Oracle® Application Express
API Reference
Release 4.2
E35127-05

August 2013
## Contents

**Preface**  
- Topic Overview .......................................................................................................................... xix  
- Audience ................................................................................................................................. xix  
- Documentation Accessibility ....................................................................................................... xxi  
- Related Documents ................................................................................................................... xxii  
- Conventions ............................................................................................................................... xxii

**Changes in This Release** .............................................................................................................. xxv  
  Changes in Oracle Application Express Release 4.2 ................................................................. xxv

### 1 APEX_APPLICATION
- Referencing Arrays .................................................................................................................. 1-3  
- Referencing Values Within an On Submit Process ................................................................. 1-4  
- Converting an Array to a Single Value .................................................................................. 1-5  
- HELP Procedure .................................................................................................................... 1-6  
- STOP_APEX_ENGINE Procedure ......................................................................................... 1-8

### 2 APEX_APPLICATION_INSTALL
- Package Overview ................................................................................................................... 2-2  
- Import Script Examples .......................................................................................................... 2-4  
- CLEAR_ALL Procedure .......................................................................................................... 2-7  
- GENERATE_APPLICATION_ID Procedure ....................................................................... 2-8  
- GENERATE_OFFSET Procedure ......................................................................................... 2-9  
- GET_APPLICATION_ALIAS Function ............................................................................... 2-10  
- GET_APPLICATION_ID Function ....................................................................................... 2-11  
- GET_APPLICATION_NAME Function ............................................................................... 2-12  
- GET_IMAGE_PREFIX Function ......................................................................................... 2-13  
- GET_OFFSET Function ....................................................................................................... 2-14  
- GET_PROXY Function .......................................................................................................... 2-15  
- GET_SCHEMA Function ....................................................................................................... 2-16  
- GET_WORKSPACE_ID Function ......................................................................................... 2-17  
- SET_APPLICATION_ALIAS Procedure .............................................................................. 2-18
SET_APPLICATION_ID Procedure ................................................................. 2-19
SET_APPLICATION_NAME Procedure ........................................................... 2-20
SET_IMAGE_PREFIX Procedure ...................................................................... 2-21
SET_OFFSET Procedure .................................................................................. 2-22
SET_PROXY Procedure ................................................................................... 2-23
SET_SCHEMA Procedure ................................................................................ 2-24
SET_WORKSPACE_ID Procedure ................................................................. 2-25

3 APEX_AUTHENTICATION

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

4 APEX_COLLECTION

About the APEX_COLLECTION API ............................................................... 4-3
Naming, Creating and Accessing Collections .................................................. 4-4
  Naming Collections ......................................................................................... 4-4
  Creating a Collection ..................................................................................... 4-4
  About the Parameter p_generate_md5 ........................................................... 4-5
  Accessing a Collection .................................................................................. 4-5
Merging, Truncating and Deleting Collections .................................................. 4-7
  Merging Collections ....................................................................................... 4-7
  Truncating a Collection ................................................................................ 4-7
  Deleting a Collection .................................................................................... 4-7
  Deleting All Collections for the Current Application ..................................... 4-7
  Deleting All Collections in the Current Session .......................................... 4-7
Adding, Updating and Deleting Collection Members ......................................... 4-8
  Adding Members to a Collection .................................................................. 4-8
  About the Parameters p_generate_md5, p_clob001, p_blob001, and p_xmltype001
  Updating Collection Members ...................................................................... 4-8
  Deleting Collection Members ....................................................................... 4-9
Managing Collections ....................................................................................... 4-10
  Obtaining a Member Count .......................................................................... 4-10
  Resequencing a Collection ......................................................................... 4-10
5  APEX_CSS

ADD Procedure ................................................................. 5-2
ADD_3RD_PARTY_LIBRARY_FILE Procedure .............................. 5-3
ADD_FILE Procedure .......................................................... 5-4

6  APEX_CUSTOM_AUTH

APPLICATION_PAGE_ITEM_EXISTS Function .......................... 6-2
CURRENT_PAGE_IS_PUBLIC Function ..................................... 6-3
DEFINE_USER_SESSION Procedure ....................................... 6-4
GET_COOKIE_PROPS Procedure ............................................. 6-5
GET_LDAP_PROPS Procedure ............................................... 6-6
GET_NEXT_SESSION_ID Function ........................................... 6-8
GET_SECURITY_GROUP_ID Function ........................................ 6-9
GET_SESSION_ID Function .................................................. 6-10
GET_SESSION_ID_FROM_COOKIE Function .............................. 6-11
GET_USER Function ........................................................... 6-12
GET_USERNAME Function ..................................................... 6-13
IS_SESSION_VALID Function ................................................. 6-14
LOGIN Procedure ............................................................. 6-15
LOGOUT Procedure ........................................................... 6-16
POST_LOGIN Procedure ..................................................... 6-17
SESSION_ID_EXISTS Function .............................................. 6-18
SET_SESSION_ID Procedure ............................................... 6-19
SET_SESSION_ID_TO_NEXT_VALUE Procedure ....................... 6-20
SET_USER Procedure ......................................................... 6-21

7  APEX_DEBUG

Constants .............................................................................. 7-3
DISABLE Procedure ........................................................... 7-4
ENABLE Procedure ............................................................ 7-5
ENTER Procedure ............................................................... 7-6
ERROR Procedure .............................................................. 7-8
INFO Procedure ................................................................. 7-9
LOG_DBMS_OUTPUT Procedure .............................................. 7-10
LOG_LONG_MESSAGE Procedure ........................................ 7-11
LOG_MESSAGE Procedure [Deprecated] ................................. 7-12
LOG_PAGE_SESSION_STATE Procedure ................................. 7-13
MESSAGE Procedure ........................................................ 7-14
REMOVE_DEBUG_BY_AGE Procedure .................................... 7-16
REMOVE_DEBUG_BY_APP Procedure ...................................... 7-17
REMOVE_DEBUG_BY_VIEW Procedure ................................... 7-18
REMOVE_SESSION_MESSAGES Procedure ................................................................. 7-19
TOCHAR Function ................................................................................................. 7-20
TRACE Procedure ................................................................................................. 7-21
WARN Procedure .................................................................................................. 7-22

8 APEX_ERROR
Data Types and Constants ...................................................................................... 8-2
  Constants used for Result Types ......................................................................... 8-3
    t_error ............................................................................................................... 8-4
    t_error_result ................................................................................................. 8-5
Example of an Error Handling Function ............................................................... 8-6
ADD_ERROR Procedure Signature 1 ...................................................................... 8-8
ADD_ERROR Procedure Signature 2 ...................................................................... 8-9
ADD_ERROR Procedure Signature 3 ...................................................................... 8-10
ADD_ERROR Procedure Signature 4 ...................................................................... 8-12
ADD_ERROR Procedure Signature 5 ...................................................................... 8-14
AUTO_SET_ASSOCIATED_ITEM Procedure ......................................................... 8-16
EXTRACT_CONSTRAINT_NAME Function ............................................................. 8-17
GET_ARIA_ERROR_ATTRIBUTES Function ............................................................ 8-18
GET_FIRST_ORA_ERROR_TEXT Function ............................................................... 8-19
INIT_ERROR_RESULT Function ............................................................................. 8-20

9 APEX_ESCAPE
Constants ................................................................................................................. 9-2
HTML Function ...................................................................................................... 9-3
HTML ATTRIBUTE Function .................................................................................. 9-5
HTML TRUNC Function .......................................................................................... 9-6
HTML WHITELIST Function .................................................................................. 9-7
JS LITERAL Function .............................................................................................. 9-8
LDAP DN Function .................................................................................................. 9-9
LDAP SEARCH FILTER Function ............................................................................ 9-10
NOOP Function ...................................................................................................... 9-11
SET_HTML_ESCAPING_MODE Procedure ............................................................ 9-12

10 APEX_INSTANCE_ADMIN
Available Parameter Values .................................................................................. 10-2
ADD_SCHEMA Procedure ...................................................................................... 10-7
ADD_WORKSPACE Procedure .............................................................................. 10-8
GET_PARAMETER Function .................................................................................... 10-9
GET_SCHEMAS Function ....................................................................................... 10-10
REMOVE_APPLICATION Procedure ................................................................. 10-11
<table>
<thead>
<tr>
<th>Procedure/Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOVE_SAVED_REPORT Procedure</td>
<td>10-12</td>
</tr>
<tr>
<td>REMOVE_SAVED_REPORTS Procedure</td>
<td>10-13</td>
</tr>
<tr>
<td>REMOVE_SCHEMA Procedure</td>
<td>10-14</td>
</tr>
<tr>
<td>REMOVE_SUBSCRIPTION Procedure</td>
<td>10-15</td>
</tr>
<tr>
<td>REMOVE_WORKSPACE Procedure</td>
<td>10-16</td>
</tr>
<tr>
<td>SET_LOG_SWITCH_INTERVAL Procedure</td>
<td>10-17</td>
</tr>
<tr>
<td>SET_PARAMETER Procedure</td>
<td>10-18</td>
</tr>
<tr>
<td>SET_WORKSPACE_CONSUMER_GROUP Procedure</td>
<td>10-19</td>
</tr>
<tr>
<td>TRUNCATE_LOG Procedure</td>
<td>10-20</td>
</tr>
<tr>
<td>11 APEX_IR</td>
<td></td>
</tr>
<tr>
<td>ADD_FILTER Procedure Signature 1</td>
<td>11-2</td>
</tr>
<tr>
<td>ADD_FILTER Procedure Signature 2</td>
<td>11-4</td>
</tr>
<tr>
<td>CHANGE_SUBSCRIPTION_LANG Procedure</td>
<td>11-6</td>
</tr>
<tr>
<td>CLEAR_REPORT Procedure Signature 1</td>
<td>11-7</td>
</tr>
<tr>
<td>CLEAR_REPORT Procedure Signature 2</td>
<td>11-8</td>
</tr>
<tr>
<td>DELETE_REPORT Procedure</td>
<td>11-9</td>
</tr>
<tr>
<td>DELETE_SUBSCRIPTION Procedure</td>
<td>11-10</td>
</tr>
<tr>
<td>GET_LAST_VIEWED_REPORT_ID Function</td>
<td>11-11</td>
</tr>
<tr>
<td>GET_REPORT Function</td>
<td>11-12</td>
</tr>
<tr>
<td>RESET_REPORT Procedure Signature 1</td>
<td>11-13</td>
</tr>
<tr>
<td>RESET_REPORT Procedure Signature 2</td>
<td>11-14</td>
</tr>
<tr>
<td>12 APEX_ITEM</td>
<td></td>
</tr>
<tr>
<td>CHECKBOX2 Function</td>
<td>12-2</td>
</tr>
<tr>
<td>DATE_POPUP Function</td>
<td>12-4</td>
</tr>
<tr>
<td>DATE_POPUP2 Function</td>
<td>12-6</td>
</tr>
<tr>
<td>DISPLAY_AND_SAVE Function</td>
<td>12-8</td>
</tr>
<tr>
<td>HIDDEN Function</td>
<td>12-9</td>
</tr>
<tr>
<td>MD5_CHECKSUM Function</td>
<td>12-11</td>
</tr>
<tr>
<td>MD5_HIDDEN Function</td>
<td>12-12</td>
</tr>
<tr>
<td>POPUP_FROM_LOV Function</td>
<td>12-13</td>
</tr>
<tr>
<td>POPUP_FROM_QUERY Function</td>
<td>12-15</td>
</tr>
<tr>
<td>POPUPKEY_FROM_LOV Function</td>
<td>12-17</td>
</tr>
<tr>
<td>POPUPKEY_FROM_QUERY Function</td>
<td>12-19</td>
</tr>
<tr>
<td>RADIOGROUP Function</td>
<td>12-21</td>
</tr>
<tr>
<td>SELECT_LIST Function</td>
<td>12-22</td>
</tr>
<tr>
<td>SELECT_LIST_FROM_LOV Function</td>
<td>12-24</td>
</tr>
<tr>
<td>SELECT_LIST_FROM_LOV_XL Function</td>
<td>12-25</td>
</tr>
<tr>
<td>SELECT_LIST_FROM_QUERY Function</td>
<td>12-27</td>
</tr>
<tr>
<td>SELECT_LIST_FROM_QUERY_XL Function</td>
<td>12-29</td>
</tr>
<tr>
<td>TEXT Function</td>
<td>12-31</td>
</tr>
</tbody>
</table>
TEXTAREA Function ................................................................. 12-32
TEXT_FROM_LOV Function ..................................................... 12-33
TEXT_FROM_LOV_QUERY Function .......................................... 12-34

13 APEX_JAVASCRIPT
ADD_3RD_PARTY_LIBRARY_FILE Procedure .......................... 13-2
ADD_ATTRIBUTE Function Signature 1 .............................. 13-3
ADD_ATTRIBUTE Function Signature 2 .............................. 13-5
ADD_ATTRIBUTE Function Signature 3 .............................. 13-6
ADD_ATTRIBUTE Function Signature 4 .............................. 13-7
ADD_INLINE_CODE Procedure ........................................... 13-8
ADD_LIBRARY Procedure .................................................. 13-9
ADD_ONLOAD_CODE Procedure ....................................... 13-10
ADD_VALUE Function Signature 1 .................................... 13-11
ADD_VALUE Function Signature 2 .................................... 13-12
ADD_VALUE Function Signature 3 .................................... 13-13
ADD_VALUE Function Signature 4 .................................... 13-14
Escape Function ............................................................... 13-15

14 APEX_LANG
CREATE_LANGUAGE_MAPPING Procedure .......................... 14-2
DELETE_LANGUAGE_MAPPING Procedure .......................... 14-4
LANG Function ................................................................. 14-6
MESSAGE Function .......................................................... 14-7
PUBLISH_APPLICATION Procedure ................................. 14-9
SEED_TRANSLATIONS Procedure ....................................... 14-10
UPDATE_LANGUAGE_MAPPING Procedure ........................ 14-11
UPDATE_MESSAGE Procedure ......................................... 14-13
UPDATE_TRANSLATED_STRING Procedure ....................... 14-14

15 APEX_LDAP
AUTHENTICATE Function .................................................. 15-2
GET_ALL_USER_ATTRIBUTES Procedure ........................... 15-3
GET_USER_ATTRIBUTES Procedure .................................... 15-5
IS_MEMBER Function ......................................................... 15-7
MEMBER_OF Function ...................................................... 15-9
MEMBER_OF2 Function .................................................... 15-10

16 APEX_MAIL
About Configuring Oracle Application Express to Send Email ...... 16-3
ADD_ATTACHMENT Procedure ........................................... 16-4
PRINT_HIDDEN_IF_READONLY Procedure .............................................................. 19-32
PRINT_JSON_HTTP_HEADER Procedure .............................................................. 19-33
PRINT_LOV_AS_JSON Procedure ......................................................................... 19-34
PRINT_OPTION Procedure ................................................................................... 19-35
REPLACE_SUBSTITUTIONS Function .................................................................. 19-36

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

21  APEX_UTIL
CACHE_GET_DATE_OF_PAGE_CACHE Function ...................................................... 21-5
CACHE_GET_DATE_OF_REGION_CACHE Function ............................................... 21-6
CACHE_PURGE_BY_APPLICATION Procedure .................................................... 21-7
CACHE_PURGE_BY_PAGE Procedure ................................................................. 21-8
CACHE_PURGE_STALE Procedure ....................................................................... 21-9
CHANGE_CURRENT_USER_PW Procedure ........................................................... 21-10
CHANGE_PASSWORD_ON_FIRST_USE Function ............................................... 21-11
CLEAR_APP_CACHE Procedure ......................................................................... 21-12
REMOVE_USER Procedure................................................................................................ 21-114
RESET_AUTHORIZATIONS Procedure.................................................................................. 21-115
RESET_PW Procedure.................................................................................................... 21-116
SAVEKEY_NUM Function................................................................................................. 21-117
SAVEKEY_VC2 Function................................................................................................. 21-118
SET_ATTRIBUTE Procedure.......................................................................................... 21-119
SET_AUTHENTICATION_RESULT Procedure .................................................................... 21-120
SET_BUILD_OPTION_STATUS Procedure.......................................................................... 21-121
SET_CUSTOM_AUTH_STATUS Procedure........................................................................... 21-122
SET_EDITION Procedure................................................................................................. 21-123
SET_EMAIL Procedure................................................................................................... 21-124
SET_FIRST_NAME Procedure.......................................................................................... 21-125
SET_LAST_NAME Procedure............................................................................................ 21-126
SET_PREFERENCE Procedure........................................................................................ 21-127
SET_SECURITY_GROUP_ID Procedure............................................................................... 21-128
SET_SESSION_HIGH_CONTRAST_OFF Procedure............................................................ 21-130
SET_SESSION_HIGH_CONTRAST_ON Procedure.............................................................. 21-131
SET_SESSION_LANG Procedure........................................................................................ 21-132
SET_SESSION_LIFETIME_SECONDS Procedure.................................................................. 21-133
SET_SESSION_MAX_IDLE_SECONDS Procedure............................................................... 21-134
SET_SESSION_SCREEN_READER_OFF Procedure............................................................. 21-135
SET_SESSION_SCREEN_READER_ON Procedure............................................................. 21-136
SET_SESSION_STATE Procedure.................................................................................... 21-137
SET_SESSION_TERRITORY Procedure............................................................................. 21-138
SET_SESSION_TIME_ZONE Procedure............................................................................. 21-139
SET_USERNAME Procedure............................................................................................ 21-140
SHOW_HIGH_CONTRAST_MODE_TOGGLE Procedure....................................................... 21-141
SHOW_SCREEN_READER_MODE_TOGGLE Procedure.................................................... 21-142
STRING_TO_TABLE Function.......................................................................................... 21-143
STRONG_PASSWORD_CHECK Procedure........................................................................... 21-144
STRONG_PASSWORD_VALIDATION Function........................................................................ 21-145
SUBMIT_FEEDBACK Procedure...................................................................................... 21-146
SUBMIT_FEEDBACK_FOLLOWUP Procedure....................................................................... 21-147
TABLE_TO_STRING Function.......................................................................................... 21-148
UNEXPIRE_END_USER_ACCOUNT Procedure...................................................................... 21-149
UNEXPIRE_WORKSPACE_ACCOUNT Procedure.............................................................. 21-150
UNLOCK_ACCOUNT Procedure......................................................................................... 21-151
URL_ENCODE Function.................................................................................................... 21-152
WORKSPACE_ACCOUNT_DAYS_LEFT Function.................................................................... 21-153

22 APEX_WEB_SERVICE

About the APEX_WEB_SERVICE API.................................................................................. 22-2
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoking a SOAP Style Web Service</td>
<td>22-3</td>
</tr>
<tr>
<td>Invoking a RESTful Style Web Service</td>
<td>22-5</td>
</tr>
<tr>
<td>Retrieving Cookies and HTTP Headers</td>
<td>22-6</td>
</tr>
<tr>
<td>Setting Cookies and HTTP Headers</td>
<td>22-7</td>
</tr>
<tr>
<td>BLOB2CLOB BASE64 Function</td>
<td>22-8</td>
</tr>
<tr>
<td>CLOB BASE642BLOB Function</td>
<td>22-9</td>
</tr>
<tr>
<td>MAKE_REQUEST Procedure</td>
<td>22-10</td>
</tr>
<tr>
<td>MAKE_REQUEST Function</td>
<td>22-12</td>
</tr>
<tr>
<td>MAKE_REST_REQUEST Function</td>
<td>22-14</td>
</tr>
<tr>
<td>PARSE_RESPONSE Function</td>
<td>22-16</td>
</tr>
<tr>
<td>PARSE_RESPONSE_CLOB Function</td>
<td>22-17</td>
</tr>
<tr>
<td>PARSE_XML Function</td>
<td>22-18</td>
</tr>
<tr>
<td>PARSE_XML_CLOB Function</td>
<td>22-20</td>
</tr>
</tbody>
</table>

## 23 JavaScript APIs

<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>apex namespace.</td>
<td>23-3</td>
</tr>
<tr>
<td>apex.confirm(pMessage, pRequest)</td>
<td>23-4</td>
</tr>
<tr>
<td>apex.confirm(pMessage, pOptions)</td>
<td>23-4</td>
</tr>
<tr>
<td>apex.submit(pOptions)</td>
<td>23-6</td>
</tr>
<tr>
<td>apex.submit(pRequest)</td>
<td>23-6</td>
</tr>
<tr>
<td>apex.event trigger(pSelector, pEvent, pData)</td>
<td>23-8</td>
</tr>
<tr>
<td>apex.item namespace</td>
<td>23-9</td>
</tr>
<tr>
<td>apex.item(pNd).addValue(pValue)</td>
<td>23-11</td>
</tr>
<tr>
<td>apex.item(pNd).disable()</td>
<td>23-12</td>
</tr>
<tr>
<td>apex.item(pNd).enable()</td>
<td>23-13</td>
</tr>
<tr>
<td>apex.item(pNd).getValue()</td>
<td>23-14</td>
</tr>
<tr>
<td>apex.item(pNd).hide(pHideRow)</td>
<td>23-15</td>
</tr>
<tr>
<td>apex.item(pNd).isEmpty()</td>
<td>23-16</td>
</tr>
<tr>
<td>apex.item(pNd).setFocus()</td>
<td>23-17</td>
</tr>
<tr>
<td>apex.item(pNd).setStyle(pPropertyName, pPropertyValue)</td>
<td>23-18</td>
</tr>
<tr>
<td>apex.item(pNd).setValue(pValue, pDisplayValue, pSuppressChangeEvent)</td>
<td>23-19</td>
</tr>
<tr>
<td>apex.navigation namespace</td>
<td>23-22</td>
</tr>
<tr>
<td>apex.navigation.popup.close(pThat, pValue)</td>
<td>23-23</td>
</tr>
<tr>
<td>apex.server namespace</td>
<td>23-24</td>
</tr>
<tr>
<td>apex.server.plugin(pAjaxIdentifier, pData, pOptions)</td>
<td>23-25</td>
</tr>
<tr>
<td>apex.server.pluginUrl(pAjaxIdentifier, pData)</td>
<td>23-29</td>
</tr>
</tbody>
</table>
apex.server.process( pAjaxIdentifier, pData, pOptions ) .............................................. 23-30
apex.storage namespace ........................................................................................................... 23-36
apex.storage.getCookie(pName) ............................................................................................. 23-37
apex.storage.setCookie(pName,pValue) .............................................................................. 23-38
apex.widget namespace ............................................................................................................ 23-39
apex.widget.initPageItem( pName, pOptions) ......................................................................... 23-40
Miscellaneous Javascript APIs ................................................................................................. 23-47
$x(pNd) ............................................................................................................................... 23-50
$v(pNd) ............................................................................................................................... 23-51
$v2(pNd) .............................................................................................................................. 23-52
$s(pNd, pValue, pDisplayValue, pSuppressChangeEvent) ...................................................... 23-53
$u_Narray(pNd) .................................................................................................................... 23-54
$u_Carray(pNd) ..................................................................................................................... 23-55
$nvl(pTest, pDefault) ........................................................................................................... 23-56
$x_Style(pNd, pStyle, pString) ............................................................................................. 23-57
$x_Hide(pNd) ....................................................................................................................... 23-58
$x_Show(pNd) ....................................................................................................................... 23-59
$x_Toggle(pNd) .................................................................................................................... 23-60
$x_Remove(pNd) .................................................................................................................... 23-61
$x_Value(pNd,pValue) .......................................................................................................... 23-62
$x_UpTill(pNd, pToTag) ........................................................................................................ 23-63
$x_ItemRow(pNd,pFunc) ........................................................................................................ 23-64
$x_HideItemRow(pNd) ........................................................................................................... 23-65
$x_ShowItemRow(pNd) .......................................................................................................... 23-66
$x_ToggleItemRow(pNd) ........................................................................................................ 23-67
$x_HideAllExcept(pNd,pNdArray) ....................................................................................... 23-68
$x_HideSiblings(pNd) ........................................................................................................... 23-69
$x_ShowSiblings(pNd) .......................................................................................................... 23-70
$x_Class(pNd,pClass) ........................................................................................................... 23-71
$x_SetSiblingsClass(pNd, pClass, pNdClass) ........................................................................ 23-72
$x_ByClass(pClass, pNd, pTag) ........................................................................................... 23-73
$x_ShowAllByClass(pNd, pClass, pTag) .............................................................................. 23-74
$x_ShowChildren(pNd) .......................................................................................................... 23-75
$x_HideChildren(pNd) .......................................................................................................... 23-76
$x_disableItem(pNd, pTest) ................................................................................................... 23-77
$f_get_emptys(pNd, pClassFail, pClass) ............................................................................... 23-78
$v_Array(pNd) ....................................................................................................................... 23-79
$f_ReturnChecked(pNd) ........................................................................................................... 23-80
$d_ClearAndHide(pNd) ............................................................................................................ 23-81
$f_SelectedOptions(pNd) ........................................................................................................ 23-82
$f_SelectValue(pNd) ............................................................................................................. 23-83
$u_ArrayToString(pArray, pDelim) ......................................................................................... 23-84
xvii

$\text{x\_CheckImageSrc}(p\text{Id}, p\text{Search})$ ........................................................................................................ 23-85
$\text{v\_CheckValueAgainst}(p\text{This}, p\text{Value})$ .................................................................................. 23-86
$\text{f\_Hide\_On\_Value\_Item}(p\text{This}, p\text{That}, p\text{Value})$ .......................................................... 23-87
$\text{f\_Show\_On\_Value\_Item}(p\text{This}, p\text{That}, p\text{Value})$ .......................................................... 23-88
$\text{f\_Hide\_On\_Value\_Item\_Row}(p\text{This}, p\text{That}, p\text{Value})$ ................................................. 23-89
$\text{f\_Show\_On\_Value\_Item\_Row}(p\text{This}, p\text{That}, p\text{Value})$ ................................................. 23-90
$\text{f\_DisableOnValue}(p\text{This}, p\text{Value}, p\text{That})$ ................................................................... 23-91
$\text{x\_ClassByClass}(p\text{Nd}, p\text{Class}, p\text{Tag}, p\text{Class2})$ ......................................................... 23-92
$\text{f\_ValuesToArray}(p\text{This}, p\text{Class}, p\text{Tag})$ ......................................................................... 23-93
$\text{x\_FormItems}(p\text{Nd}, p\text{Type})$ ............................................................................................. 23-94
$\text{f\_CheckAll}(p\text{This}, p\text{Check}, p\text{Array})$ .............................................................................. 23-95
$\text{f\_CheckFirstColumn}(p\text{Nd})$ ............................................................................................... 23-96
$\text{v\_PopupReturn}(p\text{Value}, p\text{That})$ [Deprecated] ............................................................... 23-97
$\text{x\_ToggleWithImage}(p\text{This}, p\text{Nd})$ ................................................................................... 23-98
$\text{x\_SwitchImageSrc}(p\text{Nd}, p\text{Search}, p\text{Replace})$ .............................................................. 23-99
$\text{x\_CheckImageSrc}(p\text{Nd}, p\text{Search})$ ................................................................................... 23-100
$\text{u\_SubString}(p\text{Text}, p\text{Match})$ ....................................................................................... 23-101
$\text{html\_RemoveAllChildren}(p\text{Nd})$ ....................................................................................... 23-102
$\text{v\_IsEmpty}(p\text{This})$ [Deprecated] .......................................................................................... 23-103
$\text{html\_SetSelectValue}(p\text{Id}, p\text{Value})$ .................................................................................. 23-104
$\text{addLoadEvent}(p\text{Function})$ .................................................................................................. 23-105
$\text{f\_Swap}(p\text{This}, p\text{That})$ ..................................................................................................... 23-106
$\text{submitEnter}(p\text{Nd}, e)$ [Deprecated] ..................................................................................... 23-107
$\text{f\_SetValueSequence}(p\text{Array}, p\text{Multiple})$ ..................................................................... 23-108
$\text{dom\_AddTag}(p\text{This}, p\text{Tag}, p\text{Text})$ .................................................................................. 23-109
$\text{str\_AddTD}(p\text{This}, p\text{Text})$ ................................................................................................. 23-110
$\text{str\_AddTH}(p\text{This}, p\text{Text})$ ................................................................................................. 23-111
$\text{dom\_AddInput}(p\text{This}, p\text{Type}, p\text{Id}, p\text{Name}, p\text{Value})$ .................................................. 23-112
$\text{dom\_MakeParent}(p\text{Node}, p\text{Parent})$ ................................................................................... 23-113
$\text{x\_RowHighlight}(p\text{This}, p\text{Color})$ ....................................................................................... 23-114
$\text{x\_RowHighlightOff}(p\text{This})$ ............................................................................................... 23-115
$\text{v\_Upper}(p\text{Nd})$ ...................................................................................................................... 23-116
$\text{d\_Find}(p\text{This}, p\text{String}, p\text{Tags}, p\text{Class})$ ........................................................................... 23-117
$\text{setReturn}(p\_\text{R}, p\_\text{D})$ [Deprecated] .......................................................................................... 23-118
$\text{f\_First\_field}(p\text{Nd})$ .................................................................................................................. 23-119
$\text{GetCookie}(p\text{Name})$ [Deprecated] ........................................................................................... 23-120
$\text{SetCookie}(p\text{Name}, p\text{Value})$ [Deprecated] .............................................................................. 23-121

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

Note: In Oracle Application Express 4.2, the APEX_DEBUG_MESSAGE package was renamed to APEX_DEBUG. The APEX_DEBUG_MESSAGE package name is still supported to provide backward compatibility. As a best practice, however, use the new APEX_DEBUG package for new applications unless you plan to run them in an earlier version of Oracle Application Express.

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>Changes in This Release</td>
<td>Describes changes in this document for Oracle Application Express Release 4.2</td>
</tr>
<tr>
<td>APEX_APPLICATION</td>
<td>Use the APEX_APPLICATION package to take advantage of many global variables.</td>
</tr>
<tr>
<td>APEX_APPLICATION_INSTALL</td>
<td>The APEX_APPLICATION_INSTALL package provides many methods to modify application attributes during the Application Express application installation process.</td>
</tr>
<tr>
<td>APEX_AUTHENTICATION</td>
<td>The APEX_AUTHENTICATION package provides a public API for authentication plugins.</td>
</tr>
<tr>
<td>Title</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>APEX_COLLECTION</td>
<td>Use APEX_COLLECTION to temporarily capture one or more nonscalar values. You can use collections to store rows and columns currently in session state so they can be accessed, manipulated, or processed during a user’s specific session.</td>
</tr>
<tr>
<td>APEX_CSS</td>
<td>The APEX_CSS package provides utility functions for adding CSS styles to HTTP output. This package is usually used for plug-in development.</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_DEBUG</td>
<td>The APEX_DEBUG package provides utility functions for managing the debug message log.</td>
</tr>
<tr>
<td>APEX_ESCAPE</td>
<td>The APEX_ESCAPE package provides functions for escaping special characters in strings, to ensure that the data is suitable for further processing.</td>
</tr>
<tr>
<td>APEX_ERROR</td>
<td>The APEX_ERROR package provides the interface declarations and some utility functions for an error handling function and includes procedures and functions to raise errors in an Application Express application.</td>
</tr>
<tr>
<td>APEX_INSTANCE_ADMIN</td>
<td>The APEX_INSTANCE_ADMIN package provides utilities for managing an Oracle Application Express runtime environment. Use the APEX_INSTANCE_ADMIN package to get and set email settings, wallet settings, report printing settings and to manage scheme to workspace mappings.</td>
</tr>
<tr>
<td>APEX_IR</td>
<td>The APEX_IR package provides utilities you can use when programming in the Oracle Application Express environment related to interactive reports.</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_JAVASCRIPT</td>
<td>The APEX_JAVASCRIPT package provides utility functions for adding dynamic JavaScript code to HTTP output. This package is usually used for plug-in development.</td>
</tr>
<tr>
<td>APEX_LANG</td>
<td>Use APEX_LANG API to translate messages.</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_MAIL</td>
<td>Use the APEX_MAIL package to send an email from an Oracle Application Express application.</td>
</tr>
<tr>
<td>APEX_PLSQL_JOB</td>
<td>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_PLUGIN</td>
<td>The APEX_PLUGIN package provides the interface declarations and some utility functions to work with plug-ins.</td>
</tr>
</tbody>
</table>
Audience

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

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

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

Documentation Accessibility

For information about Oracle’s commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit
Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

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

For additional application examples, go to the Learning Library. Search for free online training content, including Oracle by Example (OBE), demos, and tutorials. To access the Oracle Learning Library, go to:

http://www.oracle.com/technetwork/tutorials/index.html

Printed documentation is available for sale in the Oracle Store at

http://shop.oracle.com/

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

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

Conventions

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

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

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></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><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><strong>monospace</strong></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>
Changes in This Release

This preface contains:

- Changes in Oracle Application Express Release 4.2

Changes in Oracle Application Express Release 4.2

The following are changes in Oracle Application Express API Reference for Oracle Application Express release 4.2.

New Features

The following features are new in this release:

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

- APEX_CSS
  - Support added for media queries and IE conditions to apex_css and apex_javascript APIs. Two new parameters added to the ADD_FILE procedure. See "ADD_FILE Procedure" on page 5-4.
  - The ADD_THIRD_PARTY_LIBRARY procedure added to support content delivery networks for jQuery files. See "ADD_3RD_PARTY_LIBRARY_FILE Procedure" on page 5-3.

- APEX_DEBUG
  - In Oracle Application Express 4.2, the APEX_DEBUG_MESSAGE package was renamed to APEX_DEBUG. The APEX_DEBUG_MESSAGE package name is still supported to provide backward compatibility. As a best practice, however, use the new APEX_DEBUG package for new applications unless you plan to run them in an earlier version of Oracle Application Express. See "APEX_DEBUG" on page 7-1.

- APEX_ESCAPE
The `HTML_WHITELIST` function added to perform HTML escape on all characters in the input text except the specified whitelist tags. See "HTML_WHITELIST Function" on page 9-7.

- **APEX_IR**
  - Prior to Application Express release 4.2, the interactive report API existed in `APEX_UTIL`. A separate `APEX_IR` package implemented in Application Express release 4.2 to help developers easily find interactive report methods. See "APEX_IR" on page 11-1.

- **APEX_JAVASCRIPT**
  - Support added for media queries and IE conditions to `apex_css` and `apex_javascript` APIs. Two new parameters added to the `ADD_FILE` procedure. See "ADD_LIBRARY Procedure" on page 13-9.

- **APEX_LANG**
  - Additional procedures added in Application Express release 4.2.3 and later, for translating messages. Added procedures include: `CREATE_LANGUAGE_MAPPING`, `DELETE_LANGUAGE_MAPPING`, `PUBLISH_APPLICATION`, `SEED_TRANSLATIONS`, `UPDATE_LANGUAGE_MAPPING`, `UPDATE_MESSAGE`, `UPDATED_TRANSLATED_STRING`. See "APEX_LANG" on page 14-1.

- **APEX_MAIL**
  - Implemented easy interface to remove interactive report subscriptions. See "GET_IMAGES_URL Function" on page 16-5 and "GET_INSTANCE_URL Function" on page 16-6

- **APEX_UTIL**
  - Build options are shared components in an Application Express application used to enable and disable functionality. In Application Express release 4.1.1, an API was added to set the build option. In Application Express release 4.2, functions are available to get back the status of a specified build option. There are two flavors of the `APEX_UTIL.GET_BUILD_OPTION_STATUS` function, one where you specify the build option by ID, and another where you specify the build option by build option name.
  - Session state can be shared between applications of the same workspace by implementing an extension to application items. The following procedures have been modified:
    - **APEX_UTIL.SET_SESSION_LIFETIME_SECONDS**
      The parameter `p_scope` is obsolete. The procedure always sets the lifetime for the whole session. See "SET_SESSION_LIFETIME_SECONDS Procedure" on page 21-133.
    - **APEX_UTIL.SET_SESSION_MAX_IDLE_SECONDS**
      The parameter `p_scope` is obsolete. The procedure always sets the lifetime for the whole session. See "SET_SESSION_MAX_IDLE_SECONDS Procedure" on page 21-134.

- **APEX_PLUGIN**
- No Data Found Message added as standard region type plug-in setting. See \texttt{t\_region} in "Data Types" on page 18-2.

- Number of custom attributes for region type plug-ins increased to 25. See \texttt{t\_region} in "Data Types" on page 18-2.

- Fetched Rows added as standard region type plug-in setting. See \texttt{t\_region} in "Data Types" on page 18-2.

- CSS Classes attribute added to regions, buttons, page items and report columns. See \texttt{t\_page\_item} in "Data Types" on page 18-2.

- Support for HTML5 placeholder attribute added to several item types. See \texttt{t\_page\_item} in "Data Types" on page 18-2.

**APEX_PLUGIN_UTIL**

- Function added to return some of the standard attributes of an HTML element (for example, id, name, required, placeholder, aria-error-attributes, class) which is used if a HTML input/select/textarea/... tag is generated to get a consistent set of attributes. See "GET ELEMENT ATTRIBUTES Function" on page 19-23.

- Functions added to better support custom plug-in attributes of type Region Column Name. Instead of passing a column number, for example \texttt{p\_search\_column\_no}, the functions now also support passing a column name, such as \texttt{p\_search\_column\_name}. See "GET DATA Function Signature 1" on page 19-11, "GET DATA Function Signature 2" on page 19-13, "GET DATA2 Function Signature 1" on page 19-15, and "GET DATA2 Function Signature 2" on page 19-17.

**APEX_INSTANCE_ADMIN**

- A new parameter value added to \texttt{APEX_INSTANCE_ADMIN\_SET\_PARAMETER} and \texttt{APEX_INSTANCE_ADMIN\_GET\_PARAMETER} named \texttt{BIGFILE\_TABLESPACES\_ENABLED}. See "Available Parameter Values" on page 10-2.

- Improvements made to failed login handling. The following parameter values added \texttt{LOGIN\_THROTTLE\_DELAY}, \texttt{LOGIN\_THROTTLE\_METHODS}, \texttt{INBOUND\_PROXIES}. See "Available Parameter Values" on page 10-2.

- Interface to remove interactive report subscription implemented. Two new parameters added: \texttt{email\_images\_url}, \texttt{email\_instance\_url}. See "Available Parameter Values" on page 10-2.

- Support for Enable Application Tracing added in Application Express Instance Administration. To support this feature, \texttt{TRACING\_ENABLED} parameter value implemented. See "Available Parameter Values" on page 10-2.

- Expose the system preference for Encrypted Tablesapce in Instance Administration and in package \texttt{APEX_INSTANCE_ADMIN}. New parameter value to \texttt{APEX_INSTANCE_ADMIN\_SET\_PARAMETER} and \texttt{APEX_INSTANCE_ADMIN\_GET\_PARAMETER} named \texttt{ ENCRYPTED\_TABLESPACES\_ENABLED}. See "Available Parameter Values" on page 10-2.

- Resource Consumer Group available at workspace level. New parameter \texttt{p\_rm\_consumer\_group} added to \texttt{ADD\_WORKSPACE} procedure. See "ADD WORKSPACE Procedure" on page 10-8. Also, \texttt{SET\_WORKSPACE\_CONSUMER\_GROUP} procedure and \texttt{ADD\_WORKSPACE} procedure added. See "SET WORKSPACE CONSUMER GROUP Procedure" on page 10-19 and "ADD WORKSPACE Procedure" on page 10-8.

– Procedure added to enable an instance administrator to set the log switch interval for each of the logs maintained by Application Express. See "SET_LOG_SWITCH_INTERVAL Procedure" on page 10-17.

■ JAVASCRIPT API

– Improve AJAX functions because `html_db_get` is outdated. The following methods added to the JavaScript library: `apex.server.plugin`, `apex.server.pluginUrl`, and `apex.server.process`. See "apex.server.plugin(pAjaxIdentifier,pData,pOptions)" on page 23-25, "apex.server.pluginUrl( pAjaxIdentifier, pData )" on page 23-29, and "apex.server.process( pAjaxIdentifier, pData, pOptions )" on page 23-30.
The **APEX_APPLICATION** package is a PL/SQL package that implements the Oracle Application Express rendering engine. You can use this package to take advantage of many global variables. Table 1–1 describes the global variables available in the **APEX_APPLICATION** package.

<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 redirects 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>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 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:
- Referencing Arrays
- Referencing Values Within an On Submit Process
- Converting an Array to a Single Value
- HELP Procedure
- STOP_APEX_ENGINE 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.

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.

```sql
FOR i IN 1..3 LOOP
  APEX_ITEM.TEXT(P_IDX => 1,
              p_value =>'array element '||i ,
              p_size =>32,
              p_maxlength =>32);
END LOOP;
```
Referencing Values Within an On Submit Process

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

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

```cpp
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. You can also use it to customize how help information is displayed in your application.

Syntax

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

Parameters

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

<table>
<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 is '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 is '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>
The following example shows how to use the APEX_APPLICATION.HELP procedure to customize how help information is displayed.

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

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

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

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.

---

**Example**

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

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

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

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

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

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

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

---

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

---

**Syntax**

```
APEX_APPLICATION.STOP_APEX_ENGINE
```

**Parameters**

None

**Example 1**

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

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

**Example 2**

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

```
begin
    ... code which can raise an exception ...
    owa_util.redirect_url('http://apex.oracle.com');
apex_application.stop_apex_engine;
exception
    when apex_application.e_stop_apex_engine then
        raise; -- raise again the stop Application Express engine exception
    when others then
        ...; -- code to handle the exception
end;
```
The APEX_APPLICATION_INSTALL package provides many methods to modify application attributes during the Application Express application installation process.

**Topics:**
- Package Overview
- Import Script Examples
- CLEAR_ALL Procedure
- GENERATE_APPLICATION_ID Procedure
- GENERATE_OFFSET Procedure
- GET_APPLICATION_ALIAS Function
- GET_APPLICATION_ID Function
- GET_APPLICATION_NAME Function
- GET_IMAGE_PREFIX Function
- GET_OFFSET Function
- GET_PROXY Function
- GET_SCHEMA Function
- GET_WORKSPACE_ID Function
- SET_APPLICATION_ALIAS Procedure
- SET_APPLICATION_ID Procedure
- SET_APPLICATION_NAME Procedure
- SET_IMAGE_PREFIX Procedure
- SET_OFFSET Procedure
- SET_PROXY Procedure
- SET_SCHEMA Procedure
- SET_WORKSPACE_ID Procedure
Oracle Application Express provides two ways to import an application into an Application Express instance:

1. Upload and installation of an application export file by using the web interface of Application Express.

2. Execution of the application export file as a SQL script, typically in the command-line utility SQL*Plus.

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

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

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

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

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workspace ID</td>
<td>Workspace ID of the imported application. See GET_WORKSPACE_ID Function, SET_WORKSPACE_ID Procedure.</td>
</tr>
<tr>
<td>Offset</td>
<td>Offset value used during application import. See GENERATE_OFFSET Procedure, GET_OFFSET Function, SET_OFFSET Procedure.</td>
</tr>
<tr>
<td>Schema</td>
<td>The parsing schema (&quot;owner&quot;) of the imported application. See GET_SCHEMA Function, SET_SCHEMA Procedure.</td>
</tr>
<tr>
<td>Name</td>
<td>Application name of the imported application. See GET_APPLICATION_NAME Function, SET_APPLICATION_NAME Procedure.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Alias</td>
<td>Application alias of the imported application. See GET_APPLICATION_ALIAS Function, SET_APPLICATION_ALIAS Procedure.</td>
</tr>
<tr>
<td>Image Prefix</td>
<td>The image prefix of the imported application. See GET_IMAGE_PREFIX Function, SET_IMAGE_PREFIX Procedure.</td>
</tr>
<tr>
<td>Proxy</td>
<td>The proxy server attributes of the imported application. See GET_PROXY Function, SET_PROXY Procedure.</td>
</tr>
</tbody>
</table>
Import Script Examples

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

**Import Application without Modification**
To import this application back into the FRED_DEV workspace on the same development instance using the same application ID:

```
@f645.sql
```

**Import Application with Specified Application ID**
To import this application back into the FRED_DEV workspace on the same development instance, but using application ID 702:

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

```
@645.sql
```

**Import Application with Generated Application ID**
To import this application back into the FRED_DEV workspace on the same development instance, but using an available application ID generated by Application Express:

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

```
@f645.sql
```

**Import Application into Different Workspace using Different Schema**
To import this application into the FRED_PROD workspace on the production instance, using schema FREDDY, and the workspace ID of FRED_DEV and FRED_PROD are different:

```
declare
   l_workspace_id number;
begin
```

```
Import Script Examples

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

@f645.sql

**Import into Training Instance for Three Different Workspaces**

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

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

@f645.sql
```

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

@f645.sql
```

```sql
declare
  l_workspace_id number;
begin
  select workspace_id into l_workspace_id
    from apex_workspaces
    where workspace = 'TRAINING3';
--
apex_application_install.set_workspace_id( l_workspace_id );
apex_application_install.generate_application_id;
apex_application_install.generate_offset;
apex_application_install.set_schema('STUDENT3');
apex_application_install.set_application_alias( 'F' || apex_application.get_application_id );
end;
/

@f645.sql
```
apex_application_install.set_workspace_id( l_workspace_id );
apex_application_install.generate_application_id;
apex_application_install.generate_offset;
apex_application_install.set_schema( 'STUDENT3' );
apex_application_install.set_application_alias( 'F' || apex_application.get_application_id );
end;
/

0f645.sql
CLEAR_ALL Procedure

This procedure clears all values currently maintained in the APEX_APPLICATION_INSTALL package.

Syntax
APEX_APPLICATION_INSTALL.CLEAR_ALL;

Parameters
None.

Example
The following example clears all values currently set by the APEX_APPLICATION_INSTALL package.

begin
    apex_application_install.clear_all;
end;
GENERATE_APPLICATION_ID Procedure

This procedure generates an available application ID on the instance and sets the application ID in APEX_APPLICATION_INSTALL.

Syntax
APEX_APPLICATION_INSTALL.GENERATE_APPLICATION_ID;

Parameters
None.

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

See Also: "SET_APPLICATION_ID Procedure" on page 2-19, "GET_APPLICATION_ID Function" on page 2-11
GENERATE_OFFSET Procedure

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

Syntax
APEX_APPLICATION_INSTALL.GENERATE_OFFSET;

Parameters
None.

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

See Also: "GET_OFFSET Function" on page 2-14, "SET_OFFSET Procedure" on page 2-22
GET_APPLICATION_ALIAS Function

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

Syntax

APEX_APPLICATION_INSTALL.GET_APPLICATION_ALIAS
RETURN VARCHAR2;

Parameters
None.

Example

The following example returns the value of the application alias value in the APEX_APPLICATION_INSTALL package. The application alias cannot be more than 255 characters.

```
DECLARE
  l_alias VARCHAR2(255);
BEGIN
  l_alias := apex_application_install.get_application_alias;
END;
```

See Also: "SET_APPLICATION_ALIAS Procedure" on page 2-18
GET_APPLICATION_ID Function

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

**Syntax**

APEX_APPLICATION_INSTALL.GET_APPLICATION_ID

RETURN NUMBER;

**Parameters**

None.

**Example**

The following example returns the value of the application ID value in the APEX_APPLICATION_INSTALL package.

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

See Also:  "SET_APPLICATION_ID Procedure" on page 2-19,
"GENERATE_APPLICATION_ID Procedure" on page 2-8
GET_APPLICATION_NAME Function

This function gets the application name of the import application.

Syntax

APEX_APPLICATION_INSTALL.GET_APPLICATION_NAME
RETURN VARCHAR2;

Parameters

None.

Example

The following example returns the value of the application name value in the APEX_APPLICATION_INSTALL package.

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

See Also: “SET_APPLICATION_NAME Procedure” on page 2-20
GET_IMAGE_PREFIX Function

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

**Syntax**

APEX_APPLICATION_INSTALL.GET_IMAGE_PREFIX

RETURN VARCHAR2;

**Parameters**

None.

**Example**

The following example returns the value of the application image prefix in the APEX_APPLICATION_INSTALL package. The application image prefix cannot be more than 255 characters.

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

*See Also:* "SET_IMAGE_PREFIX Procedure" on page 2-21
GET_OFFSET Function

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

Syntax
APEX_APPLICATION_INSTALL.GET_OFFSET
RETURN NUMBER;

Parameters
None.

Example
The following example returns the value of the application offset value in the APEX_APPLICATION_INSTALL package.

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

See Also: "SET_OFFSET Procedure" on page 2-22, "GENERATE_OFFSET Procedure" on page 2-9
**GET_PROXY Function**

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

**Syntax**

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

**Parameters**

None.

**Example**

The following example returns the value of the proxy server attribute in the APEX_APPLICATION_INSTALL package. The proxy server attribute cannot be more than 255 characters.

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

**See Also:**  "SET_PROXY Procedure" on page 2-23
GET_SCHEMA Function

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

Syntax
APEX_APPLICATION_INSTALL.GET_SCHEMA
RETURN VARCHAR2;

Parameters
None.

Example
The following example returns the value of the application schema in the APEX_APPLICATION_INSTALL package.

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

See Also: "SET_SCHEMA Procedure" on page 2-24
Use this function to get the workspace ID for the application to be imported.

**Syntax**

APEX_APPLICATION_INSTALL.GET_WORKSPACE_ID
RETURN NUMBER;

**Parameters**

None.

**Example**

The following example returns the value of the workspace ID value in the APEX_APPLICATION_INSTALL package.

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

**See Also:**  "SET_WORKSPACE_ID Procedure" on page 2-25
SET_APPLICATION_ALIAS Procedure

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

Syntax
APEX_APPLICATION_INSTALL.SET_APPLICATION_ALIAS(
    p_application_alias IN VARCHAR2);

Parameters
Table 2–2 describes the parameters available in SET_APPLICATION_ALIAS procedure.

Table 2–2 SET_APPLICATION_ALIAS Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application_alias</td>
<td>The application alias. The application alias is an alphanumeric identifier. It cannot exceed 255 characters, must unique within a workspace and, ideally, is unique within an entire instance.</td>
</tr>
</tbody>
</table>

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

See Also: "GET_APPLICATION_ALIAS Function" on page 2-10
SET_APPLICATION_ID Procedure

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

**Syntax**

APEX_APPLICATION_INSTALL.SET_APPLICATION_ID (  
    p_application_id IN NUMBER);

**Parameters**

Table 2–3 describes the parameters available in SET_APPLICATION_ID procedure.

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

**Example**

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

**See Also:** "SET_APPLICATION_ID Procedure" on page 2-19,  
"GENERATE_APPLICATION_ID Procedure" on page 2-8
SET_APPLICATION_NAME Procedure

This procedure sets the application name of the import application.

Syntax
APEX_APPLICATION_INSTALL.SET_APPLICATION_NAME;(  
  p_application_name  IN VARCHAR2);

Parameters
Table 2–4 describes the parameters available in SET_APPLICATION_NAME procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application_name</td>
<td>This is the application name. The application name cannot be null and cannot be longer than 255 characters.</td>
</tr>
</tbody>
</table>

Example
The following example sets the application name in APEX_APPLICATION_INSTALL to "Executive Dashboard".

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

See Also:  "GET_APPLICATION_NAME Function" on page 2-20
SET_IMAGE_PREFIX Procedure

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

Syntax

APEX_APPLICATION_INSTALL.SET_IMAGE_PREFIX(
    p_image_prefix  IN VARCHAR2);

Parameters

Table 2–4 describes the parameters available in SET_IMAGE_PREFIX procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_image_prefix</td>
<td>The image prefix. Default is /i/.</td>
</tr>
</tbody>
</table>

Example

The following example sets the value of the image prefix variable in APEX_APPLICATION_INSTALL.

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

See Also: "GET_IMAGE_PREFIX Function" on page 2-13
This procedure sets the offset value used during application import. Use the offset value to ensure that the metadata for the Application Express application definition does not collide with other metadata on the instance. For a new application installation, it is usually sufficient to call the `generate_offset` procedure to have Application Express generate this offset value for you.

**Syntax**

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

**Parameters**

Table 2–6 describes the parameters available in `SET_OFFSET` procedure.

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

**Example**

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

```
declare
    l_offset number;
begin
    l_offset := dbms_random.value(1000000000000, 999999999999);
    apex_application_install.set_offset( p_offset => l_offset );
end;
```

**See Also:** "GET_OFFSET Function" on page 2-14, "GENERATE_OFFSET Procedure" on page 2-9
SET_PROXY Procedure

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

**Syntax**

APEX_APPLICATION_INSTALL.SET_PROXY (  
  p_proxy IN VARCHAR2);

**Parameters**

Table 2-7 describes the parameters available in SET_PROXY procedure.

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

**Example**

The following example sets the value of the proxy variable in APEX_APPLICATION_INSTALL.

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

See Also:  "SET_PROXY Procedure" on page 2-15
Use this function to set the parsing schema ("owner") of the Application Express application. The database user of this schema must already exist, and this schema name must already be mapped to the workspace used to import the application.

Syntax

APEX_APPLICATION_INSTALL.SET_SCHEMA (p_schema IN VARCHAR2);

Parameters

Table 2–8 describes the parameters available in the SET_SCHEMA procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_schema</td>
<td>The schema name.</td>
</tr>
</tbody>
</table>

Example

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

See Also: "GET_SCHEMA Function" on page 2-16
SET_WORKSPACE_ID Procedure

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

**Syntax**

APEX_APPLICATION_INSTALL.SET_WORKSPACE_ID (
    p_workspace_id  IN NUMBER);

**Parameters**

Table 2–9 describes the parameters available in the SET_WORKSPACE_ID procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_workspace_id</td>
<td>The workspace ID.</td>
</tr>
</tbody>
</table>

**Example**

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

**See Also:** "SET_WORKSPACE_ID Procedure" on page 2-25
The APEX_AUTHENTICATION package provides a public API for authentication plugins.

**Topics:**
- Constants
- CALLBACK Procedure
- GET_CALLBACK_URL Function
- GET_LOGIN_USERNAME_COOKIE Function
- IS_AUTHENTICATED Function
- IS_PUBLIC_USER Function
- LOGIN Procedure
- LOGOUT Procedure
- POST_LOGIN Procedure
- SEND_LOGIN_USERNAME_COOKIE Procedure
The following constant is used by this package.

c_default_username_cookie constant varchar2(30) := 'LOGIN_USERNAME_COOKIE';
CALLBACK Procedure

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

Syntax

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

Parameters

Table 3–1 describes the parameters available in CALLBACK procedure.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_session_id</td>
<td>The Application Express session identifier.</td>
</tr>
<tr>
<td>p_app_id</td>
<td>The database application identifier.</td>
</tr>
<tr>
<td>p_ajax_identifier</td>
<td>The system generated AJAX identifier. See &quot;GET_AJAX_IDENTIFIER Function&quot; on page 8.</td>
</tr>
<tr>
<td>p_x01 through p_x10</td>
<td>Optional parameters that the external login passes to the authentication plugin.</td>
</tr>
</tbody>
</table>

Example 1

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

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

Example 2

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

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

See Also: "GET_CALLBACK_URL Function" on page 3-5
GET_CALLBACK_URL Function

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

Syntax

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

Parameters

Table 3–2 describes the parameters available in GET_CALLBACK_URL function.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_x01 through p_x10</td>
<td>Optional parameters that the external login passes to the authentication plugin.</td>
</tr>
</tbody>
</table>

Example

See example in “CALLBACK Procedure” on page 3-3.

See Also:  "CALLBACK Procedure" on page 3-3
GET_LOGIN_USERNAME_COOKIE Function

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

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

Parameters
Table 3–3 describes the parameters available in GET_LOGIN_USERNAME_COOKIE function.

Table 3–3  APEX_AUTHENTICATION.GET_LOGIN_USERNAME_COOKIE Function
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_cookie_name</td>
<td>The cookie name that stores the username in the browser.</td>
</tr>
</tbody>
</table>

Example
In this example, GET_LOGIN_USERNAME_COOKIE saves the username cookie value into the page item P101_USERNAME.

:P101_USERNAME := apex_authentication.get_login_username_cookie;
IS_AUTHENTICATED Function

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

Syntax
APEX_AUTHENTICATION.IS_AUTHENTICATED
  return BOOLEAN;

Parameters
None.

