</td>
</tr>
<tr>
<td>p_column_name</td>
<td>Column name</td>
</tr>
<tr>
<td>p_display_in_report</td>
<td>Determines whether or not to display in the report by default, valid values are Y and N</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Table name</td>
</tr>
<tr>
<td>p_form_region_title</td>
<td>Desired form region title</td>
</tr>
</tbody>
</table>

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 => 'Deptartment 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
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Table name</td>
</tr>
<tr>
<td>p_column_name</td>
<td>Column name</td>
</tr>
<tr>
<td>p_display_height</td>
<td>Display height of any items created based upon this column</td>
</tr>
</tbody>
</table>

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_ITEMDISPLAY_HEIGHT (  
  p_table_name => 'DEPT',  
  p_column_name => 'DNAME',  
  p_display_height => 3);
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.

**Syntax**

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Table name</td>
</tr>
<tr>
<td>p_column_name</td>
<td>Column name</td>
</tr>
<tr>
<td>p_display_width</td>
<td>Display width of any items created based upon this column</td>
</tr>
</tbody>
</table>

**Example**

The following example sets a default item width of 5 when creating an item on the DEPTNO column against the DEPT table.

```sql
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**

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Table name</td>
</tr>
<tr>
<td>p_column_name</td>
<td>Column name</td>
</tr>
<tr>
<td>p_format_mask</td>
<td>Format mask to be associated with the column</td>
</tr>
</tbody>
</table>

**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'.

```sql
APEX_UI_DEFAULT_UPDATE.UPD_ITEM_FORMAT_MASK(  
    p_table_name => 'EMP',  
    p_column_name => 'HIREDATE',  
    p_format_mask => 'DD-MON-YYYY');
```
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**

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Table name</td>
</tr>
<tr>
<td>p_column_name</td>
<td>Column name</td>
</tr>
<tr>
<td>p_help_text</td>
<td>Desired help text</td>
</tr>
</tbody>
</table>

**Example**

This example demonstrates how to set the User Interface Item Help Text default for the DEPTNO column in the DEPT table.

```sql
APEX_UI_DEFAULT_UPDATE.UPD_ITEM_HELP (  
    p_table_name => 'DEPT',
    p_column_name => 'DEPTNO',
    p_help_text => 'The number assigned to the department.');
```
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Table name</td>
</tr>
<tr>
<td>p_column_name</td>
<td>Column name</td>
</tr>
<tr>
<td>p_label</td>
<td>Desired item label</td>
</tr>
</tbody>
</table>

**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.UDP_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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Table name.</td>
</tr>
<tr>
<td>p_column_name</td>
<td>Column name.</td>
</tr>
<tr>
<td>p_report_alignment</td>
<td>Defines the alignment of the column in a report. Valid values are L (left), C (center) and R (right).</td>
</tr>
</tbody>
</table>

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.UDP_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**

```sql
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Table name</td>
</tr>
<tr>
<td>p_column_name</td>
<td>Column name</td>
</tr>
<tr>
<td>p_format_mask</td>
<td>Format mask to be associated with the column whenever it is included in a report</td>
</tr>
</tbody>
</table>

**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'.

```sql
APEX_UI_DEFAULT_UPDATE.UPD_REPORT_FORMAT_MASK(  
p_table_name => 'EMP',  
p_column_name => 'HIREDATE',  
p_format_mask=> 'DD-MON-YYYY');
```
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Table name</td>
</tr>
<tr>
<td>p_report_region_title</td>
<td>Desired report region title</td>
</tr>
</tbody>
</table>