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

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

See Also:  "IS_PUBLIC_USER Function" on page 3-8
IS_PUBLIC_USER Function

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

Syntax
APEX_AUTHENTICATION.IS_PUBLIC_USER
return BOOLEAN;

Parameters
None.

Example
In this example, IS_PUBLIC_USER is used to show a notification if the user has not already logged in or the username if the user has not.

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

This procedure authenticates the user in the current session.

Login processing has the following steps:

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

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

Parameters
Table 3–4 describes the parameters available in LOGIN Procedure.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>The user’s name.</td>
</tr>
<tr>
<td>p_password</td>
<td>The user’s password.</td>
</tr>
<tr>
<td>p_uppercase_username</td>
<td>If TRUE then p_username is converted to uppercase.</td>
</tr>
</tbody>
</table>

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

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

See Also:  "POST.LOGIN Procedure" on page 3-11
LOGOUT Procedure

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

**Syntax**

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

**Parameters**

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_session_id</td>
<td>The Application Express session identifier of the session to close.</td>
</tr>
<tr>
<td>p_app_id</td>
<td>The database application identifier.</td>
</tr>
<tr>
<td>p_ws_app_id</td>
<td>The websheet application identifier.</td>
</tr>
</tbody>
</table>

**Example**

This example logs the session out.

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

Syntax

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

Parameters

Table 3–6 describes the parameters available in POST_LOGIN Procedure.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>The user’s name.</td>
</tr>
<tr>
<td>p_password</td>
<td>The user’s password.</td>
</tr>
<tr>
<td>p_uppercase_username</td>
<td>If TRUE then p_username is converted to uppercase.</td>
</tr>
</tbody>
</table>

Example

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

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

See Also:  "LOGIN Procedure" on page 3-9
SEND_LOGIN_USERNAME_COOKIE Procedure

This procedure sends a cookie with the username.

Syntax

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

Parameters

Table 3–7 describes the parameters available in SEND_LOGIN_USERNAME_COOKIE Procedure.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_username</td>
<td>The user’s name.</td>
</tr>
<tr>
<td>p_cookie_name</td>
<td>The cookie name which stores p_username in the browser.</td>
</tr>
</tbody>
</table>

Example

This example shows how to call SEND_LOGIN_USERNAME_COOKIE, to save the value of item P101_USERNAME in the login cookie.

apex_authentication.send_login_username_cookie (  
  p_username => :P101_USERNAME );
Collections enable you to temporarily capture one or more nonscalar values. You can use collections to store rows and columns currently in session state so they can be accessed, manipulated, or processed during a user’s specific session. You can think of a collection as a bucket in which you temporarily store and name rows of information.

Topics:
- About the APEX_COLLECTION API
- ADD_MEMBER Procedure
- ADD_MEMBER Function
- ADD_MEMBERS Procedure
- COLLECTION_EXISTS Function
- COLLECTION_HAS_CHANGED Function
- COLLECTION_MEMBER_COUNT Function
- CREATE_COLLECTION Procedure
- CREATE_OR_TRUNCATE_COLLECTION Procedure
- CREATE_COLLECTION_FROM_QUERY Procedure
- CREATE_COLLECTION_FROM_QUERY2 Procedure
- CREATE_COLLECTION_FROM_QUERY_B Procedure
- CREATE_COLLECTION_FROM_QUERYB2 Procedure
- DELETE_ALL_COLLECTIONS Procedure
- DELETE_ALL_COLLECTIONS_SESSION Procedure
- DELETE_COLLECTION Procedure
- DELETE_MEMBER Procedure
- DELETE_MEMBERS Procedure
- GET_MEMBER_MD5 Function
- MERGE_MEMBERS Procedure
- MOVE_MEMBER_DOWN Procedure
- MOVE_MEMBER_UP Procedure
- RESEQUENCE_COLLECTION Procedure
- RESET_COLLECTION_CHANGED Procedure
- RESET_COLLECTION_CHANGED_ALL Procedure
- SORT_MEMBERS Procedure
- TRUNCATE_COLLECTION Procedure
- UPDATE_MEMBER Procedure
- UPDATE_MEMBERS Procedure
- UPDATE_MEMBER_ATTRIBUTE Procedure Signature 1
- UPDATE_MEMBER_ATTRIBUTE Procedure Signature 2
- UPDATE_MEMBER_ATTRIBUTE Procedure Signature 3
- UPDATE_MEMBER_ATTRIBUTE Procedure Signature 4
- UPDATE_MEMBER_ATTRIBUTE Procedure Signature 5
- UPDATE_MEMBER_ATTRIBUTE Procedure Signature 6
About the APEX_COLLECTION API

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

The following are examples of when you might use collections:

- When you are creating a data-entry wizard in which multiple rows of information first need to be collected within a logical transaction. You can use collections to temporarily store the contents of the multiple rows of information, before performing the final step in the wizard when both the physical and logical transactions are completed.

- When your application includes an update page on which a user updates multiple detail rows on one page. The user can make many updates, apply these updates to a collection and then call a final process to apply the changes to the database.

- When you are building a wizard where you are collecting an arbitrary number of attributes. At the end of the wizard, the user then performs a task that takes the information temporarily stored in the collection and applies it to the database.

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

Topics:
- Naming, Creating and Accessing Collections
- Merging, Truncating and Deleting Collections
- Adding, Updating and Deleting Collection Members
- Managing Collections
Naming, Creating and Accessing Collections

**Topics:**
- Naming Collections
- Creating a Collection
- About the Parameter p_generate_md5

**Naming Collections**

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

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

**See Also:** "Accessing a Collection" on page 4-5, "CREATE_COLLECTION Procedure Parameters" on page 4-21, "CREATE_OR_TRUNCATE_COLLECTION Procedure Parameters" on page 4-22

**Creating a Collection**

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

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

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

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

- **CREATE_COLLECTION_FROM_QUERY2**
  This method creates a collection and then populates it with the results of a specified query. An exception is raised if the named collection exists. It is identical to the `CREATE_COLLECTION_FROM_QUERY` method, however, the first 5 columns of the `SELECT` clause must be numeric. After the numeric columns, there can be up to 50 character columns in the `SELECT` clause.

- **CREATE_COLLECTION_FROM_QUERY_B**
  This method offers significantly faster performance than the `CREATE_COLLECTION_FROM_QUERY` method by performing bulk SQL operations, but has the following limitations:
About the APEX_COLLECTION API

- No column value in the select list of the query can be more than 2,000 bytes. If a row is encountered that has a column value of more than 2,000 bytes, an error is raised during execution.

- The MD5 checksum is not computed for any members in the collection.

- `CREATE_COLLECTION_FROM_QUERYB2`

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

  **See Also:** "CREATE_COLLECTION Procedure" on page 4-21, "CREATE_OR_TRUNCATE_COLLECTION Procedure" on page 4-22, "CREATE_COLLECTION_FROM_QUERY Procedure" on page 4-23, "CREATE_COLLECTION_FROM_QUERY2 Procedure" on page 4-24, "CREATE_COLLECTION_FROM_QUERY_B Procedure" on page 4-25, "CREATE_COLLECTION_FROM_QUERYB2 Procedure" on page 4-27

About the Parameter `p_generate_md5`

Use the `p_generate_md5` flag to specify if the message digest of the data of the collection member should be computed. By default, this flag is set to `NO`. Use this parameter to check the MD5 of the collection member (that is, compare it with another member or see if a member has changed).

  **See Also:** "Determining Collection Status" on page 4-11 for information about using the `GET_MEMBER_MD5` function, "GET_MEMBER_MD5 Function" on page 4-34

Accessing a Collection

You can access the members of a collection by querying the database view `APEX_COLLECTIONS`. The `APEX_COLLECTIONS` view has the following definition:

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

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

```sql
SELECT c001, c002, c003, n001, clob001
FROM APEX_collections
```
WHERE collection_name = 'DEPARTMENTS'
Merging, Truncating and Deleting Collections

Topics:
- Merging Collections
- Truncating a Collection
- Deleting a Collection
- Deleting All Collections for the Current Application
- Deleting All Collections in the Current Session

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

See Also: “MERGE_MEMBERS Procedure” on page 4-35

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

See Also: “TRUNCATE_COLLECTION Procedure” on page 4-43

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

See Also: “DELETE_COLLECTION Procedure” on page 4-31

Deleting All Collections for the Current Application
Use the `DELETE_ALL_COLLECTIONS` method to delete all collections defined in the current application.

See Also: “DELETE_ALL_COLLECTIONS Procedure” on page 4-29

Deleting All Collections in the Current Session
Use the `DELETE_ALL_COLLECTIONS_SESSION` method to delete all collections defined in the current session.

See Also: “DELETE_ALL_COLLECTIONS_SESSION Procedure” on page 4-30
Adding, Updating and Deleting Collection Members

Topics:
- Adding Members to a Collection
- About the Parameters p_generate_md5, p_clob001, p_blob001, and p_xmltype001
- Updating Collection Members
- Deleting Collection Members

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

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

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

See Also: "ADD_MEMBER Procedure" on page 4-12, "ADD_MEMBER Function" on page 4-14, "ADD_MEMBERS Procedure" on page 4-16

About the Parameters p_generate_md5, p_clob001, p_blob001, and p_xmltype001
Use the `p_generate_md5` flag to specify if the message digest of the data of the collection member should be computed. By default, this flag is set to NO. Use this parameter to check the MD5 of the collection member (that is, compare it with another member or see if a member has changed).

Use `p_clob001` for collection member attributes which exceed 4,000 characters. Use `p_blob001` for binary collection member attributes. Use `p_xmltype001` to store well-formed XML.

See Also: "Determining Collection Status" on page 4-11 for information about using the function `GET_MEMBER_MD5`

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

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

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

Deleting Collection Members

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

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

See Also: "DELETE_MEMBER Procedure" on page 4-32, "DELETE_MEMBERS Procedure" on page 4-33
Managing Collections

Topics:
- Obtaining a Member Count
- Resequencing a Collection
- Verifying Whether a Collection Exists
- Adjusting a Member Sequence ID
- Sorting Collection Members
- Clearing Collection Session State

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

See Also: "COLLECTION_MEMBER_COUNT Function" on page 4-20

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

See Also: "RESEQUENCE_COLLECTION Procedure" on page 4-39

Verifying Whether a Collection Exists
Use COLLECTION_EXISTS to determine if a collection exists.

See Also: "COLLECTION_EXISTS Function" on page 4-18

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

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

See Also: "MOVE_MEMBER_DOWN Procedure" on page 4-37,
"MOVE_MEMBER_UP Procedure" on page 4-38

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

See Also: "SORT_MEMBERS Procedure" on page 4-42

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

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

APEX_COLLECTION.TRUNCATE_COLLECTION(
    p_collection_name => collection name);

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

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

**See Also:** "TRUNCATE_COLLECTION Procedure" on page 4-43

### Determining Collection Status

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

You can reset this flag manually by calling RESET_COLLECTION_CHANGED.

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

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

**See Also:** "RESET_COLLECTION_CHANGED Procedure" on page 4-40, "COLLECTION_HAS_CHANGED Function" on page 4-19, "GET_MEMBER_MD5 Function" on page 4-34
ADD_MEMBER Procedure

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

Syntax

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

Parameters

Table 4–1 describes the parameters available in the ADD_MEMBER procedure.

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

Note: Any character attribute exceeding 4,000 characters is truncated to 4,000 characters.
Example
The following is an example of the ADD_MEMBER procedure.

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

See Also: "GET_MEMBER_MD5 Function" on page 4-34, "ADD_MEMBER Function" on page 4-14, "ADD_MEMBERS Procedure" on page 4-16
ADD_MEMBER Function

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

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

Parameters
Table 4–2 describes the parameters available in the ADD_MEMBER function.

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

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

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

See Also: "GET_MEMBER_MD5 Function" on page 4-34, "ADD_MEMBER Procedure" on page 4-12, "ADD_MEMBERS Procedure" on page 4-16
ADD_MEMBERS Procedure

Use this procedure to add an array of members to a collection. An error is raised if the specified collection does not exist for the current user in the same session for the current Application ID. Gaps are not used when adding a new member, so an existing collection with members of sequence IDs (1,2,5,8) adds the new member with a sequence ID of 9. The count of elements in the p_c001 PL/SQL table is used as the total number of items across all PL/SQL tables. For example, if p_c001.count is 2 and p_c002.count is 10, only 2 members are added. If p_c001 is null an application error is raised.

Syntax

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

Parameters

Table 4–3 describes the parameters available in the ADD_MEMBERS procedure.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_collection_name</td>
<td>The name of an existing collection. Maximum length is 255 bytes. Collection names are not case sensitive and are converted to upper case.</td>
</tr>
<tr>
<td>p_c001 through p_c050</td>
<td>Array of character attribute values to be added.</td>
</tr>
<tr>
<td>p_n001 through p_n005</td>
<td>Array of numeric attribute values to be added.</td>
</tr>
<tr>
<td>p_d001 through p_d005</td>
<td>Array of date attribute values to be added.</td>
</tr>
</tbody>
</table>
Example
The following example shows how to add two new members to the EMPLOYEE table.

Begin
APEX_COLLECTION.ADD_MEMBERS(
    p_collection_name => 'EMPLOYEE',
    p_c001 => 1_arr1,
    p_c002 => 1_arr2);
End;

See Also:  "GET_MEMBER_MD5 Function" on page 4-34, "ADD_MEMBER Procedure" on page 4-12, "ADD_MEMBER Function" on page 4-14
COLLECTION_EXISTS Function

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

Syntax

APEX_COLLECTION.COLLECTION_EXISTS (    p_collection_name IN VARCHAR2)    RETURN BOOLEAN;

Parameters

Table 4–4 describes the parameters available in the COLLECTION_EXISTS function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_collection_name</td>
<td>The name of the collection. Maximum length is 255 bytes. The collection name is not case sensitive and is converted to upper case.</td>
</tr>
</tbody>
</table>

Example

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

Begin
    l_exists := APEX_COLLECTION.COLLECTION_EXISTS (    p_collection_name => 'EMPLOYEES';
End;
COLLECTION_HAS_CHANGED Function

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

**Syntax**

APEX_COLLECTION.COLLECTION_HAS_CHANGED (  
    p_collection_name IN VARCHAR2)  
RETURN BOOLEAN;

**Parameters**

Table 4–5 describes the parameters available in the COLLECTION_HAS_CHANGED function.

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

**Example**

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

Begin

    l_exists := APEX_COLLECTION.COLLECTION_HAS_CHANGED (  
        p_collection_name => 'EMPLOYEES';
    End;
COLLECTION_MEMBER_COUNT Function

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

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

Parameters
Table 4–6 describes the parameters available in the COLLECTION_MEMBER_COUNT function.

Table 4–6  COLLECTION_MEMBER_COUNT Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_collection_name</td>
<td>The name of the collection.</td>
</tr>
</tbody>
</table>

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

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

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

Syntax

APEX_COLLECTION.CREATE_COLLECTION(
   p_collection_name IN VARCHAR2);

Parameters

Table 4–7 describes the parameters available in the CREATE_COLLECTION procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_collection_name</td>
<td>The name of the collection. The maximum length is 255 characters. An error is returned if this collection exists with the specified name of the current user and in the same session.</td>
</tr>
</tbody>
</table>

Example

This example shows how to use the CREATE_COLLECTION procedure to create an empty collection named EMPLOYEES.

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

See Also:  "CREATE_OR_TRUNCATE_COLLECTION Procedure" on page 4-22, "CREATE_COLLECTION_FROM_QUERY Procedure" on page 4-23, "CREATE_COLLECTION_FROM_QUERY2 Procedure" on page 4-24, "CREATE_COLLECTION_FROM_QUERY_B Procedure" on page 4-25, "CREATE_COLLECTION_FROM_QUERYB2 Procedure" on page 4-27
CREATE_OR_TRUNCATE_COLLECTION Procedure

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

Syntax

APEX_COLLECTION.CREATE_OR_TRUNCATE_COLLECTION(
    p_collection_name IN VARCHAR2);

Parameters

Table 4–8 describes the parameters available in the CREATE_OR_TRUNCATE_COLLECTION procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_collection_name</td>
<td>The name of the collection. The maximum length is 255 characters. All members of the named collection are removed if the named collection exists for the current user in the current session.</td>
</tr>
</tbody>
</table>

Example

This example shows how to use the CREATE_OR_TRUNCATE_COLLECTION procedure to remove all members in an existing collection named EMPLOYEES.

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

See Also:  "CREATE_COLLECTION Procedure" on page 4-21,  "CREATE_COLLECTION_FROM_QUERY Procedure" on page 4-23,  "CREATE_COLLECTION_FROM_QUERY2 Procedure" on page 4-24,  "CREATE_COLLECTION_FROM_QUERY_B Procedure" on page 4-25,  "CREATE_COLLECTION_FROM_QUERYB2 Procedure" on page 4-27
CREATE_COLLECTION_FROM_QUERY Procedure

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

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

Parameters
Table 4–9 describes the parameters available in the CREATE_COLLECTION_FROM_QUERY procedure.

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

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

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

See Also:  "GET_MEMBER_MD5 Function" on page 4-34, "CREATE_COLLECTION Procedure" on page 4-21, "CREATE_OR_TRUNCATE_COLLECTION Procedure" on page 4-22, "CREATE_COLLECTION_FROM_QUERY2 Procedure" on page 4-24, "CREATE_COLLECTION_FROM_QUERY_B Procedure" on page 4-25, "CREATE_COLLECTION_FROM_QUERYB2 Procedure" on page 4-27
CREATE_COLLECTION_FROM_QUERY2 Procedure

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

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

Parameters
Table 4–10 describes the parameters available in the CREATE_COLLECTION_FROM_QUERY2 procedure.

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

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

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

See Also: "GET_MEMBER_MD5 Function" on page 4-34, "CREATE_COLLECTION Procedure" on page 4-21, "CREATE_OR_TRUNCATE_COLLECTION Procedure" on page 4-22, "CREATE_COLLECTION_FROM_QUERY Procedure" on page 4-23, "CREATE_COLLECTION_FROM_QUERY_B Procedure" on page 4-25, "CREATE_COLLECTION_FROM_QUERYB2 Procedure" on page 4-27
CREATE_COLLECTION_FROM_QUERY_B Procedure

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

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

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

Syntax

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

Parameters

Table 4–11 describes the parameters available in the CREATE_COLLECTION_FROM_QUERY_B procedure.

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

Example

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

Begin  
  l_query := 'select make, model, year from AUTOS';  
  APEX_COLLECTION.CREATE_COLLECTION_FROM_QUERY_B (  
    p_collection_name => 'AUTO',  
    p_query => l_query);  
End;
See Also: "GET_MEMBER_MD5 Function" on page 4-34, "CREATE_COLLECTION Procedure" on page 4-21, "CREATE_OR_TRUNCATE_COLLECTION Procedure" on page 4-22, "CREATE_COLLECTION_FROM_QUERY Procedure" on page 4-23, "CREATE_COLLECTION_FROM_QUERY2 Procedure" on page 4-24, "CREATE_COLLECTION_FROM_QUERYB2 Procedure" on page 4-27
CREATE_COLLECTION_FROM_QUERYB2 Procedure

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

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

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

Syntax

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

Parameters

Table 4–12 describes the parameters available in the CREATE_COLLECTION_FROM_QUERYB2 procedure.

Table 4–12 CREATE_COLLECTION_FROM_QUERYB2 Procedure Parameters

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

Example

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

Begin

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

End;

See Also: "GET_MEMBER_MD5 Function" on page 4-34, "CREATE_COLLECTION Procedure" on page 4-21, "CREATE_OR_TRUNCATE_COLLECTION Procedure" on page 4-22, "CREATE_COLLECTION_FROM_QUERY Procedure" on page 4-23, "CREATE_COLLECTION_FROM_QUERY2 Procedure" on page 4-24, "CREATE_COLLECTION_FROM_QUERY_B Procedure" on page 4-25
DELETE_ALL_COLLECTIONS Procedure

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

Syntax
APEX_COLLECTION.DELETE_ALL_COLLECTIONS;

Parameters
None.

Example
This example shows how to use the DELETE_ALL_COLLECTIONS procedure to remove all collections that belong to the current user in the current session and Application ID.

Begin
APEX_COLLECTION.DELETE_ALL_COLLECTIONS;
End;

See Also:  "DELETE_ALL_COLLECTIONS Procedure" on page 4-29, "DELETE_COLLECTION Procedure" on page 4-31, "DELETE_MEMBER Procedure" on page 4-32, "DELETE_MEMBERS Procedure" on page 4-33
DELETE_ALL_COLLECTIONS_SESSION Procedure

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

Syntax
APEX_COLLECTION.DELETE_ALL_COLLECTIONS_SESSION;

Parameters
None.

Example
This example shows how to use the DELETE_ALL_COLLECTIONS_SESSION procedure to remove all collections that belong to the current user in the current session regardless of Application ID.

Begin
    APEX_COLLECTION.DELETE_ALL_COLLECTIONS_SESSION;
End;

See Also:  "DELETE_ALL_COLLECTIONS Procedure" on page 4-29, "DELETE_COLLECTION Procedure" on page 4-31, "DELETE_MEMBER Procedure" on page 4-32, "DELETE_MEMBERS Procedure" on page 4-33
DELETE_COLLECTION Procedure

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

Syntax
APEX_COLLECTION.DELETE_COLLECTION ( p_collection_name IN VARCHAR2);

Parameters
Table 4-13 describes the parameters available in the DELETE_COLLECTION procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_collection_name</td>
<td>The name of the collection to remove all members from and drop. An error is returned if this collection does not exist with the specified name of the current user and in the same session.</td>
</tr>
</tbody>
</table>

Example
This example shows how to use the DELETE_COLLECTION procedure to remove the 'EMPLOYEE' collection.

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

See Also: "DELETE_ALL_COLLECTIONS_SESSION Procedure" on page 4-30, "DELETE_ALL_COLLECTIONS Procedure" on page 4-29, "DELETE_MEMBER Procedure" on page 4-32, "DELETE_MEMBERS Procedure" on page 4-33
DELETE_MEMBER Procedure

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

Syntax

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

Parameters

Table 4–14 describes the parameters available in the DELETE_MEMBER procedure.

Table 4–14 DELETE_MEMBER Parameters

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

Example

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

Begin

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

End;

See Also: "DELETE_ALL_COLLECTIONS_SESSION Procedure" on page 4-30, "DELETE_ALL_COLLECTIONS Procedure" on page 4-29, "DELETE_COLLECTION Procedure" on page 4-31, "DELETE_MEMBERS Procedure" on page 4-33
DELETE_MEMBERS Procedure

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

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

Syntax

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

Parameters

Table 4–15 describes the parameters available in the DELETE_MEMBERS procedure.

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

Example

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

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

See Also:  "DELETE_ALL_COLLECTIONS_SESSION Procedure" on page 4-30, "DELETE_ALL_COLLECTIONS Procedure" on page 4-29, "DELETE_COLLECTION Procedure" on page 4-31, "DELETE_MEMBER Procedure" on page 4-32
Use this function to compute and return the message digest of the attributes for the member specified by the sequence ID. This computation of message digest is equal to the computation performed natively by collections. Thus, the result of this function could be compared to the MD5_ORIGIN column of the view wwv_flow_collections.

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

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

Parameters
Table 4–16 describes the parameters available in the GET_MEMBER_MD5 function.

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

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

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

See Also:  "COLLECTION_HAS_CHANGED Function" on page 4-19, "RESET_COLLECTION_CHANGED Procedure" on page 4-40, "RESET_COLLECTION_CHANGED_ALL Procedure" on page 4-41
MERGE_MEMBERS Procedure

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

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

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

Syntax

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

Parameters

Table 4–17 describes the parameters available in the MERGE_MEMBERS procedure.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_collection_name</td>
<td>The name of the collection. Maximum length is 255 bytes. Collection names are not case sensitive and are converted to upper case.</td>
</tr>
<tr>
<td>p_c001 through p_c050</td>
<td>Array of attribute values to be merged. Maximum length is 4,000 bytes. Any character attribute exceeding 4,000 characters is truncated to 4,000 characters. The count of the <code>p_c001</code> array is used across all arrays. If no values are provided then no actions are performed.</td>
</tr>
<tr>
<td>p_c0xx</td>
<td>Attribute of NN attributes values to be merged. Maximum length can be 4,000 bytes. The attribute value is truncated to 4,000 bytes if greater than this amount.</td>
</tr>
<tr>
<td>p_seq</td>
<td>Identifies the sequence number of the collection to be merged.</td>
</tr>
</tbody>
</table>
Example
The following example creates a collection on the table of employees, and then merges the contents of the local arrays with the collection, updating the job of two employees.

```sql
DECLARE
  l_seq   APEX_APPLICATION_GLOBAL.VC_ARR2;
  l_c001  APEX_APPLICATION_GLOBAL.VC_ARR2;
  l_c002  APEX_APPLICATION_GLOBAL.VC_ARR2;
  l_c003  APEX_APPLICATION_GLOBAL.VC_ARR2;
BEGIN
  l_seq(1)  := 1;
  l_c001(1) := 7369;
  l_c002(1) := 'SMITH';
  l_c003(1) := 'MANAGER';
  l_seq(2)  := 2;
  l_c001(2) := 7499;
  l_c002(2) := 'ALLEN';
  l_c003(2) := 'CLERK';
  APEX_COLLECTION.MERGE_MEMBERS(
    p_collection_name => 'EMPLOYEES',
    p_seq => l_seq,
    p_c001 => l_c001,
    p_c002 => l_c002,
    p_c003 => l_c003,
    p_init_query => 'select empno, ename, job from emp order by empno');
END;
```
MOVE_MEMBER_DOWN Procedure

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

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

Parameters
Table 4–19 describes the parameters available in the MOVE_MEMBER_DOWN procedure.

Table 4–18  MOVE_MEMBER_DOWN Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_collection_name</td>
<td>The name of the collection. Maximum length is 255 bytes. Collection names are not case sensitive and are converted to upper case. An error is returned if this collection does not exist with the specified name of the current user in the same session.</td>
</tr>
<tr>
<td>p_seq</td>
<td>Identifies the sequence number of the collection member to be moved down by one.</td>
</tr>
</tbody>
</table>

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

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

See Also: "MOVE_MEMBER_UP Procedure" on page 4-38
**MOVE_MEMBER_UP Procedure**

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

**Syntax**

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

**Parameters**

Table 4–19 describes the parameters available in the MOVE_MEMBER_UP procedure.

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

**Example**

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

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

**See Also:** "MOVE_MEMBER_DOWN Procedure" on page 4-37
RESEQUENCE_COLLECTION Procedure

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

**Syntax**

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

**Parameters**

Table 4–20 describes the parameters available in the RESEQUENCE_COLLECTION procedure.

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

**Example**

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

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

**See Also:**

"MOVE_MEMBER_DOWN Procedure" on page 4-37,
"MOVE_MEMBER_UP Procedure" on page 4-38
Use this procedure to reset the collection changed flag (mark as not changed) for a given collection. If a collection does not exist with the specified name for the current user in the same session and for the current Application ID, an application error is raised.

**Syntax**

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

**Parameters**

Table 4–21 describes the parameters available in the `RESET_COLLECTION_CHANGED` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_collection_name</td>
<td>The name of the collection to reset the collection changed flag. An error is returned if this collection does not exist with the specified name of the current user and in the same session.</td>
</tr>
</tbody>
</table>

**Example**

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

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

**See Also:**

"RESET_COLLECTION_CHANGED_ALL Procedure" on page 4-41
Use this procedure to reset the collection changed flag (mark as not changed) for all collections in the user’s current session.

**Syntax**

```oracle
APEX_COLLECTION.RESET_COLLECTION_CHANGED_ALL; 
```

**Parameters**

None.

**Example**

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

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

**See Also:** "RESET_COLLECTION_CHANGED Procedure" on page 4-40.
SORT_MEMBERS Procedure

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

**Syntax**

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

**Parameters**

Table 4–22 describes the parameters available in the `SORT_MEMBERS` procedure.

**Table 4–22  SORT_MEMBERS Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p_collection_name</code></td>
<td>The name of the collection to sort. An error is returned if this collection does not exist with the specified name of the current user and in the same session.</td>
</tr>
<tr>
<td><code>p_sort_on_column_number</code></td>
<td>The column number used to sort the collection.</td>
</tr>
</tbody>
</table>

**Example**

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

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

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

Syntax
APEX_COLLECTION.TRUNCATE_COLLECTION ( 
    p_collection_name IN VARCHAR2);

Parameters
Table 4–23 describes the parameters available in the TRUNCATE_COLLECTION procedure.

Table 4–23 TRUNCATE_COLLECTION Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_collection_name</td>
<td>The name of the collection to truncate. An error is returned if this collection does not exist with the specified name of the current user and in the same session.</td>
</tr>
</tbody>
</table>

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

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

See Also: "CREATE_OR_TRUNCATE_COLLECTION Procedure" on page 4-22
UPDATE_MEMBER Procedure

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

Note: Using this procedure sets the columns identified and nullifies any columns not identified. To update specific columns, without affecting the values of other columns, use “UPDATE_MEMBER_ATTRIBUTE Procedure Signature 1” on page 4-48.

Syntax

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

Parameters

Table 4–24 describes the parameters available in the UPDATE_MEMBER procedure.

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

Table 4–24  UPDATE_MEMBER Parameters

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

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

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

See Also: "UPDATE_MEMBERS Procedure" on page 4-46
UPDATE_MEMBERS Procedure

Use this procedure to update the array of members for the given named collection. If a collection does not exist with the specified name for the current user in the same session and for the current Application ID, an application error is raised. The count of elements in the p_seq PL/SQL table is used as the total number of items across all PL/SQL tables. That is, if p_seq.count = 2 and p_c001.count = 10, only 2 members are updated. If p_seq is null, an application error is raised. If the member specified by sequence ID p_seq does not exist, an application error is raised.

Syntax

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

Parameters

Table 4–25 describes the parameters available in the UPDATE_MEMBERS procedure.

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

Example

DECLARE

Note: Any character attribute exceeding 4,000 characters is truncated to 4,000 characters. Also, the number of members added is based on the number of elements in the first array.
l_seq   apex_application_global.vc_arr2;
l_carr  apex_application_global.vc_arr2;
l_narr  apex_application_global.n_arr;
l_darr  apex_application_global.d_arr;
BEGIN
  l_seq(1) := 10;
l_seq(2) := 15;
l_carr(1) := 'Apples';
l_carr(2) := 'Grapes';
l_narr(1) := 100;
l_narr(2) := 150;
l_darr(1) := sysdate;
l_darr(2) := sysdate;

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

See Also:  "UPDATE_MEMBER Procedure" on page 4-44
UPDATE MEMBER_ATTRIBUTE Procedure Signature 1

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

**Syntax**

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

**Parameters**

Table 4–26 describes the parameters available in the **UPDATE MEMBER_ATTRIBUTE** procedure signature 1.

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

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

**Example**

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

```
BEGIN;
    APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (
        p_collection_name => 'Departments',
        p_seq => '2',
        p_attr_number => '1',
        p_attr_value => 'Engineering');
END;
```
See Also:  "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 2" on page 4-50, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 3" on page 4-52, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 4" on page 4-54, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 5" on page 4-56, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 6" on page 4-58
UPDATE_MEMBER_ATTRIBUTE Procedure Signature 2

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

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

Parameters
Table 4–27 describes the parameters available in the UPDATE_MEMBER_ATTRIBUTE procedure signature 2.

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

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

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

BEGIN;
APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (
    p_collection_name => 'Departments',
    p_seq => '2',
    p_clob_number => '1',
    p_clob_value => 'Engineering');
END;
See Also:  "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 1" on page 4-48, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 3" on page 4-52, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 4" on page 4-54, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 5" on page 4-56, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 6" on page 4-58
UPDATE_MEMBER_ATTRIBUTE Procedure Signature 3

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

Syntax

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

Parameters

Table 4–28 describes the parameters available in the UPDATE_MEMBER_ATTRIBUTE procedure signature 3.

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

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

Example

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

```
BEGIN;
    APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (  
        p_collection_name => 'Departments',  
        p_seq => '2',  
        p_blob_number => '1',  
        p_blob_value => l_blob_content);
END;
```
See Also: "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 1" on page 4-48, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 2" on page 4-50, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 4" on page 4-54, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 5" on page 4-56, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 6" on page 4-58
UPDATE_MEMBER_ATTRIBUTE Procedure Signature 4

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

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

Parameters
Table 4–29 describes the parameters available in the UPDATE_MEMBER_ATTRIBUTE procedure signature 4.

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

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

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

BEGIN;
APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE {
    p_collection_name => 'Departments',
    p_seq => '2',
    p_xmltype_number => '1',
    p_xmltype_value  => l_xmltype_content};
END;
See Also:  "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 1" on page 4-48, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 2" on page 4-50, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 3" on page 4-52, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 5" on page 4-56, “UPDATE_MEMBER_ATTRIBUTE Procedure Signature 6” on page 4-58
UPDATE_MEMBER_ATTRIBUTE Procedure Signature 5

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

Syntax

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

Parameters

Table 4–30 describes the parameters available in the UPDATE_MEMBER_ATTRIBUTE procedure signature 5.

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

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

Example

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

BEGIN;
    APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE (  
        p_collection_name => 'Departments',  
        p_seq => '2',  
        p_attr_number => '1',  
        p_number_value => 3000);
END;
See Also:  "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 1" on page 4-48, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 2" on page 4-50, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 3" on page 4-52, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 4" on page 4-54, “UPDATE_MEMBER_ATTRIBUTE Procedure Signature 6” on page 4-58, “UPDATE_MEMBER_ATTRIBUTE Procedure Signature 6” on page 4-58
UPDATE_MEMBER_ATTRIBUTE Procedure Signature 6

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

Syntax

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

Parameters

Table 4–30 describes the parameters available in the UPDATE_MEMBER_ATTRIBUTE procedure signature 6.

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

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

Example

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

BEGIN;
    APEX_COLLECTION.UPDATE_MEMBER_ATTRIBUTE ( 
        p_collection_name => 'Departments', 
        p_seq => '2', 
        p_attr_number => '1', 
        p_number_value => 100 );
END;
See Also:  "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 1" on page 4-48, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 2" on page 4-50, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 3" on page 4-52, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 4" on page 4-54, "UPDATE_MEMBER_ATTRIBUTE Procedure Signature 5" on page 4-56
The \texttt{APEX\_CSS} package provides utility functions for adding CSS styles to HTTP output. This package is usually used for plug-in development.

\textbf{Topics:}
- \texttt{ADD Procedure}
- \texttt{ADD\_3RD\_PARTY\_LIBRARY\_FILE Procedure}
- \texttt{ADD\_FILE Procedure}
ADD Procedure

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

Syntax

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

Parameters

Table 5–1 describes the parameters available in the ADD procedure.

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

Example

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

```
apex_css.add (  
    p_css => '.autocomplete { color:#ffffff }',  
    p_key => 'autocomplete_widget');  
```
ADD_3RD_PARTY_LIBRARY_FILE Procedure

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

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

Syntax

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

Parameters

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

Example

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

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

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

Syntax

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

Parameters

Table 5–3 describes the parameters available in the ADD_FILE procedure.

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

Example

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

apex_css.add_file (p_name => 'jquery.autocomplete', p_directory => p_plugin.image_prefix);
You can use the APEX_CUSTOM_AUTH package to perform various operations related to authentication and session management.

**Topics:**
- 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 6–1 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>p_item_name</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 6–2 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 6-21 and "SET_SESSION_ID Procedure" on page 6-19.
GET_COOKIE_PROPS Procedure

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

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

Parameters
Table 6–3 describes the parameters available in the GET_COOKIE_PROPS procedure.

Table 6–3 GET_COOKIE_PROPS Parameters

<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>
<tr>
<td>p_secure</td>
<td>Flag to set secure property of cookie.</td>
</tr>
</tbody>
</table>

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

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

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

Syntax

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

Parameters

Table 6–4 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_use_ssl</td>
<td>Whether SSL is used.</td>
</tr>
<tr>
<td>p_use_exact_dn</td>
<td>Whether exact distinguished names are used.</td>
</tr>
<tr>
<td>p_search_filter</td>
<td>The search filter used if exact DN is not used.</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_use_ssl            VARCHAR2(1);
    l_use_exact_dn       VARCHAR2(1);
    l_search_filter      VARCHAR2(256);
    l_ldap_dn            VARCHAR2(256);
    l_ldap_edit_function VARCHAR2(256);
BEGIN
    APEX_CUSTOM_AUTH.GET_LDAP_PROPS ( 
        p_ldap_host       => l_ldap_host,
        p_ldap_port       => l_ldap_port,
        p_use_ssl         => l_use_ssl,
        p_use_exact_dn    => l_use_exact_dn,
        p_search_filter   => l_search_filter,
        p_ldap_dn         => l_ldap_dn,
        p_ldap_edit_function => l_ldap_edit_function);

END;
GET_NEXT_SESSION_ID Function

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

**Syntax**

APEX_CUSTOM_AUTH.GET_NEXT_SESSION_ID
RETURN NUMBER;

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

DECLARE
   VAL NUMBER;
BEGIN
   VAL := APEX_CUSTOM_AUTH.GET_NEXT_SESSION_ID;
END;
GET_SECURITY_GROUP_ID Function

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

Syntax
APEX_CUSTOM_AUTH.GET_SECURITY_GROUP_ID
RETURN NUMBER;

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

DECLARE
    VAL NUMBER;
BEGIN
    VAL := APEX_CUSTOM_AUTH.GET_SECURITY_GROUP_ID;
END;
GET_SESSION_ID Function

This function returns APEX_APPLICATION.G_INSTANCE global variable. GET_SESSION_ID returns a number.

Syntax
APEX_CUSTOM_AUTH.GET_SESSION_ID
RETURN NUMBER;

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

DECLARE
    VAL NUMBER;
BEGIN
    VAL := APEX_CUSTOM_AUTH.GET_SESSION_ID;
END;
GET_SESSION_ID_FROM_COOKIE Function

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

Syntax
APEX_CUSTOM_AUTH.GET_SESSION_ID_FROM_COOKIE
RETURN NUMBER;

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

DECLARE
    VAL NUMBER;
BEGIN
    VAL := APEX_CUSTOM_AUTH.GET_SESSION_ID_FROM_COOKIE;
END;
GET_USER Function

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

Syntax
APEX_CUSTOM_AUTH.GET_USER
RETURN VARCHAR2;

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

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

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

Syntax
APEX_CUSTOM_AUTH.GET_USERNAME
RETURN VARCHAR2;

Example
The following example retrieves the username registered with the current application session.
DECLARE
  VAL VARCHAR2(256);
BEGIN
  VAL := APEX_CUSTOM_AUTH.GET_USERNAME;
END;
IS_SESSION_VALID Function

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

Syntax

APEX_CUSTOM_AUTH.IS_SESSION_VALID
RETURN BOOLEAN;

Example

The following example verifies whether the current session is valid.

DECLARE
  L_VAL BOOLEAN;
BEGIN
  L_VAL := APEX_CUSTOM_AUTH.IS_SESSION_VALID;
  IF L_VAL THEN
    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 6–5 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**

```sql
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 6–6 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.

```sql
BEGIN
    APEX_CUSTOM_AUTH.LOGOUT (  
        p_this_app => '1000',  
        p_next_app_page_sess => '1000:99');
END;
```
POST_LOGIN Procedure

This procedure performs session registration, assuming the authentication step has been completed. It can be called only from within an Oracle Application Express application page context.

Syntax

```
APEX_CUSTOM_AUTH.POST_LOGIN(
    p_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 6–7 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.

```
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 6–8 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 6–9 describes the parameters available in the SET_USER procedure.

Table 6–9  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;
The APEX_DEBUG package provides utility functions for managing the debug message log. Specifically, this package provides the necessary APIs to instrument and debug PL/SQL code contained within your Application Express application as well as PL/SQL code in database stored procedures and functions. Instrumenting your PL/SQL code makes it much easier to track down bugs and isolate unexpected behavior more quickly.

The package also provides the means to enable and disable debugging at different debug levels and utility procedures to clean up the message log.

You can view the message log either as described in the "Accessing Debugging Mode" section of the Oracle Application Express Application Builder User's Guide or by querying the APEX_DEBUG_MESSAGES view.

Please see the individual API descriptions for further information.

---

**Note:** In Oracle Application Express 4.2, the APEX_DEBUG_MESSAGE package was renamed to APEX_DEBUG. The APEX_DEBUG_MESSAGE package name is still supported to provide backward compatibility. As a best practice, however, use the new APEX_DEBUG package for new applications unless you plan to run them in an earlier version of Oracle Application Express.

---

**Topics:**
- Constants
- DISABLE Procedure
- ENABLE Procedure
- ENTER Procedure
- ERROR Procedure
- INFO Procedure
- LOG_DBMS_OUTPUT Procedure
- LOG_LONG_MESSAGE Procedure
- LOG_MESSAGE Procedure [Deprecated]
- LOG_PAGE_SESSION_STATE Procedure
- MESSAGE Procedure
- REMOVE_DEBUG_BY_AGE Procedure
- REMOVE_DEBUG_BY_APP Procedure
- REMOVE_DEBUG_BY_VIEW Procedure
- REMOVE_SESSION_MESSAGES Procedure
- TOCHAR Function
- TRACE Procedure
- WARN Procedure
The following constants are used by this package.

```plaintext
subtype t_log_level is pls_integer;
c_log_level_error constant t_log_level := 1; -- critical error
c_log_level_warn constant t_log_level := 2; -- less critical error
c_log_level_info constant t_log_level := 4; -- default level if debugging is
   enabled (for example, used by apex_application.debug)
c_log_level_app_enter constant t_log_level := 5; -- application: messages when
   procedures/functions are entered

c_log_level_app_trace constant t_log_level := 6; -- application: other messages
   within procedures/functions

c_log_level_engine_enter constant t_log_level := 8; -- Application Express engine:
   messages when procedures/functions are entered

c_log_level_engine_trace constant t_log_level := 9; -- Application Express engine:
   other messages within procedures/functions
```
DISABLE Procedure

This procedure turns off debug messaging.

Syntax
APEX_DEBUG.DISABLE;

Parameters
None.

Example
This example shows how you can turn off debug messaging.
BEGIN
    APEX_DEBUG.DISABLE();
END;

See Also: "ENABLE Procedure" on page 7-5
ENABLE Procedure

This procedure turns on debug messaging. You can specify, by level of importance, the types of debug messages that are monitored.

**Note:** You only need to call `ENABLE` procedure once per page view or page accept.

**Syntax**

```sql
APEX_DEBUG.ENABLE (    p_level  IN  T_LOG_LEVEL DEFAULT C_LOG_LEVEL_INFO );
```

**Parameters**

Table 7–1 describes the parameters available in the `APEX_DEBUG.ENABLE` procedure.

**Table 7–1  APEX_DEBUG.ENABLE Procedure Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_level</td>
<td>Level or levels of messages to log. Must be an integer from 1 to 9, where level 1 is the most important messages and level 9 (the default) is the least important. Setting to a specific level logs messages both at that level and below that level. For example, setting <code>p_level</code> to 2 logs any message at level 1 and 2.</td>
</tr>
</tbody>
</table>

**Example**

This example shows how to enable logging of messages for levels 1, 2 and 4. Messages at higher levels are not logged.

```sql
BEGIN
    APEX_DEBUG.ENABLE(        apex_debug.c_log_level_info);
END;
```
**ENTER Procedure**

This procedure logs messages at level `c_log_level_app_enter`. Use APEX_DEBUG.ENTER() to log the routine name and its arguments at the beginning of a procedure or function.

**Syntax**

```sql
APEX_DEBUG.ENTER ( p_routine_name IN VARCHAR2,
    p_name01 IN VARCHAR2 DEFAULT NULL,
    p_value01 IN VARCHAR2 DEFAULT NULL,
    p_name02 IN VARCHAR2 DEFAULT NULL,
    p_value02 IN VARCHAR2 DEFAULT NULL,
    p_name03 IN VARCHAR2 DEFAULT NULL,
    p_value03 IN VARCHAR2 DEFAULT NULL,
    p_name04 IN VARCHAR2 DEFAULT NULL,
    p_value04 IN VARCHAR2 DEFAULT NULL,
    p_name05 IN VARCHAR2 DEFAULT NULL,
    p_value05 IN VARCHAR2 DEFAULT NULL,
    p_name06 IN VARCHAR2 DEFAULT NULL,
    p_value06 IN VARCHAR2 DEFAULT NULL,
    p_name07 IN VARCHAR2 DEFAULT NULL,
    p_value07 IN VARCHAR2 DEFAULT NULL,
    p_name08 IN VARCHAR2 DEFAULT NULL,
    p_value08 IN VARCHAR2 DEFAULT NULL,
    p_name09 IN VARCHAR2 DEFAULT NULL,
    p_value09 IN VARCHAR2 DEFAULT NULL,
    p_name10 IN VARCHAR2 DEFAULT NULL,
    p_value10 IN VARCHAR2 DEFAULT NULL,
    p_value_max_length IN PLS_INTEGER DEFAULT 1000 );
```

**Parameters**

Table 7–2 describes the parameters available for the APEX_DEBUG.ENTER procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_routine_name</td>
<td>The name of the procedure or function.</td>
</tr>
<tr>
<td>p_namexx/p_valuexx</td>
<td>The procedure or function parameter name and value.</td>
</tr>
<tr>
<td>p_value_max_length</td>
<td>The <code>p_valuexx</code> values is truncated to this length.</td>
</tr>
</tbody>
</table>

**Example**

This example shows how to use APEX_ENTER to add a debug message at the beginning of a procedure.

```sql
procedure foo ( p_widget_id in number,
    p_additional_data in varchar2,
    p_emp_rec in emp%rowtype )
is
begin
    apex_debug.enter('foo',
        'p_widget_id', p_widget_id,
        'p_additional_data', p_additional_data,
        'p_emp_rec.id', p_emp_rec.id);
```

**Table 7–2 APEX_DEBUG.ENTER Procedure Parameters**
....do something....
end foo;

See Also:  "MESSAGE Procedure" on page 7-14, "ERROR Procedure" on page 7-8, "WARN Procedure" on page 7-22, "TRACE Procedure" on page 7-21, "INFO Procedure" on page 7-9
This procedure logs messages at level `c_log_level_error`. This procedure always logs, even if debug mode is turned off.

**Syntax**

```
APEX_DEBUG.ERROR (
    p_message IN VARCHAR2,
    p0 IN VARCHAR2 DEFAULT NULL,
    p1 IN VARCHAR2 DEFAULT NULL,
    p2 IN VARCHAR2 DEFAULT NULL,
    p3 IN VARCHAR2 DEFAULT NULL,
    p4 IN VARCHAR2 DEFAULT NULL,
    p5 IN VARCHAR2 DEFAULT NULL,
    p6 IN VARCHAR2 DEFAULT NULL,
    p7 IN VARCHAR2 DEFAULT NULL,
    p8 IN VARCHAR2 DEFAULT NULL,
    p9 IN VARCHAR2 DEFAULT NULL,
    p_max_length IN PLS_INTEGER DEFAULT 1000 );
```

**Parameters**

Table 7–3 describes parameters available for the ERROR procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_message</td>
<td>The debug message. Occurrences of '%s' are replaced by p0 to p19, as in utl_lms.format_message and C's sprintf. Occurrences of '%%' represent the special character '%'. Occurrences of '%&lt;n&gt;' are replaced by p&lt;n&gt;.</td>
</tr>
<tr>
<td>p0 through p9</td>
<td>Substitution strings for '%s' placeholders.</td>
</tr>
<tr>
<td>p_max_length</td>
<td>The p&lt;n&gt; values are truncated to this length.</td>
</tr>
</tbody>
</table>

**Example**

This example shows how to use `APEX_ERROR` to log a critical error in the debug log.

```
apex_debug.error('Critical error %s', sqlerrm);
```

**See Also:**  
"MESSAGE Procedure" on page 7-14, "ERROR Procedure" on page 7-8, "WARN Procedure" on page 7-22, "TRACE Procedure" on page 7-21, "INFO Procedure" on page 7-9
INFO Procedure

This procedure logs messages at level c_log_level_info. This procedure always logs, even if debug mode is turned off.

Syntax
APEX_DEBUG.INFO (  
    p_message IN VARCHAR2,  
    p0 IN VARCHAR2 DEFAULT NULL,  
    p1 IN VARCHAR2 DEFAULT NULL,  
    p2 IN VARCHAR2 DEFAULT NULL,  
    p3 IN VARCHAR2 DEFAULT NULL,  
    p4 IN VARCHAR2 DEFAULT NULL,  
    p5 IN VARCHAR2 DEFAULT NULL,  
    p6 IN VARCHAR2 DEFAULT NULL,  
    p7 IN VARCHAR2 DEFAULT NULL,  
    p8 IN VARCHAR2 DEFAULT NULL,  
    p9 IN VARCHAR2 DEFAULT NULL,  
    p_max_length IN PLS_INTEGER DEFAULT 1000 );

Parameters
Table 7–4 describes parameters available for the APEX_DEBUG.INFO procedure.

Table 7–4  APEX_DEBUG.INFO Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_message</td>
<td>The debug message. Occurrences of ‘%s’ are replaced by p0 to p9, as in utl_lms.format_message and C’s sprintf. Occurrences of ‘%%’ represent the special character ‘%’. Occurrences of ‘%&lt;n&gt;’ are replaced by p&lt;n&gt;.</td>
</tr>
<tr>
<td>p0 through p9</td>
<td>Substitution strings for ‘%s’ placeholders.</td>
</tr>
<tr>
<td>p_max_length</td>
<td>The p&lt;n&gt; values are truncated to this length.</td>
</tr>
</tbody>
</table>

Example
This example shows how to use APEX_DEBUG.INFO to log information in the debug log.

apex_debug.info('Important: %s', 'fnord');

See Also:  "MESSAGE Procedure" on page 7-14, "ERROR Procedure" on page 7-8, "WARN Procedure" on page 7-22, "TRACE Procedure" on page 7-21, "ENTER Procedure" on page 7-6
This procedure writes the contents of `dbms_output.get_lines` to the debug log. Messages of legacy applications which use `dbms_output` are copied into the debug log.

**Syntax**
```
APEX_DEBUG.LOG_DBMS_OUTPUT;
```

**Parameters**
None.

**Example**
This example shows how to log the contents of the DBMS_OUTPUT buffer in the debug log.
```
sys.dbms_output.put_line('some data');
sys.dbms_output.put_line('other data');
apex_debug.log_dbms_output;
```

**See Also:**  "MESSAGE Procedure" on page 7-14, "ERROR Procedure" on page 7-8, "WARN Procedure" on page 7-22, "TRACE Procedure" on page 7-21, "INFO Procedure" on page 7-9
Use this procedure to emit debug messages from PLSQL components of Application Express, or PLSQL procedures and functions. This procedure is the same as LOG_MESSAGE, except it allows logging of much longer messages, which are subsequently split into 4,000 character chunks in the debugging output (because a single debug message is constrained to 4,000 characters).

**Note:** Instead of this procedure, use "ERROR Procedure" on page 7-8, "WARN Procedure" on page 7-22, "MESSAGE Procedure" on page 7-14, "INFO Procedure" on page 7-9, "ENTER Procedure" on page 7-6, or "TRACE Procedure" on page 7-21

**Syntax**

APEX_DEBUG.LOG_LONG_MESSAGE (  
    p_message    IN VARCHAR2  DEFAULT NULL,  
    p_enabled    IN BOOLEAN   DEFAULT FALSE,  
    p_level      IN T_LOG_LEVEL DEFAULT C_LOG_LEVEL_APP_TRACE);  

**Parameters**

Table 7–5 describes parameters available for the APEX_DEBUG.LOG_LONG_MESSAGE procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_message</td>
<td>Log long message with maximum size of 32767 bytes.</td>
</tr>
<tr>
<td>p_enabled</td>
<td>Set to TRUE to always log messages, irrespective of whether debugging is enabled. Set to FALSE to only log messages if debugging is enabled.</td>
</tr>
<tr>
<td>p_level</td>
<td>Identifies the level of the long log message. See &quot;Constants&quot; on page 7-3.</td>
</tr>
</tbody>
</table>

**Example**

This example shows how to enable debug message logging for 1 and 2 level messages and display a level 1 message that could contain anything up to 32767 characters. Note, the p_enabled parameter need not be specified, as debugging has been explicitly enabled and the default of false for this parameter respects this enabling.

```sql
DECLARE  
l_msg VARCHAR2(32767) := 'Debug outputs anything up to varchar2 limit';  
BEGIN  
APEX_DEBUG.ENABLE (p_level => 2);  
APEX_DEBUG.LOG_LONG_MESSAGE(  
    p_message => l_msg,  
    p_level => 1 );  
END;  
```

**See Also:** "MESSAGE Procedure" on page 7-14, "ERROR Procedure" on page 7-8, "WARN Procedure" on page 7-22, "TRACE Procedure" on page 7-21, "INFO Procedure" on page 7-9
LOG_MESSAGE Procedure [Deprecated]

This procedure logs a debug message.

**Note:** Instead of this procedure, use "ERROR Procedure" on page 7-8, "WARN Procedure" on page 7-22, "MESSAGE Procedure" on page 7-14, "INFO Procedure" on page 7-9, "ENTER Procedure" on page 7-6, or "TRACE Procedure" on page 7-21

**Syntax**

APEX_DEBUG.LOG_MESSAGE (  
    p_message IN VARCHAR2 DEFAULT NULL,  
    p_enabled IN BOOLEAN DEFAULT FALSE,  
    p_level IN T_LOG_LEVEL DEFAULT C_LOG_LEVEL_APP_TRACE );

**Parameters**

Table 7-6 describes parameters available for the APEX_DEBUG.LOG_MESSAGE procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_message</td>
<td>The debug message with a maximum length of 1000 bytes.</td>
</tr>
<tr>
<td>p_enabled</td>
<td>Messages are logged when logging is enabled, setting a value of true enables logging.</td>
</tr>
<tr>
<td>p_level</td>
<td>Identifies the level of the log message where 1 is most important and 9 is least important. This is an integer value.</td>
</tr>
</tbody>
</table>

**Example**

This example shows how to enable debug message logging for 1 and 2 level messages and display a level 1 message showing a variable value. Note, the p_enabled parameter need not be specified, as debugging has been explicitly enabled and the default of false for this parameter respects this enabling.

```sql
DECLARE  
    l_value varchar2(100) := 'test value';  
BEGIN  
    APEX_DEBUG.ENABLE (p_level => 2);  

    APEX_DEBUG.LOG_MESSAGE (  
        p_message => 'l_value = ' || l_value,  
        p_level => 1  
    );

END;
```

**See Also:** "MESSAGE Procedure" on page 7-14, "ERROR Procedure" on page 7-8, "WARN Procedure" on page 7-22, "TRACE Procedure" on page 7-21, "INFO Procedure” on page 7-9
LOG_PAGE_SESSION_STATE Procedure

This procedure logs the session’s item values.

Syntax
APEX_DEBUG.LOG_PAGE_SESSION_STATE (   p_page_id IN NUMBER DEFAULT NULL,   p_enabled IN BOOLEAN DEFAULT FALSE,   p_level IN T_LOG_LEVEL DEFAULT C_LOG_LEVEL_APP_TRACE );

Parameters
Table 7–7 describes parameters available for the APEX_DEBUG.LOG_SESSION_STATE procedure.

Table 7–7  APEX_DEBUG.LOG_SESSION_STATE Procedure Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_page_id</td>
<td>Identifies a page within the current application and workspace context.</td>
</tr>
<tr>
<td>p_enabled</td>
<td>Messages are logged when logging is enabled, setting a value of true enables logging.</td>
</tr>
<tr>
<td>p_level</td>
<td>Identifies the level of the log message where 1 is most important, 9 is least important. Must be an integer value.</td>
</tr>
</tbody>
</table>

Example
This example shows how to enable debug message logging for 1 and 2 level messages and display a level 1 message containing all the session state for the application’s current page. Note, the p_enabled parameter need not be specified, as debugging has been explicitly enabled and the default of false for this parameter respects this enabling. Also note the p_page_id has not been specified, as this example just shows session state information for the application’s current page.

BEGIN   
APEX_DEBUG.ENABLE (p_level => 2);   
APEX_DEBUG.LOG_PAGE_SESSION_STATE (p_level => 1);   
END;
MESSAGE Procedure

This procedure logs a formatted debug message, general version.

**Syntax**

```sql
APEX_DEBUG.MESSAGE ( 
  p_message IN VARCHAR2, 
  p0 IN VARCHAR2 DEFAULT NULL, 
  p1 IN VARCHAR2 DEFAULT NULL, 
  p2 IN VARCHAR2 DEFAULT NULL, 
  p3 IN VARCHAR2 DEFAULT NULL, 
  p4 IN VARCHAR2 DEFAULT NULL, 
  p5 IN VARCHAR2 DEFAULT NULL, 
  p6 IN VARCHAR2 DEFAULT NULL, 
  p7 IN VARCHAR2 DEFAULT NULL, 
  p8 IN VARCHAR2 DEFAULT NULL, 
  p9 IN VARCHAR2 DEFAULT NULL, 
  p10 IN VARCHAR2 DEFAULT NULL, 
  p11 IN VARCHAR2 DEFAULT NULL, 
  p12 IN VARCHAR2 DEFAULT NULL, 
  p13 IN VARCHAR2 DEFAULT NULL, 
  p14 IN VARCHAR2 DEFAULT NULL, 
  p15 IN VARCHAR2 DEFAULT NULL, 
  p16 IN VARCHAR2 DEFAULT NULL, 
  p17 IN VARCHAR2 DEFAULT NULL, 
  p18 IN VARCHAR2 DEFAULT NULL, 
  p19 IN VARCHAR2 DEFAULT NULL, 
  p_max_length IN PLS_INTEGER DEFAULT 1000, 
  p_level IN T_LOG_LEVEL DEFAULT C_LOG_LEVEL_INFO, 
  p_force IN BOOLEAN DEFAULT FALSE );
```

**Parameters**

Table 7-8 describes parameters available for the APEX_DEBUG.MESSAGE procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_message</td>
<td>The debug message. Occurrences of '%s' is replaced by p0 to p19, as in utl_lms.format_message and C's sprintf. Occurrences of '%%' represent the special character '%'. Occurrences of '%&lt;n&gt;' are replaced by p&lt;n&gt;.</td>
</tr>
<tr>
<td>p0 through p19</td>
<td>Substitution strings for '%s' placeholders.</td>
</tr>
<tr>
<td>p_max_length</td>
<td>The p&lt;n&gt; values is truncated to this length.</td>
</tr>
<tr>
<td>p_level</td>
<td>The log level for the message, default is c_log_level_info. See &quot;Constants&quot; on page 7-3.</td>
</tr>
<tr>
<td>p_force</td>
<td>If true, this generates a debug message even if the page is not rendered in debug mode or p_level is greater than the configured debug messaging (using the URL or using the enable procedure).</td>
</tr>
</tbody>
</table>

**Example**

This example shows how to use the APEX_DEBUG_MESSAGE procedure to add text to the debug log.

```sql
apex_debug.message('the value of %s + %s equals %s', 3, 5, 'eight');
```
See Also:  "ERROR Procedure" on page 7-8, "WARN Procedure" on page 7-22, "TRACE Procedure" on page 7-21, "INFO Procedure" on page 7-9, "ENTER Procedure" on page 7-6
REMOVE_DEBUG_BY_AGE Procedure

Use this procedure to delete from the debug message log all data older than the specified number of days.

Syntax
APEX_DEBUG.REMOVE_DEBUG_BY_AGE (  
   p_application_id IN NUMBER,  
   p_older_than_days IN NUMBER);

Parameters
Table 7–9 describes parameters available for the APEX_DEBUG.REMOVE_DEBUG_BY_AGE procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application_id</td>
<td>The application ID of the application.</td>
</tr>
<tr>
<td>p_older_than_days</td>
<td>The number of days data can exist in the debug message log before it is deleted.</td>
</tr>
</tbody>
</table>

Example
This example demonstrates removing debug messages relating to the current application, that are older than 3 days old.

BEGIN  
   APEX_DEBUG.REMOVE_DEBUG_BY_AGE (  
       p_application_id => TO_NUMBER(:APP_ID),  
       p_older_than_days => 3 );  
END;
REMOVE_DEBUG_BY_APP Procedure

Use this procedure to delete from the debug message log all data belonging to a specified application.

Syntax
APEX_DEBUG.REMOVE_DEBUG_BY_APP {
    p_application_id IN NUMBER);

Parameters
Table 7–10 describes parameters available for the APEX_DEBUG.REMOVE_DEBUG_BY_APP procedure.

Table 7–10  APEX_DEBUG.REMOVE_DEBUG_BY_APP Procedure Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application_id</td>
<td>The application ID of the application.</td>
</tr>
</tbody>
</table>

Example
This example demonstrates removing all debug messages logged for the current application.

BEGIN
    APEX_DEBUG.REMOVE_DEBUG_BY_APP{
        p_application_id => TO_NUMBER(:APP_ID) );
END;
REMOVE_DEBUG_BY_VIEW Procedure

Use this procedure to delete all data for a specified view from the message log.

Syntax

APEX_DEBUG.REMOVE_DEBUG_BY_VIEW (  
    p_application_id IN NUMBER,  
    p_view_id IN NUMBER);  

Parameters

Table 7-11 describes parameters available for the APEX_DEBUG.REMOVE_DEBUG_BY_  
VIEW procedure.

Table 7-11  APEX_DEBUG.REMOVE_DEBUG_BY_VIEW Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application_id</td>
<td>The application ID of the application.</td>
</tr>
<tr>
<td>p_view_id</td>
<td>The view ID of the view.</td>
</tr>
</tbody>
</table>

Example

This example demonstrates the removal of debug messages within the 'View Identifier'  
of 12345, belonging to the current application.

BEGIN  
    APEX_DEBUG.REMOVE_DEBUG_BY_VIEW (  
        p_application_id => TO_NUMBER(:APP_ID),  
        p_view_id        => 12345 );  
END;  

REMOVE_SESSION_MESSAGES Procedure

This procedure deletes from the debug message log all data for a given session in your workspace defaults to your current session.

Syntax
APEX_DEBUG.REMOVE_SESSION_MESSAGES (  
   p_session    IN NUMBER  DEFAULT NULL);

Parameters
Table 7–12 describes parameters available for the APEX_DEBUG.REMOVE_SESSION_MESSAGES procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_session</td>
<td>The session ID. Defaults to your current session.</td>
</tr>
</tbody>
</table>

Example
This example demonstrates the removal of all debug messages logged within the current session. Note: As no value is passed for the p_session parameter, the procedure defaults to the current session.

BEGIN  
   APEX_DEBUG.REMOVE_SESSION_MESSAGES();
END;
TOCHAR Function

This procedure converts a BOOLEAN to a VARCHAR2.

Syntax

APEX_DEBUG.TOCHAR (  
  p_value IN BOOLEAN )  
return VARCHAR2;

Parameters

Table 7–13 describes parameters available for the APEX_DEBUG.TOCHAR function.

<table>
<thead>
<tr>
<th>Table 7–13 APEX_DEBUG.TOCHAR Procedure Parameters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>p_value</td>
<td>A BOOLEAN 0 or 1 that is converted to FALSE or TRUE respectively.</td>
</tr>
</tbody>
</table>

Example

This example shows how to use the APEX_DEBUG.TOCHAR function to convert boolean values to varchar2, so they can be passed to the other debug procedures.

declare
  l_state boolean;
begin
  ....
  apex_debug.info('Value of l_state is %s', apex_debug.tochar(l_state));
  ....
end;
TRACE Procedure

This procedure logs messages at level `c_log_level_app_trace`. This procedure always logs, even if debug mode is turned off.

Syntax
APEX_DEBUG.TRACE (  
    p_message IN VARCHAR2,  
    p0 IN VARCHAR2 DEFAULT NULL,  
    p1 IN VARCHAR2 DEFAULT NULL,  
    p2 IN VARCHAR2 DEFAULT NULL,  
    p3 IN VARCHAR2 DEFAULT NULL,  
    p4 IN VARCHAR2 DEFAULT NULL,  
    p5 IN VARCHAR2 DEFAULT NULL,  
    p6 IN VARCHAR2 DEFAULT NULL,  
    p7 IN VARCHAR2 DEFAULT NULL,  
    p8 IN VARCHAR2 DEFAULT NULL,  
    p9 IN VARCHAR2 DEFAULT NULL,  
    p_max_length IN PLS_INTEGER DEFAULT 1000 );

Parameters
Table 7–14 describes parameters available for the APEX_DEBUG.TRACE procedure.

Table 7–14 APEX_DEBUG.TRACE Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_message</td>
<td>The debug message. Occurrences of ‘%s’ are replaced by p0 to p9, as in utl_lms.format_message and C’s sprintf. Occurrences of ‘%%’ represent the special character ‘%’. Occurrences of ‘%&lt;n&gt;’ are replaced by p&lt;n&gt;.</td>
</tr>
<tr>
<td>p0 through p9</td>
<td>Substitution strings for ‘%s’ placeholders.</td>
</tr>
<tr>
<td>p_max_length</td>
<td>The p&lt;n&gt; values are truncated to this length.</td>
</tr>
</tbody>
</table>

Example
This example shows how to use APEX_DEBUG.TRACE to log low-level debug information in the debug log.

```
apex_debug.trace('Low-level information: %s+%s=%s', 1, 2, 3);
```

See Also: "MESSAGE Procedure" on page 7-14, "ERROR Procedure" on page 7-8, "WARN Procedure" on page 7-22, "ENTER Procedure" on page 7-6, "INFO Procedure" on page 7-9
WARN Procedure

This procedure logs messages at level c_log_level_warn. This procedure always logs, even if debug mode is turned off.

Syntax
APEX_DEBUG.WARN (  
  p_message IN VARCHAR2,  
  p0 IN VARCHAR2 DEFAULT NULL,  
  p1 IN VARCHAR2 DEFAULT NULL,  
  p2 IN VARCHAR2 DEFAULT NULL,  
  p3 IN VARCHAR2 DEFAULT NULL,  
  p4 IN VARCHAR2 DEFAULT NULL,  
  p5 IN VARCHAR2 DEFAULT NULL,  
  p6 IN VARCHAR2 DEFAULT NULL,  
  p7 IN VARCHAR2 DEFAULT NULL,  
  p8 IN VARCHAR2 DEFAULT NULL,  
  p9 IN VARCHAR2 DEFAULT NULL,  
  p_max_length IN PLS_INTEGER DEFAULT 1000 );

Parameters
Table 7–15 describes parameters available for the APEX_DEBUG.WARN procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_message</td>
<td>The debug message. Occurrences of '%s' are replaced by p0 to p9, as in utl_lms.format_message and C's sprintf. Occurrences of '%%%' represent the special character '%'. Occurrences of '%&lt;n&gt;' are replaced by p&lt;n&gt;.</td>
</tr>
<tr>
<td>p0 through p9</td>
<td>Substitution strings for '%s' placeholders.</td>
</tr>
<tr>
<td>p_max_length</td>
<td>The p&lt;n&gt; values are truncated to this length.</td>
</tr>
</tbody>
</table>

Example
This example shows how to use APEX_DEBUG.WARN to log highly important data in the debug log.

apex_debug.warn('Soft constraint %s violated: %s', 4711, sqlerrm);

See Also: "MESSAGE Procedure" on page 7-14, "ERROR Procedure" on page 7-8, "ENTER Procedure" on page 7-6, "TRACE Procedure" on page 7-21, "INFO Procedure" on page 7-9
The APEX_ERROR package provides the interface declarations and some utility functions for an error handling function and includes procedures and functions to raise errors in an Application Express application.

**Topics:**
- Data Types and Constants
- Example of an Error Handling Function
- ADD_ERROR Procedure Signature 1
- ADD_ERROR Procedure Signature 2
- ADD_ERROR Procedure Signature 3
- ADD_ERROR Procedure Signature 4
- ADD_ERROR Procedure Signature 5
- AUTO_SET_ASSOCIATED_ITEM Procedure
- EXTRACT_CONSTRAINT_NAME Function
- GET_ARIA_ERROR_ATTRIBUTES Function
- GET_FIRST_ORA_ERROR_TEXT Function
- INIT_ERROR_RESULT Function
Data Types and Constants

This section describes the data types and constants used by the APEX_ERROR package.

- Constants used for Result Types
- t_error
- t_error_result
Constants used for Result Types

The following constants are used for the API parameter `p_display_location` and the attribute `display_location` in the `t_error` and `t_error_result` types.

- `c_inline_with_field` constant `varchar2(40):='INLINE_WITH_FIELD';`
- `c_inline_with_field_and_notif` constant `varchar2(40):='INLINE_WITH_FIELD_AND_NOTIFICATION';`
- `c_inline_in_notification` constant `varchar2(40):='INLINE_IN_NOTIFICATION';`
- `c_on_error_page` constant `varchar2(40):='ON_ERROR_PAGE';`

The following constants are used for the API parameter `associated_type` in the `t_error` type.

- `c_ass_type_page_item` constant `varchar2(30):='PAGE_ITEM';`
- `c_ass_type_region` constant `varchar2(30):='REGION';`
- `c_ass_type_region_column` constant `varchar2(30):='REGION_COLUMN';`
The following type is passed into an error handling function and contains all of the relevant error information.

type t_error is record (    
message varchar2(32767),             /* Displayed error message */
additional_info varchar2(32767),     /* Only used for display_location ON_    
ERROR_PAGE to display additional error information */
display_location varchar2(40),       /* Use constants "used for display_    
location" below */
association_type varchar2(40),        /* Use constants "used for asocation_    
type" below */
page_item_name varchar2(255),         /* Associated page item name */
region_id number,                      /* Associated tabular form region id of    
the primary application */
column_alias varchar2(255),           /* Associated tabular form column alias */
    
row_num pls_integer,                  /* Associated tabular form row */
is_internal_error boolean,             /* Set to TRUE if it's a critical error    
raised by the Application Express engine, like an invalid SQL/PLSQL statements,    
... Internal Errors are always displayed on the Error Page */
apex_error_code varchar2(255),        /* Contains the system message code if    
it's an error raised by Application Express */
ora_sqlcode number,                    /* SQLCODE on exception stack which    
triggered the error, NULL if the error was not raised by an ORA error */
ora_sqlerrm varchar2(32767),           /* SQLERRM which triggered the error,    
NULL if the error was not raised by an ORA error */
error_backtrace varchar2(32767),       /* Output of dbms_utility.format_error_ 
backtrace or dbms_utility.format_call_stack */
component wwv_flow.t_component /* Component which has been processed    
when the error occurred */    
);
The following type is used as the result type for an error handling function.

type t_error_result is record (  
  message       varchar2(32767), /* Displayed error message */  
  additional_info varchar2(32767), /* Only used for display_location ON_ERROR_PAGE to display additional error information */  
  display_location varchar2(40),    /* Use constants 'used for display_location' below */  
  page_item_name varchar2(255),   /* Associated page item name */  
  column_alias   varchar2(255)    /* Associated tabular form column alias */  
);
Example of an Error Handling Function

create or replace function apex_error_handling_example (  
p_error in apex_error.t_error )  
return apex_error.t_error_result  
is  
l_result          apex_error.t_error_result;  
l_reference_id    number;  
l_constraint_name varchar2(255);  
begin  
l_result := apex_error.init_error_result (  
p_error => p_error );  
-- If it's an internal error raised by APEX, like an invalid statement or  
-- code which cannot be executed, the error text might contain security  
sensitive  
-- information. To avoid this security problem rewrite the error to  
-- a generic error message and log the original error message for further  
-- investigation by the help desk.  
if p_error.is_internal_error then  
-- Access Denied errors raised by application or page authorization should  
-- still show up with the original error message  
if p_error.apex_error_code <> 'APEX.AUTHORIZATION.ACCESS_DENIED'  
and p_error.apex_error_code not like 'APEX.SESSION_STATE.%' then  
-- log error for example with an autonomous transaction and return  
-- l_reference_id as reference#  
-- l_reference_id := log_error (  
--                       p_error => p_error );  
--  
-- Change the message to the generic error message which is not  
exposed  
-- any sensitive information.  
l_result.message         := 'An unexpected internal application error  
has occurred. '||  
provide '||  
'999G999G999G990'}||  
' for further investigation.';  
1_result.additional_info := null;  
end if;  
else  
-- Always show the error as inline error  
-- Note: If you have created manual tabular forms (using the package  
-- apex_item/htmldb_item in the SQL statement) you should still  
-- use 'On error page' on that pages to avoid loosing entered data  
1_result.display_location := case  
when 1_result.display_location = apex_  
error.c_on_error_page then apex_error.c_inline_in_notification  
else 1_result.display_location  
end;  
-- If it's a constraint violation like  
--  
-- -> ORA-00001: unique constraint violated  
-- -> ORA-02091: transaction rolled back (-> can hide a deferred  
constraint)
Example of an Error Handling Function

-- Example of an Error Handling Function

-- try to get a friendly error message from our constraint lookup
-- configuration.
-- If the constraint in our lookup table is not found, fallback to
-- the original ORA error message.
if p_error.ora_sqlcode in (-1, -2091, -2290, -2291, -2292) then
    l_constraint_name := apex_error.extract_constraint_name (p_error => p_error);
    begin
        select message
        into l_result.message
        from constraint_lookup
        where constraint_name = l_constraint_name;
        exception when no_data_found then null; -- not every constraint has to
        be in our lookup table
        end;
    end if;

-- If an ORA error has been raised, for example a raise_application_error(-20xxx, '...')
-- in a table trigger or in a PL/SQL package called by a process and the
-- error has not been found in the lookup table, then display
-- the actual error text and not the full error stack with all the ORA
-- error numbers.
if p_error.ora_sqlcode is not null and l_result.message = p_error.message then
    l_result.message := apex_error.get_first_ora_error_text (p_error => p_error);
end if;

-- If no associated page item/tabular form column has been set, use
-- apex_error.auto_set_associated_item to automatically guess the affected
-- error field by examine the ORA error for constraint names or column
-- names.
if l_result.page_item_name is null and l_result.column_alias is null then
    apex_error.auto_set_associated_item (p_error => p_error,
                                           p_error_result => l_result);
end if;

return l_result;
end apex_error_handling_example;
ADD_ERROR Procedure Signature 1

This procedure adds an error message to the error stack that is used to display an error on an error page or inline in a notification. It can be called in a validation or process to add one or more errors to the error stack.

---
**Note:** This procedure must be called before the Application Express application has performed the last validation or process. Otherwise, the error is ignored if it does not have a display location of `apex_error.c_on_error_page`.
---

**Syntax**

```
APEX_ERROR.ADD_ERROR (
    p_message          in varchar2,
    p_additional_info  in varchar2 default null,
    p_display_location in varchar2 );
```

**Parameters**

Table 8–1 describes the parameters available in the ADD_ERROR Procedure Signature 1.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_message</td>
<td>Displayed error message.</td>
</tr>
<tr>
<td>p_additional_info</td>
<td>Additional error information needed if the error is displayed on the error page.</td>
</tr>
<tr>
<td>p_display_location</td>
<td>Specifies where the error message is displayed. Use the constant <code>apex_error.inline_notification</code> or <code>apex_error.c_error_page</code>. See &quot;Constants used for Result Types&quot; on page 8-3.</td>
</tr>
</tbody>
</table>

**Example**

This example illustrates how to add a custom error message to the error stack. The error message is displayed inline in a notification. This example can be used in a validation or process.

```
apex_error.add_error (  
    p_message => 'This custom account is not active!',
    p_display_location => apex_error.c_inline_in_notification );
```
**ADD_ERROR Procedure Signature 2**

This procedure adds an error message to the error stack that is used to display an error for a page item inline in a notification. It can be called in a validation or process to add one or more errors to the error stack.

---

**Note:** This procedure must be called before the Application Express application has performed the last validation or process. Otherwise, the error is ignored if it does not have a display location of `apex_error.c_on_error_page`.

---

**Syntax**

```sql
APEX_ERROR.ADD_ERROR (  
  p_message          in varchar2,  
  p_additional_info  in varchar2 default null,  
  p_display_location in varchar2,  
  p_page_item_name   in varchar2);
```

**Parameters**

Table 8–2 describes the parameters available in the ADD_ERROR Procedure Signature 2.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_message</td>
<td>Displayed error message.</td>
</tr>
<tr>
<td>p_additional_info</td>
<td>Additional error information needed if the error is displayed on the error page.</td>
</tr>
<tr>
<td>p_display_location</td>
<td>Specifies where the error message is displayed. Use the constant <code>apex_error.c_inline_with_field</code> or <code>apex_error.c_inline_with_field_and_notif</code>. See &quot;Constants used for Result Types&quot; on page 8-3.</td>
</tr>
<tr>
<td>p_page_item_name</td>
<td>Name of the page item on the current page that is highlighted if <code>apex_error.c_inline_with_field</code> or <code>apex_error.c_inline_with_field_and_notif</code> are used as the display location.</td>
</tr>
</tbody>
</table>

**Example**

This example illustrates how to add a custom error message to the error stack. The P5_CUSTOMER_ID item is highlighted on the page. The error message is displayed inline in a notification. This example can be used in a validation or process.

```sql
APEX_ERROR.ADD_ERROR (  
  p_message          => 'Invalid Customer ID!',  
  p_display_location => apex_error.c_inline_with_field_and_notif,  
  p_page_item_name   => 'P5_CUSTOMER_ID');
```
ADD_ERROR Procedure Signature 3

This procedure adds an error message to the error stack that is used to display text as defined by a shared component. This error message can be displayed to all display locations. It can be called in a validation or process to add one or more errors to the error stack.

**Note:** This procedure must be called before the Application Express application has performed the last validation or process. Otherwise, the error is ignored if it does not have a display location of `apex_error.c_on_error_page`.

**Syntax**

```
APEX_ERROR.ADD_ERROR (  
p_error_code          in varchar2,  
p0                    in varchar2 default null,  
p1                    in varchar2 default null,  
p2                    in varchar2 default null,  
p3                    in varchar2 default null,  
p4                    in varchar2 default null,  
p5                    in varchar2 default null,  
p6                    in varchar2 default null,  
p7                    in varchar2 default null,  
p8                    in varchar2 default null,  
p9                    in varchar2 default null,  
p_escape_placeholders in boolean default true,  
p_additional_info     in varchar2 default null,  
p_display_location    in varchar2,  
p_page_item_name      in varchar2 );
```

**Parameters**

Table 8–3 describes the parameters available in the ADD_ERROR Procedure Signature 3.

**Table 8–3  ADD_ERROR Procedure Signature 3 Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_error_code</td>
<td>Name of shared component text message.</td>
</tr>
<tr>
<td>p_additional_info</td>
<td>Additional error information needed if the error is displayed on the error page.</td>
</tr>
<tr>
<td>p0 through p9</td>
<td>Values for %0 through %9 placeholders defined in the text message.</td>
</tr>
<tr>
<td>p_escape_placeholders</td>
<td>If set to TRUE, the values provided in p0 through p9 are escaped with <code>sys.htf.escape_sc</code> before replacing the placeholder in the text message. If set to FALSE, values are not escaped.</td>
</tr>
<tr>
<td>p_display_location</td>
<td>Specifies where the error message is displayed. Use the constants defined for p_display_location. See &quot;Constants used for Result Types&quot; on page 8-3.</td>
</tr>
<tr>
<td>p_page_item_name</td>
<td>Name of the page item on the current page that is highlighted if <code>apex_error.c_inline_with_field</code> or <code>apex_error.c_inline_with_field_andnotif</code> are used as the display location.</td>
</tr>
</tbody>
</table>
Example

This example illustrates how to add a custom error message, where the text is stored in a text message, to the error stack. The P5_CUSTOMER_ID item is highlighted on the page. The error message is displayed inline in a notification. This example can be used in a validation or process.

```python
apex_error.add_error {
    p_error_code => 'INVALID_CUSTOMER_ID',
    p0 => l_customer_id,
    p_display_location => apex_error.c_inline_with_field_and_notif,
    p_page_item_name => 'P5_CUSTOMER_ID');
```

```
ADD_ERROR Procedure Signature 4

This procedure adds an error message to the error stack that is used to display an error for a tabular form inline in a notification. It can be called in a validation or process to add one or more errors to the error stack.

Note: This procedure must be called before the Application Express application has performed the last validation or process. Otherwise, the error is ignored if it does not have a display location of apex_error.c_on_error_page.

Syntax

APEX_ERROR.ADD_ERROR (  
  p_message          in varchar2,  
  p_additional_info  in varchar2 default null,  
  p_display_location in varchar2,  
  p_region_id        in number,  
  p_column_alias     in varchar2 default null,  
  p_row_num          in number );

Parameters

Table 8–4 describes the parameters available in the ADD_ERROR Procedure Signature 4.

Table 8–4  ADD_ERROR Procedure Signature 4 Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_message</td>
<td>Displayed error message.</td>
</tr>
<tr>
<td>p_additional_info</td>
<td>Additional error information needed if the error is displayed on the error page.</td>
</tr>
<tr>
<td>p_display_location</td>
<td>Specifies where the error message is displayed. Use the constant apex_error.c_inline_with_field or apex_error.c_inline_with_field_and_notif. See “Constants used for Result Types” on page 8-3.</td>
</tr>
<tr>
<td>p_region_id</td>
<td>The ID of a tabular form region on the current page. The ID can be read from the view APEX_APPLICATION_PAGE_REGIONS.</td>
</tr>
<tr>
<td>p_column_alias</td>
<td>Name of a tabular form column alias defined for p_region_id that is highlighted if apex_error.c_inline_with_field or apex_error.c_inline_with_field_and_notif are used as a display location.</td>
</tr>
<tr>
<td>p_row_num</td>
<td>Number of the tabular form row where the error occurred.</td>
</tr>
</tbody>
</table>

Example

This example illustrates how to add a custom error message for a tabular form, where the column CUSTOMER_ID is highlighted, to the error stack. The error message is displayed inline in a notification. This example can be used in a validation or process.

```
APEX_ERROR.ADD_ERROR (  
  p_message          => 'Invalid Customer ID!',  
  p_display_location => apex_error.c_inline_with_field_and_notif,  
  p_region_id        => l_region_id,  
  p_column_alias     => 'CUSTOMER_ID',
);```
p_row_num  => l_row_num );
ADD_ERROR Procedure Signature 5

This procedure adds an error message to the error stack of a tabular form that is used to display text as defined by a shared component. This error message can be displayed to all display locations. It can be called in a validation or process to add one or more errors to the error stack.

**Note:** This procedure must be called before the Application Express application has performed the last validation or process. Otherwise, the error is ignored if it does not have a display location of `apex_error.c_on_error_page`.

**Syntax**

```sql
APEX_ERROR.ADD_ERROR (  
  p_error_code          in varchar2,  
  p0                    in varchar2 default null,  
  p1                    in varchar2 default null,  
  p2                    in varchar2 default null,  
  p3                    in varchar2 default null,  
  p4                    in varchar2 default null,  
  p5                    in varchar2 default null,  
  p6                    in varchar2 default null,  
  p7                    in varchar2 default null,  
  p8                    in varchar2 default null,  
  p9                    in varchar2 default null,  
  p_escape_placeholders in boolean  default true,  
  p_additional_info     in varchar2 default null,  
  p_display_location    in varchar2,  
  p_region_id           in number,  
  p_column_alias        in varchar2 default null,  
  p_row_num             in number );
```

**Parameters**

Table 8–5 describes the parameters available in the ADD_ERROR Procedure Signature 5.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_error_code</td>
<td>Name of shared component text message.</td>
</tr>
<tr>
<td>p0 through p9</td>
<td>Values for %0 through %9 placeholders defined in the text message.</td>
</tr>
<tr>
<td>p_escape_placeholders</td>
<td>If set to TRUE, the values provided in p0 through p9 are escaped with <code>sys.htf.escape_sc</code> before replacing the placeholder in the text message. If set to FALSE, values are not escaped.</td>
</tr>
<tr>
<td>p_additional_info</td>
<td>Additional error information needed if the error is displayed on the error page.</td>
</tr>
<tr>
<td>p_display_location</td>
<td>Specifies where the error message is displayed. Use the constants defined for p_display_location. See &quot;Constants used for Result Types&quot; on page 8-3.</td>
</tr>
</tbody>
</table>
Example

This example illustrates how to add a custom error message, where the text is stored in a text message, to the error stack. The CUSTOMER_ID column on the tabular form is highlighted. The error message is displayed inline in a notification. This example can be used in a validation or process.

```
apex_error.add_error (
    p_error_code       => 'INVALID_CUSTOMER_ID',
    p0                 => l_customer_id,
    p_display_location => apex_error.c_inline_with_field_and_notif,
    p_region_id        => l_region_id,
    p_column_alias     => 'CUSTOMER_ID',
    p_row_num          => l_row_num );
```
AUTO_SET_ASSOCIATED_ITEM Procedure

This procedure automatically sets the associated page item or tabular form column based on a constraint contained in p_error.ora_sqlerrm.

This procedure performs the following:

■ Identifies the constraint by searching for the schema.constraint pattern.
■ Only supports constraints of type P, U, R and C.
■ For constraints of type C (check constraints), the procedure parses the expression to identify those columns that are used in the constraints expression.
■ Using those columns, the procedure gets the first visible page item or tabular form column that is based on that column and set it as associated p_error_result.page_item_name or p_error_result.column_alias.
■ If a page item or tabular form column was found, p_error_result.display_location is set to apex_error.c_inline_with_field_and_notif.

Syntax

APEX_ERROR.AUTO_SET_ASSOCIATED_ITEM (  
   p_error_result in out nocopy t_error_result,  
   p_error        in            t_error );

Parameters

Table 8–10 describes the parameters available in the AUTO_SET_ASSOCIATED_ITEM procedure.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_error_result</td>
<td>The result variable of your error handling function.</td>
</tr>
<tr>
<td>p_error</td>
<td>The p_error parameter of your error handling function.</td>
</tr>
</tbody>
</table>

Example

See an example of how to use this procedure in "Example of an Error Handling Function" on page 8-6.
EXTRACT_CONSTRAINT_NAME Function

This function extracts a constraint name contained in p_error.ora_sqlerrm. The constraint must match the pattern schema.constraint.

Syntax
APEX_ERROR.EXTRACT_CONSTRAINT_NAME (  
    p_error in t_error,  
    p_include_schema in boolean default false )  
return varchar2;

Parameters
Table 8–7 describes the parameters available in the EXTRACT_CONSTRAINT_NAME function.

Table 8–7  EXTRACT_CONSTRAINT_NAME Function Parameters
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_error</td>
<td>The p_error parameter of your error handling function.</td>
</tr>
<tr>
<td>p_include_schema</td>
<td>If set to TRUE, the result is prefixed with the schema name. For example, HR.DEMO_PRODUCT_INFO_PK. If set to FALSE, only the constraint name is returned.</td>
</tr>
</tbody>
</table>

Example
See an example of how to use this procedure in "Example of an Error Handling Function" on page 8-6.
GET_ARIA_ERROR_ATTRIBUTES Function

This function is useful for item plug-in developers, to enhance screen reader usability of your item, specifically when that item is associated with an error on a page. This function is called as part of rendering of the item, where the main form element(s) are output. The returned WAI-ARIA attributes include:

- **aria-invalid="true"** - Indicates the page item’s current value is invalid. When the user is focused on the page item, the screen reader announces ‘Invalid Entry’.
- **aria-describedby="[page_item_name]_error"** - This attribute value matches up with the ID of a `<div>` tag containing the item’s associated error message, enabling a screen reader to announce the actual error, when the user is focused on the page item.

Note: Because these attributes only enhance screen reader usability, attributes are returned only if the current session is running in Screen Reader mode.

Syntax

```sql
function get_aria_error_attributes (  
    p_item_name     in varchar2  
) return varchar2;
```

Parameters

Table 8–9 describes the parameters available in the GET_ARIA_ERROR_ATTRIBUTES function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_item_name</td>
<td>The page item name. This value is available by using the name attribute of the apex_plugin.t_page_item record type, which is passed in as the 1st parameter to all item plug-in's Render Function Callback.</td>
</tr>
</tbody>
</table>

Example

This example shows how this function can be used, in rendering a SELECT element, during processing of the Render Function callback for an item plug-in. This function returns additional attributes, if the page item has errors associated with it and if the user is running in Screen Reader mode.

```sql
...  
  l_name := apex_plugin.get_input_name_for_page_item(false);  
  sys.htp.prn('<select name="'||l_name||'" id="'||p_item.name||'" ||' ||'  
        apex_error.get_aria_error_attributes(p_item.name)||'">');  
...  
```
GET_FIRST_ORA_ERROR_TEXT Function

This function returns the first ORA error message text stored in p_error.ora_sqlerrm. If p_error.ora_sqlerrm does not contain a value, NULL is returned.

Syntax
APEX_ERROR.GET_FIRST_ORA_ERROR_TEXT (  
  p_error           in t_error,  
  p_include_error_no in boolean default false )  
return varchar2;

Parameters
Table 8–9 describes the parameters available in the GET_FIRST_ORA_TEXT function.

Table 8–9  GET_FIRST_ORA_TEXT Function Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_error</td>
<td>The p_error parameter of your error handling function.</td>
</tr>
<tr>
<td>p_include_error_no</td>
<td>If set to TRUE, ORA-xxxx is included in the returned error message. If set to FALSE, only the error message text is returned.</td>
</tr>
</tbody>
</table>

Example
See an example of how to use this procedure in "Example of an Error Handling Function" on page 8-6.
INIT_ERROR_RESULT Function

This function returns the t_error_result type initialized with the values stored in p_error.

**Note:** This function must be used to ensure initialization is compatible with future changes to t_error_result.

**Syntax**

APEX_ERROR.INIT_ERROR_RESULT (p_error in t_error)
return t_error_result;

**Parameters**

Table 8–10 describes the parameters available in the INIT_ERROR_RESULT function.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_error</td>
<td>The p_error parameter of your error handling function.</td>
</tr>
</tbody>
</table>

**Example**

See an example of how to use this function in "Example of an Error Handling Function" on page 8-6.
The APEX_ESCAPE package provides functions for escaping special characters in strings to ensure that the data is suitable for further processing.

Topics:
- Constants
- HTML Function
- HTML_ATTRIBUTE Function
- HTML_TRUNC Function
- HTML_WHITELIST Function
- JS_LITERAL Function
- LDAP_DN Function
- LDAP_SEARCH_FILTER Function
- NOOP Function
- SET_HTML_ESCAPING_MODE Procedure
The APEX_ESCAPE package uses the following constants.

```
SPACE# constant binary_integer := 32;
HASH# constant binary_integer := 35;
COMMA# constant binary_integer := 44;
HYPHEN# constant binary_integer := 45;
DOT# constant binary_integer := 46;
ZERO# constant binary_integer := 48;
NINE# constant binary_integer := 57;
UP_A# constant binary_integer := 65;
UP_Z# constant binary_integer := 90;
BACKSLASH# constant binary_integer := 92;
UNDERSCORE# constant binary_integer := 95;
LOW_A# constant binary_integer := 97;
LOW_Z# constant binary_integer := 122;
```

```
c_ldap_dn_reserved_chars constant varchar2(8) := '+,;<>\';
c_ldap_search_reserved_chars constant varchar2(5) := '*()\';
c_html_whitelist_tags constant varchar2(255) :=
  '<h1>','<h2>','<h3>','<h4>','<p>','<b>','<strong>','</strong>','</h4>','</p>','</b>','</strong>','</h3>','</h2>','</h1>','<br />,','<hr />';
```
HTML Function

This function escapes characters which can change the context in an html environment. It is an extended version of the well-known sys.htf.escape_sc.

The function’s result depends on the escaping mode that is defined by using apex_escape.set_html_escaping_mode. By default, the escaping mode is "Extended", but it can be overridden by manually calling set_html_escaping_mode or by setting the application security attribute "HTML Escaping Mode" to "Basic". If the mode is "Basic", the function behaves like sys.htf.escape_sc. Otherwise, the rules below apply.

The following table, Table 9–1, depicts ascii characters that the function transforms and their escaped values:

<table>
<thead>
<tr>
<th>Raw ASCII Characters</th>
<th>Returned Escaped Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>&amp;</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>'</td>
<td>'</td>
</tr>
<tr>
<td>/</td>
<td>/</td>
</tr>
</tbody>
</table>

Syntax
APEX_ESCAPE.HTML (  
    p_string IN VARCHAR2  
)  
return VARCHAR2;

Parameters
Table 9–2 describes the parameters available in the HTML function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_string</td>
<td>The string text that is escaped</td>
</tr>
</tbody>
</table>

Example
This example tests escaping in basic ('B') and extended ('E') mode.

declare  
procedure eq(p_str1 in varchar2,p_str2 in varchar2)  
is  
begin  
    if p_str1||'.' <> p_str2||'.' then  
        raise_application_error(-20001,p_str1||'.' <> '||p_str2);  
    end if;  
end eq;  
begin  
    apex_escape.set_html_escaping_mode('B');  
    eq(apex_escape.html('hello &"<"'/>'), 'hello &amp;&quot;&lt;&gt;''/');  
    apex_escape.set_html_escaping_mode('E');
eq(apex_escape.html('hello &"<>''/'), 'hello 
&;&quot;&lt;&gt;&quot;'/); 
end;

See Also:  "HTML_TRUNC Function" on page 9-6, "HTML_WHITELIST Function" on page 9-7, "HTML_ATTRIBUTE Function" on page 9-5, "SET_HTML_ESCAPING_MODE Procedure" on page 9-12
HTML_ATTRIBUTE Function

Use this function to escape the values of html entity attributes. It hex escapes everything that is not alphanumeric or in one of the following characters ' , ', ' ', ' ', ' '.

Syntax
APEX_ESCAPE.HTML_ATTRIBUTE ( p_string IN VARCHAR2 ) return VARCHAR2;

Parameters
Table 9–3 describes the parameters available in the HTML_ATTRIBUTE function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_string</td>
<td>The text string that is escaped.</td>
</tr>
</tbody>
</table>

Example
See "HTML_TRUNC Function" on page 9-6.

See Also: "HTML_TRUNC Function" on page 9-6, "HTML Function" on page 9-3, "HTML_WHITELIST Function" on page 9-7, "SET_HTML_ESCAPING_MODE Procedure" on page 9-12
HTML_TRUNC Function

The HTML_TRUNC function escapes html and limits the returned string to \( p_{\text{length}} \) characters. This function returns the first \( p_{\text{length}} \) characters of an input clob and escapes them. You can use this function if the input clob might be too large to fit in a varchar2 variable and it is sufficient to only display the first part of it.

Syntax

\[
\text{APEX_ESCAPE.HTML_TRUNC}
\begin{array}{ll}
\text{p_string IN CLOB,} \\
\text{p_length IN NUMBER DEFAULT 4000}
\end{array}
\text{return VARCHAR2;}
\]

Parameters

Table 9–4 describes the parameters available in the HTML_TRUNC function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_string</td>
<td>The text string that is escaped.</td>
</tr>
<tr>
<td>p_length</td>
<td>The number of characters from p_string that are escaped.</td>
</tr>
</tbody>
</table>

Example

This example generates a html list of of titles and text bodies. Html entity attributes are escaped with HTML_ATTRIBUTE, whereas normal text is escaped with HTML and HTML_TRUNC.

begin
    htp.p('<ul>);
    for l_data in ( select title, cls, body 
                   from my_topics )
    loop
        sys.htp.p('<li><span class="'|| apex_escape.html_attribute(l_data.cls)||'">'|| apex_escape.html(l_data.title)||'('</li>');
        sys.htp.p(apex_escape.html_trunc(l_data.body));
        sys.htp.p('</li>');
    end loop;
    htp.p('</ul>);
end;

See Also: "HTML_ATTRIBUTE Function" on page 9-5, "HTML Function" on page 9-3, "HTML_WHITELIST Function" on page 9-7, "SET_HTML_ESCAPING_MODE Procedure" on page 9-12
HTML_WHITELIST Function

The HTML_WHITELIST function performs HTML escape on all characters in the input text except the specified whitelist tags. This function can be useful if the input text contains simple html markup but a developer wants to ensure that an attacker cannot use malicious tags for cross-site scripting.

Syntax
APEX_ESCAPE.HTML_WHITELIST ( p_html IN VARCHAR2,
                      p_whitelist_tags IN VARCHAR2 DEFAULT c_html_whitelist_tags )
return VARCHAR2;

Parameters
Table 9−5 describes the parameters available in the HTML_WHITELIST function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_html</td>
<td>The text string that is filtered.</td>
</tr>
<tr>
<td>p_whitelist_tags</td>
<td>The comma separated list of tags that stays in p_html.</td>
</tr>
</tbody>
</table>

Example
This example shows how to use HTML_WHITELIST to remove unwanted html markup from a string, while preserving whitelisted tags.

begin
  sys.htp.p(apex_escape.html_whitelist('Hello<script>alert("XSS");</script></h1>'));
end;

See Also:  "HTML_ATTRIBUTE Function" on page 9-5, "HTML Function" on page 9-3, "HTML_TRUNC Function" on page 9-6, "SET_HTML_ESCAPING_MODE Procedure" on page 9-12
JS_LITERAL Function

The JS_LITERAL function escapes and optionally enquotes a javascript string. This function replaces non-immune characters with \xHH or \uHHHH equivalents. The result can be injected into javascript code, within <script> tags or inline ("javascript:xxx"). Immune characters include a through z, A through Z, 0 through 9, commas ",", periods "." and underscores "_".

Syntax
APEX_ESCAPE.JS_LITERAL (  
   p_string IN VARCHAR2,  
   p_quote  IN VARCHAR2 DEFAULT '' )  
return VARCHAR2;

Parameters
Table 9–6 describes the parameters available in the JS_LITERAL function.

Table 9–6  JS_LITERAL Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_string</td>
<td>The text string that is escaped.</td>
</tr>
<tr>
<td>p_quote</td>
<td>If not null, this string is placed on the left and right of the result. The quotation character must be a single or a double quotation mark.</td>
</tr>
</tbody>
</table>

Example
It describes how to use JS_LITERAL to escape special characters in the l_string variable.

```sql
declare
   l_string varchar2(4000) := 'O''Brien';
begin
   sys.htp.p('<script>'||
      'alert('||apex_escape.js_literal(l_string)||');'||'</script>');
ext;
```


LDAP DN Function

The LDAP DN function escapes reserved characters in an LDAP distinguished name, according to RFC 4514. The RFC describes "+;<=\" as reserved characters (see \_reserved_chars). These are escaped by a backslash, for example, " becomes \". Non-printable characters, ascii 0 - 31, and ones with a code > 127 (see \_escape_non_ascii) are escaped as \xx, where xx is the hexadecimal character code. The space character at the beginning or end of the string and a # at the beginning is also escaped with a backslash.

Syntax

APEX_ESCAPE.LDAP_DN (  
    p_string IN VARCHAR2,  
    p_reserved_chars IN VARCHAR2 DEFAULT c_ldap_dn_reserved_chars,  
    p_escaped_non_ascii IN BOOLEAN DEFAULT TRUE  
) return VARCHAR2;

Parameters

Table 9–7 describes the parameters available in the LDAP DN function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_string</td>
<td>The text string that is escaped.</td>
</tr>
<tr>
<td>pReserved_chars</td>
<td>A list of characters that when found in p_string is escaped with a backslash.</td>
</tr>
<tr>
<td>p_escaped_non_ascii</td>
<td>If true, characters above ascii 127 in p_string are escaped with a backslash.</td>
</tr>
</tbody>
</table>

Example

This example escapes characters in l_name and places the result in l_escaped.

```
declar
    l_name varchar2(4000) := 'Joe+User';
    l_escaped varchar2(4000);
begin
    l_escaped := apex_escape.ldap_dn(l_name);
    htp.p(l_name||' becomes '|l_escaped);
end;
```

See Also: "LDAP_SEARCH_FILTER Function" on page 9-10
LDAP_SEARCH_FILTER Function

The LDAP_SEARCH_FILTER function escapes reserved characters in an LDAP search filter, according to RFC 4515. The RFC describes `*`/ as reserved characters (see `p_reserved_chars`). These, non-printable characters (ascii 0 - 31) and ones with a code > 127 (see `p_escape_non_ascii`) are escaped as \xx, where xx is the hexadecimal character code.

**Syntax**

```sql
APEX_ESCAPE.LDAP_SEARCH_FILTER (  
    p_string              IN VARCHAR2,  
    p_reserved_chars    IN VARCHAR2 DEFAULT c_ldap_search_reserved_chars,  
    p_escape_non_ascii IN BOOLEAN DEFAULT TRUE  
) return VARCHAR2;
```

**Parameters**

Table 9–8 describes the parameters available in the LDAP_SEARCH_FILTER function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_string</td>
<td>The text string that is escaped.</td>
</tr>
<tr>
<td>p_reserved_chars</td>
<td>A list of characters that when found in p_string is escaped with \xx where xx is the character’s ASCII hexadecimal code.</td>
</tr>
<tr>
<td>p_escape_non_ascii</td>
<td>If true, characters above ascii 127 in p_string are escaped with \xx where xx is the character’s ASCII hexadecimal code. This is supported by RFCs 4514, but may cause errors with older LDAP servers and Microsoft AD.</td>
</tr>
</tbody>
</table>

**Example**

This example escapes the text in l_name and places the result in l_escaped.

```sql
declare
  l_name      varchar2(4000) := 'Joe*User';
  l_escaped   varchar2(4000);
begin
  l_escaped := apex_escape.ldap_search_filter(l_name);
  htp.p(l_name||' becomes '||l_escaped);
end;
```

**See Also:**  "LDAP_DN Function" on page 9-9
NOOP Function

Return p_string unchanged. Use this function to silence automatic injection detection tests, similar to dbms_assert.noop for SQL injection.

Syntax
APEX_ESCAPE.NOOP (p_string IN VARCHAR2)
  return VARCHAR2 deterministic;

Parameters
Table 9–9 describes the parameters available in the NOOP function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_string</td>
<td>The input text string.</td>
</tr>
</tbody>
</table>

Example
This example shows how to use NOOP to show the developer's intention to explicitly not escape text.

begin
  sys.htp.p(apex_escape.noop('Cats & Dogs'));
end;
SET_HTML_ESCAPING_MODE Procedure

The SET_HTML_ESCAPING_MODE procedure configures HTML escaping mode for \texttt{wwv_flow_escape.html}.

**Syntax**

\begin{verbatim}
APEX_ESCAPE.SET_HTML_ESCAPING_MODE (p_mode IN VARCHAR2);
\end{verbatim}

**Parameters**

Table 9–10 describes the parameters available in the \texttt{SET_HTML_ESCAPING_MODE} procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_mode</td>
<td>If equal to \texttt{B}, then do basic escaping, like \texttt{sys.htf.escape_sc}. If equal to \texttt{E}, then do extended escaping.</td>
</tr>
</tbody>
</table>

**Example**

For an example, see "\texttt{HTML Function}" on page 9-3.

See Also: "\texttt{HTML_WHITELIST Function}" on page 9-7, "\texttt{HTML Function}" on page 9-3, "\texttt{HTML_TRUNC Function}" on page 9-6, "\texttt{HTML_ATTRIBUTE Function}" on page 9-5
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 scheme to workspace mappings. `APEX_INSTANCE_ADMIN` can be executed by the `SYS, SYSTEM,` and `APEX_040200` database users and any database user granted the role `APEX_ADMINISTRATOR_ROLE`.

**Topics:**
- Available Parameter Values
- ADD_SCHEMA Procedure
- ADD_WORKSPACE Procedure
- GET_PARAMETER Function
- GET_SCHEMAS Function
- REMOVE_APPLICATION Procedure
- REMOVE_SAVED_REPORTS Procedure
- REMOVE_SCHEMA Procedure
- REMOVE_SUBSCRIPTION Procedure
- REMOVE_WORKSPACE Procedure
- SET_LOG_SWITCH_INTERVAL Procedure
- SET_PARAMETER Procedure
- SET_WORKSPACE_CONSUMER_GROUP Procedure
- TRUNCATE_LOG Procedure
### Available Parameter Values

Table 10–1 lists all the available parameter values you can set within the `APEX_INSTANCE_ADMIN` package, including parameters for email, wallet, and reporting printing.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNT_LIFETIME_DAYS</td>
<td>The maximum number of days an end-user account password may be used before the account is expired.</td>
</tr>
<tr>
<td>ALLOW_DB_MONITOR</td>
<td>If set to Y, the default, database monitoring is enabled. If set to N, it is disabled.</td>
</tr>
<tr>
<td>ALLOW_PUBLIC_FILE_UPLOAD</td>
<td>If set to Y, file uploads are allowed without user authentication. If set to N, the default, they are not allowed.</td>
</tr>
<tr>
<td>ALLOW_REST</td>
<td>If set to Y, the default, developers are allowed to expose report regions as RESTful services. If set to N, the are not allowed.</td>
</tr>
<tr>
<td>APPLICATION_ACTIVITY_LOGGING</td>
<td>Controls instance wide setting of application activity log ([A]lways, [N]ever, [U]se application settings)</td>
</tr>
<tr>
<td>AUTOEXTEND_TABLESPACES</td>
<td>If set to Y, the default, provisioned tablespaces is autoextended up to a maximum size. If set to N tablesaces are not autoextended.</td>
</tr>
<tr>
<td>BIGFILE_TABLESPACES_ENABLED</td>
<td>If set to Y, the tablespaces provisioned through Oracle Application Express are created as bigfile tablespaces. If set to N, the tablespaces are created as smallfile tablespaces.</td>
</tr>
<tr>
<td>DELETE_UPLOADED_FILES_AFTER_DAYS</td>
<td>Uploaded files like application export files, websheet export files, spreadsheet data load files are automatically deleted after this number of days. Default is 14.</td>
</tr>
<tr>
<td>DISABLE_ADMIN_LOGIN</td>
<td>If set to Y, administration services are disabled. If set to N, the default, they are not disabled.</td>
</tr>
<tr>
<td>DISABLE_WORKSPACE_LOGIN</td>
<td>If set to Y, the workspace login is disabled. If set to N, the default, the login is not disabled.</td>
</tr>
<tr>
<td>DISABLE_WS_PROV</td>
<td>If set to Y, the workspace creation is disabled for requests sent out by using e-mail notification. If set to N, the default, they are not disabled.</td>
</tr>
<tr>
<td>EMAIL_IMAGES_URL</td>
<td>Specifies the full URL to the images directory of Application Express instance, including the trailing slash after the images directory. For example: http://your_server/i/</td>
</tr>
<tr>
<td>EMAIL_INSTANCE_URL</td>
<td>Specifies the URL to Application Express instance, including the trailing slash after the Database Access Descriptor. For example: http://your_server/pls/apex/</td>
</tr>
<tr>
<td>ENABLE_TRANSACTIONAL_SQL</td>
<td>If set to Y, the default, transactional SQL commands are enabled on this instance. If set to N, they are not enabled.</td>
</tr>
<tr>
<td>ENCRYPTED_TABLESPACES_ENABLED</td>
<td>If set to Y, the tablespaces provisioned through Oracle Application Express are created as encrypted tablespaces. If set to N, the tablespaces are not encrypted.</td>
</tr>
<tr>
<td>EXPIRE_FIND_USER_ACCOUNTS</td>
<td>If set to Y, expiration of Application Express accounts is enabled. If set to N, they are not enabled.</td>
</tr>
</tbody>
</table>
Table 10–1 (Cont.) Available Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INBOUND_PROXIES</strong></td>
<td>Comma-separated list of IP addresses for proxy servers through which requests come in.</td>
</tr>
<tr>
<td><strong>LOGIN_THROTTLE_DELAY</strong></td>
<td>The flag which determines the time increase in seconds after failed logins.</td>
</tr>
<tr>
<td><strong>LOGIN_THROTTLE_METHODS</strong></td>
<td>The methods to count failed logins. Colon-separated list of USERNAME_IP, USERNAME, IP.</td>
</tr>
<tr>
<td><strong>MAX_SESSION_IDLE_SEC</strong></td>
<td>The number of seconds an internal application may be idle.</td>
</tr>
<tr>
<td><strong>MAX_SESSION_LENGTH_SEC</strong></td>
<td>The number of seconds an internal application session may exist.</td>
</tr>
<tr>
<td><strong>PASSWORD_ALPHA_CHARACTERS</strong></td>
<td>The alphabetic characters used for password complexity rules. Default list of alphabetic characters include the following: abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ</td>
</tr>
<tr>
<td><strong>PASSWORD_MIN_LENGTH</strong></td>
<td>A positive integer or 0 which specifies the minimum character length for passwords for instance administrators, workspace administrators, developers, and end user Application Express accounts.</td>
</tr>
<tr>
<td><strong>PASSWORD_NEW_DIFFERS_BY</strong></td>
<td>A positive integer or 0 which specifies the number of differences required between old and new passwords. The passwords are compared character by character, and each difference that occurs in any position counts toward the required minimum difference. This setting applies to accounts for instance administrators, workspace administrators, developers, and end user Application Express accounts.</td>
</tr>
<tr>
<td><strong>PASSWORD_PUNCTUATION_CHARACTERS</strong></td>
<td>The punctuation characters used for password complexity rules. Default list of punctuation characters include the following: !&quot;#$%&amp;'()*+,-/:;&lt;=&gt;?_</td>
</tr>
<tr>
<td><strong>PLSQL_EDITING</strong></td>
<td>If set to Y, the default, the SQL Workshop Object Browser is enabled to allow users to edit and compile PL/SQL. If set to N, users are not allowed.</td>
</tr>
<tr>
<td><strong>REQUIRE_HTTPS</strong></td>
<td>If set to Y, access to the instance must be over SSL. If set to N, the default, access is not restricted to SSL.</td>
</tr>
<tr>
<td><strong>REQUIRE_VERIFICATION_CODE</strong></td>
<td>If set to Y, the Verification Code is displayed and is required for someone to request a new workspace. If set to N, the default, the Verification Code is not required.</td>
</tr>
<tr>
<td><strong>REQ_NEW_SCHEMA</strong></td>
<td>If set to Y, the option for new schema for new workspace requests is enabled. If set to N, the default, the option is disabled.</td>
</tr>
<tr>
<td><strong>RESTFULL_SERVICES_ENABLED</strong></td>
<td>If set to Y, the default, RESTful services development is enabled. If set to N, RESTful services are not enabled.</td>
</tr>
<tr>
<td><strong>SERVICE_REQUESTS_ENABLED</strong></td>
<td>If set to Y, the default, workspace service requests for schemas, storage, and termination is enabled. If set to N, these requests are disabled.</td>
</tr>
<tr>
<td><strong>SERVICE_REQUESTFLOW</strong></td>
<td>Determines default provisioning mode. Default is MANUAL.</td>
</tr>
</tbody>
</table>
### Available Parameter Values

**Table 10-1 (Cont.) Available Parameters**

<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>SMTP_PASSWORD</strong></td>
<td>Defines the password Application Express takes to authenticate itself against the SMTP server, with the parameter SMTP_USERNAME.</td>
</tr>
</tbody>
</table>
| **SMTP_TLS_MODE**       | Defines whether Application Express opens an encrypted connection to the SMTP server. Encryption is only supported on database versions 11.2.0.2 and later. On earlier database versions, the connection is not encrypted.  
If set to N, the connection is unencrypted (default).  
If set to Y, the connection is encrypted before data is sent.  
If STARTTLS is set, Application Express sends the SMTP commands EHLO <SMTP_HOST_ADDRESS> and STARTTLS before encrypting the connection. |
| **SMTP_USERNAME**       | Defines the username Application Express takes to authenticate itself against the SMTP server (default is null). Starting with database version 11.2.0.2, Application Express uses UTL_MAIL’s AUTH procedure for authentication.  
This procedure negotiates an authentication mode with the SMTP server. With earlier database versions, the authentication mode is always AUTH LOGIN.  
If SMTP_USERNAME is null, no authentication is used. |
<p>| <strong>SQL_SCRIPT_MAX_OUTPUT_SIZE</strong> | The maximum allowable size for an individual script result. Default is 200000.                                                                                                      |
| <strong>STRONG_SITE_ADMIN_PASSWORD</strong> | If set to Y, the default, the apex_admin password must conform to the default set of strong complexity rules. If set to N, the password is not required to follow the strong complexity rules. |
| <strong>SYSTEM_HELP_URL</strong>     | Location of the help and documentation accessed from the Help link within the development environment. Default is <a href="http://apex.oracle.com/doc41">http://apex.oracle.com/doc41</a>.                                                                                      |
| <strong>TRACING_ENABLED</strong>     | If set to Y (the default), an application with Debug enabled can also generate server side db trace files using &amp;p_trace=YES on the URL. If set to N, the request to create a trace file is ignored.                                           |</p>
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERNAME_VALIDATION</td>
<td>The regular expression used to validate a username if the Builder authentication scheme is not APEX. Default is as follows: <code>^[[:alnum:_.%-]+@[[:alnum:.-]+\.[[:alpha:]]][2,4]$</code></td>
</tr>
<tr>
<td>WALLET_PATH</td>
<td>The path to the wallet on the file system, for example: <code>file:/home/&lt;username&gt;/wallets</code></td>
</tr>
<tr>
<td>WALLET_PWD</td>
<td>The password associated with the wallet.</td>
</tr>
<tr>
<td>WEBSHEET_SQL_ACCESS</td>
<td>If set to Y, the default, SQL tags and SQL reports are possible in Websheet applications. If set to N, they are not possible.</td>
</tr>
<tr>
<td>WORKSPACE_EMAIL_MAXIMUM</td>
<td>Maximum number of emails allowed to be sent by using APEX_MAIL per workspace in a 24 hour period. Default is 1000.</td>
</tr>
<tr>
<td>WORKSPACE_MAX_OUTPUT_SIZE</td>
<td>The maximum space allocated for script results. Default is 2000000.</td>
</tr>
<tr>
<td>WORKSPACE_PROVISION_DEMO_OBJECTS</td>
<td>If set to Y, the default, demonstration applications and database objects are created in new workspaces. If set to N, they are not created in the current workspace.</td>
</tr>
<tr>
<td>WORKSPACE_WEBSHEET_OBJECTS</td>
<td>If set to Y, the default, Application Express Websheet database objects are created in new workspaces. If set to N, they are not created in the current workspace.</td>
</tr>
<tr>
<td>PASSWORD_HISTORY_DAYS</td>
<td>Defines the maximum number of days a developer or administrator account password may be used before the account expires. The default value is 45 days.</td>
</tr>
<tr>
<td>PRINT_BIB_LICENSED</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>PRINT_SVR_PROTOCOL</td>
<td>Valid values include:</td>
</tr>
<tr>
<td></td>
<td>■ http</td>
</tr>
<tr>
<td></td>
<td>■ https</td>
</tr>
<tr>
<td>PRINT_SVR_HOST</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>PRINT_SVR_PORT</td>
<td>Defines the port of the print server engine, for example 8888. Value must be a positive integer.</td>
</tr>
<tr>
<td>PRINT_SVR_SCRIPT</td>
<td>Defines the script that is the print server engine, for example: <code>/xmlpserver/convert</code></td>
</tr>
<tr>
<td>REQUIRE_HTTPS</td>
<td>Set to Y to allow authentication pages within the Application Express development and administration applications to be used only when the protocol is HTTPS. Select N to allow these application pages to be used when the protocol is either HTTP or HTTPS.</td>
</tr>
</tbody>
</table>
See Also:  "Configuring Email in a Runtime Environment",  
The `ADD_SCHEMA` procedure adds a schema to a workspace to schema mapping.

**Syntax**

```sql
APEX_INSTANCE_ADMIN.ADD_SCHEMA(
    p_workspace    IN VARCHAR2,
    p_schema       IN VARCHAR2);
```

**Parameters**

Table 10–2 describes the parameters available in the `ADD_SCHEMA` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p_workspace</code></td>
<td>The name of the workspace to which the schema mapping is added.</td>
</tr>
<tr>
<td><code>p_schema</code></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.