**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 => 'Deptartments');
```
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)
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)
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)
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)
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)
$x_{\text{Hide}}(pNd)$

Hides a DOM node or array of DOM nodes ($pNd$).

**Return Value**

(DOM node | Array)

**Parameters**

$pNd$ (DOM node | string ID | DOM node Array)
$x_{\text{Show}}(pNd)$

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)
$x_{\text{Remove}}(pNd)$

Removes a DOM node or array of DOM nodes.

**Return Value**

(DOM Node | Array)

**Parameters**

pNd (DOM node | string ID | DOM node Array)
$x_{\text{Value}}(p\text{Nd}, p\text{Value})$

Sets the value ($p\text{Value}$) of a DOM node or array of DOM nodes ($p\text{Nd}$).

**Return Value**

Not applicable.

**Parameters**

$p\text{Nd}$ (DOM node | string ID | DOM node Array)  
$p\text{Value}$ (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)
$x_{\text{ShowItemRow}}(pNd)$

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)
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 )
$x_{\text{Class}}(pNd,pClass)$

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 Node | 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)
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_emptys(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)
$f_{\text{SelectedOptions}}(pNd)$

Returns the DOM nodes of the selected options of a select item ($pNd$).

**Return Value**

(DOM Array)

**Parameters**

$pNd$ (DOM node | string ID)
$f_{SelectValue}(pNd)$

Returns the values of the selected options of a select item ($pNd$).

**Return Value**

(DOM Array | String)

**Parameters**

$pNd$ (DOM node | string ID)
$u_ArrayToString(pArray, pDelim)

Given an array (pArray) return a string with the values of the array delimited with a given delimiter character (pDelim).

**Return Value**
Not applicable.

**Parameters**
- pArray (pArray)
- pDelim (String)
$x_CheckImageSrc(pld,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**

pld (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)
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)
$f_{\text{CheckAll}}(p\text{This}, p\text{Check}, p\text{Array})$

Check or uncheck ($p\text{Check}$) all check boxes contained within a DOM node ($p\text{This}$). If an array of checkboxes DOM nodes ($p\text{Array}$) is provided, use that array for affected check boxes.

**Return Value**
Not applicable.

**Parameters**
- $p\text{This}$ (DOM node | string ID)
- $p\text{Check}$ (true | false)
- $p\text{Array}$ (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 (\texttt{pThat}), with (\texttt{pValue}) and then closes the popup window.

\textbf{Return Value}
Not applicable.

\textbf{Parameters}
- \texttt{pValue} (string)
- \texttt{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 Node | string ID | DOM node Array)
$x_{\text{SwitchImageSrc}}(\text{pNd}, \text{pSearch}, \text{pReplace})$

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)
$x_{\text{CheckImageSrc}}(pNd, pSearch)$

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)
$f_{\text{Swap}}(pThis,pThat)$

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 a 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 a 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_R (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
Places the user focus on the 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)
SetCookie (pName,pValue)
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;
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_job</td>
<td>The job number that identifies the submitted job you wish to purge.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_sql</td>
<td>The process you wish to run in your job. This can be any valid anonymous block, for example: <code>BEGIN &lt;your code&gt; END;' or </code>DECLARE &lt;your declaration&gt; BEGIN &lt;your code&gt; END;'</td>
</tr>
<tr>
<td>p_when</td>
<td>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: <code>sysdate + 1</code> - The job will run in 1 days time. <code>sysdate + (1/24)</code> - The job will run in 1 hours time. <code>sysdate + (10/24/60)</code> - The job will run in 10 minutes time.</td>
</tr>
<tr>
<td>p_status</td>
<td>Plain text status information for this job.</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_job</td>
<td>The job ID for the job you wish to see how long since it was submitted.</td>
</tr>
</tbody>
</table>

**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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_job</td>
<td>Passed the reserved word JOB. When this code is executed it will have visibility to the job number via the reserved word JOB.</td>
</tr>
<tr>
<td>p_status</td>
<td>Plain text that you want associated with JOB: p_job.</td>
</tr>
</tbody>
</table>

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);
  END LOOP;
END;
```
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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_primary_text_string</td>
<td>Text string of the primary language. This will be the value of the Translate From Text in the dynamic translation.</td>
</tr>
<tr>
<td>p_p0 through p_p9</td>
<td>Dynamic substitution value: p0 corresponds to 0% in the translation string; p1 corresponds to 1% in the translation string; p2 corresponds to 2% in the translation string, and so on.</td>
</tr>
<tr>
<td>p_primary_language</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>See also: Specifying the Primary Language for an Application in the Oracle Application Express Application Builder User’s Guide.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_name</td>
<td>Name of the message as defined in Shared Components &gt; Text Messages of your application in Oracle Application Express.</td>
</tr>
<tr>
<td>p_p0 through p_p9</td>
<td>Dynamic substitution value: p0 corresponds to 0% in the translation string; p1 corresponds to 1% in the translation string; p2 corresponds to 2% in the translation string, and so on.</td>
</tr>
<tr>
<td>p_lang</td>
<td>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 Oracle Application Express Application Builder User’ s Guide.</td>
</tr>
</tbody>
</table>

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
  UPD_DISPLAY_IN_FORM procedure, 8-2
  UPD_DISPLAY_IN_REPORT procedure, 8-3

Index-1
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
attribute values
setting, 1-91

authenticated user
create user group, 1-18

authentication
scheme session cookies, 5-5

C
check box
creating, 3-2

clips
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,tag,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
$nvl(pTest, pDefault), 9-9
$p(pNd, pValue), 9-6
$str_AddTD(pThis,pText), 9-65
$str_AddTH(pThis,pText), 9-66
$u_ArrayToString(pThis,pTag), 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
$s(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_Toggle(pNd), 9-15
$x_ToggleItemRow(pNd), 9-22
$x_ToggleWithImage(pThis,pNd), 9-53
$x_Target(pNd), 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

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

Index-4