```sql
BEGIN
    APEX_INSTANCE_ADMIN.ADD_SCHEMA('MY_WORKSPACE','FRANK');
END;
```
ADD_WORKSPACE Procedure

The ADD_WORKSPACE procedure adds a workspace to an Application Express Instance.

Syntax
APEX_INSTANCE_ADMIN.ADD_WORKSPACE(
    p_workspace_id        IN NUMBER DEFAULT NULL,
    p_workspace           IN VARCHAR2,
    p_source_identifier   IN VARCHAR2 DEFAULT NULL,
    p_primary_schema      IN VARCHAR2,
    p_additional_schemas  IN VARCHAR2,
    p_rm_consumer_group   IN VARCHAR2 DEFAULT NULL);

Parameters
Table 10–3 describes the parameters available in the ADD_WORKSPACE procedure.

Table 10–3   ADD_WORKSPACE Parameters

<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 is assigned.</td>
</tr>
<tr>
<td>p_workspace</td>
<td>The name of the workspace to be added.</td>
</tr>
<tr>
<td>p_source_identifier</td>
<td>A short identifier for the workspace used when synchronizing feedback between different instances.</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>
<tr>
<td>p_rm_consumer_group</td>
<td>Resource Manager consumer group which is used when executing applications of 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 (
        p_workspace_id => 8675309,
        p_workspace   => 'MY_WORKSPACE',
        p_primary_schema => 'SCOTT',
        p_additional_schemas => '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 10–4 describes the parameters available in the GET_PARAMETER function.

<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>
</tbody>
</table>

See "Available Parameter Values" on page 10-2.

Example
The following example demonstrates how to use the GET_PARAMETER function to retrieve the SMTP_HOST_ADDRESS parameter currently defined for an Oracle Application Express instance.

```sql
DECLARE
    L_VAL VARCHAR2(4000);
BEGIN
    L_VAL :=APEX_INSTANCE_ADMIN.GET_PARAMETER('SMTP_HOST_ADDRESS');
    DBMS_OUTPUT.PUT_LINE('The SMTP Host Setting Is: '||L_VAL);
END;
```
GET_SCHEMAS Function

The GET_SCHEMAS function retrieves a comma-delimited list of schemas that are mapped to a given workspace.

Syntax

APEX_INSTANCE_ADMIN.GET_SCHEMAS(
    p_workspace     IN VARCHAR2)
RETURN VARCHAR2;

Parameters

Table 10–5 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;
The REMOVE_APPLICATION procedure removes the application specified from the Application Express instance.

**Syntax**

APEX_INSTANCE_ADMIN.REMOVE_APPLICATION (
    p_application_id IN NUMBER);

**Parameters**

Table 10–6 describes the REMOVE_APPLICATION procedure parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application_id</td>
<td>The ID of the application to remove.</td>
</tr>
</tbody>
</table>

**Example**

The following example demonstrates how to use the REMOVE_APPLICATION procedure to remove an application with an ID of 100 from an Application Express instance.

BEGIN
    APEX_INSTANCE_ADMIN.REMOVE_APPLICATION(100);
END;
The `REMOVE_SAVED_REPORT` procedure removes a specific user’s saved interactive report settings for a particular application.

**Syntax**

```apl
APEX_INSTANCE_ADMIN.REMOVE_SAVED_REPORT(
    p_application_id     IN NUMBER,
    p_report_id          IN NUMBER);
```

**Parameters**

<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.</td>
</tr>
<tr>
<td>p_report_id</td>
<td>The ID of the saved user interactive report to be removed.</td>
</tr>
</tbody>
</table>

**Example**

The following example demonstrates how to use the `REMOVE_SAVED_REPORT` procedure to remove user saved interactive report with the ID 123 for the application with an ID of 100.

```apl
BEGIN
    APEX_INSTANCE_ADMIN.REMOVE_SAVED_REPORT(100,123);
END;
```
REMOVE_SAVED_REPORTS Procedure

The REMOVE_SAVED_REPORTS procedure removes all user saved interactive report settings for a particular application or for the entire instance.

Syntax

```sql
APEX_INSTANCE_ADMIN.REMOVE_SAVED_REPORTS(
    p_application_id IN NUMBER DEFAULT NULL);
```

Parameters

Table 10–8 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 is 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.

```sql
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 10–9 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 is 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;
REMOVE_SUBSCRIPTION Procedure

The REMOVE_SUBSCRIPTION procedure removes a specific interactive report subscription.

Syntax
APEX_INSTANCE_ADMIN.REMOVE_SUBSCRIPTION(
    p_subscription_id     IN NUMBER);

Parameters
Table 10–10 describes the parameters available for the REMOVE_SUBSCRIPTION procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_subscription_id</td>
<td>The ID of the interactive report subscription to be removed.</td>
</tr>
</tbody>
</table>

Example
The following example demonstrates how to use the REMOVE_SUBSCRIPTION procedure to remove interactive report subscription with the ID 12345. Use of APEX_APPLICATION_PAGE_IR_SUB view can help identifying the subscription ID to remove.

BEGIN
    APEX_INSTANCE_ADMIN.REMOVE_SUBSCRIPTION (p_subscription_id => 12345);
END;
**REMOVE_WORKSPACE Procedure**

The `REMOVE_WORKSPACE` procedure removes a workspace from an Application Express instance.

**Syntax**

```sql
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 10–11 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.

```sql
BEGIN
    APEX_INSTANCE_ADMIN.REMOVE_WORKSPACE('MY_WORKSPACE','Y','Y');
END;
```
SET_LOG_SWITCH_INTERVAL Procedure

Set the log switch interval for each of the logs maintained by Application Express.

Syntax

APEX_INSTANCE_ADMIN.SET_LOG_SWITCH_INTERVAL(
    p_log_name              IN VARCHAR2,
    p_log_switch_after_days IN NUMBER );

Parameters

Table 10–12 describes the parameters available in the SET_LOG_SWITCH_INTERVAL procedure.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_log_name</td>
<td>Specifies the name of the log. Valid values include ACCESS, ACTIVITY, CLICKTHRU, and DEBUG.</td>
</tr>
<tr>
<td>p_log_switch_after_days</td>
<td>This interval must be a positive integer between 1 and 180.</td>
</tr>
</tbody>
</table>

Example

This example sets the log switch interval for the ACTIVITY log to 30 days.

begin
    apex_instance_admin.set_log_switch_interval( p_log_name in 'ACTIVITY', p_log_switch_after_days => 30 );
    commit;
end;
SET_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 10–13 describes the parameters available in the SET_PARAMETER procedure.

<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 “Available Parameter Values” on page 10-2.</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.example.com');
END;
SET_WORKSPACE_CONSUMER_GROUP Procedure

The SET_WORKSPACE_CONSUMER_GROUP procedure sets a Resource Manager Consumer Group to a workspace.

Syntax

```sql
set_workspace_consumer_group(
    p_workspace in varchar2,
    p_rm_consumer_group in varchar2 );
```

Parameters

Table 10–14 describes the parameters available for the SET_WORKSPACE_CONSUMER_GROUP procedure.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_workspace</td>
<td>This is the name of the workspace for which the resource consumer group is to be set.</td>
</tr>
<tr>
<td>p_rm_consumer_group</td>
<td>The parameter P_RM_CONSUMER_GROUP is the Oracle Database Resource Manager Consumer Group name. The consumer group does not have to exist at the time this procedure is invoked. But if the Resource Manager Consumer Group is set for a workspace and the consumer group does not exist, then an error will be raised when anyone attempts to login to this workspace or execute any application in the workspace. If the value of P_RM_CONSUMER_GROUP is null, then the Resource Manager consumer group associated with the specified workspace is cleared.</td>
</tr>
</tbody>
</table>

Example

The following example sets the workspace to the Resource Manager consumer group "CUSTOM_GROUP1":

```sql
begin
    apex_instance_admin.set_workspace_consumer_group(
        p_workspace => 'MY_WORKSPACE',
        p_rm_consumer_group => 'CUSTOM_GROUP1' );
    commit;
end;
/
```
The TRUNCATE_LOG procedure truncates the log entries specified by the input parameter.

**Syntax**

```
APEX_INSTANCE_ADMIN.TRUNCATE_LOG(
    p_log     IN VARCHAR2);
```

**Parameters**

Table 10–15 describes the parameters available in the TRUNCATE_LOG procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_log</td>
<td>This parameter can have one of the following values: ACTIVITY - removes all entries that record page access. USER_ACCESS - removes all entries that record user login. MAIL - removes all entries that record mail sent. DEVELOPER - removes all entries that record developer activity. CLICKS - removes all entries that record clicks tracked to external sites. LOCK_PAGE - removes all entries that record developer locking of pages. WORKSPACE_HIST - removes all entries that record daily workspace summary. PURGE - removes all entries that record automatic workspace purge activity. FILE - removes all entries that record automatic file purge activity. SCRIPT - removes all entries that record results of SQL scripts executed in SQL Workshop. SQL - removes all entries that record the history of commands executed in SQL Workshop SQL Commands</td>
</tr>
</tbody>
</table>

**Example**

The following example demonstrates how to use the TRUNCATE_LOG procedure to remove all log entries that record access to Application Express application pages.

```
BEGIN
    APEX_INSTANCE_ADMIN.TRUNCATE_LOG('ACTIVITY');
END;
```
The APEX_IR package provides utilities you can use when programming in the Oracle Application Express environment related to interactive reports. You can use the APEX_IR package to get an interactive report runtime query, add filters, reset or clear report settings, delete saved reports and manage subscriptions.

Topics:
- ADD_FILTER Procedure Signature 1
- ADD_FILTER Procedure Signature 2
- CHANGE_SUBSCRIPTION_LANG Procedure
- CLEAR_REPORT Procedure Signature 1
- CLEAR_REPORT Procedure Signature 2
- DELETE_REPORT Procedure
- DELETE_SUBSCRIPTION Procedure
- GET_LAST_VIEWED_REPORT_ID Function
- GET_REPORT Function
- RESET_REPORT Procedure Signature 1
- RESET_REPORT Procedure Signature 2
ADD_FILTER Procedure Signature 1

This procedure creates a filter on an interactive report using a report ID.

Syntax
APEX_IR.ADD_FILTER(
    p_page_id       IN NUMBER,
    p_region_id     IN NUMBER,
    p_report_column IN VARCHAR2,
    p_filter_value  IN VARCHAR2,
    p_operator_abbr IN VARCHAR2 DEFAULT NULL,
    p_report_id     IN NUMBER DEFAULT NULL);

Parameters
Table 11–1 describes the available parameters for the ADD_FILTER procedure signature 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_page_id</td>
<td>Page of the current Application Express application that contains an interactive report.</td>
</tr>
<tr>
<td>p_region_id</td>
<td>The interactive report region (ID).</td>
</tr>
<tr>
<td>p_report_column</td>
<td>Name of the report SQL column, or column alias, to be filtered.</td>
</tr>
<tr>
<td>p_filter_value</td>
<td>The filter value. This value is not used for N and NN.</td>
</tr>
<tr>
<td>p_operator_abbr</td>
<td>Filter type. Valid values are as follows:</td>
</tr>
<tr>
<td></td>
<td>EQ = Equals</td>
</tr>
<tr>
<td></td>
<td>NEQ = Not Equals</td>
</tr>
<tr>
<td></td>
<td>LT = Less than</td>
</tr>
<tr>
<td></td>
<td>LTE = Less then or equal to</td>
</tr>
<tr>
<td></td>
<td>GT = Greater Than</td>
</tr>
<tr>
<td></td>
<td>GTE = Greater than or equal to</td>
</tr>
<tr>
<td></td>
<td>LIKE = SQL Like operator</td>
</tr>
<tr>
<td></td>
<td>NLIKE = Not Like</td>
</tr>
<tr>
<td></td>
<td>N = Null</td>
</tr>
<tr>
<td></td>
<td>NN = Not Null</td>
</tr>
<tr>
<td></td>
<td>C = Contains</td>
</tr>
<tr>
<td></td>
<td>NC = Not Contains</td>
</tr>
<tr>
<td></td>
<td>IN = SQL In Operator</td>
</tr>
<tr>
<td></td>
<td>NIN = SQL Not In Operator</td>
</tr>
<tr>
<td>p_report_id</td>
<td>The saved report ID within the current application page. If p_report_id is null, it adds the filter to the last viewed report settings.</td>
</tr>
</tbody>
</table>

Example
The following example shows how to use the ADD_FILTER procedure to filter the interactive report with report ID of 880629800374638220 in page 1, region 2505704029884282 of the current application with DEPTNO equals 30.
BEGIN
    APEX_IR.ADD_FILTER(
        p_page_id => 1,
        p_region_id => 2505704029884282,
        p_report_column => 'DEPTNO',
        p_filter_value => '30',
        p_operator_abbr => 'EQ',
        p_report_id => 880629800374638220);
END;
ADD_FILTER Procedure Signature 2

This procedure creates a filter on an interactive report using a report alias.

Syntax
APEX_IR.ADD_FILTER(
    p_page_id IN NUMBER,
    p_region_id IN NUMBER,
    p_report_column IN VARCHAR2,
    p_filter_value IN VARCHAR2,
    p_operator_abbr IN VARCHAR2 DEFAULT NULL,
    p_report_alias IN VARCHAR2 DEFAULT NULL);

Parameters
Table 11–2 describes the available parameters for the ADD_FILTER Procedure Signature 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_page_id</td>
<td>Page of the current Application Express application that contains an interactive report.</td>
</tr>
<tr>
<td>p_region_id</td>
<td>The interactive report region (ID).</td>
</tr>
<tr>
<td>p_report_column</td>
<td>Name of the report SQL column, or column alias, to be filtered.</td>
</tr>
<tr>
<td>p_filter_value</td>
<td>This is the filter value. This value is not used for N and NN.</td>
</tr>
<tr>
<td>p_operator_abbr</td>
<td>Filter type. Valid values are as follows:</td>
</tr>
<tr>
<td></td>
<td>EQ = Equals</td>
</tr>
<tr>
<td></td>
<td>NEQ = Not Equals</td>
</tr>
<tr>
<td></td>
<td>LT = Less than</td>
</tr>
<tr>
<td></td>
<td>LTE = Less than or equal to</td>
</tr>
<tr>
<td></td>
<td>GT = Greater Than</td>
</tr>
<tr>
<td></td>
<td>GTE = Greater than or equal to</td>
</tr>
<tr>
<td></td>
<td>LIKE = SQL Like operator</td>
</tr>
<tr>
<td></td>
<td>NLIKE = Not Like</td>
</tr>
<tr>
<td></td>
<td>N = Null</td>
</tr>
<tr>
<td></td>
<td>NN = Not Null</td>
</tr>
<tr>
<td></td>
<td>C = Contains</td>
</tr>
<tr>
<td></td>
<td>NC = Not Contains</td>
</tr>
<tr>
<td></td>
<td>IN = SQL In Operator</td>
</tr>
<tr>
<td></td>
<td>NIN = SQL Not In Operator</td>
</tr>
<tr>
<td>p_report_alias</td>
<td>The saved report alias within the current application page. If p_report_alias is null, it adds filter to the last viewed report settings.</td>
</tr>
</tbody>
</table>

Example
The following example shows how to use the ADD_FILTER procedure to filter an interactive report with a report alias of CATEGORY_REPORT in page 1, region 250570429884282 of the current application with DEPTNO equals 30.
BEGIN
    APEX_IR.ADD_FILTER(
        p_page_id       => 1,
        p_region_id     => 2505704029884282,
        p_report_column => 'DEPTNO',
        p_filter_value  => '30',
        p_operator_abbr => 'EQ',
        p_report_alias  => 'CATEGORY_REPORT');
END;
CHANGE_SUBSCRIPTION_LANG Procedure

This procedure changes the interactive report subscription language.

Syntax

APEX_IR.CHANGE_SUBSCRIPTION_LANG(
   p_subscription_id IN NUMBER,
   p_language        IN VARCHAR2);

Parameters

Table 11–3 describes the available parameters for the

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_subscription_id</td>
<td>Subscription ID to change the language within the current workspace.</td>
</tr>
<tr>
<td>p_language</td>
<td>This is an IANA language code. Some examples include: en, de, de-at, zh-cn, and pt-br.</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use the CHANGE_SUBSCRIPTION_LANG procedure to change the subscription with the ID of 567890123 to German in the current workspace.

BEGIN
   APEX_IR.CHANGE_SUBSCRIPTION_LANG(
       p_subscription_id => 567890123,
       p_language        => 'de');
END;
CLEAR_REPORT Procedure Signature 1

This procedure clears report settings using the report ID.

Syntax
APEX_IR.CLEAR_REPORT(
   p_page_id   IN NUMBER,
   p_region_id IN NUMBER,
   p_report_id IN NUMBER DEFAULT NULL);

Parameters
Table 11–4 describes the available parameters for the CLEAR_REPORT procedure signature 1.

Table 11–4  CLEAR_REPORT Procedure Signature 1 Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_page_id</td>
<td>Page of the current Application Express application that contains an interactive report.</td>
</tr>
<tr>
<td>p_region_id</td>
<td>The interactive report region (ID).</td>
</tr>
<tr>
<td>p_report_id</td>
<td>The saved report ID within the current application page. If p_report_id is null, it clears the last viewed report settings.</td>
</tr>
</tbody>
</table>

Example
The following example shows how to use the CLEAR_REPORT procedure to clear interactive report settings with a report ID of 880629800374638220 in page 1, region 2505704029884282 of the current application.

BEGIN
   APEX_IR.CLEAR_REPORT(
      p_page_id => 1,
      p_region_id => 2505704029884282,
      p_report_id => 880629800374638220);
END;
This procedure clears report settings using report alias.

Syntax
APEX_IR.CLEAR_REPORT(
    p_page_id      IN NUMBER,
    p_region_id    IN NUMBER,
    p_report_alias IN VARCHAR2 DEFAULT NULL);

Parameters
Table 11–5 describes the available parameters for the CLEAR_REPORT procedure signature 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_page_id</td>
<td>Page of the current Application Express application that contains an interactive report.</td>
</tr>
<tr>
<td>p_region_id</td>
<td>The interactive report region (ID).</td>
</tr>
<tr>
<td>p_report_alias</td>
<td>The saved report alias within the current application page. If p_report_alias is null, it clears the last viewed report settings.</td>
</tr>
</tbody>
</table>

Example
The following example shows how to use the CLEAR_REPORT procedure to clear interactive report settings with report alias of CATEGORY_REPORT in page 1, region 250570429884282 of the current application.

BEGIN
    APEX_IR.CLEAR_REPORT(
        p_page_id => 1,
        p_region_id => 250570429884282,
        p_report_alias => 'CATEGORY_REPORT');
END;
DELETE_REPORT Procedure

This procedure deletes saved interactive reports. It deletes all saved reports except the Primary Default report.

Syntax
APEX_IR.DELETE_REPORT(
    p_report_id IN NUMBER);

Parameters
Table 11–6 describes the available parameters for the DELETE_REPORT procedure.

Table 11–6  DELETE_REPORT Procedure Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_report_id</td>
<td>Report ID to delete within the current Application Express application.</td>
</tr>
</tbody>
</table>

Example
The following example shows how to use the DELETE_REPORT procedure to delete the saved interactive report with ID of 880629800374638220 in the current application.

BEGIN
    APEX_IR.DELETE_REPORT (
        p_report_id => 880629800374638220);
END;
DELETE_SUBSCRIPTION Procedure

This procedure deletes interactive report subscriptions.

Syntax

APEX_IR.DELETE_SUBSCRIPTION(
    p_subscription_id IN NUMBER);

Parameters

Table 11–7 describes the available parameters for the DELETE_SUBSCRIPTION procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_subscription_id</td>
<td>Subscription ID to delete within the current workspace.</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use the DELETE_SUBSCRIPTION procedure to delete the subscription with ID of 567890123 in the current workspace.

BEGIN
    APEX_IR.DELETE_SUBSCRIPTION(
        p_subscription_id => 567890123);
END;
GET_LAST_VIEWED_REPORT_ID Function

This function returns the last viewed base report ID of the specified page and region.

Syntax

APEX_IR.GET_LAST_VIEWED_REPORT_ID(
    p_page_id   IN NUMBER,
    p_region_id IN NUMBER);

Parameters

Table 11–8 describes the available parameters for the GET_LAST_VIEWED_REPORT_ID function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_page_id</td>
<td>Page of the current Application Express application that contains an interactive report.</td>
</tr>
<tr>
<td>p_region_id</td>
<td>The interactive report region ID.</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use the GET_LAST_VIEWED_REPORT_ID function to retrieve the last viewed report ID in page 1, region 2505704029884282 of the current application.

DECLARE
    l_report_id number;
BEGIN
    l_report_id := APEX_IR.GET_LAST_VIEWED_REPORT_ID (p_page_id => 1,
                                                    p_region_id => 2505704029884282);
END;
GET_REPORT Function

This function returns an interactive report runtime query.

Syntax
APEX_IR.GET_REPORT(
    p_page_id   IN NUMBER,
    p_region_id IN NUMBER,
    p_report_id IN NUMBER DEFAULT NULL);

Parameters
Table 11–9 describes the available parameters for the GET_REPORT function.

Table 11–9  GET_REPORT Function Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_page_id</td>
<td>Page of the current Application Express application that contains an interactive report.</td>
</tr>
<tr>
<td>p_region_id</td>
<td>The interactive report region ID.</td>
</tr>
<tr>
<td>p_report_id</td>
<td>The saved report ID within the current application page. If p_report_id is null, it gets last viewed report query.</td>
</tr>
</tbody>
</table>

Example
The following example shows how to use the GET_REPORT function to retrieve the runtime report query with bind variable information with report ID of 880629800374638220 in page 1, region 2505704029884282 of the current application.

DECLARE
    l_report  apex_ir.t_report;
    l_query   varchar2(32767);
BEGIN
    l_report := APEX_IR.GET_REPORT (p_page_id => 1,
                                     p_region_id => 2505704029884282,
                                     p_report_id => 880629800374638220);
    l_query := l_report.sql_query;
    for i in 1..l_report.binds.count
    loop
        dbms_output.put_line(i||'. '|||l_report.binds(i).name||'='||l_report.binds(i).value);
    end loop;
END;
RESET_REPORT Procedure Signature 1

This procedure resets report settings to the developer defined default settings using the report ID.

Syntax
APEX_IR.RESET_REPORT(
    p_page_id   IN NUMBER,
    p_region_id IN NUMBER,
    p_report_id IN NUMBER DEFAULT NULL);

Parameters
Table 11–10 describes the available parameters for the RESET_REPORT procedure signature 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_page_id</td>
<td>Page of the current Application Express application that contains an interactive report.</td>
</tr>
<tr>
<td>p_region_id</td>
<td>The interactive report region ID.</td>
</tr>
<tr>
<td>p_report_id</td>
<td>The saved report ID within the current application page. If p_report_id is null, it resets the last viewed report settings.</td>
</tr>
</tbody>
</table>

Example
The following example shows how to use the RESET_REPORT procedure signature 1 to reset interactive report settings with report ID of 880629800374638220 in page 1, region 2505704029884282 of the current application.

```
BEGIN
    APEX_IR.RESET_REPORT(
        p_page_id => 1,
        p_region_id => 2505704029884282,
        p_report_id => 880629800374638220);
END;
```
RESET_REPORT Procedure Signature 2

This procedure resets report settings using the report alias.

Syntax
APEX_IR.RESET_REPORT(
    p_page_id      IN NUMBER,
    p_region_id    IN NUMBER,
    p_report_alias IN VARCHAR2 DEFAULT NULL);

Parameters
Table 11–11 describes the available parameters for the RESET_REPORT procedure signature 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_page_id</td>
<td>Page of the current Application Express application that contains an interactive report.</td>
</tr>
<tr>
<td>p_region_id</td>
<td>The interactive report region ID.</td>
</tr>
<tr>
<td>p_report_alias</td>
<td>The saved report alias within the current application page. If p_report_alias is null, it resets the last viewed report settings.</td>
</tr>
</tbody>
</table>

Example
The following example shows how to use the RESET_REPORT procedure to reset interactive report settings with a report alias of CATEGORY_REPORT in page 1, region 250570429884282 of the current application.

BEGIN
    APEX_IR.RESET_REPORT(
        p_page_id   => 1,
        p_region_id => 250570429884282,
        p_report_alias => 'CATEGORY_REPORT');
END;
You can use the APEX_ITEM package to create form elements dynamically based on a SQL query instead of creating individual items page by page.

Topics:
- CHECKBOX2 Function
- DATE_POPUP Function
- DATE_POPUP2 Function
- DISPLAY_AND_SAVE Function
- HIDDEN Function
- MD5_CHECKSUM Function
- MD5_HIDDEN Function
- POPUP_FROM_LOV Function
- POPUP_FROM_QUERY Function
- POPUPKEY_FROM_LOV Function
- POPUPKEY_FROM_QUERY Function
- RADIOGROUP Function
- SELECT_LIST Function
- SELECT_LIST_FROM_LOV Function
- SELECT_LIST_FROM_LOV_XL Function
- SELECT_LIST_FROM_QUERY Function
- SELECT_LIST_FROM_QUERY_XL Function
- TEXT Function
- TEXTAREA Function
- TEXT_FROM_LOV Function
- TEXT_FROM_LOV_QUERY Function
CHECKBOX2 Function

This function creates check boxes.

Syntax
APEX_ITEM.CHECKBOX2(
  p_idx                       IN    NUMBER,
  p_value                     IN    VARCHAR2 DEFAULT NULL,
  p_attributes                IN    VARCHAR2 DEFAULT NULL,
  p_checked_values            IN    VARCHAR2 DEFAULT NULL,
  p_checked_values_delimiter  IN    VARCHAR2 DEFAULT ':',
  p_item_id                   IN    VARCHAR2 DEFAULT NULL,
  p_item_label                IN    VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;

Parameters
Table 12–1 describes the parameters available in the CHECKBOX2 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 is 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_delimiter</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.CHECKBOX2(1,empno,'CHECKED') "Select",
       ename, job
FROM   emp
ORDER BY 1
```

The following example demonstrates how to have all check boxes for employees display without being selected.

```sql
SELECT APEX_ITEM.CHECKBOX2(1,empno) 'Select",
       ename, job
FROM   emp
ORDER BY 1
```
The following example demonstrates how to select the check boxes for employees who work in department 10.

```sql
SELECT APEX_ITEM.CHECKBOX2(1,empno,DECODE(deptno,10,'CHECKED',NULL)) "Select",
       ename, job
FROM   emp
ORDER BY 1
```

The next example demonstrates how to select the check boxes for employees who work in department 10 or department 20.

```sql
SELECT APEX_ITEM.CHECKBOX2(1,deptno,NULL,'10:20',':') "Select",
       ename, job
FROM   emp
ORDER BY 1
```

### Creating an On-Submit Process

If you are using check boxes in your application, you might need to create an On Submit process to perform a specific type of action on the selected rows. For example, you could have a Delete button that uses the following logic:

```sql
SELECT APEX_ITEM.CHECKBOX2(1,empno) "Select",
       ename, job
FROM   emp
ORDER BY 1
```

Consider the following sample on-submit process:

```sql
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:

```sql
SELECT APEX_ITEM.CHECKBOX2(1,empno,NULL,NULL,NULL,'f01_#ROWNUM#') "Select",
       ename, job
FROM   emp
ORDER BY 1
```
DATE_POPUP Function

Use this function with forms that include date fields. The DATE_POPUP function dynamically generates a date field that has a popup calendar button.

Syntax
APEX_ITEM.DATE_POPUP(
    p_idx                       IN    NUMBER,
    p_row                       IN    NUMBER,
    p_value                     IN    VARCHAR2 DEFAULT NULL,
    p_date_format               IN    DATE DEFAULT 'DD-MON-YYYY',
    p_size                      IN    NUMBER DEFAULT 20,
    p_maxlength                 IN    NUMBER DEFAULT 2000,
    p_attributes                IN    VARCHAR2 DEFAULT NULL,
    p_item_id                   IN    VARCHAR2 DEFAULT NULL,
    p_item_label                IN    VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;

Parameters
Table 12–2 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 is 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 is 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
DATE_POPUP2 Function

Use this function with forms that include date fields. The DATE_POPUP2 function dynamically generates a date field that has a jQuery based popup calendar with button.

Syntax

APEX_ITEM.DATE_POPUP2(
  p_idx                 in number,
  p_value               in date     default null,
  p_date_format         in varchar2 default null,
  p_size                in number   default 20,
  p_maxLength           in number   default 2000,
  p_attributes          in varchar2 default null,
  p_item_id             in varchar2 default null,
  p_item_label          in varchar2 default null,
  p_default_value       in varchar2 default null,
  p_max_value           in varchar2 default null,
  p_min_value           in varchar2 default null,
  p_show_on             in varchar2 default 'button',
  p_number_of_months    in varchar2 default null,
  p_navigation_list_for in varchar2 default 'NONE',
  p_year_range          in varchar2 default null,
  p_validation_date     in varchar2 default null)
RETURN VARCHAR2;

Parameters

Table 12–3 describes the parameters available in the DATE_POPUP2 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 is 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 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>
<tr>
<td>p_default_value</td>
<td>The default date which should be selected in DatePicker calendar popup</td>
</tr>
<tr>
<td>p_max_value</td>
<td>The Maximum date that can be selected from the datepicker</td>
</tr>
</tbody>
</table>
Table 12–3  (Cont.) DATE_POPUP2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_min_value</td>
<td>The Minimum date that can be selected from the datepicker.</td>
</tr>
<tr>
<td>p_show_on</td>
<td>Determines when the datepicker displays, on button click or on focus of the item or both.</td>
</tr>
<tr>
<td>p_number_of_months</td>
<td>Determines number of months displayed. Value should be in array formats follows: [row,column]</td>
</tr>
<tr>
<td>p_navigation_list_for</td>
<td>Determines if a select list is displayed for Changing Month, Year or Both. Possible values include: MONTH,YEAR,MONTH_AND_YEAR and default is null.</td>
</tr>
<tr>
<td>p_year_range</td>
<td>The range of years displayed in the year selection list.</td>
</tr>
<tr>
<td>p_validation_date</td>
<td>Used to store the Date value for the which date validation failed</td>
</tr>
</tbody>
</table>

See Also:  *Oracle Database SQL Language Reference* for information about the TO_CHAR or TO_DATE functions
DISPLAY_AND_SAVE Function

Use this function to display an item as text, but save its value to session state.

Syntax

APEX_ITEM.DISPLAY_AND_SAVE(
    p_idx         IN    NUMBER,
    p_value       IN    VARCHAR2 DEFAULT NULL,
    p_item_id     IN    VARCHAR2 DEFAULT NULL,
    p_item_label  IN    VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;

Parameters

Table 12–4 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 is 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**

```sql
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 12–5* 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 determines which <em>G_FXX</em> global is populated. See Also: &quot;APEX_APPLICATION&quot; on page 1-1</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 <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**

Typically, the primary key of a table is stored as a hidden column and used for subsequent update processing, for example:

```sql
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:

```sql
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),
```

APEX_ITEM 12-9
hiredate=TO_DATE(APEX_APPLICATION.G_F04(i), 'dd-mon-yyyy'),
sal=APEX_APPLICATION.G_F05(i),
comm=APEX_APPLICATION.G_F06(i)
WHERE empno=TO_NUMBER(APEX_APPLICATION.G_F01(i));
END LOOP;
END;

Note that the G_F01 column (which corresponds to the hidden EMPNO) is used as the key to update each row.
MD5_CHECKSUM Function

Use this function for lost update detection. Lost update detection ensures data integrity in applications where data can be accessed concurrently.

This function produces hidden form field(s) with a name attribute equal to 'fcs' and includes 50 inputs. APEX_ITEM.MD5_CHECKSUM also produces an MD5 checksum using Oracle database DBMS_CRYPTO:

```
UTL_RAW.CAST_TO_RAW(DBMS_CRYPTO.MD5())
```

An MD5 checksum provides data integrity through hashing and sequencing to ensure that data is not altered or stolen as it is transmitted over a network.

Syntax

```
APEX_ITEM.MD5_CHECKSUM(
    p_value01   IN    VARCHAR2 DEFAULT NULL,
    p_value02   IN    VARCHAR2 DEFAULT NULL,
    p_value03   IN    VARCHAR2 DEFAULT NULL,
    ...
    p_value50   IN    VARCHAR2 DEFAULT NULL,
    p_col_sep   IN    VARCHAR2 DEFAULT '|',
    p_item_id   IN    VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 12–6 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>
<tr>
<td>p_item_id</td>
<td>ID of the HTML form item</td>
</tr>
</tbody>
</table>

Example

This function generates hidden form elements with the name 'fcs'. The values can subsequently be accessed by using the APEX_APPLICATION.G_FCS array.

```
SELECT APEX_ITEM.MD5_CHECKSUM(ename, job, sal) md5_cks,
    ename, job, sal
FROM emp
```
MD5_HIDDEN Function

Use this function for lost update detection. Lost update detection ensures data integrity in applications where data can be accessed concurrently.

This function produces a hidden form field and includes 50 inputs. APEX_ITEM.MD5_HIDDEN also produces an MD5 checksum using Oracle database DBMS_CRYPTO:

\[
\text{UTL_RAW.CAST TO RAW(DBMS_CRYPTO.MD5())}
\]

An MD5 checksum provides data integrity through hashing and sequencing to ensure that data is not altered or stolen as it is transmitted over a network.

**Syntax**

```sql
APEX_ITEM.MD5_HIDDEN(
    p_idx       IN    NUMBER,
    p_value01   IN    VARCHAR2 DEFAULT NULL,
    p_value02   IN    VARCHAR2 DEFAULT NULL,
    p_value03   IN    VARCHAR2 DEFAULT NULL,
    ...
    p_value50  IN    VARCHAR2 DEFAULT NULL,
    p_col_sep   IN    VARCHAR2 DEFAULT '|',
    p_item_id   IN    VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

**Parameters**

Table 12–7 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>
<tr>
<td>p_item_id</td>
<td>ID of the HTML form item</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 is generated.

```sql
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 12–8 describes the available 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 website). If this form comes before the #FORM_OPEN# substitution string, then its index is zero and the form opened automatically by Oracle Application Express must be referenced as form 1. This functionality supports the JavaScript used in the popup LOV that passes a value back to a form element.</td>
</tr>
<tr>
<td>p_escape_html</td>
<td></td>
</tr>
<tr>
<td>p_max_elements</td>
<td></td>
</tr>
<tr>
<td>p_attributes</td>
<td></td>
</tr>
<tr>
<td>p_ok_to_query</td>
<td></td>
</tr>
<tr>
<td>p_item_id</td>
<td></td>
</tr>
<tr>
<td>p_item_label</td>
<td></td>
</tr>
</tbody>
</table>
Table 12–8 (Cont.) POPUP_FROM_LOV Parameters

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

Example

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

```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

```sql
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 12–9 describes the available 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></td>
<td>SELECT dname, deptno FROM dept</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 website). If this form comes before the #FORM_OPEN# substitution string, then its index is zero and the form opened automatically by Oracle Application Express must be referenced as form 1. This functionality supports the JavaScript used in the popup LOV that passes a value back to a form element.</td>
</tr>
<tr>
<td>p_escape_html</td>
<td>NULL</td>
</tr>
<tr>
<td>p_max_elements</td>
<td>NULL</td>
</tr>
<tr>
<td>p_attributes</td>
<td>NULL</td>
</tr>
<tr>
<td>p_ok_to_query</td>
<td>‘YES’</td>
</tr>
<tr>
<td>p_item_id</td>
<td>NULL</td>
</tr>
<tr>
<td>p_item_label</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Table 12–9  (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 &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 are escaped.</td>
</tr>
<tr>
<td></td>
<td>This parameter is useful if you know your query returns 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</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 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>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 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
```
POPUPKEY_FROM_LOV Function

This function generates a popup key select list from a shared list of values (LOV). Similar to other available functions in the APEX_ITEM package, the POPUPKEY_FROM_LOV function is designed to generate forms with F01 to F50 form array elements.

Syntax
APEX_ITEM.POPUPKEY_FROM_LOV(
    p_idx              IN    NUMBER,  
    p_value            IN    VARCHAR2 DEFAULT NULL,  
    p_lov_name         IN    VARCHAR2,  
    p_width            IN    VARCHAR2 DEFAULT NULL,  
    p_max_length       IN    VARCHAR2 DEFAULT NULL,  
    p_form_index       IN    VARCHAR2 DEFAULT '0',  
    p_escape_html      IN    VARCHAR2 DEFAULT NULL,  
    p_max_elements     IN    VARCHAR2 DEFAULT NULL,  
    p_attributes       IN    VARCHAR2 DEFAULT NULL,  
    p_ok_to_query      IN    VARCHAR2 DEFAULT 'YES',  
    p_item_id          IN    VARCHAR2 DEFAULT NULL,  
    p_item_label       IN    VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;

Although the text field associated with the popup displays in the first column in the LOV query, the actual value is specified in the second column in the query.

Parameters
Table 12–10 describes the available parameters in the POPUPKEY_FROM_LOV function.

Table 12–10 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: 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 website). If this form comes before the #FORM_OPEN# substitution string, then its index is zero and the form opened automatically by Oracle Application Express must be referenced as form 1. This functionality supports the JavaScript used in the popup LOV that passes a value back to a form element.</td>
</tr>
</tbody>
</table>
Table 12–10  (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 returns 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 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>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 12–11 describes the available parameters in the POPUPKEY_FROM_QUERY function.

Table 12–11 POPUPKEY_FROM_QUERY 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, 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_QUERY (1,deptno,'SELECT dname, deptno FROM dept') dt, APEX_ITEM.HIDDEN(3,empno) eno</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 website). If this form comes before the #FORM_OPEN# substitution string, then its index is zero and the form opened automatically by Oracle Application Express must be referenced as form 1. This functionality supports the JavaScript used in the popup LOV that passes a value back to a form element.</td>
</tr>
</tbody>
</table>
The following example demonstrates how to generate a popup select list from a SQL query.

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

### Table 12–11 (Cont.) POPUPKEY_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 &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 returns 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</td>
</tr>
<tr>
<td></td>
<td>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>
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 12–12 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 is 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
```
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 12–13 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.

```sql
SELECT APEX_ITEM.SELECT_LIST(1, 'Y', 'Yes;Y, No;N') yn
```
The following example demonstrates the use of APEX_ITEM.SELECT_LIST to generate a static select list where:

- A form array element F03 is generated (p_idx parameter).
- The initial value for each element is equal to the value for deptno for the row from emp (p_value parameter).
- The select list contains 4 options (p_list_values parameter).
- The text within the select list displays in red (p_attributes parameter).
- A null option is displayed (p_show_null) and this option displays -Select- as the text (p_null_text parameter).
- An HTML ID attribute is generated for each row, where #ROWNUM# is substituted for the current row rownum (p_item_id parameter). (So an ID of 'f03_4' is generated for row 4.)
- A HTML label element is generated for each row (p_item_label parameter).
- The current value for deptno is displayed, even if it is not contained with the list of values passed in the p_list_values parameter (p_show_extra parameter).

```
SELECT empno 'Employee #',
ename 'Name',
APEX_ITEM.SELECT_LIST(
    p_idx => 3,
    p_value => deptno,
    p_list_values => 'ACCOUNTING;10,RESEARCH;20,SALES;30,OPERATIONS;40',
    p_attributes => 'style="color:red;"',
    p_show_null => 'YES',
    p_null_value => NULL,
    p_null_text => '-Select-',
    p_item_id => 'f03_#ROWNUM#',
    p_item_label => 'Label for f03_#ROWNUM#',
    p_show_extra => 'YES') "Department"
FROM emp;
```
SELECT_LIST_FROM_LOV Function

This function dynamically generates select lists from a shared list of values (LOV). Similar to other functions available in the APEX_ITEM package, these select list functions are designed to generate forms with F01 to F50 form array elements.

Syntax

APEX_ITEM.SELECT_LIST_FROM_LOV(
    p_idx            IN   NUMBER,
    p_value          IN   VARCHAR2 DEFAULT NULL,
    p_lov            IN   VARCHAR2,
    p_attributes     IN   VARCHAR2 DEFAULT NULL,
    p_show_null      IN   VARCHAR2 DEFAULT 'YES',
    p_null_value     IN   VARCHAR2 DEFAULT '%NULL%',
    p_null_text      IN   VARCHAR2 DEFAULT '%',
    p_item_id        IN   VARCHAR2 DEFAULT NULL,
    p_item_label     IN   VARCHAR2 DEFAULT NULL,
    p_show_extra     IN   VARCHAR2 DEFAULT 'YES')
RETURN VARCHAR2;

Parameters

Table 12–14 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. Use this function in SQL queries where you need to handle a column value longer than 4000 characters.

Syntax

APEX_ITEM.SELECT_LIST_FROM_LOV_XL(
    p_idx           IN   NUMBER,
    p_value         IN   VARCHAR2 DEFAULT NULL,
    p_lov           IN   VARCHAR2,
    p_attributes    IN   VARCHAR2 DEFAULT NULL,
    p_show_null     IN   VARCHAR2 DEFAULT 'YES',
    p_null_value    IN   VARCHAR2 DEFAULT '%NULL%',
    p_null_text     IN   VARCHAR2 DEFAULT '%',
    p_item_id       IN   VARCHAR2 DEFAULT NULL,
    p_item_label    IN   VARCHAR2 DEFAULT NULL,
    p_show_extra    IN   VARCHAR2 DEFAULT 'YES')
RETURN CLOB;

Parameters

Table 12-15 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 12–16 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 is used for both display and return purposes. |
| p_attributes     | Extra HTML parameters you want to add.                                      |
| p_show_null      | Extra select option to enable the NULL selection. Range of values is YES and NO. |
| p_null_value     | Value to be returned when a user selects the NULL option. Only relevant when p_show_null equals YES. |
| p_null_text      | Value to be displayed when a user selects the NULL option. Only relevant when p_show_null equals YES. |
| p_item_id        | HTML attribute ID for the <select> tag.                                     |
| p_item_label     | Invisible label created for the item.                                       |
| p_show_extra     | Show the current value even if the value of p_value is not located in the select list. |
Example
The following example demonstrates a select list based on a SQL query.

```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 12–17 describes the parameters available in the SELECT_LIST_FROM_QUERY_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_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>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>
### Example

The following example demonstrates a select list based on a SQL query.

```sql
SELECT APEX_ITEM.SELECT_LIST_FROM_QUERY_XL(3, 'job', 'SELECT DISTINCT job FROM emp') → job
FROM emp
```

**Table 12-17 (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>
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 12–18 describes the parameters available in the TEXT function.

Table 12–18   TEXT Parameters

<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 determines which G_FXX global is populated.</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 &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 sample query demonstrates how to generate one update field for each row. Note that the ename, sal, and comm columns use the APEX_ITEM.TEXT function to generate an HTML text field for each row. Also, notice that each item in the query is passed a unique p_idx parameter to ensure that each column is stored in its own array.

SELECT
    empno,
    APEX_ITEM.HIDDEN(1,empno) ||
    APEX_ITEM.TEXT(2,ename) ename,
    APEX_ITEM.TEXT(3,job) job,
    mgr,
    APEX_ITEM.DATE_POPUP(4,rownum,hiredate,'dd-mon-yyyy') hiredate,
    APEX_ITEM.TEXT(5,sal) sal,
    APEX_ITEM.TEXT(6,comm) comm,
    deptno
FROM emp
ORDER BY 1
TEXTAREA Function

This function creates text areas.

Syntax

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

Parameters

Table 12–19 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 determines which G_FXX global is populated. See Also: &quot;APEX_APPLICATION&quot; on page 1-1</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 &lt;textarea&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 create a text area based on a SQL query.

```
SELECT APEX_ITEM.TEXTAREA(3, ename, 5, 80) a
FROM emp
```
TEXT_FROM_LOV Function

Use this function to display an item as text, deriving the display value of the named LOV.

Syntax

```sql
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 12–20 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 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 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).

```sql
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 12–21 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>
</tbody>
</table>
| p_query         | SQL query that is expected to select two columns, a display column and a return column. For example:

```sql
SELECT dname, deptno FROM dept
```

Note if only one column is specified in the select clause of this query, the value for this column is used for both display and return purposes.

| p_null_text     | Value to be displayed when the value of the field item is NULL or a corresponding entry is not located for the value p_value in the list of values query. |

Example

The following example demonstrates how to derive the display value from a query.

```sql
SELECT APEX_ITEM.TEXT_FROM_LOV_QUERY(empno,'SELECT ename, empno FROM emp') c from emp
```
The `APEX_JAVASCRIPT` package provides utility functions for adding dynamic JavaScript code to HTTP output. This package is usually used for plug-in development.

**Topics:**

- `ADD_3RD_PARTY_LIBRARY_FILE Procedure`
- `ADD_ATTRIBUTE Function Signature 1`
- `ADD_ATTRIBUTE Function Signature 2`
- `ADD_ATTRIBUTE Function Signature 3`
- `ADD_ATTRIBUTE Function Signature 4`
- `ADD_INLINE_CODE Procedure`
- `ADD_LIBRARY Procedure`
- `ADD_ONLOAD_CODE Procedure`
- `ADD_VALUE Function Signature 1`
- `ADD_VALUE Function Signature 2`
- `ADD_VALUE Function Signature 3`
- `ADD_VALUE Function Signature 4`
- `Escape Function`
ADD_3RD_PARTY_LIBRARY_FILE Procedure

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

Syntax

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

Parameters

Table 13–1 describes the parameters available for the ADD_3RD_PARTY_LIBRARY_FILE procedure.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_library</td>
<td>Use one of the c_library_* constants</td>
</tr>
<tr>
<td>p_file_name</td>
<td>Specifies the file name without version, .min and .js</td>
</tr>
<tr>
<td>p_directory</td>
<td>Directory where the file p_file_name is located (optional)</td>
</tr>
<tr>
<td>p_version</td>
<td>If no value is provided then the same version Application Express ships is used (optional)</td>
</tr>
</tbody>
</table>

Example

This example loads the JavaScript file of the Draggable feature of jQuery UI.

```sql
apex_javascript.add_3rd_party_library_file (  
p_library => apex_javascript.c_library_jquery_ui,  
p_file_name => 'jquery.ui.draggable' )
```
ADD_ATTRIBUTE Function Signature 1

This function returns the attribute and the attribute’s escaped text surrounded by double quotation marks.

**Note:** This function does not escape HTML tags. It only prevents HTML tags from breaking the JavaScript object attribute assignment. To prevent XSS (cross site scripting) attacks, you must also call SYS.HTF.ESCAPE_SC to prevent embedded JavaScript code from being executed when you inject the string into the HTML page.

**Syntax**

APEX_JAVASCRIPT.ADD_ATTRIBUTE (  
  p_name       IN VARCHAR2,  
  p_value      IN VARCHAR2,  
  p_omit_null  IN BOOLEAN:=TRUE,  
  p_add_comma  IN BOOLEAN:=TRUE)  
RETURN VARCHAR2;

**Parameters**

Table 13–2 describes the parameters available in the ADD_ATTRIBUTE function signature 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_name</td>
<td>Name of the JavaScript object attribute.</td>
</tr>
<tr>
<td>p_value</td>
<td>Text to be assigned to the JavaScript object attribute.</td>
</tr>
<tr>
<td>p_omit_null</td>
<td>If set to TRUE and p_value is empty, returns NULL.</td>
</tr>
<tr>
<td>p_add_comma</td>
<td>If set to TRUE, a trailing comma is added when a value is returned.</td>
</tr>
</tbody>
</table>

**Example**

Adds a call to the addEmployee JavaScript function and passes in a JavaScript object with different attribute values. The output of this call looks like:

```javascript
addEmployee(
  {'FirstName':'John',
   'LastName':'Doe',
   'Salary':2531.29,
   'Birthday':new Date(1970,1,15,0,0,0),
   'isSalesman':true
 });
```

As the last attribute you should use the parameter combination FALSE (p_omit_null), FALSE (p_add_comma) so that the last attribute is always generated. This avoids that you have to check for the other parameters if a trailing comma should be added or not.

```javascript
apex_javascript.add_onload_code {
  'addEmployee('||
   apex_javascript.add_attribute('FirstName', sys.htf.escape_sc(l_first_
apex_javascript.add_attribute('LastName', sys.htf.escape_sc(l_last_name))||
apex_javascript.add_attribute('Salary', l_salary)||
apex_javascript.add_attribute('Birthday', l_birthday)||
apex_javascript.add_attribute('isSalesman', l_is_salesman, false, false)||
');


This function returns the attribute and the attribute’s number.

Syntax

APEX_JAVASCRIPT.ADD_ATTRIBUTE (   p_name       IN VARCHAR2,   p_value      IN NUMBER,   p_omit_null  IN BOOLEAN:=TRUE,   p_add_comma  IN BOOLEAN:=TRUE) RETURN VARCHAR2;

Parameters

Table 13–3 describes the parameters available in the ADD_ATTRIBUTE function signature 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_name</td>
<td>Name of the JavaScript object attribute.</td>
</tr>
<tr>
<td>p_value</td>
<td>Number which should be assigned to the JavaScript object attribute.</td>
</tr>
<tr>
<td>p_omit_null</td>
<td>If set to TRUE and p_value is empty, returns NULL.</td>
</tr>
<tr>
<td>p_add_comma</td>
<td>If set to TRUE, a trailing comma is added when a value is returned.</td>
</tr>
</tbody>
</table>

Example

See example for ADD_ATTRIBUTE Function Signature 1 on page 13-3.
ADD_ATTRIBUTE Function Signature 3

This function returns the attribute and a JavaScript boolean of true, false, or null.

Syntax

APEX_JAVASCRIPT.ADD_ATTRIBUTE (  
  p_name       IN VARCHAR2,  
  p_value      IN BOOLEAN,  
  p_omit_null  IN BOOLEAN:=TRUE,  
  p_add_comma  IN BOOLEAN:=TRUE)  
RETURN VARCHAR2;  

Parameters

Table 13–4 describes the parameters available in the ADD_ATTRIBUTE function signature 3.

Table 13–4  ADD_ATTRIBUTE Signature 3 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_name</td>
<td>Name of the JavaScript object attribute.</td>
</tr>
<tr>
<td>p_value</td>
<td>Boolean assigned to the JavaScript object attribute.</td>
</tr>
<tr>
<td>p_omit_null</td>
<td>If p_omit_null is TRUE and p_value is NULL the function returns NULL.</td>
</tr>
<tr>
<td>p_add_comma</td>
<td>If set to TRUE a trailing comma is added when a value is returned.</td>
</tr>
</tbody>
</table>

Example

See example for ADD_ATTRIBUTE Function Signature 1 on page 13-3
ADD_ATTRIBUTE Function Signature 4

This function returns the attribute and the attribute’s date. If p_value is null the value null is returned.

Syntax
APEX_JAVASCRIPT.ADD_ATTRIBUTE (p_name IN VARCHAR2,
                                  p_value IN DATE,
                                  p_omit_null IN BOOLEAN:=TRUE,
                                  p_add_comma IN BOOLEAN:=TRUE)
RETURN VARCHAR2;

Parameters
Table 13–5 describes the parameters available in the ADD_ATTRIBUTE function signature 4.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_name</td>
<td>Name of the JavaScript object attribute.</td>
</tr>
<tr>
<td>p_value</td>
<td>Date assigned to the JavaScript object attribute.</td>
</tr>
<tr>
<td>p_omit_null</td>
<td>If p_omit_null is TRUE and p_value is NULL the function returns NULL.</td>
</tr>
<tr>
<td>p_add_comma</td>
<td>If set to TRUE a trailing comma is added when a value is returned.</td>
</tr>
</tbody>
</table>

Example
See example for ADD_ATTRIBUTE Function Signature 1 on page 13-3
ADD_INLINE_CODE Procedure

This procedure adds a code snippet that is included inline into the HTML output. For example, you can use this procedure to add new functions or global variable declarations. If you want to execute code you should use ADD_ONLOAD_CODE Procedure.

Syntax
APEX_JAVASCRIPT.ADD_INLINE_CODE {
    p_code       IN VARCHAR2,
    p_key        IN VARCHAR2 DEFAULT NULL);

Parameters
Table 13–6 describes the parameters available in the ADD_INLINE_CODE procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_code</td>
<td>JavaScript code snippet. For example: $s('P1_TEST',123);</td>
</tr>
<tr>
<td>p_key</td>
<td>Identifier for the code snippet. If specified and a code snippet with the same name has already been added, the new code snippet is ignored. If p_key is NULL the snippet is always added.</td>
</tr>
</tbody>
</table>

Example
The following example includes the JavaScript function initMySuperWidget in the HTML output. If the plug-in is used multiple times on the page and the add_inline_code is called multiple times, it is added once to the HTML output because all calls have the same value for p_key.

    apex_javascript.add_inline_code {
        p_code => 'function initMySuperWidget(){'||chr(10)||
            '  // do something'||chr(10)||
            '};',
        p_key => 'my_super_widget_function' );
ADD_LIBRARY Procedure

This procedure adds the script tag to load a JavaScript library. If a library has been added, it is not added a second time.

Syntax
APEX_JAVASCRIPT.ADD_LIBRARY (  
    p_name                   IN VARCHAR2,  
    p_directory              IN VARCHAR2,  
    p_version                IN VARCHAR2 DEFAULT NULL,  
    p_check_to_add_minified  IN BOOLEAN DEFAULT FALSE,  
    p_skip_extension         IN BOOLEAN DEFAULT FALSE,  
    p_ie_condition           IN VARCHAR2 DEFAULT NULL,  
    p_key                    IN VARCHAR2 DEFAULT NULL);  

Parameters
Table 13–7 describes the parameters available in the ADD_LIBRARY procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_name</td>
<td>Name of the JavaScript file. Must not use .js when specifying.</td>
</tr>
<tr>
<td>p_directory</td>
<td>Directory where JavaScript library is loaded. Must have a trailing slash.</td>
</tr>
<tr>
<td>p_version</td>
<td>Version identifier.</td>
</tr>
<tr>
<td>p_check_to_add_minified</td>
<td>If TRUE, the procedure tests if it is appropriate to add .min extension and add it if appropriate. This is added if an application is not running in DEBUG mode, and omitted when in DEBUG mode.</td>
</tr>
<tr>
<td>p_skip_extension</td>
<td>If TRUE the extension .js is NOT added.</td>
</tr>
<tr>
<td>p_ie_condition</td>
<td>Condition which is used as Internet Explorer condition.</td>
</tr>
<tr>
<td>p_key</td>
<td>Name used to indicate if the library has already been loaded. If not specified, defaults to p_directory</td>
</tr>
</tbody>
</table>

Example
The following example includes the JavaScript library file named my_library.1.2.min.js (if the application is not running in DEBUG mode), or my_library.1.2.js (if the application is running in DEBUG mode), from the directory specified by p_plugin.file_prefix. The addition of the .min extension if the application is not running in DEBUG mode is carried out because p_check_to_add_minified is set to true. Since p_skip_extension is not specified, this defaults to .js. Also, since p_key is not specified, the key defaults to p_plugin.file_prefix||mylibrary.1.2.

```apl
APEX_JAVASCRIPT.add_library (  
    p_name                  => 'mylibrary.1.2',  
    p_directory             => p_plugin.file_prefix,  
    p_check_to_add_minified => true  
);```
ADD_ONLOAD_CODE Procedure

This procedure adds a javascript code snippet to the HTML output which is executed by the onload event. If an entry with the same key exists it is ignored. If p_key is NULL the snippet is always added.

Syntax
APEX_JAVASCRIPT.ADD_ONLOAD_CODE (  
    p_code           IN VARCHAR2,  
    p_key            IN VARCHAR2 DEFAULT NULL);  

Parameters
Table 13–8 describes the parameters available in the ADD_ONLOAD_CODE procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_code</td>
<td>Javascript code snippet to be executed during the onload event.</td>
</tr>
<tr>
<td>p_key</td>
<td>Any name to identify the specified code snippet. If specified, the code snippet is added if there has been no other call with the same p_key. If p_key is NULL the code snippet is always added.</td>
</tr>
</tbody>
</table>

Example
Adds the JavaScript call initMySuperWidget() to the onload buffer. If the plug-in is used multiple times on the page and the add_onload_code is called multiple times, it is added once to the HTML output because all calls have the same value for p_key

    apex_javascript.add_onload_code (  
        p_code => 'initMySuperWidget();'  
        p_key  => 'my_super_widget'  
    );
ADD_VALUE Function Signature 1

This function returns the escaped text surrounded by double quotation marks. For example, this string could be returned "That\'s a test".

**Note:** This function does not escape HTML tags. It only prevents HTML tags from breaking the JavaScript object attribute assignment. To prevent XSS (cross site scripting) attacks, you must also call SYS.HTF.ESCAPE_SC to prevent embedded JavaScript code from being executed when you inject the string into the HTML page.

**Syntax**

APEX_JAVASCRIPT.ADD_VALUE (  
  p_value          IN VARCHAR2,  
  p_add_comma      IN BOOLEAN :=TRUE)  
RETURN VARCHAR2;

**Parameters**

Table 13–9 describes the parameters available in the ADD_VALUE signature 1 function.

**Table 13–9  ADD_VALUE Signature 1 Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_value</td>
<td>Text to be escaped and wrapped by double quotation marks.</td>
</tr>
<tr>
<td>p_add_comma</td>
<td>If p_add_comma is TRUE a trailing comma is added.</td>
</tr>
</tbody>
</table>

**Example**

This example adds some JavaScript code to the onload buffer. The value of p_item.attribute_01 is first escaped with htf.escape_sc to prevent XSS attacks and then assigned to the JavaScript variable lTest by calling apex_javascript.add_value. Add_value takes care of properly escaping the value and wrapping it with double quotation marks. Because commas are not wanted, p_add_comma is set to FALSE.

```javascript
apex_javascript.add_onload_code (  
  'var lTest = '||apex_javascript.add_value(sys.htf.escape_sc(p_item.attribute_01), FALSE)||';'||chr(10)||  
  'showMessage(lTest);' );
```
ADD_VALUE Function Signature 2

This function returns `p_value` as JavaScript number, if `p_value` is NULL the value null is returned.

**Syntax**

```sql
APEX_JAVASCRIPT.ADD_VALUE (
    p_value          IN NUMBER,
    p_add_comma      IN BOOLEAN :=TRUE)
RETURN VARCHAR2;
```

**Parameters**

Table 13–9 describes the parameters available in the ADD_VALUE signature 2 function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p_value</code></td>
<td>Number which should be returned as JavaScript number.</td>
</tr>
<tr>
<td><code>p_add_comma</code></td>
<td>If <code>p_add_comma</code> is TRUE a trailing comma is added. Default is TRUE.</td>
</tr>
</tbody>
</table>

**Example**

See example for ADD_VALUE Function Signature 1 on page 13-11.
ADD_VALUE Function Signature 3

This function returns `p_value` as JavaScript boolean. If `p_value` is NULL the value `null` is returned.

**Syntax**

```apex
APEX_JAVASCRIPT.ADD_VALUE (p_value         IN BOOLEAN,
                            p_add_comma      IN BOOLEAN :=TRUE)
RETURN VARCHAR2;
```

**Parameters**

Table 13–11 describes the parameters available in the ADD_VALUE signature 3 function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p_value</code></td>
<td>Boolean which should be returned as JavaScript boolean.</td>
</tr>
<tr>
<td><code>p_add_comma</code></td>
<td>If <code>p_add_comma</code> is TRUE a trailing comma is added. Default is TRUE.</td>
</tr>
</tbody>
</table>

**Example**

See example for ADD_VALUE Function Signature 1 on page 13-11.
**ADD_VALUE Function Signature 4**

This function returns \( p\_value \) as JavaScript date object, if \( p\_value \) is NULL the value null is returned.

**Syntax**

```
APEX_JAVASCRIPT.ADD_VALUE ( 
  p_value          IN NUMBER, 
  p_add_comma      IN BOOLEAN :=TRUE) 
RETURN VARCHAR2;
```

**Parameters**

Table 13–12 describes the parameters available in the ADD_VALUE signature 4 function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_value )</td>
<td>Date which should be returned as JavaScript date object.</td>
</tr>
<tr>
<td>( p_add_comma )</td>
<td>If ( p_add_comma ) is TRUE a trailing comma is added. Default is TRUE.</td>
</tr>
</tbody>
</table>

**Example**

See example for ADD_VALUE Function Signature 1 on page 13-11.
Escape Function

This function escapes text to be used in JavaScript. This function makes the following replacements:

<table>
<thead>
<tr>
<th>Replacement</th>
<th>After replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>\u003c</td>
</tr>
<tr>
<td>&gt;</td>
<td>\u003e</td>
</tr>
<tr>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>&quot;</td>
<td>\u0022</td>
</tr>
<tr>
<td>'</td>
<td>\u0027</td>
</tr>
<tr>
<td>tab</td>
<td>\t</td>
</tr>
<tr>
<td>chr(10)</td>
<td>\n</td>
</tr>
</tbody>
</table>

**Note:** This function prevents HTML tags from breaking the JavaScript object attribute assignment and also escapes the HTML tags `<` and `>`. It does not escape other HTML tags, therefore to be sure to prevent XSS (cross site scripting) attacks, you must also call `SYS.HTF.ESCAPE_SC` to prevent embedded JavaScript code from being executed when you inject the string into the HTML page.

**Syntax**

```sql
APEX_JAVASCRIPT.ESCAPE (p_text  IN VARCHAR2)
RETURN VARCHAR2;
```

**Parameters**

Table 13–14 describes the parameters available in the ESCAPE function.

**Table 13–14 ESCAPE Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_text</td>
<td>Text to be escaped.</td>
</tr>
</tbody>
</table>

**Example**

Adds some JavaScript code to the onload buffer. The value of p_item.attribute_01 is first escaped with htf.escape_sc to prevent XSS attacks and then escaped with apex_javascript.escape to prevent that special characters like a quotation mark break the JavaScript code.

```javascript
apex_javascript.add_onload_code {
    'var lTest = "'||apex_javascript.escape(sys.htf.escape_sc(p_item.attribute_01))||"'; ||chr(10)||
    'showMessage(lTest);' );
```
You can use APEX_LANG API to translate messages.

Topics:
- CREATE_LANGUANGE_MAPPING Procedure
- DELETE_LANGUANGE_MAPPING Procedure
- LANG Function
- MESSAGE Function
- PUBLISH_APPLICATION Procedure
- SEED_TRANSLATIONS Procedure
- UPDATE_LANGUANGE_MAPPING Procedure
- UPDATE_MESSAGE Procedure
- UPDATE_TRANSLATED_STRING Procedure
CREATE_LANGUAGE_MAPPING Procedure

Use this procedure to create the language mapping for the translation of an application. Translated applications are published as new applications, but are not directly editable in the Application Builder.

Note: This procedure is available in Application Express release 4.2.3 and later.

Syntax

```
APEX_LANG.CREATE_LANGUAGE_MAPPING ( 
    p_application_id IN NUMBER,
    p_language IN VARCHAR2,
    p_translation_application_id IN NUMBER )
```

Parameters

<table>
<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 you want to create the language mapping. This is the ID of the primary language application.</td>
</tr>
<tr>
<td>p_language</td>
<td>The IANA language code for the mapping. Examples include en-us, fr-ca, ja, he.</td>
</tr>
<tr>
<td>p_translation_application_id</td>
<td>Unique integer value for the ID of the underlying translated application. This number cannot end in 0.</td>
</tr>
</tbody>
</table>

Example

The following example demonstrates the creation of the language mapping for an existing Application Express application.

```
begin
    --
    -- If running from SQL*Plus, we need to set the environment
    -- for the Application Express workspace associated with this schema. The
    -- call to apex_util.set_security_group_id is not necessary if
    -- you’re running within the context of the Application Builder
    -- or an Application Express application.
    --
    for c1 in (select workspace_id
                  from apex_workspaces) loop
        apex_util.set_security_group_id( c1.workspace_id );
        exit;
    end loop;

    -- Now, actually create the language mapping
    apex_lang.create_language_mapping( 
        p_application_id => 63969,
        p_language => 'ja',
        p_translation_application_id => 778899 );
    commit;

    --
    -- Print what we just created to confirm
```
--
for c1 in (select *
    from apex_application_trans_map
    where primary_application_id = 63969) loop
    dbms_output.put_line('translated_application_id: ' || c1.translated_application_id);
    dbms_output.put_line('translated_app_language: ' || c1.translated_app_language);
    end loop;
end;
/
DELETE_LANGUAGE_MAPPING Procedure

Use this procedure to delete the language mapping for the translation of an application. This procedure deletes all translated strings in the translation repository for the specified language and mapping. Translated applications are published as new applications, but are not directly editable in the Application Builder.

Note: This procedure is available in Application Express release 4.2.3 and later.

Syntax

APEX_LANG.DELETE_LANGUAGE_MAPPING (  
  p_application_id IN NUMBER,  
  p_language IN VARCHAR2 )

Parameters

Table 14–2 DELETE_LANGUAGE_MAPPING Parameters

<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 you want to delete the language mapping. This is the ID of the primary language application.</td>
</tr>
<tr>
<td>p_language</td>
<td>The IANA language code for the existing mapping. Examples include en-us, fr-ca, ja, he.</td>
</tr>
</tbody>
</table>

Example

The following example demonstrates the deletion of the language mapping for an existing Application Express application and existing translation mapping.

begin  
  --  
  -- If running from SQL*Plus, we need to set the environment  
  -- for the Application Express workspace associated with this schema. The  
  -- call to apex_util.set_security_group_id is not necessary if  
  -- you’re running within the context of the Application Builder  
  -- or an Application Express application.  
  --  
  for c1 in (select workspace_id  
              from apex_workspaces) loop  
      apex_util.set_security_group_id( c1.workspace_id );  
      exit;  
  end loop;  
  -- Now, delete the language mapping  
  apex_lang.delete_language_mapping(  
    p_application_id => 63969,  
    p_language => 'ja' );  
  commit;  
  --  
  -- Print what we just updated to confirm  
  --  
  for c1 in (select count(*) thecount  
              from apex_application_trans_map  
              where primary_application_id = 63969) loop
DELETE_LANGUAGE_MAPPING Procedure

        dbms_output.put_line('Translation mappings found: ' || cl.thecount);
        end loop;
    end;
    /
LANG Function

Use this function to return a translated text string for translations defined in dynamic translations.

Syntax

```
APEX_LANG.LANG (  
    p_primary_text_string IN VARCHAR2 DEFAULT NULL,  
    p0 IN VARCHAR2 DEFAULT NULL,  
    p1 IN VARCHAR2 DEFAULT NULL,  
    p2 IN VARCHAR2 DEFAULT NULL,  
    ...  
    p9 IN VARCHAR2 DEFAULT NULL,  
    p_primary_language IN VARCHAR2 DEFAULT NULL)  
RETURN VARCHAR2;
```

Parameters

<table>
<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 is the value of the Translate From Text in the dynamic translation.</td>
</tr>
<tr>
<td>p0 through 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. 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 14–4 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 Text Messages under Shared Components of your application in Oracle Application Express.</td>
</tr>
<tr>
<td>p0 through 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  
  --  
  HTP.P(APEX_LANG.MESSAGE('GREETING_MSG', V('APP_USER')));  
END;

How the p_lang attribute is defined depends on how the Application Express engine derives the Application Primary Language. For example, if you are running the application in German and the previous call is made to the APEX_LANG.MESSAGE API, the Application Express engine first looks for a message called GREETING_MSG with a LANG_CODE of de. If it does not find anything, then it is reverted to the Application Primary Language attribute. If it still does not find anything, the Application Express engine looks for a message by this name with a language code of en.
See also: Specifying the Primary Language for an Application in the Oracle Application Express Application Builder User’s Guide.
PUBLISH_APPLICATION Procedure

Use this procedure to publish the translated version of an application. This procedure creates an underlying, hidden replica of the primary application and merges the strings from the translation repository in this new application. Perform a seed and publish process each time you want to update the translated version of your application and synchronize it with the primary application.

This application is not visible in the Application Builder. It can be published and exported, but not directly edited.

---

**Note:** This procedure is available in Application Express release 4.2.3 and later.

---

Syntax

```sql
APEX_LANG.PUBLISH_APPLICATION (  
  p_application_id IN NUMBER,  
  p_language IN VARCHAR2 )
```

Parameters

Table 14–5  PUBLISH_APPLICATION Parameters

<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 you want to publish and create the translated version. This is the ID of the primary language application.</td>
</tr>
<tr>
<td>p_language</td>
<td>The IANA language code for the existing translation mapping. Examples include en-us, fr-ca, ja, he.</td>
</tr>
</tbody>
</table>

Example

The following example demonstrates the publish process for an Application Express application and language.

```sql
begin  
  --  
  -- If running from SQL*Plus, we need to set the environment  
  -- for the Application Express workspace associated with this schema. The  
  -- call to apex_util.set_security_group_id is not necessary if  
  -- you're running within the context of the Application Builder  
  -- or an Application Express application.  
  --  
  for c1 in (select workspace_id  
    from apex_workspaces) loop  
    apex_util.set_security_group_id( c1.workspace_id );  
    exit;  
  end loop;  
  -- Now, publish the translated version of the application  
  apex_lang.publish_application(  
    p_application_id => 63969,  
    p_language => 'ja' );  
  commit;  
end;  
/
```

Note:
This procedure is available in Application Express release 4.2.3 and later.
SEED_TRANSLATIONS Procedure

Use this procedure to seed the translation repository for the specified application and language. This procedure populates the translation repository with all of the new, updated and removed translatable strings from your application. Perform a seed and publish process each time you want to update the translated version of your application and synchronize it with the primary application.

Syntax

```
APEX_LANG.SEED_TRANSLATIONS ( 
    p_application_id IN NUMBER, 
    p_language IN VARCHAR2 )
```

Parameters

<table>
<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 you want to update the translation repository. This is the ID of the primary language application.</td>
</tr>
<tr>
<td>p_language</td>
<td>The IANA language code for the existing translation mapping. Examples include en-us, fr-ca, ja, he.</td>
</tr>
</tbody>
</table>

Example

The following example demonstrates the seeding process of the translation repository for an Application Express application and language.

```
begin
    --
    -- If running from SQL*Plus, we need to set the environment
    -- for the Application Express workspace associated with this schema. The
    -- call to apex_util.set_security_group_id is not necessary if
    -- you're running within the context of the Application Builder
    -- or an Application Express application.
    --
    for c1 in (select workspace_id
                from apex_workspaces) loop
        apex_util.set_security_group_id( c1.workspace_id );
        exit;
    end loop;
    -- Now, seed the translation repository
    apex_lang.seed_translations( 
        p_application_id => 63969, 
        p_language => 'ja' );
    commit;
    -- Print out the total number of potentially translatable strings
    --
    for c1 in (select count(*) thecount
                from apex_application_trans_repos 
                where application_id = 63969) loop
        dbms_output.put_line( 'Potentially translatable strings found: ' || c1.thecount );
    end loop;
end;
/
UPDATE_LANGUAGE_MAPPING Procedure

Use this procedure to update the language mapping for the translation of an application. Translated applications are published as new applications, but are not directly editable in the Application Builder.

**Note:** This procedure is available in Application Express release 4.2.3 and later.

**Syntax**

APEX_LANG.UPDATE_LANGUAGE_MAPPING (  
    p_application_id IN NUMBER,  
    p_language IN VARCHAR2,  
    p_new_trans_application_id IN NUMBER )

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application_id</td>
<td>The ID of the application for which you want to update the language mapping. This is the ID of the primary language application.</td>
</tr>
<tr>
<td>p_language</td>
<td>The IANA language code for the existing mapping. Examples include en-us, fr-ca, ja, he. The language of the mapping cannot be updated with this procedure, only the new translation application ID.</td>
</tr>
<tr>
<td>p_new_trans_application_id</td>
<td>New unique integer value for the ID of the underlying translated application. This number cannot end in 0.</td>
</tr>
</tbody>
</table>

**Example**

The following example demonstrates the update of the language mapping for an existing Application Express application and existing translation mapping.

```
begin
    --
    -- If running from SQL*Plus, we need to set the environment
    -- for the Application Express workspace associated with this schema. The
    -- call to apex_util.set_security_group_id is not necessary if
    -- you're running within the context of the Application Builder
    -- or an Application Express application.
    --
    for c1 in (select workspace_id
                from apex_workspaces) loop
        apex_util.set_security_group_id( c1.workspace_id );
        exit;
    end loop;
    -- Now, update the language mapping
    apex_lang.update_language_mapping(  
        p_application_id => 63969,  
        p_language => 'ja',  
        p_new_trans_application_id => 881188 );
    commit;
    --
```

Note: This procedure is available in Application Express release 4.2.3 and later.
-- Print what we just updated to confirm
--
for c1 in (select *
    from apex_application_trans_map
    where primary_application_id = 63969) loop
    dbms_output.put_line( 'translated_application_id: ' || c1.translated_application_id );
    dbms_output.put_line( 'translated_app_language: ' || c1.translated_app_language );
    end loop;
end;
/
UPDATE_MESSAGE Procedure

Use this procedure to update a translatable text message for the specified application.

**Note:** This procedure is available in Application Express release 4.2.3 and later.

**Syntax**

```
APEX_LANG.UPDATE_MESSAGE ( 
    p_id IN NUMBER, 
    p_message_text IN VARCHAR2 )
```

**Parameters**

*Table 14–8 UPDATE_MESSAGE Parameters*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_id</td>
<td>The ID of the text message.</td>
</tr>
<tr>
<td>p_message_text</td>
<td>The new text for the translatable text message.</td>
</tr>
</tbody>
</table>

**Example**

The following example demonstrates an update of an existing translatable text message.

```
begin
    --
    -- If running from SQL*Plus, we need to set the environment
    -- for the Application Express workspace associated with this schema. The
    -- call to apex_util.set_security_group_id is not necessary if
    -- you're running within the context of the Application Builder
    -- or an Application Express application.
    --
    for c1 in (select workspace_id
        from apex_workspaces) loop
        apex_util.set_security_group_id( c1.workspace_id );
        exit;
    end loop;
    -- Locate the ID of the specific message and update it with the new text
    for c1 in (select translation_entry_id
        from apex_application_translations
        where application_id = 63969
        and translatable_message = 'TOTAL_COST'
        and language_code = 'ja') loop
        apex_lang.update_message(
            p_id => c1.translation_entry_id,
            p_message_text => 'The total cost is: %0');
        commit;
        exit;
    end loop;
end;
/```
Update this procedure to update a translated string in the seeded translation repository.

**Note:** This procedure is available in Application Express release 4.2.3 and later.

**Syntax**

```sql
APEX_LANG.UPDATE_TRANSLATED_STRING ( p_id IN NUMBER,
                                           p_language IN VARCHAR2,
                                           p_string IN VARCHAR2 )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_id</td>
<td>The ID of the string in the translation repository.</td>
</tr>
<tr>
<td>p_language</td>
<td>The IANA language code for the existing translation mapping. Examples include en-us, fr-ca, ja, he. The language of the mapping cannot be updated with this procedure, only the new translation application ID.</td>
</tr>
<tr>
<td>p_string</td>
<td>The new value for the string in the translation repository.</td>
</tr>
</tbody>
</table>

**Example**

The following example demonstrates an update of an existing string in the translation repository.

```sql
begin
  -- If running from SQL*Plus, we need to set the environment
  -- for the Application Express workspace associated with this schema. The
  -- call to apex_util.set_security_group_id is not necessary if
  -- you're running within the context of the Application Builder
  -- or an Application Express application.
  --
  for c1 in (select workspace_id
              from apex_workspaces) loop
    apex_util.set_security_group_id( c1.workspace_id );
    exit;
  end loop;
  -- Locate all strings in the repository for the specified application
  -- which are 'Search' and change to 'Find'
  for cl in (select id
              from apex_application_trans_repos
              where application_id = 63969
              and dbms_lob.compare(from_string, to_nclob('Search')) = 0
              and language_code = 'ja') loop
    apex_lang.update_translated_string( p_id => cl.id,
                                        p_language => 'ja',
                                        p_string => 'Find');
    commit;
end;
```

**Note:** This procedure is available in Application Express release 4.2.3 and later.
exit;
end loop;
end;
/

UPDATE_TRANSLATED_STRING Procedure
You can use APEX_LDAP to perform various operations related to Lightweight Directory Access Protocol (LDAP) authentication.

**Topics:**
- 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,
    p_use_ssl      IN VARCHAR2 DEFAULT 'N')
RETURN BOOLEAN;

Parameters

Table 15–1 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>
<tr>
<td>p_use_ssl</td>
<td>Set to 'Y' to use SSL in bind to LDAP server. Set to 'A' to use SSL with one way authentication (requires LDAP server certificate configured in an Oracle wallet). Set to 'N' to not use SSL (default).</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_use_ssl           IN VARCHAR2 DEFAULT 'N',
    p_attributes        OUT wwv_flow_global.vc_arr2,
    p_attribute_values  OUT wwv_flow_global.vc_arr2);

Parameters

Table 15–2 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_use_ssl</td>
<td>Set to 'Y' to use SSL in bind to LDAP server. Set to 'A' to use SSL with one way authentication (requires LDAP server certificate configured in an Oracle wallet). Set to 'N' to not use SSL (default).</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_sever.my_company.com',
        p_port => '389',
        p_attributes => L_ATTRIBUTES,
        p_attribute_values => L_ATTRIBUTE_VALUES);

Table 15–2 GET_ALL_USER_ATTRIBUTES Parameters
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_use_ssl           IN VARCHAR2 DEFAULT 'N',
  p_attributes        IN  wwv_flow_global.vc_arr2,
  p_attribute_values  OUT wwv_flow_global.vc_arr2);

Parameters

Table 15–3 describes the parameters available in the GET_USER_ATTRIBUTES procedure.

Table 15–3  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_use_ssl</td>
<td>Set to 'Y' to use SSL in bind to LDAP server. Set to 'A' to use SSL with one way authentication (requires LDAP server certificate configured in an Oracle wallet). Set to 'N' to not use SSL (default).</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_sever.my_company.com',
    p_port => '389',
    p_attribute_values => L_ATTRIBUTES);
p_attributes => L_ATTRIBUTES,
p_attribute_values => L_ATTRIBUTE_VALUES);
IS_MEMBER Function

The IS_MEMBER 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**

\begin{verbatim}
APEX_LDAP.IS_MEMBER(
    p_username     IN VARCHAR2,
    p_pass         IN VARCHAR2 DEFAULT NULL,
    p_auth_base    IN VARCHAR2,
    p_host         IN VARCHAR2,
    p_port         IN VARCHAR2 DEFAULT 389,
    p_use_ssl      IN VARCHAR2 DEFAULT 'N',
    p_group        IN VARCHAR2,
    p_group_base   IN VARCHAR2)
RETURN BOOLEAN;
\end{verbatim}

**Parameters**

Table 15–4 describes the parameters available in the IS_MEMBER function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{p_username}</td>
<td>Login name of the user.</td>
</tr>
<tr>
<td>\texttt{p_pass}</td>
<td>Password for \texttt{p_username}.</td>
</tr>
<tr>
<td>\texttt{p_auth_base}</td>
<td>LDAP search base, for example, \texttt{dc=users,dc=my,dc=org}.</td>
</tr>
<tr>
<td>\texttt{p_host}</td>
<td>LDAP server host name.</td>
</tr>
<tr>
<td>\texttt{p_port}</td>
<td>LDAP server port number.</td>
</tr>
<tr>
<td>\texttt{p_use_ssl}</td>
<td>Set to \texttt{Y} to use SSL in bind to LDAP server. Set to \texttt{A} to use SSL with one way authentication (requires LDAP server certificate configured in an Oracle wallet). Set to \texttt{N} to not use SSL.</td>
</tr>
<tr>
<td>\texttt{p_group}</td>
<td>Name of the group to be search for membership.</td>
</tr>
<tr>
<td>\texttt{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.

\begin{verbatim}
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.');
    END IF;
END;
\end{verbatim}
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,
  p_use_ssl      IN VARCHAR2 DEFAULT 'N')
RETURN wwv_flow_global.vc_arr2;

Parameters
Table 15–5 describes the parameters available in the MEMBER_OF function.

Table 15–5 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>
<tr>
<td>p_use_ssl</td>
<td>Set to 'Y' to use SSL in bind to LDAP server. Set to 'A' to use SSL with one way authentication (requires LDAP server certificate configured in an Oracle wallet). Set to 'N' to not use SSL (default).</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_sever.my_company.com',
    p_port             => '389');
  FOR i IN L_MEMBERSHIP.FIRST..L_MEMBERSHIP.LAST LOOP
    htp.p('Member of: '||L_MEMBERSHIP(i));
  END LOOP;
END;
MEMBER_OF2 Function

The MEMBER_OF2 function returns a VARCHAR2 colon delimited list of groups the user name designated by p_username (with password if required) belongs to, using the provided auth base, host, and port.

Syntax

APEX_LDAP.MEMBER_OF2(
    p_username     IN VARCHAR2 DEFAULT NULL,
    p_pass         IN VARCHAR2 DEFAULT NULL,
    p_auth_base    IN VARCHAR2,
    p_host         IN VARCHAR2,
    p_port         IN VARCHAR2 DEFAULT 389,
    p_use_ssl      IN VARCHAR2 DEFAULT 'N')
RETURN VARCHAR2;

Parameters

Table 15–6 describes the parameters available in the MEMBER_OF2 function.

Table 15–6 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>
<tr>
<td>p_use_ssl</td>
<td>Set to 'Y' to use SSL in bind to LDAP server. Set to 'A' to use SSL with one way authentication (requires LDAP server certificate configured in an Oracle wallet). Set to 'N' to not use SSL (default).</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;
You can use the APEX_MAIL package to send an email from an Oracle Application Express application. This package is built on top of the Oracle supplied UTL_SMTP package. Because of this dependence, the UTL_SMTP package must be installed and functioning to use APEX_MAIL.

See Also: Oracle Database PL/SQL Packages and Types Reference for more information about the UTL_SMTP package

APEX_MAIL contains three procedures. Use APEX_MAIL.SEND to send an outbound email message from your application. Use APEX_MAIL.PUSH_QUEUE to deliver mail messages stored in APEX_MAIL_QUEUE. Use APEX_MAIL.ADD_ATTACHMENT to send an outbound email message from your application as an attachment.

Topics:
- About Configuring Oracle Application Express to Send Email
- ADD_ATTACHMENT Procedure
- GET_IMAGES_URL Function
- GET_INSTANCE_URL Function
- PUSH_QUEUE Procedure
- SEND Procedure

Note: The most efficient approach to sending email is to create a background job (using the DBMS_JOB or DBMS_SCHEDULER package) to periodically send all mail messages stored in the active mail queue. To call the APEX_MAIL package from outside the context of an Application Express application, you must call apex_util.set_security_group_id as in the following example:

```sql
for c1 in (  
    select workspace_id  
    from apex_applications  
    where application_id = p_app_id )  
loop  
    apex_util.set_security_group_id(p_security_group_id =>  
    c1.workspace_id)  
end loop;
```
See Also:  "Sending Email from an Application" in Oracle Application Express Application Builder User’s Guide
Before you can send email from an Application Builder application, you must:

1. Log in to Oracle Application Express Administration Services and configure the email settings on the Instance Settings page. See in Oracle Application Express Administration Guide.

2. If you are running Oracle Application Express with Oracle Database 11g release 1 (11.1), you must enable outbound mail. In Oracle Database 11g release 1 (11.1), the ability to interact with network services is disabled by default. See “Enabling Network Services in Oracle Database 11g” in Oracle Application Express Application Builder User’s Guide.

**Tip:** You can configure Oracle Application Express to automatically email users their login credentials when a new workspace request has been approved. To learn more, see “Specifying a Provisioning Mode” in Oracle Application Express Administration Guide.
ADD_ATTACHMENT Procedure

This procedure sends an outbound email message from an application as an attachment. To add multiple attachments to a single email, APEX_MAIL.ADD_ATTACHMENT can be called repeatedly for a single email message.

Syntax

APEX_MAIL.ADD_ATTACHMENT(  
  p_mail_id                   IN    NUMBER,  
  p_attachment                IN    BLOB,  
  p_filename                  IN    VARCHAR2,  
  p_mime_type                 IN    VARCHAR2);  

Parameters

Table 16–1 describes the parameters available in the ADD_ATTACHMENT procedure.

Table 16–1  ADD_ATTACHMENT Parameters

<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;  
/

16-4  Oracle Application Express API Reference
GET_IMAGES_URL Function

Use this function to get the image prefixed URL, if the email includes Application Express instance images.

Syntax
APEX_MAIL.GET_IMAGES_URL return VARCHAR2;

Parameters
None.

Example
The following example sends an Order Confirmation email which includes the Oracle Logo image.

```
declare
    l_body      clob;
    l_body_html clob;
begin
    l_body := 'To view the content of this message, please use an HTML enabled mail client.' || utl_tcp.crlf;
    l_body_html := '<html><body>' || utl_tcp.crlf ||
                   '<p>Please confirm your order on the <a href="' ||
                apex_mail.get_instance_url || 'f?p=100:10">Order Confirmation</a> page.</p>' || utl_tcp.crlf ||
                   '<p>Sincerely,<br />' || utl_tcp.crlf ||
                'The Application Express Dev Team<br />' || utl_tcp.crlf ||
                   '<img src="' || apex_mail.get_images_url || 'oracle.gif" alt="Oracle Logo">" || utl_tcp.crlf ||
                   '</body></html>';  
apex_mail.send (
    p_to        => 'some_user@somewhere.com',   -- change to your email address
    p_from      => 'some_sender@somewhere.com', -- change to a real senders email address
    p_body      => l_body,
    p_body_html => l_body_html,
    p_subj      => 'Order Confirmation' );
end;
```
GET_INSTANCE_URL Function

GET_INSTANCE_URL Function
If an email includes a link to an Application Express instance, use this function to get
the instance URL.
Note: This function requires that the instance setting Application
Express Instance URL for emails is set.

Syntax
APEX_MAIL.GET_IMAGES_URL return VARCHAR2;

Parameters
None.
Example
The following example sends an Order Confirmation email which includes an absolute
URL to page 10 of application 100.
declare
l_body
clob;
l_body_html clob;
begin
l_body := 'To view the content of this message, please use an HTML enabled
mail client.' || utl_tcp.crlf;
l_body_html := '<html><body>' || utl_tcp.crlf ||
'<p>Please confirm your order on the <a href="' ||
apex_mail.get_instance_url || 'f?p=100:10">Order
Confirmation</a> page.</p>' || utl_tcp.crlf ||
'</body></html>';
apex_mail.send (
p_to
=> 'some_user@somewhere.com',
-- change to your email
address
p_from
=> 'some_sender@somewhere.com', -- change to a real senders
email address
p_body
=> l_body,
p_body_html => l_body_html,
p_subj
=> 'Order Confirmation' );
end;

16-6 Oracle Application Express API Reference


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 16–2 describes the parameters available in the PUSH_QUEUE procedure.

Table 16–2  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 results in erroneous email messages, including partial messages or messages with extraneous exclamation points.

- **Plain text and HTML email content.** Passing a value to p_body, but not p_body_html results in a plain text message. Passing a value to p_body and p_body_html yields a multi-part message that includes both plain text and HTML content. The settings and capabilities of the recipient’s email client determine what displays. Although most modern email clients can read an HTML formatted email, remember that some users disable this functionality to address security issues.

- **Avoid images.** When referencing images in p_body_html using the `<img />` tag, remember that the images must be accessible to the recipient’s email client in order for them to see the image.

For example, suppose you reference an image on your network called hello.gif as follows:

```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 is not displayed. For this reason, avoid using images. If you must include images, be sure to include the ALT attribute to provide a textual description in the event the image is not accessible.

**Syntax**

```sql
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 16–3 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.

```sql
-- Example One: Plain Text only message
DECLARE
    l_body      CLOB;
BEGIN
    l_body := 'Thank you for your interest in the APEX_MAIL package.'||utl_tcp.crlf||utl_tcp.crlf;
    l_body := l_body ||'  Sincerely,'||utl_tcp.crlf;
    l_body := l_body ||'  The Application Express Dev Team'||utl_tcp.crlf;
    apex_mail.send(
        p_to       => 'some_user@somewhere.com',   -- change to your email address
        p_from     => 'some_sender@somewhere.com', -- change to a real senders email address
        p_body     => l_body,
        p_subj     => 'APEX_MAIL Package - Plain Text message');
END;
/
```

---

### Table 16–3 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 is 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 is sent (required). This email address must be a valid address. Otherwise, the message is not 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 <code>&lt;html&gt;</code> and <code>&lt;body&gt;</code> tags. A single line cannot exceed 1000 characters without a carriage return or line feed (CRLF)</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 send these messages, but the automatic replies 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.

```sql
-- 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;}
        </style>
        <span.sig{font-style:italic;
            font-weight:bold;
            color:#811919;}
      </head>
      <body>
        Thank you for your interest in the APEX_MAIL package.
        Sincerely,
        The Application Express Dev Team
      </body>
    </html>

  apex_mail.send(
    p_to   => 'some_user@somewhere.com',   -- change to your email address
    p_from => 'some_sender@somewhere.com', -- change to a real senders email address
    p_body      => l_body,
    p_body_html => l_body_html,
    p_subj      => 'APEX_MAIL Package - HTML formatted message');
END;
/```
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:**
- About the `APEX_PLSQL_JOB` Package
- `JOBS_ARE_ENABLED` Function
- `PURGE_PROCESS` Procedure
- `SUBMIT_PROCESS` Function
- `TIME_ELAPSED` Function
- `UPDATE_JOB_STATUS` Procedure
About the APEX_PLSQL_JOB Package

APEX_PLSQL_JOB is a wrapper package around DBMS_JOB functionality offered in the Oracle database. Note that the APEX_PLSQL_JOB package only exposes that functionality which is necessary to run PL/SQL in the background.

Table 17–1 describes the functions available in the APEX_PLSQL_JOB package.

<table>
<thead>
<tr>
<th>Function or Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBMIT_PROCESS</td>
<td>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.</td>
</tr>
<tr>
<td>UPDATE_JOB_STATUS</td>
<td>Call this procedure to update the status of the currently running job. This procedure is most effective when called from the submitted PL/SQL.</td>
</tr>
<tr>
<td>TIME_ELAPSED</td>
<td>Use this function to determine how much time has elapsed since the job was submitted.</td>
</tr>
<tr>
<td>JOBS_ARE_ENABLED</td>
<td>Call this function to determine whether the database is currently in a mode that supports submitting jobs to the APEX_PLSQL_JOB package.</td>
</tr>
<tr>
<td>PURGE_PROCESS</td>
<td>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.</td>
</tr>
</tbody>
</table>

You can view all jobs submitted to the APEX_PLSQL_JOB package using the APEX_PLSQL_JOBS view.
JOBS_ARE_ENABLED Function

Call this function to determine whether the database is currently in a mode that supports submitting jobs to the APEX_PLSQL_JOB package.

Syntax
APEX_PLSQL_JOB.JOBS_ARE_ENABLED
RETURN BOOLEAN;

Parameters
None.

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

BEGIN
  IF APEX_PLSQL_JOB.JOBS_ARE_ENABLED THEN
    HTP.P('Jobs are enabled on this database instance.');
  ELSE
    HTP.P('Jobs are not enabled on this database instance.');
  END IF;
END;
PURGE_PROCESS Procedure

Call this procedure to clean up submitted jobs. Submitted jobs stay in the APEX_PLSQL_JOBS view until either Oracle Application Express cleans out those records, or you call PURGE_PROCESS to manually remove them.

Syntax
APEX_PLSQL_JOB.PURGE_PROCESS (  
   p_job IN NUMBER);  

Parameters
Table 17–2 describes the parameters available in the PURGE_PROCESS procedure.

Table 17–2  PURGE_PROCESS Parameters
<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 want 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 submitted jobs by referencing the APEX_PLSQL_JOBS view.

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

Use this function to submit background PL/SQL. This function 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 17–3 describes the parameters available in the SUBMIT_PROCESS function.

Table 17–3 SUBMIT_PROCESS Parameters

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

Example
The following example shows how to use the SUBMIT_PROCESS function to submit a background process that starts 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 17–4 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 for which you want to determine the time that has passed 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 17–5 describes the parameters available in the UPDATE_JOB_STATUS procedure.

Table 17–5 UPDATE_JOB_STATUS Parameters

<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 want to update the status of.</td>
</tr>
<tr>
<td>p_status</td>
<td>The string of up to 100 characters to be used as the current status of the 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 is assigned to this process as it is submitted to APEX_PLSQL_JOB. By specifying this reserved item inside your process code, it is 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 are not 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 happens regardless of whether the job succeeds or fails. This last point is important for two reasons:
  
  1. Even if your status shows "100 rows inserted", it does not mean the entire operation was successful. If an error occurred at the time the block of code tried to commit, the user_status column of APEX_PLSQL_JOBS would not be affected because status updates are committed separately.
  2. Updates are performed autonomously. You can view the job status before the job has completed. This gives you the ability to display status text about ongoing operations in the background as they are happening.

BEGIN
  FOR i IN 1 .. 100 LOOP
    INSERT INTO emp(a,b) VALUES (:APP_JOB,i);
    IF MOD(i,10) = 0 THEN
      APEX_PLSQL_JOB.UPDATE_JOB_STATUS(
        P_JOB => :APP_JOB,
UPDATE_JOB_STATUS Procedure

P_STATUS => i || ' rows inserted');
END IF;
APEX_UTIL.PAUSE(2);
END LOOP;
END;
The APEX_PLUGIN package provides the interface declarations and some utility functions to work with plug-ins.

Topics:
- Data Types
- GET_AJAX_IDENTIFIER Function
- GET_INPUT_NAME_FOR_PAGE_ITEM Function
Data Types

The data types used by the APEX_PLUGIN package are described in this section.

Data Types:
- c_*
- t_authentication
- t_authentication_ajax_result
- t_authentication_auth_result
- t_authentication_inval_result
- t_authentication_logout_result
- t_authentication_sentry_result
- t_dynamic_action
- t_dynamic_action_ajax_result
- t_dynamic_action_render_result
- t_page_item
- t_page_item_ajax_result
- t_page_item_render_result
- t_page_item_validation_result
- t_plugin
- t_process
- t_process_exec_result
- t_region
- t_region_ajax_result
- t_region_render_result

The following constants are used for display_location in the page item validation function result type t_page_item_validation_result.

```plaintext
c_inline_with_field   constant varchar2(40) := 'INLINE_WITH_FIELD';
c_inline_with_field_and_notif constant varchar2(40) := 'INLINE_WITH_FIELD_AND_NOTIFICATION';
c_inline_in_notification constant varchar2(40) := 'INLINE_IN_NOTIFICATION';
c_on_error_page       constant varchar2(40) := 'ON_ERROR_PAGE';
```

**t_authentication**

```plaintext
type t_authentication is record {
    id                   number,
    name                 varchar2(255),
    invalid_session_url varchar2(4000),
    logout_url           varchar2(4000),
    plsql_code           clob,
    attribute_01         varchar2(32767),
}
```
attribute_02     varchar2(32767),
attribute_03     varchar2(32767),
attribute_04     varchar2(32767),
attribute_05     varchar2(32767),
attribute_06     varchar2(32767),
attribute_07     varchar2(32767),
attribute_08     varchar2(32767),
attribute_09     varchar2(32767),
attribute_10     varchar2(32767),
attribute_11     varchar2(32767),
attribute_12     varchar2(32767),
attribute_13     varchar2(32767),
attribute_14     varchar2(32767),
attribute_15     varchar2(32767),
--
session_id      number,
username        varchar2(255) );

t_authentication_ajax_result
type t_authentication_ajax_result is record {
    dummy        boolean );

t_authentication_auth_result
type t_authentication_auth_result is record {
    is_authenticated   boolean,
    redirect_url       varchar2(4000),
    log_code           number,
    log_text           varchar2(4000),
    display_text       varchar2(4000) );

t_authentication_inval_result
type t_authentication_inval_result is record {
    redirect_url       varchar2(4000) );

t_authentication_logout_result
type t_authentication_logout_result is record {
    redirect_url       varchar2(4000) );

t_authentication_sentry_result
type t_authentication_sentry_result is record {
    is_valid           boolean );

t_dynamic_action
The following type is passed into all dynamic action plug-in functions and contains information about the current dynamic action.
type t_dynamic_action is record {
    id        number,
    action    varchar2(50),
    attribute_01 varchar2(32767),
    attribute_02 varchar2(32767),
    attribute_03 varchar2(32767),
    attribute_04 varchar2(32767),
    attribute_05 varchar2(32767),
    attribute_06 varchar2(32767),
    attribute_07 varchar2(32767),
    attribute_08 varchar2(32767),
attribute_09 varchar2(32767),
attribute_10 varchar2(32767) );

**t_dynamic_action_ajax_result**
The following type is used as the result type for the AJAX function of a dynamic action type plug-in.

type t_dynamic_action_ajax_result is record {
    dummy boolean /* not used yet */
};

**t_dynamic_action_render_result**
The following type is used as the result type for the rendering function of a dynamic action plug-in.

type t_dynamic_action_render_result is record (
    javascript_function varchar2(32767),
    ajax_identifier varchar2(255),
    attribute_01 varchar2(32767),
    attribute_02 varchar2(32767),
    attribute_03 varchar2(32767),
    attribute_04 varchar2(32767),
    attribute_05 varchar2(32767),
    attribute_06 varchar2(32767),
    attribute_07 varchar2(32767),
    attribute_08 varchar2(32767),
    attribute_09 varchar2(32767),
    attribute_10 varchar2(32767) );

**t_page_item**
The following type is passed into all item type plug-in functions and contains information about the current page item.

type t_page_item is record {
    id                          number,
    name                        varchar2(255),
    label                       varchar2(4000),
    plain_label                 varchar2(4000),
    placeholder                 varchar2(255),
    format_mask                 varchar2(255),
    is_required                 boolean,
    lov_definition              varchar2(4000),
    lov_display_extra           boolean,
    lov_display_null            boolean,
    lov_null_text               varchar2(255),
    lov_null_value              varchar2(255),
    lov_cascade_parent_items    varchar2(255),
    ajax_items_to_submit       varchar2(255),
    ajax_optimize_refresh      boolean,
    element_width               number,
    element_max_length          number,
    element_height              number,
    element_css_classes         varchar2(255),
    element_attributes          varchar2(4000),
    element_option_attributes   varchar2(4000),
    escape_output               boolean,
    attribute_01                varchar2(32767),
    attribute_02                varchar2(32767),
    attribute_03                varchar2(32767),
Data Types

attribute_04        varchar2(32767),
attribute_05        varchar2(32767),
attribute_06        varchar2(32767),
attribute_07        varchar2(32767),
attribute_08        varchar2(32767),
attribute_09        varchar2(32767),
attribute_10        varchar2(32767) );

t_page_item_ajax_result
The following type is used as the result type for the AJAX function of an item type plug-in.

```plsql
type t_page_item_ajax_result is record {
    dummy boolean /*! not used yet */
};
```

t_page_item_render_result
The following type is used as the result type for the rendering function of an item type plug-in.

```plsql
type t_page_item_render_result is record (
    is_navigable     boolean default false,
    navigable_dom_id varchar2(255)          /* should only be set if navigable
    element is not equal to item name */
);  
```

t_page_item_validation_result
The following type is used as the result type for the validation function of an item type plug-in.

```plsql
type t_page_item_validation_result is record (
    message          varchar2(32767),
    display_location varchar2(40),    /* if not set the application default is
    page_item_name   varchar2(255)   */; /* if not set the validated page item name
    is used */
);  
```

t_plugin
The following type is passed into all plug-in functions and contains information about the current plug-in.

```plsql
type t_plugin is record {
    name         varchar2(45),
    file_prefix  varchar2(4000),
    attribute_01 varchar2(32767),
    attribute_02 varchar2(32767),
    attribute_03 varchar2(32767),
    attribute_04 varchar2(32767),
    attribute_05 varchar2(32767),
    attribute_06 varchar2(32767),
    attribute_07 varchar2(32767),
    attribute_08 varchar2(32767),
    attribute_09 varchar2(32767),
    attribute_10 varchar2(32767) );
```

t_process
The following type is passed into all process type plug-in functions and contains information about the current process.
**Data Types**

```plsql
type t_process is record {
    id                   number,
    name                 varchar2(255),
    success_message      varchar2(32767),
    attribute_01         varchar2(32767),
    attribute_02         varchar2(32767),
    attribute_03         varchar2(32767),
    attribute_04         varchar2(32767),
    attribute_05         varchar2(32767),
    attribute_06         varchar2(32767),
    attribute_07         varchar2(32767),
    attribute_08         varchar2(32767),
    attribute_09         varchar2(32767),
    attribute_10         varchar2(32767));

T_PROCESS_EXEC_RESULT
The following type is used as the result type for the execution function of a process
plug-in.

```plsql
type t_process_exec_result is record {
    success_message varchar2(32767)
};

T_REGION
The following type is passed into all region type plug-in functions and contains
information about the current region.

```plsql
type t_region is record {
    id                   number,
    static_id            varchar2(255),
    name                 varchar2(255),
    type                 varchar2(255),
    source               varchar2(32767),
    ajax_items_to_submit varchar2(32767),
    fetched_rows         pls_integer,
    escape_output        boolean,
    no_data_found_message varchar2(32767),
    attribute_01         varchar2(32767),
    attribute_02         varchar2(32767),
    attribute_03         varchar2(32767),
    attribute_04         varchar2(32767),
    attribute_05         varchar2(32767),
    attribute_06         varchar2(32767),
    attribute_07         varchar2(32767),
    attribute_08         varchar2(32767),
    attribute_09         varchar2(32767),
    attribute_10         varchar2(32767),
    attribute_11         varchar2(32767),
    attribute_12         varchar2(32767),
    attribute_13         varchar2(32767),
    attribute_14         varchar2(32767),
    attribute_15         varchar2(32767),
    attribute_16         varchar2(32767),
    attribute_17         varchar2(32767),
    attribute_18         varchar2(32767),
    attribute_19         varchar2(32767),
    attribute_20         varchar2(32767),
    attribute_21         varchar2(32767),
    attribute_22         varchar2(32767),
    attribute_23         varchar2(32767),
```
attribute_24         varchar2(32767),
attribute_25         varchar2(32767));

t_region_ajax_result
The following type is used as result type for the AJAX function of a region type plug-in.

type t_region_ajax_result is record {
  dummy boolean /* not used yet */
};

t_region_render_result
The following type is used as the result type for the rendering function of a region type plug-in.

type t_region_render_result is record {
  dummy boolean /* not used yet */
};
GET_AJAX_IDENTIFIER Function

This function returns the AJAX identifier used to call the AJAX callback function defined for the plug-in.

**Note:** This function only works in the context of a plug-in rendering function call and only if the plug-in has defined an AJAX function callback in the plug-in definition.

**Syntax**

APEX_PLUGIN.GET_AJAX_IDENTIFIER
RETURN VARCHAR2;

**Parameters**

None.

**Example**

This is an example of a dynamic action plug-in rendering function that supports an AJAX callback.

```sql
function render_set_value (p_dynamic_action in apex_plugin.t_dynamic_action )
return apex_plugin.t_dynamic_action_render_result
is
  l_result                apex_plugin.t_dynamic_action_render_result;
begin
  l_result.javascript_function := 'com_oracle_apex_set_value';
  l_result.ajax_identifier     := wwv_flow_plugin.get_ajax_identifier;
  return l_result;
end;
```
GET_INPUT_NAME_FOR_PAGE_ITEM Function

Use this function when you want to render an HTML input element in the rendering function of an item type plug-in.

For the HTML input element, for example, `<input type="text" id="P1_TEST" name="xxx">`, you have to provide a value for the `name` attribute so that Oracle Application Express can map the submitted value to the actual page item in session state. This function returns the mapping `name` for your page item. If the HTML input element has multiple values, such as a select list with `multiple="multiple"`, then set `p_is_multi_value` to true.

Note: This function is only useful when called in the rendering function of an item type plug-in.

Syntax

APEX_PLUGIN.GET_INPUT_NAME_FOR_PAGE_ITEM (  
  p_is_multi_value  IN BOOLEAN)  
RETURN VARCHAR2;

Parameters

Table 18–1 describes the parameters available in the GET_INPUT_NAME_FOR_PAGE_ITEM function.

Table 18–1  GET_INPUT_NAME_FOR_PAGE_ITEM Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_is_multi_value</td>
<td>Set to TRUE if the HTML input element has multiple values. If not, set to FALSE. HTML input elements with multiple values can be checkboxes and multi select lists.</td>
</tr>
</tbody>
</table>

Example

The following example outputs the necessary HTML code to render a text field where the value gets stored in session state when the page is submitted.

```sql
sys.htp.prn (  
  '<input type="text" id="'||p_item.name||'" |'  
  'name='||wwv_flow_plugin.get_input_name_for_page_item(false)||'" |'  
  'value='||sys.htf.escape_sc(p_value)||'" |'  
  'size='||p_item.element_width||'" |'  
  'maxlength='||p_item.element_max_length||'" |'  
  coalesce(p_item.element_attributes, 'class="text_field"')||' />' );
```
The APEX_PLUGIN_UTIL package provides utility functions that solve common problems when writing a plug-in.

**Topics:**
- DEBUG_DYNAMIC_ACTION Procedure
- DEBUG_PAGE_ITEM Procedure Signature 1
- DEBUG_PAGE_ITEM Procedure Signature 2
- DEBUG_PROCESS Procedure
- DEBUG_REGION Procedure Signature 1
- DEBUG_REGION Procedure Signature 2
- ESCAPE Function
- EXECUTE_PLSQL_CODE Procedure
- GET_DATA Function Signature 1
- GET_DATA Function Signature 2
- GET_DATA2 Function Signature 1
- GET_DATA2 Function Signature 2
- GET_DISPLAY_DATA Function Signature 1
- GET_DISPLAY_DATA Function Signature 2
- GET_ELEMENT_ATTRIBUTES Function
- GET_PLSQL_EXPRESSION_RESULT Function
- GET_PLSQL_FUNCTION_RESULT Function
- GET_POSITION_IN_LIST Function
- GET_SEARCH_STRING Function
- IS_EQUAL Function
- PAGE_ITEM_NAMES_TO_JQUERY Function
- PRINT_DISPLAY_ONLY Procedure
- PRINT_ESCAPE_VALUE Procedure
- PRINT_HIDDEN_IF_READONLY Procedure
- PRINT_JSON_HTTP_HEADER Procedure
- PRINT_LOV_AS_JSON Procedure
- PRINT_OPTION Procedure
- REPLACE_SUBSTITUTIONS Function
DEBUG_DYNAMIC_ACTION Procedure

This procedure writes the data of the dynamic action meta data to the debug output if debugging is enabled.

Syntax
APEX_PLUGIN_UTIL.DEBUG_DYNAMIC_ACTION (
    p_plugin         IN apex_plugin.t_plugin,
    p_dynamic_action IN apex_plugin.t_dynamic_action);

Parameters
Table 19–2 describes the parameters available in the DEBUG_DYNAMIC_ACTION procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_plugin</td>
<td>This is the p_plugin parameter of your plug-in function.</td>
</tr>
<tr>
<td>p_dynamic_action</td>
<td>This is the p_dynamic_action parameter of your plug-in function.</td>
</tr>
</tbody>
</table>

Example
This example shows how to collect helpful debug information during the plug-in development cycle to see what values are actually passed into the rendered function or AJAX callback function of the plug-in.

```
APEX_PLUGIN_UTIL.debug_dynamic_action (
    p_plugin         => p_plugin,
    p_dynamic_action => p_dynamic_action );
```
DEBUG_PAGE_ITEM Procedure Signature 1

This procedure writes the data of the page item meta data to the debug output if debugging is enabled.

Syntax
APEX_PLUGIN_UTIL.DEBUG_PAGE_ITEM (  
  p_plugin    IN apex_plugin.t_plugin,  
  p_page_item IN apex_plugin.t_page_item);  

Parameters
Table 19–2 describes the parameters available in the DEBUG_PAGE_ITEM procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_plugin</td>
<td>This is the p_plugin parameter of your plug-in function.</td>
</tr>
<tr>
<td>p_page_item</td>
<td>This is the p_page_item parameter of your plug-in function.</td>
</tr>
</tbody>
</table>

Example
This example shows how to collect helpful debug information during the plug-in development cycle to see what values are actually passed into the renderer, AJAX callback or validation function.

```
APEX_PLUGIN_UTIL.DEBUG_PAGE_ITEM (  
  p_plugin    => p_plugin,  
  p_page_item => p_page_item );
```
DEBUG_PAGE_ITEM Procedure Signature 2

This procedure writes the data of the page item meta data to the debug output if debugging is enabled.

Syntax

APEX_PLUGIN_UTIL.DEBUG_PAGE_ITEM (  
  p_plugin              IN apex_plugin.t_plugin,  
  p_page_item           IN apex_plugin.t_page_item,  
  p_value               IN VARCHAR2,  
  p_is_readonly         IN BOOLEAN,  
  p_is_printer_friendly IN BOOLEAN);  

Parameters

Table 19–3 describes the parameters available in the DEBUG_PAGE_ITEM procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_plugin</td>
<td>This is the p_plugin parameter of your plug-in function.</td>
</tr>
<tr>
<td>p_page_item</td>
<td>This is the p_page_item parameter of your plug-in function.</td>
</tr>
<tr>
<td>p_value</td>
<td>This is the p_value parameter of your plug-in function.</td>
</tr>
<tr>
<td>p_is_readonly</td>
<td>This is the p_is_readonly parameter of your plug-in function.</td>
</tr>
<tr>
<td>p_is_printer_friendly</td>
<td>This is the p_is_printer_friendly parameter of your plug-in function.</td>
</tr>
</tbody>
</table>

Example

This example shows how to collect helpful debug information during the plug-in development cycle to see what values are actually passed into the renderer, AJAX callback or validation function.

```sql
APEX_PLUGIN_UTIL.debug_page_item (  
  p_plugin => p_plugin,  
  p_page_item => p_page_item,  
  p_value => p_value,  
  p_is_readonly => p_is_readonly,  
  p_is_printer_friendly => p_is_printer_friendly);  
```
DEBUG_PROCESS Procedure

This procedure writes the data of the process meta data to the debug output if debugging is enabled.

Syntax
APEX_PLUGIN_UTIL.DEBUG_PROCESS (
  p_plugin         IN apex_plugin.t_plugin,
  p_process        IN apex_plugin.t_process);

Parameters
Table 19–4 describes the parameters available in the DEBUG_PROCESS procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_plugin</td>
<td>This is the p_plugin parameter of your plug-in function.</td>
</tr>
<tr>
<td>p_process</td>
<td>This is the p_process parameter of your plug-in function.</td>
</tr>
</tbody>
</table>

Example
This example shows how to collect helpful debug information during the plug-in development cycle to see what values are actually passed into the execution function of the plug-in.

```javascript
apex_plugin_util.debug_process {
  p_plugin => p_plugin,
  p_process => p_process);
```
DEBUG_REGION Procedure Signature 1

This procedure writes the data of the region meta data to the debug output if debugging is enabled.

Syntax
APEX_PLUGIN_UTIL.DEBUG_REGION (  
p_plugin IN apex_plugin.t_plugin,  
p_region IN apex_plugin.t_region);

Parameters
Table 19–5 describes the parameters available in the DEBUG_REGION procedure.

Table 19–5  DEBUG_REGION Signature 1 Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_plugin</td>
<td>This is the p_plugin parameter of your plug-in function.</td>
</tr>
<tr>
<td>p_region</td>
<td>This is the p_region parameter of your plug-in function.</td>
</tr>
</tbody>
</table>

Example
This example shows how to collect helpful debug information during the plug-in development cycle to see what values are actually passed into the render function or AJAX callback function of the plug-in.

apex_plugin_util.debug_process (  
p_plugin => p_plugin,  
p_region => p_region);


DEBUG_REGION Procedure Signature 2

This procedure writes the data of the region meta data to the debug output if debugging is enabled. This is the advanced version of the debugging procedure which is used for the rendering function of a region plug-in.

Syntax

APEX_PLUGIN_UTIL.DEBUG_REGION (  
    p_plugin                IN apex_plugin.t_plugin,  
    p_region                IN apex_plugin.t_region,  
    p_is_printer_friendly   IN BOOLEAN);

Parameters

Table 19–6 describes the parameters available in the DEBUG_REGION procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_plugin</td>
<td>This is the p_plugin parameter of your plug-in function</td>
</tr>
<tr>
<td>p_region</td>
<td>This is the p_region parameter of your plug-in function</td>
</tr>
<tr>
<td>p_is_printer_friendly</td>
<td>This is the p_is_printer_friendly parameter of your plug-in function</td>
</tr>
</tbody>
</table>

Example

This example shows how to collect helpful debug information during the plug-in development cycle to see what values are actually passed into the render function or AJAX callback function of the plug-in.

apex_plugin_util.debug_process (  
    p_plugin => p_plugin,  
    p_region => p_region,  
    p_is_printer_friendly => p_is_printer_friendly);
ESCAPE Function

This function is used if you have checked the standard attribute "Has Escape Output Attribute" option for your item type plug-in which allows a developer to decide if the output should be escaped or not.

Syntax
APEX_PLUGIN_UTIL.ESCAPE (p_value IN VARCHAR2, p_escape IN BOOLEAN)
RETURN VARCHAR2;

Parameters
Table 19–7 describes the parameters available in the ESCAPE function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_value</td>
<td>This is the value you want to escape depending on the p_escape parameter.</td>
</tr>
<tr>
<td>p_escape</td>
<td>If set to TRUE, the return value is escaped. If set to FALSE, the value is not escaped.</td>
</tr>
</tbody>
</table>

Example
This example outputs all values of the array l_display_value_list as a HTML list and escapes the value of the array depending on the setting the developer as picked when using the plug-in.

```sql
for i in 1 .. l_display_value_list.count loop
  sys.htp.prn ('<li>' ||
    apex_plugin_util.escape (p_value => l_display_value_list(i),
    p_escape => p_item.escape_output ||
      '</li> ');
end loop;
```
EXECUTE_PLSQL_CODE Procedure

This procedure executes a PL/SQL code block and performs binding of bind variables in the provided PL/SQL code. This procedure is usually used for plug-in attributes of type PL/SQL Code.

Syntax

APEX_PLUGIN_UTIL.EXECUTE_PLSQL_CODE (
    p_plsql_code  IN VARCHAR2);

Parameters

Table 19–8 describes the parameters available in the EXECUTE_PLSQL_CODE procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_plsql_code</td>
<td>PL/SQL code to be executed.</td>
</tr>
</tbody>
</table>

Example

Text which should be escaped and then printed to the HTTP buffer.

declare
    l_plsql_code VARCHAR(32767) := p_process.attribute_01;
begin
    apex_plugin_util.execute_plsql_code (
        p_plsql_code => l_plsql_code );
end;
GET_DATA Function Signature 1

Executes the specified SQL query restricted by the provided search string (optional) and returns the values for each column. All column values are returned as a string, independent of their data types. The search column is identified by providing a column number in the p_search_column_no parameter.

Syntax

```sql
APEX_PLUGIN_UTIL.GET_DATA (  
p_sql_statement    IN VARCHAR2,  
p_min_columns      IN NUMBER,  
p_max_columns      IN NUMBER,  
p_component_name   IN VARCHAR2,  
p_search_type      IN VARCHAR2 DEFAULT 2,  
p_search_column_no IN VARCHAR2 DEFAULT 2,  
p_search_string    IN VARCHAR2 DEFAULT NULL,  
p_first_row        IN NUMBER DEFAULT NULL,  
p_max_rows         IN NUMBER DEFAULT NULL)  
RETURN t_column_value_list;
```

Parameters

Table 19–9 describes the parameters available in the GET_DATA function signature 1.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_sql_statement</td>
<td>SQL statement used for the lookup.</td>
</tr>
<tr>
<td>p_min_columns</td>
<td>Minimum number of return columns.</td>
</tr>
<tr>
<td>p_max_columns</td>
<td>Maximum number of return columns.</td>
</tr>
<tr>
<td>p_component_name</td>
<td>In case an error is returned, this is the name of the page item or report column used to display the error message.</td>
</tr>
<tr>
<td>p_search_type</td>
<td>Must be one of the c_search_* constants. They are as follows: c_search_contains_case, c_search_contains_ignore, c_search_exact_case, c_search_exact_ignore</td>
</tr>
<tr>
<td>p_search_column_no</td>
<td>Number of the column used to restrict the SQL statement. Must be within the p_min_columns though p_max_columns range.</td>
</tr>
<tr>
<td>p_search_string</td>
<td>Value used to restrict the query.</td>
</tr>
<tr>
<td>p_first_row</td>
<td>Start query at the specified row. All rows before the specified row are skipped.</td>
</tr>
<tr>
<td>p_max_rows</td>
<td>Maximum number of return rows allowed.</td>
</tr>
</tbody>
</table>

Return

Table 19–10 describes the return value by the GET_DATA function signature 1.

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_column_value_list</td>
<td>Table of wwv_flow_global.vc_arr2 indexed by column number.</td>
</tr>
</tbody>
</table>
**Example**

The following example shows a simple item type plug-in rendering function which executes the LOV defined for the page item and does a case sensitive LIKE filtering with the current value of the page item. The result is then generated as a HTML list.

```sql
function render_list ( 
    p_item                in apex_plugin.t_page_item,
    p_value               in varchar2,
    p_is_readonly         in boolean,
    p_is_printer_friendly in boolean ) 
return apex_plugin.t_page_item_render_result 
is 
  l_column_value_list   apex_plugin_util.t_column_value_list;
begin
  l_column_value_list := 
    apex_plugin_util.get_data ( 
      p_sql_statement    => p_item.lov_definition,
      p_min_columns      => 2,
      p_max_columns      => 2,
      p_component_name   => p_item.name,
      p_search_type      => apex_plugin_util.c_search_contains_case,
      p_search_column_no => 1,
      p_search_string    => p_value );

  sys.htp.p('<ul>);
  for i in 1 .. l_column_value_list(1).count
    loop
    sys.htp.p(
      '<li>'||
      sys.htf.escape_sc(l_column_value_list(1)(i))|| -- display column
      '-'||
      sys.htf.escape_sc(l_column_value_list(2)(i))|| -- return column
      '</li>');
    end loop;
  sys.htp.p('</ul>');
end render_list;
```
GET_DATA Function Signature 2

Executes the specified SQL query restricted by the provided search string (optional) and returns the values for each column. All column values are returned as a string, independent of their data types. The search column is identified by providing a column name in the `p_search_column_name` parameter.

Syntax

```
APEX_PLUGIN_UTIL.GET_DATA (
    p_sql_statement      IN VARCHAR2,
    p_min_columns        IN NUMBER,
    p_max_columns        IN NUMBER,
    p_component_name     IN VARCHAR2,
    p_search_type        IN VARCHAR2 DEFAULT NULL,
    p_search_column_name IN VARCHAR2 DEFAULT NULL,
    p_search_string      IN VARCHAR2 DEFAULT NULL,
    p_first_row          IN NUMBER DEFAULT NULL,
    p_max_rows           IN NUMBER DEFAULT NULL)
RETURN t_column_value_list;
```

Parameters

Table 19–11 describes the parameters available for GET_DATA function signature 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p_sql_statement</code></td>
<td>SQL statement used for the lookup.</td>
</tr>
<tr>
<td><code>p_min_columns</code></td>
<td>Minimum number of return columns.</td>
</tr>
<tr>
<td><code>p_max_columns</code></td>
<td>Maximum number of return columns.</td>
</tr>
<tr>
<td><code>p_component_name</code></td>
<td>In case an error is returned, this is the name of the page item or report column used to display the error message.</td>
</tr>
<tr>
<td><code>p_search_type</code></td>
<td>Must be one of the <code>c_search_*</code> constants. They are as follows: <code>c_search_contains_case</code>, <code>c_search_contains_ignore</code>, <code>c_search_exact_case</code>, <code>c_search_exact_ignore</code>.</td>
</tr>
<tr>
<td><code>p_search_column_name</code></td>
<td>This is the column name used to restrict the SQL statement.</td>
</tr>
<tr>
<td><code>p_search_string</code></td>
<td>Value used to restrict the query.</td>
</tr>
<tr>
<td><code>p_first_row</code></td>
<td>Start query at the specified row. All rows before the specified row are skipped.</td>
</tr>
<tr>
<td><code>p_max_rows</code></td>
<td>Maximum number of return rows allowed.</td>
</tr>
</tbody>
</table>

Return

Table 19–12 describes the return value by the GET_DATA function signature 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>t_column_value_list</code></td>
<td>Table of <code>wwv_flow_global.vc_arr2</code> indexed by column number.</td>
</tr>
</tbody>
</table>
Example

The following example shows a simple item type plug-in rendering function which executes the LOV defined for the page item and does a case sensitive LIKE filtering with the current value of the page item. The result is then generated as a HTML list.

function render_list ( 
  p_item                in apex_plugin.t_page_item,
  p_value               in varchar2,
  p_is_readonly         in boolean,
  p_is_printer_friendly in boolean )
return apex_plugin.t_page_item_render_result
is
  l_column_value_list   apex_plugin_util.t_column_value_list;
begin
  l_column_value_list :=
    apex_plugin_util.get_data ( 
      p_sql_statement    => p_item.lov_definition,
      p_min_columns      => 2,
      p_max_columns      => 2,
      p_component_name   => p_item.name,
      p_search_type      => apex_plugin_util.c_search_contains_case,
      p_search_column_name => 'ENAME',
      p_search_string    => p_value );
  sys.htp.p('<ul>');
  for i in 1 .. l_column_value_list(1).count
    loop
    sys.htp.p(' 
      <li>'||
        sys.htf.escape_sc(l_column_value_list(1)(i)) || -- display column
      '-'||
        sys.htf.escape_sc(l_column_value_list(2)(i)) || -- return column
    '</li>');
  end loop;
  sys.htp.p('</ul>');
end render_list;
GET_DATA2 Function Signature 1

Executes the specified SQL query restricted by the provided search string (optional) and returns the values for each column. All column values are returned along with their original data types. The search column is identified by providing a column number in the p_search_column_no parameter.

Syntax

APEX_PLUGIN_UTIL.GET_DATA2 (  
  p_sql_statement    IN VARCHAR2,  
  p_min_columns      IN NUMBER,  
  p_max_columns      IN NUMBER,  
  p_data_type_list   IN WWV_GLOBAL.VC_ARR2 DEFAULT C_EMPTY_DATA_TYPE_LIST,  
  p_component_name   IN VARCHAR2,  
  p_search_type      IN VARCHAR2 DEFAULT 2,  
  p_search_column_no IN VARCHAR2 DEFAULT 2,  
  p_search_string    IN VARCHAR2 DEFAULT NULL,  
  p_first_row        IN NUMBER DEFAULT NULL,  
  p_max_rows         IN NUMBER DEFAULT NULL)  
RETURN t_column_value_list2;

Parameters

Table 19–13 describes the parameters available in the GET_DATA2 function.

Table 19–13   GET_DATA2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_sql_statement</td>
<td>SQL statement used for the lookup.</td>
</tr>
<tr>
<td>p_min_columns</td>
<td>Minimum number of return columns.</td>
</tr>
<tr>
<td>p_max_columns</td>
<td>Maximum number of return columns.</td>
</tr>
<tr>
<td>p_data_type_list</td>
<td>If provided, checks to make sure the data type for each column matches the specified data type in the array. Use the constants c_data_type_* for available data types.</td>
</tr>
<tr>
<td>p_component_name</td>
<td>In case an error is returned, this is the name of the page item or report column used to display the error message.</td>
</tr>
<tr>
<td>p_search_type</td>
<td>Must be one of the c_search_* constants. They are as follows: c_search_contains_case, c_search_contains_ignore, c_search_exact_case, c_search_exact_ignore</td>
</tr>
<tr>
<td>p_search_column_no</td>
<td>Number of the column used to restrict the SQL statement. Must be within the p_min_columns though p_max_columns range.</td>
</tr>
<tr>
<td>p_search_string</td>
<td>Value used to restrict the query.</td>
</tr>
<tr>
<td>p_first_row</td>
<td>Start query at the specified row. All rows before the specified row are skipped.</td>
</tr>
<tr>
<td>p_max_rows</td>
<td>Maximum number of return rows allowed.</td>
</tr>
</tbody>
</table>

Return

Table 19–14 describes the return value by the GET_DATA2 function.
Example

The following example is a simple item type plug-in rendering function which executes the LOV defined for the page item and does a case sensitive LIKE filtering with the current value of the page item. The result is then generated as a HTML list. This time, the first column of the LOV SQL statement is checked if it is of type VARCHAR2 and the second is of type NUMBER.

```sql
function render_list ( p_item                in apex_plugin.t_page_item,
                       p_value               in varchar2,
                       p_is_readonly         in boolean,
                       p_is_printer_friendly in boolean )
return apex_plugin.t_page_item_render_result
is
  l_data_type_list    wwv_flow_global.vc_arr2;
  l_column_value_list apex_plugin_util.t_column_value_list2;
begin
  -- The first LOV column has to be a string and the second a number
  l_data_type_list(1) := apex_plugin_util.c_data_type_varchar2;
  l_data_type_list(2) := apex_plugin_util.c_data_type_number;

  l_column_value_list :=
    apex_plugin_util.get_data2 ( p_sql_statement    => p_item.lov_definition,
                               p_min_columns      => 2,
                               p_max_columns      => 2,
                               p_data_type_list   => l_data_type_list,
                               p_component_name   => p_item.name,
                               p_search_type      => apex_plugin_util.c_search_contains_case,
                               p_search_column_no => 1,
                               p_search_string    => p_value );

  sys.htp.p('<ul>');
  for i in 1 .. l_column_value_list.count(1)
  loop
    sys.htp.p( '<li>'||
                sys.htf.escape_sc(l_column_value_list(1).value_list(i).varchar2_value)|| -- display column
                ' - '||
                sys.htf.escape_sc(l_column_value_list(2).value_list(i).number_value)|| -- return column
                '</li>');
  end loop;
  sys.htp.p('</ul>');
end render_list;
```

Table 19–14 GET_DATA2 Return

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_column_value_list2</td>
<td>Table of t_column_values indexed by column number.</td>
</tr>
</tbody>
</table>
GET_DATA2 Function Signature 2

Executes the specified SQL query restricted by the provided search string (optional) and returns the values for each column. All column values are returned along with their original data types. The search column is identified by providing a column number in the p_search_column_no parameter.

Syntax

APEX_PLUGIN_UTIL.GET_DATA2 (  
p_sql_statement IN VARCHAR2,  
p_min_columns IN NUMBER,  
p_max_columns IN NUMBER,  
p_data_type_list IN WNV_GLOBAL.VC_ARR2 DEFAULT C_EMPTY_DATA_TYPE_LIST,  
p_component_name IN VARCHAR2,  
p_search_type IN VARCHAR2 DEFAULT 2,  
p_search_column_name IN VARCHAR2 DEFAULT 2,  
p_search_string IN VARCHAR2 DEFAULT NULL,  
p_first_row IN NUMBER DEFAULT NULL,  
p_max_rows IN NUMBER DEFAULT NULL)  
RETURN t_column_value_list2;

Parameters

Table 19–15 describes the parameters available in the GET_DATA2 function signature 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_sql_statement</td>
<td>SQL statement used for the lookup.</td>
</tr>
<tr>
<td>p_min_columns</td>
<td>Minimum number of return columns.</td>
</tr>
<tr>
<td>p_max_columns</td>
<td>Maximum number of return columns.</td>
</tr>
<tr>
<td>p_data_type_list</td>
<td>If provided, checks to make sure the data type for each column matches the specified data type in the array. Use the constants c_data_type_* for available data types.</td>
</tr>
<tr>
<td>p_component_name</td>
<td>In case an error is returned, this is the name of the page item or report column used to display the error message.</td>
</tr>
<tr>
<td>p_search_type</td>
<td>Must be one of the c_search_* constants. They are as follows: c_search_contains_case, c_search_contains_ignore, c_search_exact_case, c_search_exact_ignore</td>
</tr>
<tr>
<td>p_search_column_name</td>
<td>The column name used to restrict the SQL statement.</td>
</tr>
<tr>
<td>p_search_string</td>
<td>Value used to restrict the query.</td>
</tr>
<tr>
<td>p_first_row</td>
<td>Start query at the specified row. All rows before the specified row are skipped.</td>
</tr>
<tr>
<td>p_max_rows</td>
<td>Maximum number of return rows allowed.</td>
</tr>
</tbody>
</table>

Return

Table 19–16 describes the return value by the GET_DATA2 function signature 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_column_value_list2</td>
<td>Table of t_column_values indexed by column number.</td>
</tr>
</tbody>
</table>
Example

The following example is a simple item type plug-in rendering function which executes the LOV defined for the page item and does a case sensitive LIKE filtering with the current value of the page item. The result is then generated as a HTML list. This time, the first column of the LOV SQL statement is checked if it is of type VARCHAR2 and the second is of type number.

```lisp
function render_list ( 
  p_item                in apex_plugin.t_page_item, 
  p_value               in varchar2, 
  p_is_readonly         in boolean, 
  p_is_printer_friendly in boolean ) 
return apex_plugin.t_page_item_render_result 
is 
  l_data_type_list    wwv_flow_global.vc_arr2; 
  l_column_value_list apex_plugin_util.t_column_value_list2; 
begin 
  -- The first LOV column has to be a string and the second a number 
  l_data_type_list(1) := apex_plugin_util.c_data_type_varchar2; 
  l_data_type_list(2) := apex_plugin_util.c_data_type_number; 
  --
  l_column_value_list := 
    apex_plugin_util.get_data2 ( 
      p_sql_statement    => p_item.lov_definition, 
      p_min_columns      => 2, 
      p_max_columns      => 2, 
      p_data_type_list   => l_data_type_list, 
      p_component_name   => p_item.name, 
      p_search_type      => apex_plugin_util.c_search_contains_case, 
      p_search_column_name => 'ENAME', 
      p_search_string    => p_value );
  --
  sys.htp.p('<ul>'); 
  for i in 1 .. l_column_value_list.count(1) 
  loop 
    sys.htp.p( 
      '<li>'|| 
      sys.htf.escape_sc(l_column_value_list(1).value_list(i).varchar2_value)|| -- display column 
      '-'|| 
      sys.htf.escape_sc(l_column_value_list(2).value_list(i).number_value)|| 
      -- return column 
      '</li>');
  end loop; 
  sys.htp.p('</ul>');
end render_list;
```
GET_DISPLAY_DATA Function Signature 1

This function gets the display lookup value for the value specified in `p_search_string`.

Syntax

```
APEX_PLUGIN_UTIL.GET_DISPLAY_DATA (   
    p_sql_statement    IN VARCHAR2,   
    p_min_columns      IN NUMBER,   
    p_max_columns      IN NUMBER,   
    p_component_name   IN VARCHAR2,   
    p_display_column_no IN BINARY_INTEGER DEFAULT 1,   
    p_search_column_no  IN BINARY_INTEGER DEFAULT 2,   
    p_search_string    IN VARCHAR2 DEFAULT NULL,   
    p_display_extra    IN BOOLEAN DEFAULT TRUE)   
RETURN VARCHAR2;
```

Parameters

Table 19–17 describes the parameters available in the GET_DISPLAY_DATA function signature 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_sql_statement</td>
<td>SQL statement used for the lookup.</td>
</tr>
<tr>
<td>p_min_columns</td>
<td>Minimum number of return columns.</td>
</tr>
<tr>
<td>p_max_columns</td>
<td>Maximum number of return columns.</td>
</tr>
<tr>
<td>p_component_name</td>
<td>In case an error is returned, this is the name of the page item or report</td>
</tr>
<tr>
<td></td>
<td>column used to display the error message.</td>
</tr>
<tr>
<td>p_display_column_no</td>
<td>Number of the column returned from the SQL statement. Must be within the</td>
</tr>
<tr>
<td></td>
<td>p_min_columns though p_max_columns range.</td>
</tr>
<tr>
<td>p_search_column_no</td>
<td>Number of the column used to restrict the SQL statement. Must be within the</td>
</tr>
<tr>
<td></td>
<td>p_min_columns though p_max_columns range.</td>
</tr>
<tr>
<td>p_search_string</td>
<td>Value used to restrict the query.</td>
</tr>
<tr>
<td>p_display_extra</td>
<td>If set to TRUE, and a value is not found, the search value is added to the</td>
</tr>
<tr>
<td></td>
<td>result instead.</td>
</tr>
</tbody>
</table>

Return

Table 19–18 describes the return value by the GET_DISPLAY_DATA function signature 1.

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2</td>
<td>Value of the first record of the column specified by <code>p_display_column_no</code>.</td>
</tr>
<tr>
<td></td>
<td>If no record was found it contains the value of <code>p_search_string</code> if the</td>
</tr>
<tr>
<td></td>
<td>parameter <code>p_display_extra</code> is set to TRUE. Otherwise NULL is returned.</td>
</tr>
</tbody>
</table>
Example
The following example does a lookup with the value provided in p_value and returns the display column of the LOV query.

```sql
function render_value (  
    p_item                in apex_plugin.t_page_item, 
    p_value               in varchar2, 
    p_is_readonly         in boolean, 
    p_is_printer_friendly in boolean )  
  return apex_plugin.t_page_item_render_result  
is
  begin
    sys.htp.p(sys.htf.escape_sc(  
      apex_plugin_util.get_display_data (  
        p_sql_statement   => p_item.lov_definition, 
        p_min_columns     => 2,  
        p_max_columns     => 2,  
        p_component_name  => p_item.name,  
        p_display_column_no => 1,  
        p_search_column_no => 2,  
        p_search_string   => p_value  
      )));
  end render_value;
```
GET_DISPLAY_DATA Function Signature 2

This function looks up all the values provided in the p_search_value_list instead of just a single value lookup.

Syntax
APEX_PLUGIN_UTIL.GET_DISPLAY_DATA (  
  p_sql_statement    IN VARCHAR2,  
  p_min_columns      IN NUMBER,  
  p_max_columns      IN NUMBER,  
  p_component_name   IN VARCHAR2,  
  p_display_column_no IN BINARY_INTEGER DEFAULT 1,  
  p_search_column_no  IN BINARY_INTEGER DEFAULT 2,  
  p_search_value_list IN ww_flow_global.vc_arr2,  
  p_display_extra     IN BOOLEAN DEFAULT TRUE)  
RETURN wwv_flow_global.vc_arr2;

Parameters
Table 19–19 describes the parameters available in the GET_DISPLAY_DATA function signature 2.

Table 19–19  GET_DISPLAY_DATA Signature 2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_sql_statement</td>
<td>SQL statement used for the lookup.</td>
</tr>
<tr>
<td>p_min_columns</td>
<td>Minimum number of return columns.</td>
</tr>
<tr>
<td>p_max_columns</td>
<td>Maximum number of return columns.</td>
</tr>
<tr>
<td>p_component_name</td>
<td>In case an error is returned, this is the name of the page item or report column used to display the error message.</td>
</tr>
<tr>
<td>p_display_column_no</td>
<td>Number of the column returned from the SQL statement. Must be within the p_min_columns though p_max_columns range.</td>
</tr>
<tr>
<td>p_search_column_no</td>
<td>Number of the column used to restrict the SQL statement. Must be within the p_min_columns though p_max_columns range.</td>
</tr>
<tr>
<td>p_search_value_list</td>
<td>Array of values to look up.</td>
</tr>
<tr>
<td>p_display_extra</td>
<td>If set to TRUE, and a value is not found, the search value is added to the result instead.</td>
</tr>
</tbody>
</table>

Return
Table 19–20 describes the return value by the GET_DISPLAY_DATA function signature 2.

Table 19–20  GET_DISPLAY_DATA Signature 2 Return

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wwv_flow_global.vc_arr2</td>
<td>List of VARCHAR2 indexed by pls_integer. For each entry in p_search_value_list the resulting array contains the value of the first record of the column specified by p_display_column_no in the same order as in p_search_value_list. If no record is found it contains the value of p_search_string if the parameter p_display_extra is set to TRUE. Otherwise the value is skipped.</td>
</tr>
</tbody>
</table>
Example

Looks up the values 7863, 7911 and 7988 and generates a HTML list with the value of the corresponding display column in the LOV query.

```plaintext
function render_list (   p_plugin              in apex_plugin.t_plugin,   p_item                in apex_plugin.t_page_item,   p_value               in varchar2,   p_is_readonly         in boolean,   p_is_printer_friendly in boolean )   return apex_plugin.t_page_item_render_result is   l_search_list wwv_flow_global.vc_arr2;   l_result_list wwv_flow_global.vc_arr2;
begin   l_search_list(1) := '7863';
   l_search_list(2) := '7911';
   l_search_list(3) := '7988';
   l_result_list :=
      apex_plugin_util.get_display_data (   p_sql_statement     => p_item.lov_definition,   p_min_columns       => 2,   p_max_columns       => 2,   p_component_name    => p_item.name,   p_search_column_no  => 1,   p_search_value_list => l_search_list );
   --
   sys.htp.p('<ul>');
   for i in 1 .. l_result_list.count loop
      sys.htp.p(     '<li>'||
         sys.htf.escape_sc(l_result_list(i))||
       '</li>');
   end loop;
   sys.htp.p('</ul>');
end render_list;
```
GET_ELEMENT_ATTRIBUTES Function

This function returns some of the standard attributes of an HTML element (for example, id, name, required, placeholder, aria-error-attributes, class) which is used if a HTML input/select/textarea/... tag is generated to get a consistent set of attributes.

Syntax

APEX_PLUGIN_UTIL.GET_ELEMENT_ATTRIBUTES (  
p_item           IN APEX_PLUGIN.T_PAGE_ITEM,  
p_name           IN VARCHAR2 DEFAULT NULL,  
p_default_class  IN VARCHAR2 DEFAULT NULL,  
p_add_id         in boolean  default true )  
return varchar2;

Parameters

Table 19–21 describes the available parameters for GET_ELEMENT_ATTRIBUTES function.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_item</td>
<td>This is the p_item parameter of your plug-in function.</td>
</tr>
<tr>
<td>p_name</td>
<td>This is the value which has been return by apex_plugin.get_input_name_or_page_item</td>
</tr>
<tr>
<td>p_default_class</td>
<td>Default CSS class which which should be contained in the result string.</td>
</tr>
<tr>
<td>p_add_id</td>
<td>If set to TRUE then the id attribute is also contained in the result string.</td>
</tr>
</tbody>
</table>

Example

This example emits an INPUT tag of type text which uses apex_plugin_util.get_element_attributes to automatically include the most common attributes.

sys.htp.prn (  
    '<input type="text" ' ||  
    apex_plugin_util.get_element_attributes(p_item, l_name, 'text_field') ' ||  
    'value="'||l_escaped_value||'" '||  
    'size="'||p_item.element_width||'" '||  
    'maxlength="'||p_item.element_max_length||'" '||  
    ' />'
);
GET_PLSQL_EXPRESSION_RESULT Function

This function executes a PL/SQL expression and returns a result. This function also performs the binding of any bind variables in the provided PL/SQL expression. This function is usually used for plug-in attributes of type PL/SQL Expression.

Syntax

APEX_PLUGIN_UTIL.GET_PLSQL_EXPRESSION_RESULT (  
  p_plsql_expression IN VARCHAR2)  
RETURN VARCHAR2;

Parameters

Table 19–22 describes the parameters available in the GET_PLSQL_EXPRESSION_RESULT function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_plsql_expression_result</td>
<td>A PL/SQL expression that returns a string.</td>
</tr>
</tbody>
</table>

Return

Table 19–23 describes the return value by the function GET_PLSQL_EXPRESSION_RESULT.

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2</td>
<td>String result value returned by the PL/SQL Expression.</td>
</tr>
</tbody>
</table>

Example

This example executes and returns the result of the PL/SQL expression which is specified in attribute_03 of an item type plug-in attribute of type "PL/SQL Expression".

```plaintext
l_result := apex_plugin_util.get_plsql_expression_result (  
  p_plsql_expression => p_item.attribute_03 );
```
GET_PSQL_FUNCTION_RESULT Function

This function executes a PL/SQL function block and returns the result. This function also performs binding of bind variables in the provided PL/SQL Function Body. This function is usually used for plug-in attributes of type PL/SQL Function Body.

Syntax
APEX_PLUGIN_UTIL.GET_PSQL_FUNCTION_RESULT (p_plsql_function IN VARCHAR2) RETURN VARCHAR2;

Parameters
Table 19–24 describes the parameters available in the GET_PSQL_FUNCTION_RESULT function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_plsql_function</td>
<td>A PL/SQL function block that returns a result of type string.</td>
</tr>
</tbody>
</table>

Return
Table 19–25 describes the return value by the function GET_PSQL_FUNCTION_RESULT.

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2</td>
<td>String result value returned by the PL/SQL function block.</td>
</tr>
</tbody>
</table>

Example
The following example executes and returns the result of the PL/SQL function body that is specified in attribute_03 of an item type plug-in attribute of type PL/SQL Function Body.

```plaintext
l_result := apex_plugin_util.get_plsql_function_result (p_plsql_function => p_item.attribute_03);
```
GET_POSITION_IN_LIST Function

This function returns the position in the list where p_value is stored. If it is not found, null is returned.

Syntax
APEX_PLUGIN_UTIL.GET_POSITION_IN_LIST(
    p_list IN wwv_flow_global.vc_arr2,
    p_value IN VARCHAR2)
RETURN NUMBER;

Parameters
Table 19–26 describes the parameters available in the GET_POSITION_IN_LIST function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_list</td>
<td>Array of type wwv_flow_global.vc_arr2 that contains entries of type VARCHAR2.</td>
</tr>
<tr>
<td>p_value</td>
<td>Value located in the p_list array.</td>
</tr>
</tbody>
</table>

Return
Table 19–27 describes the return value by the GET_POSITION_IN_LIST function.

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER</td>
<td>Returns the position of p_value in the array p_list. If it is not found NULL is returned.</td>
</tr>
</tbody>
</table>

Example
The following example searches for "New York" in the provided list and returns 2 into l_position.

```sql
declare
    l_list     wwv_flow_global.vc_arr2;
    l_position number;
begin
    l_list(1) := 'Rome';
    l_list(2) := 'New York';
    l_list(3) := 'Vienna';

    l_position := apex_plugin_util.get_position_in_list (
        p_list => l_list,
        p_value => 'New York' );
end;
```
GET_SEARCH_STRING Function

Based on the provided value in `p_search_type` the passed in value of `p_search_string` is returned unchanged or is converted to uppercase. Use this function with the `p_search_string` parameter of `get_data` and `get_data2`.

Syntax

```apl
APEX_PLUGIN_UTIL.GET_SEARCH_STRING(
    p_search_type IN VARCHAR2,
    p_search_string IN VARCHAR2)
RETURN VARCHAR2;
```

Parameters

Table 19–28 describes the parameters available in the GET_SEARCH_STRING function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_search_type</td>
<td>Type of search when used with <code>get_data</code> and <code>get_data2</code>. Use one of the c_search_* constants.</td>
</tr>
<tr>
<td>p_search_string</td>
<td>Search string used for the search with <code>get_data</code> and <code>get_data2</code>.</td>
</tr>
</tbody>
</table>

Return

Table 19–29 describes the return value by the function GET_SEARCH_STRING.

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2</td>
<td>Returns <code>p_search_string</code> unchanged or in uppercase if <code>p_search_type</code> is of type c_search_contains_ignore or c_search_exact_ignore.</td>
</tr>
</tbody>
</table>

Example

This example uses a call to `get_data` or `get_data2` to make sure the search string is using the correct case.

```apl
l_column_value_list :=
    apex_plugin_util.get_data (;
        p_sql_statement => p_item.lov_definition,
        p_min_columns => 2,
        p_max_columns => 2,
        p_component_name => p_item.name,
        p_search_type => apex_plugin_util.c_search_contains_ignore,
        p_search_column_no => 1,
        p_search_string => apex_plugin_util.get_search_string (;
            p_search_type => apex_plugin_util.c_search_contains_ignore,
            p_search_string => p_value ) );
```
IS_EQUAL Function

This function returns TRUE if both values are equal and FALSE if not. If both values are NULL, TRUE is returned.

Syntax
APEXPLUGIN_UTIL.IS_EQUAL (  
    p_value1 IN VARCHAR2  
    p_value2 IN VARCHAR2)  
RETURN BOOLEAN;

Parameters
Table 19–30 describes the parameters available in the IS_EQUAL function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_value1</td>
<td>First value to compare.</td>
</tr>
<tr>
<td>p_value2</td>
<td>Second value to compare.</td>
</tr>
</tbody>
</table>

Return
Table 19–31 describes the return value by the function IS_EQUAL.

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>Returns TRUE if both values are equal or both values are NULL, otherwise it returns FALSE.</td>
</tr>
</tbody>
</table>

Example
In the following example, if the value in the database is different from what is entered, the code in the if statement is executed.

```sql
if NOT apex_plugin_util.is_equal(l_database_value, l_current_value) then
    -- value has changed, do something
    null;
end if;
```
PAGE_ITEM_NAMES_TO_JQUERY Function

This function returns a jQuery selector based on a comma delimited string of page item names. For example, you could use this function for a plug-in attribute called "Page Items to Submit" where the JavaScript code has to read the values of the specified page items.

Syntax

APEX_PLUGIN_UTIL.PAGE_ITEM_NAMES_TO_JQUERY (p_page_item_names IN VARCHAR2)
RETURN VARCHAR2;

Parameters

Table 19–32 describes the parameters available in the PAGE_ITEM_NAMES_TO_JQUERY function.

Table 19–32 PAGE_ITEM_NAMES_TO_JQUERY Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_page_item_names</td>
<td>Comma delimited list of page item names.</td>
</tr>
</tbody>
</table>

Return

Table 19–33 describes the return value by the PAGE_ITEM_NAMES_TO_JQUERY function.

Table 19–33 PAGE_ITEM_NAMES_TO_JQUERY Return

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2</td>
<td>Transforms the page items specified in p_page_item_names into a jQuery selector.</td>
</tr>
</tbody>
</table>

Example

The following example shows the code to construct the initialization call for a JavaScript function called myOwnWidget. This function gets an object with several attributes where one attribute is pageItemsToSubmit which is expected to be a jQuery selector.

```javascript
apex_javascript.add_onload_code {
    p_code => 'myOwnWidget('||
    '"'||p_item.name||'"', ||
    '"'||apex_javascript.add_attribute('ajaxIdentifier', apex_plugin.get_ajax_identifier)||
    '"'||apex_javascript.add_attribute('dependingOnSelector', apex_plugin_util.page_item_names_to_jquery(p_item.lov_cascade_parent_items))||
    '"'||apex_javascript.add_attribute('optimizeRefresh', p_item.ajax_optimize_refresh)||
    '"'||apex_javascript.add_attribute('pageItemsToSubmit', apex_plugin_util.page_item_names_to_jquery(p_item.ajax_items_to_submit))||
    '"'||apex_javascript.add_attribute('nullValue', p_item.lov_null_value, false, false)||
    '});');
```
PRINT_DISPLAY_ONLY Procedure

This procedure outputs a SPAN tag for a display only field.

Syntax
APEX_PLUGIN_UTIL.PRINT_DISPLAY_ONLY (  
  p_item_name        IN VARCHAR2,  
  p_display_value    IN VARCHAR2,  
  p_show_line_breaks IN BOOLEAN,  
  p_attributes       IN VARCHAR2,  
  p_id_postfix       IN VARCHAR2 DEFAULT '_DISPLAY');

Parameters
Table 19–34 describes the parameters available in the PRINT_DISPLAY_ONLY procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_item_name</td>
<td>Name of the page item. This parameter should be called with p_item.name.</td>
</tr>
<tr>
<td>p_display_value</td>
<td>Text to be displayed.</td>
</tr>
<tr>
<td>p_show_line_breaks</td>
<td>If set to TRUE line breaks in p_display_value are changed to &lt;br /&gt; so that the browser renders them as line breaks.</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Additional attributes added to the SPAN tag.</td>
</tr>
<tr>
<td>p_id_postfix</td>
<td>Postfix which is getting added to the value in p_item_name to get the ID for the SPAN tag. Default is _DISPLAY.</td>
</tr>
</tbody>
</table>

Example
The following code could be used in an item type plug-in to render a display only page item.

```plaintext
apex_plugin_util.print_display_only (  
  p_item_name => p_item.name,  
  p_display_value => p_value,  
  p_show_line_breaks => false,  
  p_escape => true,  
  p_attributes => p_item.element_attributes );
```
PRINT_ESCAPED_VALUE Procedure

This procedure outputs the value in an escaped form and chunks big strings into smaller outputs.

Syntax
APEX_PLUGIN_UTIL.PRINT_ESCAPED_VALUE (    p_value   IN VARCHAR2);

Parameters
Table 19–35 describes the parameters available in the PRINT_ESCAPED_VALUE procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_value</td>
<td>Text which should be escaped and then printed to the HTTP buffer.</td>
</tr>
</tbody>
</table>

Example
Prints a hidden field with the current value of the page item.

```sql
sys.htp.prn('<input type="hidden" name="'|l_name||'" id="'|p_item_name||'" value="');
print_escaped_value(p_value);
sys.htp.prn('">');
```
PRINT_HIDDEN_IF_READONLY Procedure

This procedure outputs a hidden field to store the page item value if the page item is rendered as readonly and is not printer friendly. If this procedure is called in an item type plug-in, the parameters of the plug-in interface should directly be passed in.

Syntax
APEX_PLUGIN_UTIL.PRINT_HIDDEN_IF_READONLY (  
  p_item_name   IN VARCHAR2,  
  p_value       IN VARCHAR2,  
  p_is_readonly IN BOOLEAN,  
  p_is_printer_friendly IN BOOLEAN,  
  p_id_postfix  IN VARCHAR2 DEFAULT NULL);

Parameters
Table 19–36 describes the parameters available in the PRINT_HIDDEN_IF_READONLY procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_item_name</td>
<td>Name of the page item. For this parameter the p_item.name should be passed in.</td>
</tr>
<tr>
<td>p_value</td>
<td>Current value of the page item. For this parameter p_value should be passed in.</td>
</tr>
<tr>
<td>p_is_readonly</td>
<td>Is the item rendered readonly. For this parameter p_is_readonly should be passed in.</td>
</tr>
<tr>
<td>p_is_printer_friendly</td>
<td>Is the item rendered in printer friendly mode. For this parameter p_is_printer_friendly should be passed in.</td>
</tr>
<tr>
<td>p_id_postfix</td>
<td>Used to generate the ID attribute of the hidden field. It is build based on p_item_name and the value in p_id_postfix.</td>
</tr>
</tbody>
</table>

Example
Writes a hidden field with the current value to the HTTP output if p_is_readonly is TRUE and p_is_printer_friendly is FALSE.

```sql
APEX_PLUGIN_UTIL.PRINT_HIDDEN_IF_READONLY (  
  p_item_name => p_item.name,  
  p_value => p_value,  
  p_is_readonly => p_is_readonly,  
  p_is_printer_friendly => p_is_printer_friendly );
```
PRINT_JSON_HTTP_HEADER Procedure

This procedure outputs a standard HTTP header for a JSON output.

Syntax
APEX_PLUGIN_UTIL.PRINT_JSON_HTTP_HEADER;

Parameters
None.

Example
This example shows how to use this procedure in the AJAX callback function of a plugin. This code outputs a JSON structure in the following format:

```
[{
  "d":"Display 1","r":"Return 1"},
  {
    "d":"Display 2","r":"Return 2"
  }
]
```

-- Write header for the JSON stream.
apex_plugin_util.print_json_http_header;
-- initialize the JSON structure
sys.htp.p('[');
-- loop through the value array
for i in 1 .. l_values.count
loop
  -- add array entry
  sys.htp.p(' {
    case when i > 1 then ',' end||
    '{'||
    apex_javascript.add_attribute('d', sys.htf.escape_sc(l_values(i).display_value), false, true)||
    apex_javascript.add_attribute('r', sys.htf.escape_sc(l_values(i).return_value), false, false)||
    '}'
  });
end loop;
-- close the JSON structure
sys.htp.p(']');
PRINT_LOV_AS_JSON Procedure

PRINT_LOV_AS_JSON Procedure
This procedure outputs a JSON response based on the result of a two column LOV in
the format:
[{"d:"display","r":"return"},{"d":....,"r":....},....]

The HTTP header is initialized with MIME type
"application/json" as well.

Note:

Syntax
APEX_PLUGIN_UTIL.PRINT_LOV_AS_JSON (
p_sql_statement
IN VARCHAR2,
p_component_name
IN VARCHAR2,
p_escape
IN BOOLEAN,
p_replace_substitutions IN BOOLEAN DEFAULT FALSE);

Parameters
Table 19–37 describes the parameters available in the PRINT_LOV_AS_JSON
procedure.
Table 19–37

PRINT_LOV_AS_JSON Parameters

Parameter

Description

p_sql_statement

A SQL statement which returns two columns from the
SELECT.

p_component_name

The name of the page item or report column that is used
in case an error is displayed.

p_escape

If set to TRUE the value of the display column is escaped,
otherwise it is output as is.

p_replace_substitutions

If set to TRUE, apex_plugin_util.replace_
substitutions is called for the value of the display
column, otherwise, it is output as is.

Example
This example shows how to use the procedure in an AJAX callback function of an item
type plug-in. The following call writes the LOV result as a JSON array to the HTTP
output.
apex_plugin_util.print_lov_as_json (
p_sql_statement => p_item.lov_definition,
p_component_name => p_item.name,
p_escape
=> true );

19-34 Oracle Application Express API Reference


PRINT_OPTION Procedure

This procedure outputs an OPTION tag.

Syntax

APEX_PLUGIN_UTIL.PRINT_OPTION ( 
  p_display_value       IN VARCHAR2,  
  p_return_value        IN VARCHAR2,  
  p_is_selected         IN BOOLEAN,  
  p_attributes          IN VARCHAR2,  
  p_escape              IN BOOLEAN DEFAULT TRUE);  

Parameters

Table 19–38 describes the parameters available in the PRINT_OPTION procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_display_value</td>
<td>Text which is displayed by the option.</td>
</tr>
<tr>
<td>p_return_value</td>
<td>Value which is set when the option is picked.</td>
</tr>
<tr>
<td>p_is_selected</td>
<td>Set to TRUE if the selected attribute should be set for this option.</td>
</tr>
<tr>
<td>p_attributes</td>
<td>Additional HTML attributes which should be set for the OPTION tag.</td>
</tr>
<tr>
<td>p_escape</td>
<td>Set to TRUE if special characters in p_display_value should be escaped.</td>
</tr>
</tbody>
</table>

Example

The following example could be used in an item type plug-in to create a SELECT list. Use apex_plugin_util.is_equal to find out which list entry should be marked as current.

```sql
sys.htp.p('"<select id="'||p_item.name||'" size="'||nvl(p_item.element_height, 5)||"'" ||coalesce(p_item.element_attributes, 'class="new_select_list"')||'">');  
-- loop through the result and add list entries  
for i in 1 .. l_values.count  
  loop  
    apex_plugin_util.print_option (  
      p_display_value => l_values(i).display_value,  
      p_return_value  => l_values(i).return_value,  
      p_is_selected   => apex_plugin_util.is_equal(l_values(i).return_value, p_value),  
      p_attributes    => p_item.element_option_attributes,  
      p_escape        => true );  
  end loop;  
sys.htp.p('</select>');  
```
REPLACE_SUBSTITUTIONS Function

This function replaces any &ITEM. substitution references with their actual value. If p_escape is set to TRUE, any special characters contained in the value of the referenced item are escaped to prevent Cross-site scripting (XSS) attacks.

Syntax

```sql
apex_plugin_util.replace_substitutions(
p_value    in varchar2,
p_escape   in boolean default true)
return varchar2;
```

Parameters

**Table 19–39** replaces the parameters available in the REPLACE_SUBSTITUTION function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_value</td>
<td>This value is a string which can contain several &amp;ITEM. references which are replaced by their actual page item values.</td>
</tr>
<tr>
<td>p_escape</td>
<td>If set to TRUE any special characters contained in the value of the referenced item are escaped to prevent Cross-site scripting (XSS) attacks. If set to FALSE, the referenced items are not escaped.</td>
</tr>
</tbody>
</table>

Example

The following example replaces any substitution syntax references in the region plug-in attribute 05 with their actual values. Any special characters in the values are escaped.

```sql
l_advanced_formatting := apex_plugin_util.replace_substitutions(
p_value => p_region.attribute_05,
p_escape => true );
```
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:
- ADD_AD_COLUMN Procedure
- ADD_AD_SYNONYM Procedure
- DEL_AD_COLUMN Procedure
- DEL_AD_SYNONYM Procedure
- DEL_COLUMN Procedure
- DEL_GROUP Procedure
- DEL_TABLE Procedure
- SYNCH_TABLE Procedure
- UPD_AD_COLUMN Procedure
- UPD_AD_SYNONYM Procedure
- UPD_COLUMN Procedure
- UPD_DISPLAY_IN_FORM Procedure
- UPD_DISPLAY_IN_REPORT Procedure
- UPD_FORM_REGION_TITLE Procedure
- UPD_GROUP Procedure
- UPD_ITEM_DISPLAY_HEIGHT Procedure
- UPD_ITEMDISPLAY_WIDTH Procedure
- UPD_ITEM_FORMAT_MASK Procedure
- UPD_ITEM_HELP Procedure
- UPD_LABEL Procedure
- UPD_REPORT_ALIGNMENT Procedure
- UPD_REPORT_FORMAT_MASK Procedure
- UPD_REPORT_REGION_TITLE Procedure
- UPD_TABLE Procedure

See Also: "Managing User Interface Defaults" in Oracle Application Express SQL Workshop Guide
ADD_AD_COLUMN Procedure

Adds a User Interface Default Attribute Dictionary entry with the provided definition. Up to three synonyms can be provided during the creation. Additional synonyms can be added post-creation using apex_ui_default_update.add_ad_synonym. Synonyms share the column definition of their base column.

Syntax

APEX_UI_DEFAULT_UPDATE.ADD_AD_COLUMN (  
  p_column_name           IN  VARCHAR2,  
  p_label                 IN  VARCHAR2  DEFAULT NULL,  
  p_help_text             IN  VARCHAR2  DEFAULT NULL,  
  p_format_mask           IN  VARCHAR2  DEFAULT NULL,  
  p_default_value         IN  VARCHAR2  DEFAULT NULL,  
  p_form_format_mask      IN  VARCHAR2  DEFAULT NULL,  
  p_form_display_width    IN  VARCHAR2  DEFAULT NULL,  
  p_form_display_height   IN  VARCHAR2  DEFAULT NULL,  
  p_form_data_type        IN  VARCHAR2  DEFAULT NULL,  
  p_report_format_mask    IN  VARCHAR2  DEFAULT NULL,  
  p_report_col_alignment  IN  VARCHAR2  DEFAULT NULL,  
  p_syn_name1             IN  VARCHAR2  DEFAULT NULL,  
  p_syn_name2             IN  VARCHAR2  DEFAULT NULL,  
  p_syn_name3             IN  VARCHAR2  DEFAULT NULL);  

Parameters

Table 20–5 describes the parameters available in the ADD_AD_COLUMN procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_column_name</td>
<td>Name of column to be created.</td>
</tr>
<tr>
<td>p_label</td>
<td>Used for item label and report column heading.</td>
</tr>
<tr>
<td>p_help_text</td>
<td>Used for help text for items and interactive report columns</td>
</tr>
<tr>
<td>p_format_mask</td>
<td>Used as the format mask for items and report columns. Can be overwritten by report for form specific format masks.</td>
</tr>
<tr>
<td>p_default_value</td>
<td>Used as the default value for items.</td>
</tr>
<tr>
<td>p_form_format_mask</td>
<td>If provided, used as the format mask for items, overriding any value for the general format mask.</td>
</tr>
<tr>
<td>p_form_display_width</td>
<td>Used as the width of any items using this Attribute Definition.</td>
</tr>
<tr>
<td>p_form_display_height</td>
<td>Used as the height of any items using this Attribute Definition (only used by item types such as text areas and shuttles).</td>
</tr>
<tr>
<td>p_form_data_type</td>
<td>Used as the data type for items (results in an automatic validation). Valid values are VARCHAR, NUMBER and DATE.</td>
</tr>
<tr>
<td>p_report_format_mask</td>
<td>If provided, used as the format mask for report columns, overriding any value for the general format mask.</td>
</tr>
<tr>
<td>p_report_col_alignment</td>
<td>Used as the alignment for report column data (for example, number are usually right justified). Valid values are LEFT, CENTER, and RIGHT.</td>
</tr>
</tbody>
</table>
Example
The following example creates a new attribute to the UI Defaults Attribute Dictionary within the workspace associated with the current schema. It also creates a synonym for that attribute.

BEGIN
    apex_ui_default_update.add_ad_column {
        p_column_name => 'CREATED_BY',
        p_label => 'Created By',
        p_help_text => 'User that created the record.',
        p_form_display_width => 30,
        p_form_data_type => 'VARCHAR',
        p_report_col_alignment => 'LEFT',
        p_syn_name1 => 'CREATED_BY_USER',
    );
END;
ADD_AD_SYNONYM Procedure

If the column name is found within the User Interface Default Attribute Dictionary, the synonym provided is created and associated with that column. Synonyms share the column definition of their base column.

Syntax

APEX_UI_DEFAULT_UPDATE.ADD_AD_SYNONYM (  
  p_column_name           IN VARCHAR2,  
  p_syn_name              IN VARCHAR2);  

Parameters

Table 20–2 describes the parameters available in the ADD_AD_SYNONYM procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_column_name</td>
<td>Name of column with the Attribute Dictionary that the synonym is being created for.</td>
</tr>
<tr>
<td>p_syn_name</td>
<td>Name of synonym to be created.</td>
</tr>
</tbody>
</table>

Example

The following example add the synonym CREATED_BY_USER to the CREATED_BY attribute of the UI Defaults Attribute Dictionary within the workspace associated with the current schema.

```
BEGIN  
apex_ui_default_update.add_ad_synonym (  
  p_column_name => 'CREATED_BY',  
  p_syn_name    => 'CREATED_BY_USER' );  
END;
```
DEL_AD_COLUMN Procedure

If the column name is found within the User Interface Default Attribute Dictionary, the column, along with any associated synonyms, is deleted.

Syntax

APEX_UI_DEFAULT_UPDATE.DEL_AD_COLUMN (p_column_name           IN VARCHAR2);

Parameters

Table 20–3 describes the parameters available in the DEL_AD_COLUMN procedure.

Table 20–3  DEL_AD_COLUMN Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_column_name</td>
<td>Name of column to be deleted</td>
</tr>
</tbody>
</table>

Example

The following example deletes the attribute CREATED_BY from the UI Defaults Attribute Dictionary within the workspace associated with the current schema.

BEGIN
    apex_ui_default_update.del_ad_column (p_column_name => 'CREATED_BY');
END;
DEL_AD_SYNONYM Procedure

If the synonym name is found within the User Interface Default Attribute Dictionary, the synonym name is deleted.

Syntax
APEX_UI_DEFAULT_UPDATE.DEL_AD_SYNONYM (  
p_syn_name IN VARCHAR2);

Parameters
Table 20–4 describes the parameters available in the DEL_AD_SYNONYM procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_syn_name</td>
<td>Name of synonym to be deleted</td>
</tr>
</tbody>
</table>

Example
The following example deletes the synonym CREATED_BY_USER from the UI Defaults Attribute Dictionary within the workspace associated with the current schema.

```
BEGIN
    apex_ui_default_update.del_ad_synonym (  
        p_syn_name => 'CREATED_BY_USER' );
END;
```
DEL_COLUMN Procedure

If the provided table and column exists within the user’s schema’s table based User Interface Defaults, the UI Defaults for it are deleted.

Syntax

APEX_UI_DEFAULT_UPDATE.DEL_COLUMN (
    p_table_name            IN VARCHAR2,
    p_column_name           IN VARCHAR2);

Parameters

Table 20–5 describes the parameters available in the DEL_COLUMN procedure.

Table 20–5  DEL_COLUMN Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Name of table whose column’s UI Defaults are to be deleted.</td>
</tr>
<tr>
<td>p_column_name</td>
<td>Name of columns whose UI Defaults are to be deleted.</td>
</tr>
</tbody>
</table>

Example

The following example deletes the column CREATED_BY from the EMP table definition within the UI Defaults Table Dictionary within the current schema.

BEGIN
    apex_ui_default_update.del_column {
        p_table_name => ‘EMP’,
        p_column_name => ‘CREATED_BY’
    };
END;
DEL_GROUP Procedure

If the provided table and group exists within the user’s schema’s table based User Interface Defaults, the UI Defaults for it are deleted and any column within the table that references that group has the group_id set to null.

Syntax
APEX_UI_DEFAULT_UPDATE.DEL_GROUP (  
    p_table_name            IN VARCHAR2,  
    p_group_name            IN VARCHAR2);  

Parameters
Table 20–6 describes the parameters available in the DEL_GROUP procedure.

Table 20–6 DEL_GROUP Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Name of table whose group UI Defaults are to be deleted</td>
</tr>
<tr>
<td>p_group_name</td>
<td>Name of group whose UI Defaults are to be deleted</td>
</tr>
</tbody>
</table>

Example
The following example deletes the group AUDIT_INFO from the EMP table definition within the UI Defaults Table Dictionary within the current schema.

BEGIN  
    apex_ui_default_update.del_group (  
        p_table_name => 'EMP',  
        p_group_name => 'AUDIT_INFO' );  
END;
DEL_TABLE Procedure

If the provided table exists within the user's schema's table based User Interface Defaults, the UI Defaults for it is deleted. This includes the deletion of any groups defined for the table and all the columns associated with the table.

Syntax

APEX_UI_DEFAULT_UPDATE.DEL_TABLE (  
    p_table_name               IN VARCHAR2);  

Parameters

Table 20–7 describes the parameters available in the DEL_TABLE procedure.

Table 20–7 DEL_TABLE Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Table name</td>
</tr>
</tbody>
</table>

Example

The following example removes the UI Defaults for the EMP table that are associated with the current schema.

begin  
    apex_ui_default_update.del_table (  
        p_table_name => 'EMP' );  
end;  
/
SYNCH_TABLE Procedure

If the Table Based User Interface Defaults for the table do not already exist within the user’s schema, they are defaulted. If they do exist, they are synchronized, meaning, the columns in the table is matched against the column in the UI Defaults Table Definitions. Additions and deletions are used to make them match.

Syntax
APEX_UI_DEFAULT_UPDATE.SYNCH_TABLE (
   p_table_name              IN VARCHAR2);

Parameters
Table 20–8 describes the parameters available in the SYNCH_TABLE procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Table name</td>
</tr>
</tbody>
</table>

Example
The following example synchronizes the UI Defaults for the EMP table that are associated with the current schema.

BEGIN
   apex_ui_default_update.synch_table (
      p_table_name => 'EMP');
END;
If the column name is found within the User Interface Default Attribute Dictionary, the column entry is updated using the provided parameters. If ‘null%’ is passed in, the value of the associated parameter is set to null.

Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_AD_COLUMN (  
p_column_name           IN  VARCHAR2,  
p_new_column_name       IN  VARCHAR2  DEFAULT NULL,  
p_label                 IN  VARCHAR2  DEFAULT NULL,  
p_help_text             IN  VARCHAR2  DEFAULT NULL,  
p_format_mask           IN  VARCHAR2  DEFAULT NULL,  
p_default_value         IN  VARCHAR2  DEFAULT NULL,  
p_form_format_mask      IN  VARCHAR2  DEFAULT NULL,  
p_form_display_width    IN  VARCHAR2  DEFAULT NULL,  
p_form_display_height   IN  VARCHAR2  DEFAULT NULL,  
p_form_data_type        IN  VARCHAR2  DEFAULT NULL,  
p_report_format_mask    IN  VARCHAR2  DEFAULT NULL,  
p_report_col_alignment  IN  VARCHAR2  DEFAULT NULL);  
```

Parameters

Table 20–9 describes the parameters available in the UPD_AD_COLUMN procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_column_name</td>
<td>Name of column to be updated</td>
</tr>
<tr>
<td>p_new_column_name</td>
<td>New name for column, if column is being renamed</td>
</tr>
<tr>
<td>p_label</td>
<td>Used for item label and report column heading</td>
</tr>
<tr>
<td>p_help_text</td>
<td>Used for help text for items and interactive report columns</td>
</tr>
<tr>
<td>p_format_mask</td>
<td>Used as the format mask for items and report columns. Can be overwritten by report for form specific format masks.</td>
</tr>
<tr>
<td>p_default_value</td>
<td>Used as the default value for items.</td>
</tr>
<tr>
<td>p_form_format_mask</td>
<td>If provided, used as the format mask for items, overriding any value for the general format mask.</td>
</tr>
<tr>
<td>p_form_display_width</td>
<td>Used as the width of any items using this Attribute Definition.</td>
</tr>
<tr>
<td>p_form_display_height</td>
<td>Used as the height of any items using this Attribute Definition (only used by item types such as text areas and shuttles).</td>
</tr>
<tr>
<td>p_form_data_type</td>
<td>Used as the data type for items (results in an automatic validation). Valid values are VARCHAR, NUMBER and DATE.</td>
</tr>
<tr>
<td>p_report_format_mask</td>
<td>If provided, used as the format mask for report columns, overriding any value for the general format mask.</td>
</tr>
<tr>
<td>p_report_col_alignment</td>
<td>Used as the alignment for report column data (for example, number are usually right justified). Valid values are LEFT, CENTER, and RIGHT.</td>
</tr>
</tbody>
</table>
**Note:** If p_label through p_report_col_alignment are set to 'null%', the value is nullified. If no value is passed in, that column is not updated.

**Example**

The following example updates the CREATED_BY column in the UI Defaults Attribute Dictionary within the workspace associated with the current schema, setting the form_format_mask to null.

```sql
BEGIN
    apex_ui_default_update.upd_ad_column (p_column_name => 'CREATED_BY',
                                           p_form_format_mask => 'null%');
END;
```
UPD_AD_SYNONYM Procedure

If the synonym name is found within the User Interface Default Attribute Dictionary, the synonym name is updated.

Syntax

APEX_UI_DEFAULT_UPDATE.UPD_AD_SYNONYM {
  p_syn_name           IN VARCHAR2,
  p_new_syn_name       IN VARCHAR2 DEFAULT NULL);

Parameters

Table 20–10 describes the parameters available in the UPD_AD_SYNONYM procedure.

Table 20–10  UPD_AD_SYNONYM Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_syn_name</td>
<td>Name of synonym to be updated</td>
</tr>
<tr>
<td>p_new_syn_name</td>
<td>New name for synonym</td>
</tr>
</tbody>
</table>

Example

The following example updates the CREATED_BY_USER synonym in the UI Defaults Attribute Dictionary within the workspace associated with the current schema.

BEGIN
  apex_ui_default_update.upd_ad_synonym {
    p_syn_name => 'CREATED_BY_USER',
    p_new_syn_name => 'USER_CREATED_BY'};
END;
UPD_COLUMN Procedure

If the provided table and column exists within the user's schema's table based User Interface Defaults, the provided parameters are updated. If 'null%' is passed in, the value of the associated parameter is set to null.

Syntax
APEX_UI_DEFAULT_UPDATE.UPD_COLUMN (  
    p_table_name            IN VARCHAR2,  
    p_column_name           IN VARCHAR2,  
    p_group_id              IN VARCHAR2  DEFAULT NULL,  
    p_label                 IN VARCHAR2  DEFAULT NULL,  
    p_help_text             IN VARCHAR2  DEFAULT NULL,  
    p_display_in_form       IN VARCHAR2  DEFAULT NULL,  
    p_display_seq_form      IN VARCHAR2  DEFAULT NULL,  
    p_mask_form             IN VARCHAR2  DEFAULT NULL,  
    p_default_value         IN VARCHAR2  DEFAULT NULL,  
    p_required              IN VARCHAR2  DEFAULT NULL,  
    p_display_width         IN VARCHAR2  DEFAULT NULL,  
    p_max_width             IN VARCHAR2  DEFAULT NULL,  
    p_height                IN VARCHAR2  DEFAULT NULL,  
    p_display_in_report     IN VARCHAR2  DEFAULT NULL,  
    p_display_seq_report    IN VARCHAR2  DEFAULT NULL,  
    p_mask_report           IN VARCHAR2  DEFAULT NULL,  
    p_alignment             IN VARCHAR2  DEFAULT NULL);  

Parameters
Table 20–11 describes the parameters available in the UPD_COLUMN procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Name of table whose column's UI Defaults are being updated</td>
</tr>
<tr>
<td>p_column_name</td>
<td>Name of column whose UI Defaults are being updated</td>
</tr>
<tr>
<td>p_group_id</td>
<td>id of group to be associated with the column</td>
</tr>
<tr>
<td>p_label</td>
<td>When creating a form against this table or view, this is used as the label for the item if this column is included. When creating a report or tabular form, this is used as the column heading if this column is included.</td>
</tr>
<tr>
<td>p_help_text</td>
<td>When creating a form against this table or view, this becomes the help text for the resulting item.</td>
</tr>
<tr>
<td>p_display_in_form</td>
<td>When creating a form against this table or view, this determines whether this column is displayed in the resulting form page. Valid values are Y and N.</td>
</tr>
<tr>
<td>p_display_seq_form</td>
<td>When creating a form against this table or view, this determines the sequence in which the columns is displayed in the resulting form page.</td>
</tr>
<tr>
<td>p_mask_form</td>
<td>When creating a form against this table or view, this specifies the mask that is applied to the item, such as 999-99-9999. This is not used for character based items.</td>
</tr>
<tr>
<td>p_default_value</td>
<td>When creating a form against this table or view, this specifies the default value for the item resulting from this column.</td>
</tr>
</tbody>
</table>
The following example updates the column `DEPT_NO` within the `EMP` table definition within the UI Defaults Table Dictionary within the current schema, setting the `group_id` to null.

```sql
BEGIN
    apex_ui_default_update.upd_column (
        p_table_name => 'EMP',
        p_column_name => 'DEPT_NO',
        p_group_id => 'null%');
END;
```

### Table 20-11 (Cont.) UPD_COLUMN Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_required</td>
<td>When creating a form against this table or view, this specifies to generate a validation in which the resulting item must be NOT NULL. Valid values are Y and N.</td>
</tr>
<tr>
<td>p_display_width</td>
<td>When creating a form against this table or view, this specifies the display width of the item resulting from this column.</td>
</tr>
<tr>
<td>p_max_width</td>
<td>When creating a form against this table or view, this specifies the maximum string length that a user is allowed to enter in the item resulting from this column.</td>
</tr>
<tr>
<td>p_height</td>
<td>When creating a form against this table or view, this specifies the display height of the item resulting from this column.</td>
</tr>
<tr>
<td>p_display_in_report</td>
<td>When creating a report against this table or view, this determines whether this column is displayed in the resulting report. Valid values are Y and N.</td>
</tr>
<tr>
<td>p_display_seq_report</td>
<td>When creating a report against this table or view, this determines the sequence in which the columns are displayed in the resulting report.</td>
</tr>
<tr>
<td>p_mask_report</td>
<td>When creating a report against this table or view, this specifies the mask that is applied against the data, such as 999-99-9999. This is not used for character based items.</td>
</tr>
<tr>
<td>p_alignment</td>
<td>When creating a report against this table or view, this determines the alignment for the resulting report column. Valid values are L for Left, C for Center, and R for Right.</td>
</tr>
</tbody>
</table>

**Note:** If `p_group_id` through `p_alignment` are set to 'null%', the value is nullified. If no value is passed in, that column is not updated.

**Example**
The following example updates the column `DEPT_NO` within the `EMP` table definition within the UI Defaults Table Dictionary within the current schema, setting the `group_id` to null.
UPD_DISPLAY_IN_FORM Procedure

The UPD_DISPLAY_IN_FORM procedure sets the display in form user interface defaults. This user interface default is used by wizards when you select to create a form based upon the table. It controls whether the column is included by default or not.

Syntax

APEX_UI_DEFAULT_UPDATE.UPD_DISPLAY_IN_FORM (  
    p_table_name           IN VARCHAR2,  
    p_column_name           IN VARCHAR2,  
    p_display_in_form       IN VARCHAR2);  

Parameters

Table 20–12 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 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 defaults 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 is used by wizards when you select to create a report based upon the table and controls whether the column is included by default or not.

**Syntax**

```sql
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 20–13 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 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 defaults to 'No'.

```sql
APEX_UI_DEFAULT_UPDATE.UPD_DISPLAY_IN_REPORT(
    p_table_name => 'DEPT',
    p_column_name => 'DEPTNO',
    p_display_in_report => 'N');
```
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 20–14 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><code>p_table_name</code></td>
<td>Table name</td>
</tr>
<tr>
<td><code>p_form_region_title</code></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_GROUP Procedure

If the provided table and group exist within the user's schema's table based User Interface Defaults, the group name, description and display sequence of the group are updated. If 'null%' is passed in for p_description or p_display_sequence, the value is set to null.

Syntax

APEX_UI_DEFAULT_UPDATE.UPD_GROUP (  
  p_table_name            IN VARCHAR2,  
  p_group_name            IN VARCHAR2,  
  p_new_group_name        IN VARCHAR2 DEFAULT NULL,  
  p_description           IN VARCHAR2 DEFAULT NULL,  
  p_display_sequence      IN VARCHAR2 DEFAULT NULL);  

Parameters

Table 20–15 describes the parameters available in the UPD_GROUP procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Name of table whose group is being updated</td>
</tr>
<tr>
<td>p_group_name</td>
<td>Group being updated</td>
</tr>
<tr>
<td>p_new_group_name</td>
<td>New name for group, if group is being renamed</td>
</tr>
<tr>
<td>p_description</td>
<td>Description of group</td>
</tr>
<tr>
<td>p_display_sequence</td>
<td>Display sequence of group.</td>
</tr>
</tbody>
</table>

Note: If p_description or p_display_sequence are set to 'null%', the value is nullified. If no value is passed in, that column is not updated.

Example

The following example updates the description of the group AUDIT_INFO within the EMP table definition within the UI Defaults Table Dictionary within the current schema.

BEGIN  
apex_ui_default_update.upd_group (  
  p_table_name => 'EMP',  
  p_group_name => 'AUDIT_INFO',  
  p_description => 'Audit columns' );  
END;
UPD_ITEM_DISPLAY_HEIGHT Procedure

The UPD_ITEM_DISPLAY_HEIGHT procedure sets the item display height user interface default. This user interface default is used by wizards when you select to create a form based upon the table and include the specified column. Display height controls if the item is a text box or a text area.

Syntax
APEX_UI_DEFAULT_UPDATE.UPD_ITEM_DISPLAY_HEIGHT (  
  p_table_name            IN VARCHAR2,  
  p_column_name           IN VARCHAR2,  
  p_display_height        IN NUMBER);

Parameters
Table 20–16 describes the parameters available in the UPD_ITEM_DISPLAY_HEIGHT 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_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_ITEM_DISPLAY_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 is 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>
<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 is used by wizards when you select to create a form based upon the table and include the specified column. Item format mask is typically used to format numbers and dates.

**Syntax**

```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 20–18 describes the parameters available in the **UPD_ITEM_FORMAT_MASK** procedure.

**Table 20–18  **UPD_ITEM_FORMAT_MASK 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_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 is 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 20–19 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 is used when you create a form or report based on the specified table and include a specific column.

### Syntax

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

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p_table_name</code></td>
<td>Table name</td>
</tr>
<tr>
<td><code>p_column_name</code></td>
<td>Column name</td>
</tr>
<tr>
<td><code>p_label</code></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.

```sql
APEX_UI_DEFAULT_UPDATE.UPD_LABEL(  
  p_table_name => 'DEPT',  
  p_column_name => 'DEPTNO',  
  p_label => 'Department Number');
```
The UPD_REPORT_ALIGNMENT procedure sets the report alignment user interface default. This user interface default is used by wizards when you select to create a report based upon the table and include the specified column and determines if the report column should be left, center, or right justified.

Syntax

```sql
APEX_UI_DEFAULT_UPDATE.UPD_REPORT_ALIGNMENT (  
p_table_name            IN VARCHAR2,  
p_column_name           IN VARCHAR2,  
p_report_alignment      IN VARCHAR2);
```

Parameters

Table 20–21 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.

```sql
APEX_UI_DEFAULT_UPDATE.UPD_REPORT_ALIGNMENT (  
p_table_name => 'DEPT',  
p_column_name => 'DEPTNO',  
p_report_alignment => 'R');
```
The `UPD_REPORT_FORMAT_MASK` procedure sets the report format mask user interface default. This user interface default is used by wizards when you select to create a report based upon the table and include the specified column. Report format mask is typically used to format numbers and dates.

**Syntax**

```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 20–22 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><code>p_table_name</code></td>
<td>Table name</td>
</tr>
<tr>
<td><code>p_column_name</code></td>
<td>Column name</td>
</tr>
<tr>
<td><code>p_format_mask</code></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 20–23 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 => 'Departments');
```
UPD_TABLE Procedure

If the provided table exists within the user's schema's table based User Interface Defaults, the form region title and report region title are updated to match those provided. If `null%` is passed in for `p_form_region_title` or `p_report_region_title`, the value is set to null.

Syntax

```
APEX_UI_DEFAULT_UPDATE.UPD_TABLE (  
    p_table_name            IN VARCHAR2,  
    p_form_region_title     IN VARCHAR2 DEFAULT NULL,  
    p_report_region_title   IN VARCHAR2 DEFAULT NULL);  
```

Parameters

Table 20-24 describes the parameters available in the UPD_TABLE procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_table_name</td>
<td>Name of table being updated.</td>
</tr>
<tr>
<td>p_form_region_title</td>
<td>Region title used for forms.</td>
</tr>
<tr>
<td>p_report_region_title</td>
<td>Region title used for reports and tabular forms.</td>
</tr>
</tbody>
</table>

Note: if `null%` is passed in for `p_form_region_title` or `p_report_region_title`, the value is set to null. If no value is passed in, that column is not updated.

Example

The following example updates the EMP table definition within the UI Defaults Table Dictionary within the current schema.

```
bEGIN
    apex_ui_default_update.upd_table (
    p_table_name => 'EMP',
    p_form_region_title => 'Employee Details',
    p_report_region_title => 'Employees');
END;
/```
The APEX_UTIL package provides utilities you can use when programming in the Oracle Application Express environment. You can use the APEX_UTIL package to get and set session state, get files, check authorizations for users, reset different states for users, get and purge cache information and also to get and set preferences for users.

Topics:
- 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
- CUSTOMCALENDAR Procedure
- CREATE_USER_GROUP Procedure Signature 1
- CREATE_USER_GROUP Procedure Signature 2
- DOWNLOAD_PRINT_DOCUMENT Procedure Signature 1
- DOWNLOAD_PRINT_DOCUMENT Procedure Signature 2
- DOWNLOAD_PRINT_DOCUMENT Procedure Signature 3
- DOWNLOAD_PRINT_DOCUMENT Procedure Signature 4
- EDIT_USER Procedure
- END_USER_ACCOUNT_DAYS_LEFT Function
- EXPIRE_END_USER_ACCOUNT Procedure
- EXPIRE_WORKSPACE_ACCOUNT Procedure
- EXPORT_USERS Procedure
- FETCH_APP_ITEM Function
- FETCH_USER Procedure Signature 1
- FETCH_USER Procedure Signature 2
- FETCH_USER Procedure Signature 3
- FIND_SECURITY_GROUP_ID Function
- FIND_WORKSPACE Function
- GET_ACCOUNT_LOCKED_STATUS Function
- GET_ATTRIBUTE Function
- GET_AUTHENTICATION_RESULT Function
- GET_BLOB_FILE_SRC Function
- GET_BUILD_OPTION_STATUS Function Signature 1
- GET_BUILD_OPTION_STATUS Function Signature 2
- GET_CURRENT_USER_ID Function
- GET_DEFAULT_SCHEMA Function
- GET_EDITION Function
- GET_EMAIL Function
- GET_FEEDBACK_FOLLOW_UP Function
- GET_FILE Procedure
- GET_FILE_ID Function
- GET_FIRST_NAME Function
- GET_GROUPS_USER_BELONGS_TO Function
- GET_GROUP_ID Function
- GET_GROUP_NAME Function
- GET_HIGH_CONTRAST_MODE_TOGGLE Function
- GET_LAST_NAME Function
- GET_NUMERIC_SESSION_STATE Function
- GET_PREFERENCE Function
- GET_PRINT_DOCUMENT Function Signature 1
- GET_PRINT_DOCUMENT Function Signature 2
- GET_PRINT_DOCUMENT Function Signature 3
- GET_PRINT_DOCUMENT Function Signature 4
- GET_SCREEN_READER_MODE_TOGGLE Function
- GET_SESSION_LANG Function
- GET_SESSION_STATE Function
- GET_SESSION_TERRITORY Function
- GET_SESSION_TIME_ZONE Function
- GET_USER_ID Function
- GET_USER_ROLES Function
- GET_USERNAME Function
- HOST_URL Function
- HTML_PCT_GRAPH_MASK Function
- INCREMENT_CALENDAR Procedure
- IR_CLEAR Procedure
- IR_DELETE_REPORT Procedure
- IR_DELETE_SUBSCRIPTION Procedure
- IR_FILTER Procedure
- IR_RESET Procedure
- IS_HIGH_CONTRAST_SESSION Function
- IS_HIGH_CONTRAST_SESSION_YN Function
- IS_LOGIN_PASSWORD_VALID Function
- IS_SCREEN_READER_SESSION Function
- IS_SCREEN_READER_SESSION_YN Function
- IS_USERNAME_UNIQUE Function
- KEYVAL_NUM Function
- KEYVAL_VC2 Function
- LOCK_ACCOUNT Procedure
- PASSWORD_FIRST_USE_OCCURRED Function
- PREPARE_URL Function
- PUBLIC_CHECK_AUTHORIZATION Function
- PURGE_REGIONS_BY_APP Procedure
- PURGE_REGIONS_BY_NAME Procedure
- PURGE_REGIONS_BY_PAGE Procedure
- REDIRECT_URL Procedure
- REMOVE_PREFERENCE Procedure
- REMOVE_SORT_PREFERENCES Procedure
- REMOVE_USER Procedure
- RESET_AUTHORIZATIONS Procedure
- RESET_PW Procedure
- SAVEKEY_NUM Function
- SAVEKEY_VC2 Function
- SET_ATTRIBUTE Procedure
- SET_AUTHENTICATION_RESULT Procedure
SET_BUILD_OPTION_STATUS Procedure
SET_CUSTOM_AUTH_STATUS Procedure
SET_EDITION Procedure
SET_EMAIL Procedure
SET_FIRST_NAME Procedure
SET_LAST_NAME Procedure
SET_PREFERENCE Procedure
SET_SECURITY_GROUP_ID Procedure
SET_SESSION_HIGH_CONTRAST_OFF Procedure
SET_SESSION_HIGH_CONTRAST_ON Procedure
SET_SESSION_LANG Procedure
SET_SESSION_LIFETIME_SECONDS Procedure
SET_SESSION_MAX_IDLE_SECONDS Procedure
SET_SESSION_SCREEN_READER_OFF Procedure
SET_SESSION_SCREEN_READER_ON Procedure
SET_SESSION_STATE Procedure
SET_SESSION_TERRITORY Procedure
SET_SESSION_TIME_ZONE Procedure
SET_USERNAME Procedure
SHOW_HIGH_CONTRAST_MODE_TOGGLE Procedure
SHOW_SCREEN_READER_MODE_TOGGLE Procedure
STRING_TO_TABLE Function
SHOW_SCREEN_READER_MODE_TOGGLE Procedure
STRONG_PASSWORD_VALIDATION Function
SUBMIT_FEEDBACK Procedure
SUBMIT_FEEDBACK_FOLLOWUP Procedure
TABLE_TO_STRING Function
UNEXPIRE_END_USER_ACCOUNT Procedure
UNEXPIRE_WORKSPACE_ACCOUNT Procedure
UNLOCK_ACCOUNT Procedure
URL_ENCODE Function
WORKSPACE_ACCOUNT_DAYS_LEFT Function
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 21–1 describes the parameters available in the CACHE_GET_DATE_OF_PAGE_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>
</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 21–2 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 21–3 describes the parameters available in the CACHE_PURGE_BY_APPLICATION 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_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 is also purged.

Syntax

APEX_UTIL.CACHE_PURGE_BY_PAGE (  
p_application  IN    NUMBER,  
p_page         IN    NUMBER,  
p_user_name    IN    VARCHAR2 DEFAULT NULL);

Parameters

Table 21–4 describes the parameters available in the CACHE_PURGE_BY_PAGE procedure.

Table 21–4  CACHE_PURGE_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 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 users cache (only relevant if the cache is set to be by user).

BEGIN

APEX_UTIL.CACHE_PURGE_BY_PAGE(  
    p_application => :APP_ID,  
    p_page => 9);  

END;
This procedure deletes all cached pages and regions for a specified application that have passed the defined active time period. When you cache a page or region, you specify an active time period (or Cache Timeout). Once that period has passed, the cache is no longer used, thus removing those unusable pages or regions from the cache.

Syntax

APEX_UTIL.CACHE_PURGE_STALE(
    p_application IN NUMBER);

Parameters

Table 21–5 describes the parameters available in the CACHE_PURGE_STALE procedure.

Table 21–5  CACHE_PURGE_STALE 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>
</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 21–6 describes the parameters available in the CHANGE_CURRENT_USER_PW procedure.

Table 21–6  CHANGE_CURRENT_USER_PW Parameters

<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 21-116
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 21–7 describes the parameters available in the CHANGE_PASSWORD_ON_FIRST_USE function.

Table 21–7 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 21-104
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 21–8 describes the parameters available in the CLEAR_APP_CACHE procedure.

<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 is 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 21–9 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 is 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, in place of APEX_UTIL.COUNT_CLICK.

Syntax

APEX_UTIL.COUNT_CLICK (   p_url         IN    VARCHAR2,   p_cat         IN    VARCHAR2,   p_id          IN    VARCHAR2    DEFAULT NULL,   p_user        IN    VARCHAR2    DEFAULT NULL,   p_workspace   IN    VARCHAR2    DEFAULT NULL);

Parameters

Table 21–10 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 by using the APEX_WORKSPACE_CLICKS view and in the reports on this view available to workspace and site administrators.

```plsql
DECLARE
    l_url VARCHAR2(255);
    l_cat VARCHAR2(30);
    l_workspace_id VARCHAR2(30);
BEGIN
    l_url := 'http://yahoo.com';
    l_cat := 'yahoo';
    l_workspace_id := TO_CHAR(APEX_UTIL.FIND_SECURITY_GROUP_ID('MY_WORKSPACE'));
    HTP.P('<a href=APEX_UTIL.COUNT_CLICK?p_url=' || l_url || '&p_cat=' || l_cat || '&p_workspace=' || l_workspace_id || '>Click</a>');
END;
```

See Also:  "FIND_SECURITY_GROUP_ID Function" on page 21-48 in this document and "Purging the External Click Count Log" in Oracle Application Express Administration Guide, Defining Authorized URLs in Oracle Application Express Administration Guide.
CREATE_USER Procedure

This procedure creates a new account record in the Application Express user account table. To execute this procedure, the current user must have administrative privileges.

Syntax

```
APEX_UTIL.CREATE_USER(
    p_user_id                       IN      NUMBER      DEFAULT NULL,
    p_user_name                     IN      VARCHAR2,
    p_first_name                    IN      VARCHAR2    DEFAULT NULL,
    p_last_name                     IN      VARCHAR2    DEFAULT NULL,
    p_description                   IN      VARCHAR2    DEFAULT NULL,
    p_email_address                 IN      VARCHAR2    DEFAULT NULL,
    p_web_password                  IN      VARCHAR2,
    p_web_password_format           IN      VARCHAR2    DEFAULT 'CLEAR_TEXT',
    p_group_ids                     IN      VARCHAR2    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,
    p_allow_app_building_yn         IN      VARCHAR2    DEFAULT NULL,
    p_allow_sql_workshop_yn         IN      VARCHAR2    DEFAULT NULL,
    p_allow_websheet_dev_yn         IN      VARCHAR2    DEFAULT NULL,
    p_allow_team_development_yn     IN      VARCHAR2    DEFAULT NULL);
```

Parameters

Table 21–11 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>
</tbody>
</table>
Table 21–11 (Cont.) CREATE_USER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
| p_developer_privs           | Colon separated list of developer privileges. If p_developer_privs is not null, the user is given access to Team Development. If p_developer_privs contains ADMIN, the user is given Application Builder and SQL Workshop access. If p_developer_privs does not contain ADMIN but contains EDIT, the user is given Application Builder Access. If p_developer_privs does not contain ADMIN but contains SQL, the user is given SQL Workshop access. The following are acceptable values for this parameter:  
null - To create an end user (a user who can only authenticate to developed applications).  
CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL - To create a user with developer privileges with access to Application Builder and SQL Workshop.  
ADMIN:CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL - To create a user with full workspace administrator and developer privileges with access to Application Builder, SQL Workshop and Team Development.  
Note: Currently this parameter is named inconsistently between the CREATE_USER, EDIT_USER and FETCH_USER APIs, although they all relate to the DEVELOPER_ROLE field stored in the named user account record. CREATE_USER uses p_developer_privs, EDIT_USER uses p_developer_roles and FETCH_USER uses p_developer_role. |
| p_default_schema            | A database schema assigned to the user's workspace, used by default for browsing.                                                             |
| p_allow_access_to_schemas   | Colon separated list of schemas assigned to the user’s workspace to which the user is restricted (leave null for all).                           |
| p_account_expiry            | Date password was last updated, which defaults to today's date on creation.                                                                     |
| p_account_locked            | 'Y' or 'N' indicating if account is locked or unlocked.                                                                                          |
| p_failed_access_attempts    | Number of consecutive login failures that have occurred, defaults to 0 on creation.                                                             |
| p_change_password_on_first_use | 'Y' or 'N' to indicate whether password must be changed on first use, defaults to 'Y' on creation.                                              |
| p_first_password_use_occurred | 'Y' or 'N' to indicate whether login has occurred since password change, defaults to 'N' on creation.                                             |
| p_attribute_01              | Arbitrary text accessible with an API                                                                                                        |
| ...                         |                                                                                                                                             |
| p_attribute_10              |                                                                                                                                             |
**CREATE_USER Procedure**

**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.

```sql
BEGIN
APEX_UTIL.CREATE_USER(
    p_user_name    => 'NEWUSER1',
    p_web_password => 'secret99');
END;
```

**Example 2**
The following example creates a 'Workspace Administrator' called 'NEWUSER2'. Where the user 'NEWUSER2':

- Has full workspace administration and developer privilege (`p_developer_privs` parameter set to 'ADMIN:CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL').
- Has access to 2 schemas, both their browsing default 'MY_SCHEMA' (`p_default_schema` parameter set to 'MY_SCHEMA') and also 'MY_SCHEMA2' (`p_allow_access_to_schemas` parameter set to 'MY_SCHEMA2').
- Does not have to change their password when they first login (`p_change_password_on_first_use` parameter set to 'N').
- Has their phone number stored in the first additional attribute (`p_attribute_01` parameter set to '123 456 7890').

```sql
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;
```

### Table 21–11 (Cont.) CREATE_USER Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p_allow_app_building_yn</code></td>
<td>'Y' or 'N' to indicate whether access is allowed to Application Builder.</td>
</tr>
<tr>
<td><code>p_allow_sql_workshop_yn</code></td>
<td>'Y' or 'N' to indicate whether access is allowed to SQL Workshop.</td>
</tr>
<tr>
<td><code>p_allow_websheet_dev_yn</code></td>
<td>'Y' or 'N' to indicate whether access is allowed to Websheet development.</td>
</tr>
<tr>
<td><code>p_allow_team_development_yn</code></td>
<td>'Y' or 'N' to indicate whether access is allowed to Team Development.</td>
</tr>
</tbody>
</table>
See Also: "FETCH_USER Procedure Signature 3" on page 21-45, "EDIT_USER Procedure" on page 21-31, and "GET_GROUP_ID Function" on page 21-67
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 21–12 describes the parameters available in the CREATE_USER_GROUP procedure.

<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 assigns PK
        p_group_name => 'Managers',
        p_security_group_id => null,  -- defaults to current workspace ID
        p_group_desc => 'text'};
END;

CURRENT_USER_IN_GROUP Function

This function returns a Boolean result based on whether the current user is a member of the specified group. You can use the group name or group ID to identify the group.

Syntax

APEX_UTIL.CURRENT_USER_IN_GROUP(
    p_group_name    IN VARCHAR2)
RETURN BOOLEAN;

APEX_UTIL.CURRENT_USER_IN_GROUP(
    p_group_id    IN NUMBER)
RETURN BOOLEAN;

Parameters

Table 21–13 describes the parameters available in the CURRENT_USER_IN_GROUP function.

Table 21–13  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;
CUSTOM_CALENDAR Procedure

Use this procedure to change the existing calendar view to Custom Calendar.

Syntax
APEX_UTIL.CUSTOM_CALENDAR(
    p_date_type_field IN VARCHAR2);

Parameters
Table 21–14 describes the parameters available in the CUSTOM_CALENDAR procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_date_type_field</td>
<td>Identifies the item name used to define the type of calendar to be displayed.</td>
</tr>
</tbody>
</table>

Example 1
The following example defines a custom calendar based on the hidden calendar type field. Assuming the Calendar is created in Page 9, the following example hides the column called P9CALENDAR_TYPE.

APEX_UTIL.CUSTOM_CALENDAR('P9CALENDAR_TYPE');
Assuming you are using Application Express authentication, this procedure deletes a user group by providing the primary key of the group. To execute this procedure, the current user must have administrative privileges in the workspace.

Syntax

APEX_UTIL.DELETE_USER_GROUP(
   p_group_id IN NUMBER);

Parameter

Table 21–12 describes the parameters available in the DELETE_USER_GROUP procedure signature 1.

Table 21–15 DELETE_USER_GROUP Procedure Signature 1 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_group_id</td>
<td>Primary key of group</td>
</tr>
</tbody>
</table>

Example

The following example demonstrates how to use the DELETE_USER_GROUP procedure signature 1 to remove the user group called 'Managers', by providing the user group’s primary key.

DECLARE
   VAL NUMBER;
BEGIN
   VAL := APEX_UTIL.GET_GROUP_ID (
      p_group_name => 'Managers');
   APEX_UTIL.DELETE_USER_GROUP (
      p_group_id => VAL);
END;
Assuming you are using Application Express authentication, this procedure deletes a user group by providing the name of the group. To execute this procedure, the current user must have administrative privileges in the workspace.

Syntax

```sql
APEX_UTIL.DELETE_USER_GROUP(
    p_group_name IN VARCHAR2);
```

Parameter

Table 21–12 describes the parameters available in the DELETE_USER_GROUP procedure signature 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_group_name</td>
<td>Name of group</td>
</tr>
</tbody>
</table>

Example

The following example demonstrates how to use the DELETE_USER_GROUP procedure signature 2 to remove the user group called 'Managers', by providing the name of the user group.

```sql
BEGIN
    APEX_UTIL.DELETE_USER_GROUP (
        p_group_name => 'Managers');
END;
```
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 21–17 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</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>is derived from preferences.</td>
</tr>
</tbody>
</table>

**See Also:** "Printing Report Regions" in Oracle Application Express Application Builder User’s Guide.
DOWNLOAD_PRINT_DOCUMENT Procedure Signature 2

This procedure initiates the download of a print document using pre-defined report query and RTF and XSL-FO based report layout.

Syntax
APEX_UTIL.DOWNLOAD_PRINT_DOCUMENT (  
  p_file_name           IN VARCHAR,  
  p_content_disposition IN VARCHAR,  
  p_application_id      IN NUMBER,  
  p_report_query_name   IN VARCHAR2,  
  p_report_layout       IN CLOB,  
  p_report_layout_type  IN VARCHAR2 default 'xsl-fo',  
  p_document_format     IN VARCHAR2 default 'pdf',  
  p_print_server        IN VARCHAR2 default null);

Parameters
Table 21–18 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 is 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');

See Also: "Printing Report Regions" in Oracle Application Express
Application Builder User's Guide.
DOWNLOAD_PRINT_DOCUMENT Procedure Signature 3

This procedure initiates the download of a print document using pre-defined report query and pre-defined report layout.

Syntax

APEX_UTIL.DOWNLOAD_PRINT_DOCUMENT (  
p_file_name           IN VARCHAR,  
p_content_disposition IN VARCHAR,  
p_application_id      IN NUMBER,  
p_report_query_name   IN VARCHAR2,  
p_report_layout_name  IN VARCHAR2,  
p_report_layout_type  IN VARCHAR2 default 'xsl-fo',  
p_document_format     IN VARCHAR2 default 'pdf',  
p_print_server        IN VARCHAR2 default null);

Parameters

Table 21–19 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 is 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 want to provision dynamic layouts, instead of specifying 'ReportLayout' for the p_report_layout_name parameter, you could reference a page item that allowed the user to select one of multiple saved Report Layouts. This example also provides a way for the user to specify how they want to receive the document (as an attachment or inline), through passing the value of P1_CONTENT_DISP to the p_content_disposition.
parameter. `P1_CONTENT_DISP` is a page item of type 'Select List' with the following List of Values Definition:

```
STATIC2:In Browser;inline,Save / Open in separate Window;attachment
```

```
BEGIN
APEX_UTIL.DOWNLOAD_PRINT_DOCUMENT (  
p_file_name => 'myreport123',
p_content_disposition => :P1_CONTENT_DISP,  
p_application_id => :APP_ID,  
p_report_query_name => 'ReportQuery',  
p_report_layout_name => 'ReportLayout',  
p_report_layout_type => 'rtf',  
p_document_format => 'pdf');
END;
```

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 21–19 describes the parameters available in the DOWNLOAD_PRINT_DOCUMENT procedure for Signature 4.

Table 21–20  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 is 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 21–21 describes the parameters available in the EDIT_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_user_name</td>
<td>Alphanumeric name used for login.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;SET_USERNAME Procedure&quot; on page 21-140</td>
</tr>
<tr>
<td>p_first_name</td>
<td>Informational.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;SET_FIRST_NAME Procedure&quot; on page 21-125</td>
</tr>
<tr>
<td>p_last_name</td>
<td>Informational.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;SET_LAST_NAME Procedure&quot; on page 21-126</td>
</tr>
<tr>
<td>p_web_password</td>
<td>Clear text password. If using this procedure to update the password for the</td>
</tr>
<tr>
<td></td>
<td>user, values for both p_web_password and p_new_password must not be null</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>the user, values for both p_web_password and p_new_password must not be</td>
</tr>
<tr>
<td></td>
<td>null and must be identical.</td>
</tr>
</tbody>
</table>

Table 21–21 EDIT_USER Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_email_address</td>
<td>Informational.</td>
<td>&quot;SET_EMAIL Procedure&quot; on page 21-124</td>
</tr>
<tr>
<td>p_start_date</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>p_end_date</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>p_employee_id</td>
<td>Unused</td>
<td></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>
<td></td>
</tr>
<tr>
<td>p_person_type</td>
<td>Unused</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>
<td></td>
</tr>
<tr>
<td>p_group_ids</td>
<td>Colon-separated list of numeric group IDs</td>
<td></td>
</tr>
<tr>
<td>p_developer_roles</td>
<td>Colon-separated list of developer privileges. The following are acceptable values for this parameter:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>· <strong>null</strong> - To update the user to be an end user (a user who can only authenticate to developed applications)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>· <strong>CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL</strong> - To update the user to have developer privilege</td>
<td></td>
</tr>
<tr>
<td></td>
<td>· <strong>ADMIN:CREATE:DATA_LOADER:EDIT:HELP:MONITOR:SQL</strong> - To update the user to have full workspace administrator and developer privilege</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> 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></td>
<td><strong>See Also:</strong> &quot;GET_USER_ROLES Function&quot; on page 21-84</td>
<td></td>
</tr>
<tr>
<td>p_description</td>
<td>Informational</td>
<td></td>
</tr>
<tr>
<td>p_account_expiry</td>
<td>Date password was last updated.</td>
<td>&quot;EXPIRE_END_USER_ACCOUNT Procedure&quot; on page 21-36, &quot;EXPIRE_WORKSPACE_ACCOUNT Procedure&quot; on page 21-37, &quot;UNEXPIRE_END_USER_ACCOUNT Procedure&quot; on page 21-153, &quot;UNEXPIRE_WORKSPACE_ACCOUNT Procedure&quot; on page 21-154</td>
</tr>
<tr>
<td>p_account_locked</td>
<td>'Y' or 'N' indicating if account is locked or unlocked.</td>
<td>&quot;LOCK_ACCOUNT Procedure&quot; on page 21-103, &quot;UNLOCK_ACCOUNT Procedure&quot; on page 21-155</td>
</tr>
<tr>
<td>p_failed_access_attempts</td>
<td>Number of consecutive login failures that have occurred.</td>
<td></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>
<td>&quot;CHANGE_PASSWORD_ON_FIRST_USE Function&quot; on page 21-11</td>
</tr>
</tbody>
</table>
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
DECLARE
    l_user_id                       NUMBER;
    l_workspace                     VARCHAR2(255);
    l_user_name                     VARCHAR2(100);
    l_first_name                    VARCHAR2(255);
    l_last_name                     VARCHAR2(255);
    l_web_password                  VARCHAR2(255);
    l_email_address                 VARCHAR2(240);
    l_start_date                    DATE;
    l_end_date                      DATE;
    l_employee_id                   NUMBER(15,0);
    l_allow_access_to_schemas       VARCHAR2(4000);
    l_person_type                   VARCHAR2(1);
    l_default_schema                VARCHAR2(30);
    l_groups                        VARCHAR2(1000);
    l_developer_role                VARCHAR2(60);
    l_description                   VARCHAR2(240);
    l_account_expiry                DATE;
    l_account_locked                VARCHAR2(1);
    l_failed_access_attempts       NUMBER;
    l_change_password_on_first_use  VARCHAR2(1);
    l_first_password_use_occurred   VARCHAR2(1);
BEGIN
    l_user_id := APEX_UTIL.GET_USER_ID('FRANK');

    APEX_UTIL.FETCH_USER(
        p_user_id => l_user_id,
        p_workspace => l_workspace,
        p_user_name => l_user_name,
        p_first_name => l_first_name,
        p_last_name => l_last_name,
        p_web_password => l_web_password,
        p_email_address => l_email_address,
        p_start_date => l_start_date,
        p_end_date => l_end_date,
        p_employee_id => l_employee_id,
        p_allow_access_to_schemas => l_allow_access_to_schemas,
        p_person_type => l_person_type,
        p_default_schema => l_default_schema,
        p_groups => l_groups,
        p_developer_role => l_developer_role,
    )

    APEX_UTIL.EDIT_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,
    )
END;
```

See Also: "PASSWORD_FIRST_USE_OCCURRED Function" on page 21-104
EDIT_USER Procedure

```sql
APEX_UTIL.EDIT_USER (  
  p_user_id                       => l_user_id,  
  p_user_name                     => l_user_name, 
  p_first_name                    => l_first_name,  
  p_last_name                     => l_last_name, 
  p_web_password                  => l_web_password,  
  p_new_password                  => l_web_password, 
  p_email_address                 => l_email_address, 
  p_start_date                    => l_start_date,  
  p_end_date                      => l_end_date,  
  p_employee_id                   => l_employee_id,  
  p_allow_access_to_schemas       => l_allow_access_to_schemas,  
  p_person_type                   => l_person_type, 
  p_default_schema                => l_default_schema, 
  p_group_ids                     => l_groups,  
  p_description                   => l_description, 
  p_account_expiry                => l_account_expiry, 
  p_account_locked                => l_account_locked, 
  p_failed_access_attempts       => l_failed_access_attempts,  
  p_change_password_on_first_use  => l_change_password_on_first_use, 
  p_first_password_use_occurred   => l_first_password_use_occurred);  
END;
```

See Also:  
"FETCH_USER Procedure Signature 3" on page 21-45
END_USER_ACCOUNT_DAYS_LEFT Function

Returns the number of days remaining before a end user account password expires. This function may be run in a page request context by any authenticated user.

Syntax
APEX_UTIL.END_USER_ACCOUNT_DAYS_LEFT (  
   p_user_name IN VARCHAR2)  
RETURN NUMBER;

Parameters
Table 21–22 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 expires.

DECLARE
   l_days_left NUMBER;
BEGIN
   FOR c1 IN (SELECT user_name from wwv_flow_users) LOOP
      l_days_left := APEX_UTIL.END_USER_ACCOUNT_DAYS_LEFT(p_user_name =>
      c1.user_name);
      htp.p('End User Account:'||c1.user_name||' expires in '||l_days_left||' days.');
   END LOOP;
END;

See Also: "EXPIRE_END_USER_ACCOUNT Procedure" on page 21-36 and "UNEXPIRE_END_USER_ACCOUNT Procedure" on page 21-153
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 21–24 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 for its use by end users to authenticate to developed applications, but it may also expire the account for its use by developers or administrators to log in to a workspace.

Note that this procedure must be run by a user having administration privileges in the current workspace.

BEGIN
   FOR c1 IN (select user_name from wwv_flow_users) LOOP
      APEX_UTIL.EXPIRE_END_USER_ACCOUNT (p_user_name => c1.user_name);
      htp.p('End User Account:'||c1.user_name||' is now expired.');
   END LOOP;
END;

See Also: "UNEXPIRE_END_USER_ACCOUNT Procedure" on page 21-153
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 21–24 describes the parameters available in the EXPIRE_WORKSPACE_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_WORKSPACE_ACCOUNT procedure. Use this procedure to expire an Application Express account (workspace administrator, developer, or end user) in the current workspace. This action specifically expires the account for its use by developers or administrators to log in to a workspace, but it may also expire the account for its use by end users to authenticate to developed applications.

BEGIN
   FOR c1 IN (SELECT user_name FROM wwv_flow_users) LOOP
      APEX_UTIL.EXPIRE_WORKSPACE_ACCOUNT(p_user_name => c1.user_name);
      htp.p('Workspace Account:'||c1.user_name||' is now expired.');
   END LOOP;
END;

See Also: "UNEXPIRE_WORKSPACE_ACCOUNT Procedure" on page 21-154
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 21–25 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 are formatted. Specify 'UNIX' to have the resulting file contain rows delimited by line feeds. Specify 'DOS' to have the resulting file contain rows delimited by carriage returns and line feeds</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 is formatted with rows delimited by line feeds.

BEGIN
  APEX_UTIL.EXPORT_USERS;
END;
FETCH_APP_ITEM Function

This function fetches session state for the current or specified application in the current or specified session.

Syntax
APEX_UTIL.FETCH_APP_ITEM(
    p_item  IN VARCHAR2,
    p_app   IN NUMBER DEFAULT NULL,
    p_session IN NUMBER DEFAULT NULL)
RETURN VARCHAR2;

Parameters
Table 21–26 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.

DECLARE
    VAL VARCHAR2(30);
BEGIN
    VAL := APEX_UTIL.FETCH_APP_ITEM(
        p_item => 'F300_NAME',
        p_app => 300);
END;
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 21–27 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><strong>See Also:</strong> &quot;GET_USERNAME Function&quot; on page 21-85</td>
</tr>
<tr>
<td>p_first_name</td>
<td>Informational.</td>
</tr>
<tr>
<td></td>
<td><strong>See Also:</strong> &quot;GET_FIRST_NAME Function&quot; on page 21-65</td>
</tr>
<tr>
<td>p_last_name</td>
<td>Informational.</td>
</tr>
<tr>
<td></td>
<td><strong>See Also:</strong> &quot;GET_LAST_NAME Function&quot; on page 21-70</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><strong>See Also:</strong> &quot;GET_EMAIL Function&quot; on page 21-60</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>
Example for Signature 1

The following example shows how to use the `FETCH_USER` procedure with Signature 1. This procedure is passed the ID of the currently authenticated user for the only IN parameter `p_user_id`. The code then stores all the other OUT parameter values in local variables.

```
DECLARE
    l_workspace    VARCHAR2(255);
    l_user_name    VARCHAR2(100);
    l_first_name   VARCHAR2(255);
    l_last_name    VARCHAR2(255);
    l_web_password VARCHAR2(255);
    l_email_address VARCHAR2(240);
    l_start_date   DATE;
    l_end_date     DATE;
    l_employee_id  NUMBER(15,0);
    l_allow_access_to_schemas VARCHAR2(4000);
    l_person_type  VARCHAR2(1);
    l_default_schema VARCHAR2(30);
    l_groups       VARCHAR2(1000);
    l_developer_role VARCHAR2(60);
    l_description  VARCHAR2(240);
BEGIN
    APEX_UTIL.FETCH_USER(
        p_user_id => APEX_UTIL.GET_CURRENT_USER_ID,
        p_workspace => l_workspace,
        p_user_name => l_user_name,
        p_first_name => l_first_name,
        p_last_name => l_last_name,
        p_web_password => l_web_password,
        p_email_address => l_email_address,
        p_start_date => l_start_date,
        p_end_date => l_end_date,
        p_employee_id => l_employee_id,
        p_allow_access_to_schemas => l_allow_access_to_schemas,
        p_person_type => l_person_type,
        p_default_schema => l_default_schema,
        p_groups => l_groups,
        p_developer_role => l_developer_role,
        p_description => l_description
    );
END;
```

### Table 21-27 (Cont.) Fetch_User Parameters Signature 1

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</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>
</tr>
<tr>
<td></td>
<td>See Also: “GET_DEFAULT_SCHEMA Function” on page 21-58</td>
</tr>
<tr>
<td>p_groups</td>
<td>List of groups of which user is a member.</td>
</tr>
<tr>
<td></td>
<td>See Also: “GET_GROUPS_USER_BELONGS_TO Function” on page 21-66 and “CURRENT_USER_IN_GROUP Function” on page 21-21</td>
</tr>
<tr>
<td>p_developer_role</td>
<td>Colon-separated list of developer roles. The following are acceptable values for this parameter:</td>
</tr>
<tr>
<td></td>
<td>null - Indicates an end user (a user who can only authenticate to developed applications).</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></td>
<td>See Also: “GET_USER_ROLES Function” on page 21-84</td>
</tr>
<tr>
<td>p_description</td>
<td>Informational</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 21-31 and "GET_CURRENT_USER_ID Function" on page 21-37
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 21–28 describes the parameters available in the FETCH_USER procedure for signature 2.

Table 21–28 Fetch_User Parameters 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></td>
<td>See Also: &quot;GET_USERNAME Function&quot; on page 21-85</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 21-65</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 21-70</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 21-60</td>
</tr>
<tr>
<td>p_groups</td>
<td>List of groups of which user is a member.</td>
</tr>
<tr>
<td></td>
<td>See Also: &quot;GET_GROUPS_USER_BELONGS_TO Function&quot; on page 21-66 and &quot;CURRENT_USER_IN_GROUP Function&quot; on page 21-21</td>
</tr>
</tbody>
</table>
Example for Signature 2

The following example shows how to use the FETCH_USER procedure with Signature 2. This procedure is passed the ID of the currently authenticated user for the only IN parameter p_user_id. The code then stores all the other OUT parameter values in local variables.

```
DECLARE
  l_user_name         VARCHAR2(100);
  l_first_name        VARCHAR2(255);
  l_last_name         VARCHAR2(255);
  l_email_address     VARCHAR2(240);
  l_groups            VARCHAR2(1000);
  l_developer_role    VARCHAR2(60);
  l_description       VARCHAR2(240);
BEGIN
  APEX_UTIL.FETCH_USER(
    p_user_id           => APEX_UTIL.GET_CURRENT_USER_ID,
    p_user_name         => l_user_name,
    p_first_name        => l_first_name,
    p_last_name         => l_last_name,
    p_email_address     => l_email_address,
    p_groups            => l_groups,
    p_developer_role    => l_developer_role,
    p_description       => l_description);
END;
```

See Also: "EDIT_USER Procedure" on page 21-31 and "GET_CURRENT_USER_ID Function" on page 21-57
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**

```sql
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 21–29 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 21-85</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 21-65</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 21-70</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 21-60</td>
</tr>
<tr>
<td>p_start_date</td>
<td>Unused</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</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>
<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></td>
<td><strong>See Also:</strong> &quot;GET_DEFAULT_SCHEMA Function&quot; on page 21-58</td>
</tr>
<tr>
<td>p_groups</td>
<td>List of groups of which user is a member.</td>
</tr>
<tr>
<td></td>
<td><strong>See Also:</strong> &quot;GET_GROUPS_USER_BELONGS_TO Function&quot; on page 21-66 and &quot;CURRENT_USER_IN_GROUP Function&quot; on page 21-21</td>
</tr>
<tr>
<td>p_developer_role</td>
<td>Colon-separated list of developer roles. The following are acceptable values for this parameter:</td>
</tr>
<tr>
<td></td>
<td>null - Indicates an end user (a user who can only authenticate to developed applications).</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> 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_prives, EDIT_USER uses p_developer_roles and FETCH_USER uses p_developer_role.</td>
</tr>
<tr>
<td></td>
<td><strong>See Also:</strong> &quot;GET_USER_ROLES Function&quot; on page 21-84</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></td>
<td><strong>See Also:</strong> &quot;END_USER_ACCOUNT_DAYS_LEFT Function&quot; on page 21-35 and &quot;WORKSPACE_ACCOUNT_DAYS_LEFT Function&quot; on page 21-158</td>
</tr>
<tr>
<td>p_account_locked</td>
<td>Locked/Unlocked indicator Y or N.</td>
</tr>
<tr>
<td></td>
<td><strong>See Also:</strong> &quot;GET_ACCOUNT_LOCKED_STATUS Function&quot; on page 21-50</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.

```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);
  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 21-31 and "GET_CURRENT_USER_ID Function" on page 21-57
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 21–30 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'.

DECLARE
    VAL NUMBER;
BEGIN
    VAL := APEX_UTIL.FIND_SECURITY_GROUP_ID ('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 21–31 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 21–32 describes the parameters available in the GET_ACCOUNT_LOCKED_STATUS function.

Table 21–32 GET_ACCOUNT_LOCKED_STATUS 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 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:  LINK_ACCOUNT Procedure on page 21-103 and UNLOCK_ACCOUNT Procedure on page 21-155.
GET_ATTRIBUTE Function

This function returns the value of one of the attribute values (1 through 10) of a named user in the Application Express accounts table. Please note these are only accessible by using the APIs.

Syntax

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

Parameters

Table 21–33 describes the parameters available in the GET_ATTRIBUTE function.

Table 21–33  GET_ATTRIBUTE Parameters

<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'.

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 21-119
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 21-120 and "SET_CUSTOM_AUTH_STATUS Procedure" on page 21-122
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**

```sql
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 21–34 describes the parameters available in GET_BLOB_FILE_SRC function.

**Table 21–34   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:

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

As a Region Source of type SQL:

```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_BUILD_OPTION_STATUS Function Signature 1

Use this function to get the build option status of a specified application by providing the ID of the application build option.

Syntax

APEX_UTIL.GET_BUILD_OPTION_STATUS(
    p_application_id  IN NUMBER
    p_id              IN NUMBER;
)

Parameters

Table 21–35 describes the parameters available in the GET_BUILD_OPTION_STATUS function signature 1.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application_id</td>
<td>The ID of the application that owns the build option under shared components.</td>
</tr>
<tr>
<td>p_id</td>
<td>The ID of the build option in the application.</td>
</tr>
</tbody>
</table>

Example

The following code retrieves the current status of the specified build option that is identified by ID.

DECLARE
    l_status VARCHAR2(255);
BEGIN
    l_status := APEX_UTIL.GET_BUILD_OPTION_STATUS(
        P_APPLICATION_ID => 101,
        P_ID => 245935500311121039);
END;
/
GET_BUILD_OPTION_STATUS Function Signature 2

Use this function to get the build option status of a specified application by providing the name of the application build option.

Syntax

APEX_UTIL.GET_BUILD_OPTION_STATUS(
    p_application_id     IN NUMBER
    p_build_option_name IN VARCHAR2);

Parameters

Table 21–36 describes the parameters available in the GET_BUILD_OPTION_STATUS function signature 2.

Table 21–36  GET_BUILD_OPTION_STATUS Function Signature 2 Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_application_id</td>
<td>The ID of the application that owns the build option under shared components.</td>
</tr>
<tr>
<td>p_build_option_name</td>
<td>The name of the build option in the application.</td>
</tr>
</tbody>
</table>

Example

The following code retrieves the current status of the specified build option that is identified by name.

DECLARE
    l_status VARCHAR2(255);
BEGIN
    l_status := APEX_UTIL.GET_BUILD_OPTION_STATUS(
        P_APPLICATION_ID => 101,
        P_BUILD_OPTION_NAME => 'EXCLUDE_FROM_PRODUCTION');
END;
/


GET_CURRENT_USER_ID Function

This function returns the numeric user ID of the current user.

**Syntax**

APEX_UTIL.GET_CURRENT_USER_ID
RETURN NUMBER;

**Parameters**

None.

**Example**

This following example shows how to use the GET_CURRENT_USER_ID function. It returns the numeric user ID of the current user into a local variable.

```declare
VAL NUMBER;
begin
    VAL := APEX_UTIL.GET_CURRENT_USER_ID;
end;
```
GET_DEFAULT_SCHEMA Function

This function returns the default schema name associated with the current user.

Syntax

APEX_UTIL.GET_DEFAULT_SCHEMA
RETURN VARCHAR2;

Parameters

None.

Example

The following example shows how to use the GET_DEFAULT_SCHEMA function. It returns the default schema name associated with the current user into a local variable.

DECLARE
    VAL VARCHAR2(30);
BEGIN
    VAL := APEX_UTIL.GET_DEFAULT_SCHEMA;
END;
GET_EDITION Function

This function returns the edition for the current page view.

Syntax
APEX_UTIL.GET_EDITION
RETURN VARCHAR2;

Parameters
None.

Example
The following example shows how to use the GET_EDITION function. It returns the edition name for the current page view into a local variable.

DECLARE
   VAL VARCHAR2(30);
BEGIN
   VAL := APEX_UTIL.GET_EDITION;
END;
GET_EMAIL Function

This function returns the email address associated with the named user.

Syntax

APEX_UTIL.GET_EMAIL(
    p_username IN VARCHAR2);
RETURN VARCHAR2;

Parameters

Table 21–37 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 21-124
GET_FEEDBACK_FOLLOW_UP Function

Use this function to retrieve any remaining follow up associated with a specific feedback.

Syntax
APEX_UTIL.GET_FEEDBACK_FOLLOW_UP ( 
    p_feedback_id    IN NUMBER,
    p_row            IN NUMBER DEFAULT 1,
    p_template       IN VARCHAR2 DEFAULT '<br/>#CREATED_ON# (#CREATED_BY#) #FOLLOW_UP#')
RETURN VARCHAR2;

Parameters
Table 21–38 describes the parameters available in GET_FEEDBACK_FOLLOW_UP function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_feedback_id</td>
<td>The unique identifier of the feedback item.</td>
</tr>
<tr>
<td>p_row</td>
<td>Identifies which follow-up to retrieve and is ordered by created_on_desc.</td>
</tr>
<tr>
<td>p_template</td>
<td>The template to use to return the follow up. Given the &lt;br/&gt; in the default template, the function can be used in a loop to return all the follow up to a feedback.</td>
</tr>
</tbody>
</table>

Example
The following example displays all the remaining follow-up for feedback with the ID of 123.

declare
    l_feedback_count  number;
begin
    select count(*)
    into l_feedback_count
    from apex_team_feedback_followup
    where feedback_id = 123;

    for i in 1..l_feedback_count loop
        htp.p(apex_util.get_feedback_follow_up ( 
            p_feedback_id => 123,
            p_row         => i,
            p_template    => '<br/>#FOLLOW_UP# was created on #CREATED_ON# by #CREATED_BY#') );
    end loop;
end;
/
GET_FILE Procedure

This procedure downloads files from the Oracle Application Express file repository. Please note if you are invoking this procedure during page processing, you must ensure that no page branch is invoked under the same condition, as it interferes with the file retrieval. This means that branches with any of the following conditions should not be set to fire:

- Branches with a 'When Button Pressed' attribute equal to the button that invokes the procedure.
- Branches with conditional logic defined that would succeed during page processing when the procedure is being invoked.
- As unconditional.

Syntax

```sql
APEX_UTIL.GET_FILE (  
    p_file_id    IN   VARCHAR2,  
    p_inline     IN   VARCHAR2 DEFAULT 'NO');
```

Parameters

Table 21–39 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>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 is displayed inline in the browser.

```sql
BEGIN
    APEX_UTIL.GET_FILE(  
        p_file_id   => '8675309',  
        p_inline    => 'YES');
END;
```
END;

See Also: "GET_FILE_ID Function" on page 21-64
GET_FILE_ID Function

This function obtains the primary key of a file in the Oracle Application Express file repository.

**Syntax**

```sql
APEX_UTIL.GET_FILE_ID (  
    p_name    IN   VARCHAR2)  
RETURN NUMBER;
```

**Parameters**

Table 21–40 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 21–41 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 21-125
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 21-42 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 21-31
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 21–43 describes the parameters available in GET_GROUP_ID function.

Table 21–43  GET_GROUP_ID Parameters

<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 VARCHAR2;

Parameters
Table 21–44 describes the parameters available in GET_GROUP_NAME function.

Table 21–44  GET_GROUP_NAME Parameters

<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_HIGH_CONTRAST_MODE_TOGGLE Function

This function returns a link to the current page that enables you to turn on or off, toggle, the mode. For example, if you are in standard mode, this function displays a link that when clicked switches high contrast mode on.

Syntax

APEX_UTIL.GET_HIGH_CONTRAST_MODE_TOGGLE (  
  p_on_message  IN VARCHAR2 DEFAULT NULL,  
  p_off_message  IN VARCHAR2 DEFAULT NULL)  
RETURN VARCHAR2;

Parameters

Table 21–45 describes the parameters available in GET_HIGH_CONTRAST_MODE_TOGGLE function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_on_message</td>
<td>Optional text used for the link to switch to high contrast mode, when you are in standard mode. If this parameter is not passed, the default 'Set High Contrast Mode On' text is returned in the link.</td>
</tr>
<tr>
<td>p_off_message</td>
<td>Optional text used for the link to switch to standard mode, when you are in high contrast mode. If this parameter is not passed, the default 'Set High Contrast Mode Off' text is returned in the link.</td>
</tr>
</tbody>
</table>

Example

When running in standard mode, this function returns a link with the text 'Set High Contrast Mode On'. When the link is clicked the current page is refreshed and high contrast mode is switched on. When running in high contrast mode, a link 'Set High Contrast Mode Off' is returned. When the link is clicked the current page is refreshed and switched back to standard mode.

BEGIN
  htp.p(apex_util.get_high_contrast_mode_toggle);
END;

See Also:  "SHOW_HIGH_CONTRAST_MODE_TOGGLE Procedure" on page 21-141

Note:  There are also 2 translatable system messages that can be overridden at application level to change the default link text that is returned for this toggle. They include:

- APEX.SET_HIGH_CONTRAST_MODE_OFF - Default text = Set High Contrast Mode Off
- APEX.SET_HIGH_CONTRAST_MODE_ON - Default text = Set High Contrast Mode On
GET_LAST_NAME Function

This function returns the LAST_NAME field stored in the named user account record.

Syntax
APEX_UTIL.GET_LAST_NAME(
    p_username IN VARCHAR2)
RETURN VARCHAR2;

Parameters
Table 21–46 describes the parameters available in GET_LAST_NAME function.

Table 21–46  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 21-126
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 21–47 describes the parameters available in GET_NUMERIC_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 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 21-80 and "SET_SESSION_STATE Procedure" on page 21-137
GET_PREFERENCE Function

This function retrieves the value of a previously saved preference for a given user.

Syntax

APEX_UTIL.GET_PREFERENCE (
    p_preference  IN    VARCHAR2 DEFAULT NULL,
    p_user        IN    VARCHAR2 DEFAULT V('USER'))
RETURN VARCHAR2;

Parameters

Table 21–48 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.

```
DECLARE
    l_default_view    VARCHAR2(255);
BEGIN
    l_default_view := APEX_UTIL.GET_PREFERENCE(
        p_preference => 'default_view',
        p_user       => :APP_USER);
END;
```

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 21–49 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 is 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 21–50 describes the parameters available in the GET_PRINT_DOCUMENT function.

**Table 21–50 GET_PRINT_DOCUMENT Signature 2 Parameters**

<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 is 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 21–51 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 “xsl-fo” or “rtf”</td>
</tr>
<tr>
<td>p_document_format</td>
<td>Defines the document format, that is “pdf”, “rtf”, “xls”, “htm”, or “xml”</td>
</tr>
<tr>
<td>p_print_server</td>
<td>URL of the print server. If not specified, the print server is 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 21–52 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 is 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 with APEX_MAIL_SEND and APEX_MAIL_ADD_ATTACHMENT to send an email with an attachment of the file returned by GET_PRINT_DOCUMENT. Both the report data and layout are taken from values stored in page items (P1_XML and P1_XSL).

```
DECLARE

  l_id number;
  l_document BLOB;
BEGIN

  l_document := APEX_UTIL.GET_PRINT_DOCUMENT ( 
    p_report_data => :P1_XML, 
    p_report_layout => :P1_XSL, 
    p_report_layout_type => 'xsl-fo', 
    p_document_format => 'pdf');

  l_id := APEX_MAIL_SEND ( 
    p_to => :P35_MAIL_TO, 
    p_from => 'noreplies@oracle.com', 
    p_subj => 'sending PDF by using print API', 
    p_body => 'Please review the attachment.', 
    p_body_html => 'Please review the attachment');

```

21-76   Oracle Application Express API Reference
APEX_MAIL.ADD_ATTACHMENT {
    p_mail_id => l_id,
    p_attachment => l_document,
    p_filename => 'mydocument.pdf',
    p_mime_type => 'application/pdf');
END;
GET_SCREEN_READER_MODE_TOGGLE Function

This function returns a link to the current page to turn on or off, toggle, the mode. For example, if you are in standard mode, this function displays a link that when clicked switches screen reader mode on.

Syntax
APEX_UTIL.GET_SCREEN_READER_MODE_TOGGLE (  
    p_on_message  IN VARCHAR2 DEFAULT NULL,  
    p_off_message IN VARCHAR2 DEFAULT NULL)  
RETURN VARCHAR2;

Parameters
Table 21–53 describes the parameters available in GET_SCREEN_READER_MODE_TOGGLE function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_on_message</td>
<td>Optional text used for the link to switch to screen reader mode, when you are in standard mode. If this parameter is not passed, the default ‘Set Screen Reader Mode On’ text is returned in the link.</td>
</tr>
<tr>
<td>p_off_message</td>
<td>Optional text used for the link to switch to standard mode, when you are in screen reader mode. If this parameter is not passed, the default ‘Set Screen Reader Mode Off’ text is returned in the link.</td>
</tr>
</tbody>
</table>

Example
When running in standard mode, this function returns a link with the text ‘Set Screen Reader Mode On’. When the link is clicked the current page is refreshed and screen reader mode is switched on. When running in screen reader mode, a link ‘Set Screen Reader Mode Off’ is returned. When the link is clicked the current page is refreshed and switched back to standard mode.

BEGIN  
  htp.p(apex_util.get_screen_reader_mode_toggle);  
END;

See Also:  "SHOW_SCREEN_READER_MODE_TOGGLE Procedure" on page 21-142
GET_SESSION_LANG Function

This function returns the language setting for the current user in the current Application Express session.

Syntax
APEX_UTIL.GET_SESSION_LANG
RETURN VARCHAR2;

Parameters
None.

Example
The following example shows how to use the GET_SESSION_LANG function. It returns the session language for the current user in the current Application Express session into a local variable.

DECLARE
    VAL VARCHAR2(5) ;
BEGIN
    VAL := APEX_UTIL.GET_SESSION_LANG;
END;
GET_SESSION_STATE Function

This function returns the value for an item. You can use this function in your Oracle Application Express applications wherever you can use PL/SQL or SQL. You can also use the shorthand, function V, in place of APEX_UTIL.GET_SESSION_STATE.

Syntax

APEX_UTIL.GET_SESSION_STATE (  
    p_item    IN   VARCHAR2)  
RETURN VARCHAR2;

Parameters

Table 21–54 describes the parameters available in GET_SESSION_STATE function.

Table 21–54  GET_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 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 21-71 and "SET_SESSION_STATE Procedure" on page 21-137
GET_SESSION_TERRITORY Function

This function returns the territory setting for the current user in the current Application Express session.

**Syntax**

APEX_UTIL.GET_SESSION_TERRITORY
RETURN VARCHAR2;

**Parameters**

None.

**Example**

The following example shows how to use the GET_SESSION_TERRITORY function. It returns the session territory setting for the current user in the current Application Express session into a local variable.

```sql
DECLARE
    VAL VARCHAR2(30);
BEGIN
    VAL := APEX_UTIL.GET_SESSION_TERRITORY;
END;
```
GET_SESSION_TIME_ZONE Function

This function returns the time zone for the current user in the current Application Express session. This value is null if the time zone is not explicitly set by using APEX_UTIL.SET_SESSION_TIME_ZONE or if an application's automatic time zone attribute is enabled.

Syntax
APEX_UTIL.GET_SESSION_TIME_ZONE
RETURN VARCHAR2;

Parameters
None.

Example
The following example shows how to use the GET_SESSION_TIME_ZONE function. It returns the session time zone for the current user in the current Application Express session into a local variable.

BEGIN
    VAL := APEX_UTIL.GET_SESSION_TIME_ZONE;
END;
GET_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 21–55 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 21–56 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**

```sql
APEX_UTIL.GET_USERNAME(
    p_userid IN NUMBER
) RETURN VARCHAR2;
```

**Parameters**

Table 21–57 describes the parameters available in GET_USERNAME function.

<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.

```sql
DECLARE
    VAL VARCHAR2(100);
BEGIN
    VAL := APEX_UTIL.GET_USERNAME(p_userid => 228922003);
END;
```

**See Also:** "SET_USERNAME Procedure" on page 21-140
HOST_URL Function

This function returns the URL to the Application Express instance, depending on the option passed.

Syntax
APEX_UTIL.HOST_URL (  
    p_option IN VARCHAR2 DEFAULT NULL)  
RETURN VARCHAR2;

Parameters
Table 21–58 describes the parameters available in the HOST_URL function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_option</td>
<td>Specifies the parts of the URL to include.</td>
</tr>
</tbody>
</table>

Possible values for p_option include:
- NULL - Return URL up to port number. For example:
  http://myserver.com:7778
- SCRIPT - Return URL to include script name. For example:
  https://myserver.com:7778/pls/apex/
- IMGPRE - Return URL to include image prefix. For example:
  https://myserver.com:7778/i/

Example
The following example demonstrates how to use the HOST_URL function to return the URL, including the script name, to the current Application Express instance.

declare
    l_host_url    varchar2(4000);
    l_url         varchar2(4000);
    l_application varchar2(30) := 'f?p=100:1';
    l_email_body  varchar2(32000);
begin
    l_host_url := apex_util.host_url('SCRIPT');
    l_url := l_host_url||l_application;
    l_email_body := 'The URL to the application is: '||l_url;
end;
HTML_PCT_GRAPH_MASK Function

Use this function to scale a graph. This function can also be used by classic and interactive reports with format mask of GRAPH. This generates a <div> tag with inline styles.

Syntax
APEX_UTIL.HTML_PCT_GRAPH_MASK (  
    p_number         IN NUMBER    DEFAULT NULL,  
    p_size           IN NUMBER    DEFAULT 100,  
    p_background     IN VARCHAR2  DEFAULT NULL,  
    p_bar_background IN VARCHAR2  DEFAULT NULL,  
    p_format         IN VARCHAR2  DEFAULT NULL)  
RETURN VARCHAR2;

Parameters
Table 21–59 describes the parameters available in HTML_PCT_GRAPH_MASK function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_number</td>
<td>Number between 0 and 100.</td>
</tr>
<tr>
<td>p_size</td>
<td>Width of graph in pixels.</td>
</tr>
<tr>
<td>p_background</td>
<td>Six character hexadecimal background color of chart bar (not bar color)</td>
</tr>
<tr>
<td>p_bar_background</td>
<td>Six character hexadecimal background color of chart bar (bar color)</td>
</tr>
</tbody>
</table>
| p_format     | If this parameter is supplied, p_size, p_background and p_bar_background are ignored.  

This parameter uses the following format:  
- position 1: PCT_GRAPH format mask indicator  
- position 2: Background color in hexadecimal, 6 characters (optional)  
- position 3: Foreground "bar" color in hexadecimal, 6 characters (optional)  
- position 4: Chart width in pixels. Numeric and defaults to 100.  

p_number is automatically scaled so that 50 is half of chart_width (optional)

Example
The following is an SQL example.

```sql
select apex_util.html_pct_graph_mask(33) from dual
```

The following is a report numeric column format mask example.

```
PCT_GRAPH:777777:111111:200
```
INCREMENT_CALENDAR Procedure

Use this procedure to navigate to the next set of days in the calendar. Depending on what the calendar view is, this procedure navigates to the next month, week or day. If it is a Custom Calendar the total number of days between the start date and end date are navigated.

Syntax
APEX_UTIL.INCREMENT_CALENDAR;

Parameter
None.

Example
In this example, if you create a button called NEXT in the Calendar page and create a process that fires when the create button is clicked the following code navigates the calendar.

APEX_UTIL.INCREMENT_CALENDAR
IR_CLEAR Procedure

This procedure clears report settings.

Syntax

APEX_UTIL.IR_CLEAR(
   p_page_id IN NUMBER,
   p_report_alias IN VARCHAR2 DEFAULT NULL);

Parameters

Table 21–60 describes the parameters available in IR_CLEAR procedure.

Table 21–60  IR_CLEAR Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_page_id</td>
<td>Page of the current Application Express application that contains an interactive report.</td>
</tr>
<tr>
<td>p_report_alias</td>
<td>Identifies the saved report alias within the current application page. To clear a Primary report, p_report_alias must be 'PRIMARY' or leave as NULL. To clear a saved report, p_report_alias must be the name of the saved report. For example, to clear report '1234', p_report_alias must be '1234'</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use the IR_CLEAR procedure to clear Interactive report settings with alias of '8101021' in page 1 of the current application.

BEGIN
   APEX_UTIL.IR_CLEAR(
      p_page_id => 1,
      p_report_alias => '8101021'
   );
END;
IR_DELETE_REPORT Procedure

This procedure deletes saved Interactive reports. It deletes all saved reports except the Primary Default report.

Syntax

APEX_UTIL.IR_DELETE_REPORT(
    p_report_id IN NUMBER);

Parameters

Table 21–61 describes the parameters available in IR_DELETE_REPORT procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_report_id</td>
<td>Report ID to delete within the current Application Express application.</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use the IR_DELETE_REPORT procedure to delete the saved Interactive report with ID of '880629800374638220' in the current application.

BEGIN
    APEX_UTIL.IR_DELETE_REPORT(
        p_report_id => '880629800374638220');
END;
IR_DELETE_SUBSCRIPTION Procedure

This procedure deletes Interactive subscriptions.

**Syntax**

APEX_UTIL.IR_DELETE_SUBSCRIPTION(
    p_subscription_id IN NUMBER);

**Parameters**

Table 21–61 describes the parameters available in IR_DELETE_SUBSCRIPTION procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_subscription_id</td>
<td>Subscription ID to delete within the current workspace.</td>
</tr>
</tbody>
</table>

**Example**

The following example shows how to use the IR_DELETE_SUBSCRIPTION procedure to delete the subscription with ID of '880629800374638220' in the current workspace.

BEGIN
    APEX_UTIL.IR_DELETE_SUBSCRIPTION(
        p_subscription_id => '880629800374638220');
END;
IR_FILTER Procedure

This procedure creates a filter on an interactive report.

Syntax

APEX_UTIL.IR_FILTER(
    p_page_id       IN NUMBER,
    p_report_column IN VARCHAR2,
    p_operator_abbr IN VARCHAR2 DEFAULT NULL,
    p_filter_value  IN VARCHAR2,
    p_report_alias  IN VARCHAR2 DEFAULT NULL);

Parameters

Table 21–63 describes the parameters available in IR_FILTER procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_page_id</td>
<td>Page of the current Application Express application that contains an interactive report.</td>
</tr>
<tr>
<td>p_report_column</td>
<td>Name of the report SQL column, or column alias, to be filtered.</td>
</tr>
<tr>
<td>p_operator_abbr</td>
<td>Filter type. Valid values are as follows:</td>
</tr>
<tr>
<td></td>
<td>■ EQ = Equals</td>
</tr>
<tr>
<td></td>
<td>■ NEQ = Not Equals</td>
</tr>
<tr>
<td></td>
<td>■ LT = Less than</td>
</tr>
<tr>
<td></td>
<td>■ LTE = Less then or equal to</td>
</tr>
<tr>
<td></td>
<td>■ GT = Greater Than</td>
</tr>
<tr>
<td></td>
<td>■ GTE = Greater than or equal to</td>
</tr>
<tr>
<td></td>
<td>■ LIKE = SQL Like operator</td>
</tr>
<tr>
<td></td>
<td>■ N = Null</td>
</tr>
<tr>
<td></td>
<td>■ NN = Not Null</td>
</tr>
<tr>
<td></td>
<td>■ C = Contains</td>
</tr>
<tr>
<td></td>
<td>■ NC = Not Contains</td>
</tr>
<tr>
<td></td>
<td>■ IN = SQL In Operator</td>
</tr>
<tr>
<td></td>
<td>■ NIN = SQL Not In Operator</td>
</tr>
<tr>
<td>p_filter_value</td>
<td>Filter value. This value is not used for ‘N’ and ‘NN’.</td>
</tr>
<tr>
<td>p_report_alias</td>
<td>Identifies the saved report alias within the current application page. To create a filter on a Primary report, p_report_alias must be ‘PRIMARY’ or leave as NULL. To create a filter on a saved report, p_report_alias must be the name of the saved report. For example, to create a filter on report ‘1234’, p_report_alias must be ‘1234’.</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use the IR_FILTER procedure to filter Interactive report with alias of ‘8101021’ in page 1 of the current application with DEPTNO equals 30.

BEGIN
APEX_UTIL.IR_FILTER {
    p_page_id => 1,
    p_report_column => 'DEPTNO',
    p_operator_abbr => 'EQ',
    p_filter_value => '30'
    p_report_alias => '8101021'
};
END;
IR_RESET Procedure

This procedure resets report settings back to the default report settings. Resetting a report removes any customizations you have made.

Syntax
APEX_UTIL.IR_RESET(
    p_page_id IN NUMBER,
    p_report_alias IN VARCHAR2 DEFAULT NULL);

Parameters
Table 21–64 describes the parameters available in IR_RESET procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_page_id</td>
<td>Page of the current Application Express application that contains an interactive report.</td>
</tr>
<tr>
<td>p_report_alias</td>
<td>Identifies the saved report alias within the current application page. To reset a Primary report, p_report_alias must be 'PRIMARY' or leave as NULL. To reset a saved report, p_report_alias must be the name of the saved report. For example, to reset report '1234', p_report_alias must be '1234'.</td>
</tr>
</tbody>
</table>

Example
The following example shows how to use the IR_RESET procedure to reset Interactive report settings with alias of '8101021' in page 1 of the current application.

BEGIN
    APEX_UTIL.IR_RESET(          
        p_page_id => 1,             
        p_report_alias => '8101021' 
    );
END;
IS_HIGH_CONTRAST_SESSION Function

This function returns a boolean true if the session is in high contrast mode and returns a boolean false if not in high contrast mode.

**Syntax**

APEX_UTIL.IS_HIGH_CONTRAST_SESSION
RETURN BOOLEAN;

**Parameters**

None.

**Example**

In this example, if the current session is running in high contrast mode, a high contrast specific CSS file 'my_app_hc.css' is added to the HTML output of the page.

BEGIN
    IF apex_util.is_high_contrast_session THEN
        apex_css.add_file {
            p_name => 'my_app_hc');
        END IF;
    END IF;
END;
IS_HIGH_CONTRAST_SESSION_YN Function

This function returns Y if the session is in high contrast mode and N if not in high contrast mode.

Syntax
APEX_UTIL.IS_HIGH_CONTRAST_SESSION_YN
RETURN VARCHAR2;

Parameters
None.

Example
In this example, if the current session is running in high contrast mode, a high contrast specific CSS file, my_app_hc.css, is added to the HTML output of the page.

BEGIN
    IF apex_util.is_high_contrast_session_yn = 'Y' THEN
        apex_css.add_file (
            p_name => 'my_app_hc');
    END IF;
END;
IS_LOGIN_PASSWORD_VALID Function

This function returns a Boolean result based on the validity of the password for a named user account in the current workspace. This function returns true if the password matches and it returns false if the password does not match.

Syntax
APEX_UTIL.IS_LOGIN_PASSWORD_VALID(
    p_username IN VARCHAR2,
    p_password IN VARCHAR2)
RETURN BOOLEAN;

Parameters
Table 21–65 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 is returned if this is a valid password for 'FRANK', FALSE is returned if not.

```sql
DECLARE
    VAL BOOLEAN;
BEGIN
    VAL := APEX_UTIL.IS_LOGIN_PASSWORD_VALID (p_username=>'FRANK', p_password=>'tiger');
END;
```
IS_SCREEN_READER_SESSION Function

This function returns a boolean true if the session is in screen reader mode and returns a boolean false if not in screen reader mode.

Syntax
APEX_UTIL.IS_SCREEN_READER_SESSION
RETURN BOOLEAN;

Parameters
None

Example
BEGIN
    IF apex_util.is_screen_reader_session then
        htp.p('Screen Reader Mode');
    END IF;
END;
IS_SCREEN_READER_SESSION_YN Function

This function returns 'Y' if the session is in screen reader mode and 'N' if not in screen reader mode.

**Syntax**

APEX_UTIL.IS_SCREEN_READER_SESSION_YN
RETURN VARCHAR2;

**Parameters**

None

**Example**

BEGIN
  IF apex_util.is_screen_reader_session_yn = 'Y' then
    htp.p('Screen Reader Mode');
  END IF;
END;
IS_USERNAME_UNIQUE Function

This function returns a Boolean result based on whether the named user account is unique in the workspace.

Syntax
APEX_UTIL.IS_USERNAME_UNIQUE(
    p_username IN VARCHAR2)
RETURN BOOLEAN;

Parameters
Table 21–66 describes the parameters available in IS_USERNAME_UNIQUE function.

<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 is returned, otherwise TRUE is returned.

DECLARE
    VAL BOOLEAN;
BEGIN
    VAL := APEX_UTIL.IS_USERNAME_UNIQUE(
        p_username=>'FRANK');
END;
KEYVAL_NUM Function

This function gets the value of the package variable (wwv_flow_utilities.g_val_num) set by APEX_UTIL.SAVEKEY_NUM.

Syntax
APEX_UTIL.KEYVAL_NUM
RETURN NUMBER;

Parameters
None

Example
The following example shows how to use the KEYVAL_NUM function to return the current value of the package variable wwv_flow_utilities.g_val_num.

DECLARE
    VAL NUMBER;
BEGIN
    VAL := APEX_UTIL.KEYVAL_NUM;
END;

See Also: "SAVEKEY_NUM Function" on page 21-117
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.

```sql
DECLARE
    VAL VARCHAR2(4000);
BEGIN
    VAL := APEX_UTIL.KEYVAL_VC2;
END;
```

See Also: "SAVEKEY_VC2 Function" on page 21-118
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 21–67 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 21-155 and "GET_ACCOUNT_LOCKED_STATUS Function" on page 21-50
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 21–68 describes the parameters available in the PASSWORD_FIRST_USE_OCCURRED procedure.

Table 21–68  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 21-11
PREPARE_URL Function

The PREPARE_URL function serves two purposes:

1. To return an f?p URL with the Session State Protection checksum argument (&cs=) if one is required.
2. To return an f?p URL with the session ID component replaced with zero (0) if the zero session ID feature is in use and other criteria are met.

**Note:** The PREPARE_URL functions returns the f?p URL with &cs=<large hex value> appended. If you use this returned value, for example in JavaScript, it may be necessary to escape the ampersand in the URL to conform with syntax rules of the particular context. One place you may encounter this is in SVG chart SQL queries which might include PREPARE_URL calls.

### Syntax

```sql
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 21–69 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
The following example shows how to use the PREPARE_URL function to return a URL with a zero session ID. In a PL/SQL Dynamic Content region that generates f?p URLs (anchors), call PREPARE_URL to ensure that the session ID is set to zero when the zero session ID feature is in use, when the user is a public user (not authenticated), and when the target page is a public page in the current application:

```sql
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"
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 21–70 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 21–71 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 21–72 describes the parameters available in PURGE_REGIONS_BY_NAME.

Table 21–72  PURGE_REGIONS_BY_NAME 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 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 'my_cached_region' on page 1 of the current application.

BEGIN  
    APEX_UTIL.PURGE_REGIONS_BY_NAME(  
        p_application => :APP_ID,  
        p_page => 1,  
        p_region_name => 'my_cached_region');  
END;
PURGE_REGIONS_BY_PAGE Procedure

Deletes all cached regions by application and page.

Syntax

APEX_UTIL.PURGE_REGIONS_BY_PAGE (
    p_application IN NUMBER,
    p_page     IN NUMBER);

Parameters

Table 21–73 describes the parameters available in PURGE_REGIONS_BY_PAGE.

<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;
REDIRECT_URL Procedure

This procedure calls owa_util.redirect_url to tell the browser to redirect to a new URL. Afterwards, it automatically calls apex_application.stop_apex_engine to abort further processing of the Application Express application.

Syntax
APEX_UTIL.REDIRECT_URL (  
    p_url in varchar2,  
    p_reset_htp_buffer in boolean default true );

Parameters
Table 21–74 describes the parameters available in the REDIRECT_URL procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_url</td>
<td>The URL the browser requests.</td>
</tr>
<tr>
<td>p_reset_htp_buffer</td>
<td>Set to TRUE to reset the HTP buffer to make sure the browser understands the redirect to the new URL and is not confused by data that is already written to the HTP buffer. Set to FALSE if the application has it's own cookie to use in the response.</td>
</tr>
</tbody>
</table>

Example
The following example tells the browser to redirect to http://www.oracle.com and immediately stops further processing.

```sql
APEX_UTIL.REDIRECT_URL (  
    p_url => 'http://www.oracle.com/' );
```
## 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 21–75 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 21-72, "SET_PREFERENCE Procedure" on page 21-127 and "Managing Session State and User Preferences" in Oracle Application Express Administration Guide.
REMOVE_SORT_PREFERENCES Procedure

This procedure removes the user’s column heading sorting preference value.

Syntax
APEX_UTIL.REMOVE_SORT_PREFERENCES (
    p_user  IN   VARCHAR2 DEFAULT V('USER'));

Parameters
Table 21–76 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 are 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.

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 21–77 describes the parameters available in the REMOVE_USER procedure.

Table 21–77  REMOVE_USER Parameters

<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 21–78 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 21-10
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 21–79 describes the parameters available in the SAVEKEY_NUM procedure.

Table 21–79  SAVEKEY_NUM Parameters

<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 21-101
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 21–80 describes the parameters available in the SAVEKEY_VC2 function.

Table 21–80  SAVEKEY_VC2 Parameters
<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'.

DECLARE
    VAL VARCHAR2(4000);
BEGIN
    VAL := APEX_UTIL.SAVEKEY_VC2(p_val => 'XXX');
END;

See Also:  "KEYVAL_VC2 Function" on page 21-102
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 21–81 describes the parameters available in the SET_ATTRIBUTE procedure.

Table 21–81  SET_ATTRIBUTE Parameters

<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 21-51
**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**

```sql
APEX_UTIL.SET_AUTHENTICATION_RESULT(
    p_code IN NUMBER);
```

**Parameters**

Table 21–24 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.

```sql
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 21-52 and "SET_CUSTOM_AUTH_STATUS Procedure" on page 21-122
SET_BUILD_OPTION_STATUS Procedure

Use this procedure to change the build option status of a specified application.

**Syntax**

```sql
apex_util.set_build_option_status(p_application_id IN NUMBER,
                                  p_id IN NUMBER,
                                  p_build_status IN VARCHAR2);
```

**Parameters**

Table 21–83 describes the parameters available in the SET_BUILD_OPTION_STATUS 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 that owns the build option under shared components.</td>
</tr>
<tr>
<td>p_id</td>
<td>The ID of the build option in the application.</td>
</tr>
<tr>
<td>p_build_status</td>
<td>The new status of the build option. Possible values are INCLUDE, EXCLUDE both upper case.</td>
</tr>
</tbody>
</table>

**Example**

The following example demonstrates how to use the SET_BUILD_OPTION_STATUS procedure to change the current status of build option.

```sql
BEGIN
APEX_UTIL.SET_BUILD_OPTION_STATUS(
    P_APPLICATION_ID  => 101,
    P_ID => 2459350031121039, P_BUILD_STATUS=>'INCLUDE');
END;
```
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 21–84 describes the parameters available in the SET_CUSTOM_AUTH_STATUS procedure.

Table 21–84  SET_CUSTOM_AUTH_STATUS Parameters

<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 21-120 and "GET_AUTHENTICATION_RESULT Function" on page 21-52
SET_EDITION Procedure

This procedure sets the name of the edition to be used in all application SQL parsed in the current page view or page submission.

Syntax
APEX_UTIL.SET_EDITION(
    p_edition IN VARCHAR2);

Parameters
Table 21–84 describes the parameters available in the SET_EDITION procedure.

Table 21–85 SET_EDITION Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>

Example
The following example shows how to use the SET_EDITION procedure. It sets the edition name for the database session of the current page view.

BEGIN
    APEX_UTIL.SET_EDITION( p_EDITION => 'Edition1' );
END;

Note: Support for Edition-Based Redefinition is only available in database version 11.2.0.1 or higher.
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**
```sql
APEX_UTIL.SET_EMAIL(
    p_userid IN NUMBER,
    p_email  IN VARCHAR2);
```

**Parameters**
Table 21–86 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@somewhere.com' for the user 'FRANK'.

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

**See Also:** 
"GET_EMAIL Function" on page 21-60 and "GET_USER_ID Function" on page 21-83
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 21–87 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 21-65 and "GET_USER_ID Function” on page 21-83
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 21–88 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 21-70 and "GET_USER_ID Function" on page 21-83
SET_PREFERENCE Procedure

This procedure sets a preference that persists beyond the user's current session.

Syntax
APEX_UTIL.SET_PREFERENCE (  
  p_preference   IN    VARCHAR2 DEFAULT NULL,  
  p_value        IN    VARCHAR2 DEFAULT NULL,  
  p_user         IN    VARCHAR2 DEFAULT NULL);  

Parameters
Table 21–89 describes the parameters available in the SET_PREFERENCE procedure.

Table 21–89  SET_PREFERENCE Parameters

<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 persists beyond session for the currently authenticated user.

BEGIN  
APEX_UTIL.SET_PREFERENCE(  
  p_preference => 'default_view',  
  p_value      => 'WEEKLY',  
  p_user       => :APP_USER);  
END;  

See Also: "GET_PREFERENCE Function" on page 21-72 and "REMOVE_PREFERENCE Procedure" on page 21-112
SET_SECURITY_GROUP_ID Procedure

Use this procedure with `apex_util.find_security_group_id` to ease the use of the mail package in batch mode. This procedure is especially useful when a schema is associated with more than one workspace. For example, you might want to create a procedure that is run by a nightly job to email all outstanding tasks.

Syntax

```
APEX_UTIL.SET_SECURITY_GROUP_ID (  
    p_security_group_id  IN NUMBER);
```

Parameters

Table 21–90 describes the parameters available in the `SET_SECURITY_GROUP_ID` procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_security_group_id</td>
<td>This is the security group id of the workspace you are working in.</td>
</tr>
</tbody>
</table>

Example

The following example sends an alert to each user that has had a task assigned within the last day.

```sql
create or replace procedure new_tasks  
is  
l_workspace_id      number;
l_subject           varchar2(2000);
l_body              clob;
l_body_html         clob;
begin  
l_workspace_id := apex_util.find_security_group_id (p_workspace =>'PROJECTS');  
apex_util.set_security_group_id (p_security_group_id => l_workspace_id);  
    l_body := ' ';  
l_subject := 'You have new tasks';  
for c1 in (select distinct(p.email_address) email_address, p.user_id  
    from teamsp_user_profile p, teamsp_tasks t  
    where p.user_id = t.assigned_to_user_id  
    and t.created_on > sysdate - 1  
    and p.email_address is not null ) loop  
l_body_html := '<p />The following tasks have been added.';  
for c2 in (select task_name, due_date  
    from teamsp_tasks  
    where assigned_to_user_id = c1.user_id  
    and created_on > sysdate - 1 ) loop  
l_body_html := l_body_html || '<p />Task: '||c2.task_name||', due  
    '||c2.due_date;  
    end loop;
apex_mail.send (  
p_to => c1.email_address,  
p_from => c1.email_address,  
p_body => l_body,  
p_body_html => l_body_html,
```

Table 21–90  SET_SECURITY_GROUP_ID Parameters
p_subj => l_subject );
end loop;
end;
SET_SESSION_HIGH_CONTRAST_OFF Procedure

This procedure switches off high contrast mode for the current session.

Syntax
APEX_UTIL.SET_SESSION_HIGH_CONTRAST_OFF;

Parameters
None.

Example
In this example, high contrast mode is switched off for the current session.

BEGIN
    apex_util.set_session_high_contrast_off;
END;
SET_SESSION_HIGH_CONTRAST_ON Procedure

This procedure switches on high contrast mode for the current session.

**Syntax**
APEX_UTIL.SET_SESSION_HIGH_CONTRAST_ON;

**Parameters**
None.

**Example**
In this example, the current session is put into high contrast mode.

```sql
BEGIN
    apex_util.set_session_high_contrast_on;
END;
```
SET_SESSION_LANG Procedure

This procedure sets the language to be used for the current user in the current Application Express session. The language must be a valid IANA language name.

Syntax

APEX_UTIL.SET_SESSION_LANG(
   p_lang IN VARCHAR2);

Parameters

Table 21–91 describes the parameters available in the SET_SESSION_LANG procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_lang</td>
<td>This is an IANA language code. Some examples include: en, de, de-at, zh-cn, and pt-br.</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use the SET_SESSION_LANG procedure. It sets the language for the current user for the duration of the Application Express session.

BEGIN
   APEX_UTIL.SET_SESSION_LANG( P_LANG => 'en');
END;
SET_SESSION_LIFETIME_SECONDS Procedure

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

Syntax

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

Parameters

Table 21–92 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>This parameter is obsolete. The procedure always sets the lifetime for the whole session.</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).

By allowing the p_scope input parameter to use the default value of 'SESSION', the following example would actually apply to all applications using the current session. This would be the most common use case when multiple Application Express applications use a common authentication scheme and are designed to operate as a suite in a common session.

BEGIN
    APEX_UTIL.SET_SESSION_LIFETIME_SECONDS(p_seconds => 7200);
END;

Example 2

The following example shows how to use the SET_SESSION_LIFETIME_SECONDS procedure to set the current application’s Maximum Session Length in Seconds attribute to 3600 seconds (one hour).

BEGIN
    APEX_UTIL.SET_SESSION_LIFETIME_SECONDS(p_seconds => 3600);
END;
SET_SESSION_MAX_IDLE SECONDS Procedure

Sets the current application’s Maximum Session Idle Time in Seconds value for the current session, overriding the corresponding application attribute. This allows developers to dynamically shorten or lengthen the maximum idle time allowed between page requests based on criteria determined after the user authenticates.

Syntax

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

Parameters

Table 21–93 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>This parameter is obsolete. The procedure always sets the lifetime for the whole 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). The following example applies to all applications using the current session.

BEGIN
    APEX_UTIL.SET_SESSION_MAX_IDLE_SECONDS (p_seconds => 1200);
END;

Example 2

The following example shows how to use the SET_SESSION_MAX_IDLE SECONDS procedure to set the current application’s Maximum Session Idle Time in Seconds attribute to 600 seconds (ten minutes). This example applies to all applications using the current session.

BEGIN
    APEX_UTIL.SET_SESSION_MAX_IDLE_SECONDS (p_seconds => 600);
END;
SET_SESSION_SCREEN_READER_OFF Procedure

This procedure switches off screen reader mode for the current session.

Syntax
APEX_UTIL.SET_SESSION_SCREEN_READER_OFF;

Parameters
None

Example
In this example, the current session is put into standard mode.
BEGIN
  IF apex_util.set_session_screen_reader_off;
END;
This procedure puts the current session into screen reader mode.

**Syntax**

APEX_UTIL.SET_SESSION SCREEN_READER_ON;

**Parameters**

None

**Example**

In this example, the current session is put into screen reader mode.

```
BEGIN
    IF apex_util.set_session_screen_reader_on;
END;
```
SET_SESSION_STATE Procedure

This procedure sets session state for a current Oracle Application Express session.

**Syntax**

```sql
APEX_UTIL.SET_SESSION_STATE (  
    p_name     IN    VARCHAR2 DEFAULT NULL,  
    p_value    IN    VARCHAR2 DEFAULT NULL);
```

**Parameters**

Table 21–94 describes the parameters available in the SET_SESSION_STATE procedure.

### Table 21–94 SET_SESSION_STATE Parameters

<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 sessions 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.

```sql
BEGIN  
    APEX_UTIL.SET_SESSION_STATE('my_item','myvalue');  
END;
```

**See Also:**  "GET_SESSION_STATE Function" on page 21-80, "GET_NUMERIC_SESSION_STATE Function" on page 21-71, and "Understanding Session State Management" in Oracle Application Express Application Builder User’s Guide
SET_SESSION_TERRITORY Procedure

This procedure sets the territory to be used for the current user in the current Application Express session. The territory name must be a valid Oracle territory.

Syntax

APEX_UTIL.SET_SESSION_TERRITORY(
   p_territory IN VARCHAR2);

Parameters

Table 21–95 describes the parameters available in the SET_SESSION_TERRITORY procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_territory</td>
<td>A valid Oracle territory name. Examples include: AMERICA, UNITED KINGDOM, ISRAEL, AUSTRIA, and UNITED ARAB EMIRATES.</td>
</tr>
</tbody>
</table>

Example

The following example shows how to use the SET_SESSION_TERRITORY procedure. It sets the territory for the current user for the duration of the Application Express session.

BEGIN
   APEX_UTIL.SET_SESSION_TERRITORY( P_TERRITORY => 'UNITED KINGDOM');
END;
This procedure sets the time zone to be used for the current user in the current Application Express session.

**Syntax**

APEX_UTIL.SET_SESSION_TIME_ZONE(
  p_time_zone IN VARCHAR2);

**Parameters**

Table 21–96 describes the parameters available in the SET_SESSION_TIME_ZONE procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_timezone</td>
<td>A time zone value in the form of hours and minutes. Examples include: +09:00, 04:00, -05:00.</td>
</tr>
</tbody>
</table>

**Example**

The following example shows how to use the SET_SESSION_TIME_ZONE procedure. It sets the time zone for the current user for the duration of the Application Express session.

```
BEGIN
  APEX_UTIL.SET_SESSION_TIME_ZONE( P_TIME_ZONE => '-05:00');
END;
```
SET_USERNAME Procedure

This procedure updates a user account with a new USER_NAME value. To execute this procedure, the current user must have administrative privileges in the workspace.

Syntax
APEX_UTIL.SET_USERNAME(
   p_userid   IN NUMBER,
   p_username IN VARCHAR2);

Parameters
Table 21–97 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 USER_NAME 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 21-85 and "GET_USER_ID Function" on page 21-83
**SHOW_HIGH_CONTRAST_MODE_TOGGLE Procedure**

This procedure displays a link to the current page to turn on or off, toggle, the mode. For example, if you are in standard mode, this function displays a link that when clicked switches the high contrast mode on.

**Syntax**

APEX_UTIL.SHOW_HIGH_CONTRAST_MODE_TOGGLE (  
  p_on_message  in varchar2 default null,  
  p_off_message in varchar2 default null);  

**Parameters**

Table 21–98 describes the parameters available in the SHOW_HIGH_CONTRAST_MODE_TOGGLE procedure.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_on_message</td>
<td>Optional text used for the link to switch to high contrast mode, when you are in standard mode. If this parameter is not passed, the default 'Set High Contrast Mode On' text is displayed.</td>
</tr>
<tr>
<td>p_off_message</td>
<td>Optional text used for the link to switch to standard mode, when you are in high contrast mode. If this parameter is not passed, the default 'Set High Contrast Mode Off' text is displayed.</td>
</tr>
</tbody>
</table>

**Example**

When running in standard mode, this procedure displays a link, Set High Contrast Mode On, that when clicked refreshes the current page and switches on high contrast mode. When running in high contrast mode, a link, Set High Contrast Mode Off, is displayed, that refreshes the current page and switches back to standard mode when clicked.

BEGIN  
  htp.p(apex_util.show_high_contrast_mode_toggle);  
END;

See Also: "GET_HIGH_CONTRAST_MODE_TOGGLE Function" on page 21-69

**Note:** There are also 2 translatable system messages that can be overridden at application level to change the default link text that is returned for this toggle. They include:

- APEX.SET_HIGH_CONTRAST_MODE_OFF - Default text = Set High Contrast Mode Off
- APEX.SET_HIGH_CONTRAST_MODE_ON - Default text = Set High Contrast Mode On
SHOW_SCREEN_READER_MODE_TOGGLE Procedure

This procedure displays a link to the current page to turn on or off, toggle, the mode. For example, if you are in standard mode, this function displays a link that when clicked switches the screen reader mode on.

Syntax

APEX_UTIL.SHOW_SCREEN_READER_MODE_TOGGLE (  
  p_on_message  IN VARCHAR2 DEFAULT NULL,  
  p_off_message IN VARCHAR2 DEFAULT NULL)

Parameters

Table 21–99 describes the parameters available in SHOW_SCREEN_READER_MODE_TOGGLE function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_on_message</td>
<td>Optional text used for the link to switch to screen reader mode, when you are in standard mode. If this parameter is not passed, the default 'Set Screen Reader Mode On' text is displayed.</td>
</tr>
<tr>
<td>p_off_message</td>
<td>Optional text used for the link to switch to standard mode, when you are in screen reader mode. If this parameter is not passed, the default 'Set Screen Reader Mode Off' text is displayed.</td>
</tr>
</tbody>
</table>

Example

When running in standard mode, this procedure displays a link 'Set Screen Reader Mode On', that when clicked refreshes the current page and switches on screen reader mode. When running in screen reader mode, a link 'Set Screen Reader Mode Off' is displayed, that when clicked refreshes the current page and switches back to standard mode.

BEGIN
  htp.p(apex_util.show_screen_reader_mode_toggle);
END;
STRING_TO_TABLE Function

Given a string, this function returns a PL/SQL array of type APEX_APPLICATION_GLOBAL.VC_ARR2. This array is a VARCHAR2 (32767) table.

Syntax
APEX_UTIL.STRING_TO_TABLE (
    p_string       IN VARCHAR2,
    p_separator    IN VARCHAR2 DEFAULT ':')
RETURN APEX_APPLICATION_GLOBAL.VC_ARR2;

Parameters
Table 21–100 describes the parameters available in the STRING_TO_TABLE function.

Table 21–100 STRING_TO_TABLE Parameters

<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_APPLICATION_GLOBAL.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 21-152
This procedure returns Boolean OUT values based on whether a proposed password meets the password strength requirements as defined by the Oracle Application Express site administrator.

Syntax

APEX_UTIL.STRONG_PASSWORD_CHECK(
    p_username                    IN  VARCHAR2,
    p_password                    IN  VARCHAR2,
    p_old_password                IN  VARCHAR2,
    p_workspace_name              IN  VARCHAR2,
    p_use_strong_rules            IN  BOOLEAN,
    p_min_length_err              OUT BOOLEAN,
    p_new_differs_by_err          OUT BOOLEAN,
    p_one_alpha_err               OUT BOOLEAN,
    p_one_numeric_err             OUT BOOLEAN,
    p_one_punctuation_err         OUT BOOLEAN,
    p_one_upper_err               OUT BOOLEAN,
    p_one_lower_err               OUT BOOLEAN,
    p_not_like_username_err       OUT BOOLEAN,
    p_not_like_workspace_name_err OUT BOOLEAN,
    p_not_like_words_err          OUT BOOLEAN,
    p_not_reusable_err            OUT BOOLEAN);         

Parameters

Table 21–101 describes the parameters available in the STRONG_PASSWORD_CHECK procedure.

Table 21–101   STRONG_PASSWORD_CHECK Parameters

<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.

```
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 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 21–102 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;
```
SUBMIT_FEEDBACK Procedure

This procedure enables you to write a procedure to submit feedback, rather than using the page that can be generated by create page of type feedback.

Syntax

```sql
APEX_UTIL.SUBMIT_FEEDBACK (  
  p_comment          IN VARCHAR2 DEFAULT NULL,  
  p_type             IN NUMBER   DEFAULT '1',  
  p_application_id   IN VARCHAR2 DEFAULT NULL,  
  p_page_id          IN VARCHAR2 DEFAULT NULL,  
  p_email            IN VARCHAR2 DEFAULT NULL,  
  p_screen_width     IN VARCHAR2 DEFAULT NULL,  
  p_screen_height    IN VARCHAR2 DEFAULT NULL,  
  p_attribute_01     IN VARCHAR2 DEFAULT NULL,  
  p_attribute_02     IN VARCHAR2 DEFAULT NULL,  
  p_attribute_03     IN VARCHAR2 DEFAULT NULL,  
  p_attribute_04     IN VARCHAR2 DEFAULT NULL,  
  p_attribute_05     IN VARCHAR2 DEFAULT NULL,  
  p_attribute_06     IN VARCHAR2 DEFAULT NULL,  
  p_attribute_07     IN VARCHAR2 DEFAULT NULL,  
  p_attribute_08     IN VARCHAR2 DEFAULT NULL,  
  p_label_01         IN VARCHAR2 DEFAULT NULL,  
  p_label_02         IN VARCHAR2 DEFAULT NULL,  
  p_label_03         IN VARCHAR2 DEFAULT NULL,  
  p_label_04         IN VARCHAR2 DEFAULT NULL,  
  p_label_05         IN VARCHAR2 DEFAULT NULL,  
  p_label_06         IN VARCHAR2 DEFAULT NULL,  
  p_label_07         IN VARCHAR2 DEFAULT NULL,  
  p_label_08         IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 21–103 describes the parameters available in the SUBMIT_FEEDBACK procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_comment</td>
<td>Comment to be submitted</td>
</tr>
<tr>
<td>p_type</td>
<td>Type of feedback (1 is General Comment, 2 is Enhancement Request, 3 is Bug)</td>
</tr>
<tr>
<td>p_application_id</td>
<td>ID of application related to the feedback</td>
</tr>
<tr>
<td>p_page_id</td>
<td>ID of page related to the feedback</td>
</tr>
<tr>
<td>p_email</td>
<td>Email of the user providing the feedback</td>
</tr>
<tr>
<td>p_screen_width</td>
<td>Width of screen at time feedback was provided</td>
</tr>
<tr>
<td>p_screen_height</td>
<td>Height of screen at time feedback was provided</td>
</tr>
<tr>
<td>p_attribute_01</td>
<td>Custom attribute for collecting feedback</td>
</tr>
<tr>
<td>p_attribute_02</td>
<td>Custom attribute for collecting feedback</td>
</tr>
<tr>
<td>p_attribute_03</td>
<td>Custom attribute for collecting feedback</td>
</tr>
<tr>
<td>p_attribute_04</td>
<td>Custom attribute for collecting feedback</td>
</tr>
<tr>
<td>p_attribute_05</td>
<td>Custom attribute for collecting feedback</td>
</tr>
</tbody>
</table>
Example
The following example submits a bug about page 22 within application 283.

begin
  apex_util.submit_feedback ( 
    p_comment => 'This page does not render properly for me',
    p_type => 3,
    p_application_id => 283,
    p_page_id => 22,
    p_email => 'user@xyz.corp',
    p_attribute_01 => 'Charting',
    p_label_01 => 'Component' );
end;
/

Table 21–103 (Cont.) SUBMIT_FEEDBACK Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_attribute_06</td>
<td>Custom attribute for collecting feedback</td>
</tr>
<tr>
<td>p_attribute_07</td>
<td>Custom attribute for collecting feedback</td>
</tr>
<tr>
<td>p_attribute_08</td>
<td>Custom attribute for collecting feedback</td>
</tr>
<tr>
<td>p_label_01</td>
<td>Label for corresponding custom attribute</td>
</tr>
<tr>
<td>p_label_02</td>
<td>Label for corresponding custom attribute</td>
</tr>
<tr>
<td>p_label_03</td>
<td>Label for corresponding custom attribute</td>
</tr>
<tr>
<td>p_label_04</td>
<td>Label for corresponding custom attribute</td>
</tr>
<tr>
<td>p_label_05</td>
<td>Label for corresponding custom attribute</td>
</tr>
<tr>
<td>p_label_06</td>
<td>Label for corresponding custom attribute</td>
</tr>
<tr>
<td>p_label_07</td>
<td>Label for corresponding custom attribute</td>
</tr>
<tr>
<td>p_label_08</td>
<td>Label for corresponding custom attribute</td>
</tr>
</tbody>
</table>
SUBMIT_FEEDBACK_FOLLOWUP Procedure

This procedure enables you to submit follow up to a feedback.

Syntax

APEX_UTIL.SUBMIT_FEEDBACK_FOLLOWUP (  
   p_feedback_id IN NUMBER,  
   p_follow_up    IN VARCHAR2 DEFAULT NULL,  
   p_email        IN VARCHAR2 DEFAULT NULL);

Parameters

Table 21–104 describes the parameters available in the SUBMIT_FEEDBACK_FOLLOWUP procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_feedback_followup</td>
<td>ID of feedback that this is a follow up to</td>
</tr>
<tr>
<td>p_follow_up</td>
<td>Text of follow up</td>
</tr>
<tr>
<td>p_email</td>
<td>Email of user providing the follow up</td>
</tr>
</tbody>
</table>

Example

The following example submits follow up to a previously filed feedback.

begin
   apex_util.submit_feedback_followup (  
      p_feedback_id => 12345,  
      p_follow_up   => 'I tried this on another instance and it does not work there either',  
      p_email       => 'user@xyz.corp' );
end;
/

TABLE_TO_STRING Function

Given a PL/SQL table of type APEX_APPLICATION_GLOBAL.VC_ARR2, this function returns a delimited string separated by the supplied separator, or by the default separator, a colon (:).

Syntax

APEX_UTIL.TABLE_TO_STRING (  
  p_table       IN     APEX_APPLICATION_GLOBAL.VC_ARR2,  
  p_string      IN     VARCHAR2 DEFAULT ':')  
RETURN VARCHAR2;

Parameters

Table 21–105 describes the parameters available in the TABLE_TO_STRING function.

Table 21–105  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 function returns a comma delimited string of contact names that are associated with the provided cust_id.

```sql
create or replace function get_contacts (  
  p_cust_id  in  number  
)  
return varchar2  
is  
  l_vc_arr2   apex_application_global.vc_arr2;  
  l_contacts  varchar2(32000);  
begin  
  select contact_name  
    bulk collect  
    into l_vc_arr2  
    from contacts  
    where cust_id = p_cust_id  
    order by contact_name;  
  l_contacts := apex_util.table_to_string (  
    p_table => l_vc_arr2,  
    p_string => ',', ' ');  
  return l_contacts;  
end get_contacts;
```

See Also:  "STRING_TO_TABLE Function" on page 21-143
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 21–106 describes the parameters available in the UNEXPIRE_END_USER_ACCOUNT procedure.

Table 21–106  UNEXPIRE_END_USER_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_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 21-36 and "END_USER_ACCOUNT_DAYS_LEFT Function" on page 21-35
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 21–107 describes the parameters available in the UNEXPIRE_WORKSPACE_ACCOUNT procedure.

Table 21–107 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 for its use by end users to authenticate to developed applications.

This procedure must be run by a user having administration privileges in the current workspace.

BEGIN
    FOR c1 IN (select user_name from wwv_flow_users) loop
        APEX_UTIL.UNEXPIRE_WORKSPACE_ACCOUNT(p_user_name => c1.user_name);
        htp.p('Workspace Account:'||c1.user_name||' is now valid.');
    END LOOP;
END;

See Also:  "EXPIRE_WORKSPACE_ACCOUNT Procedure" on page 21-37 and "WORKSPACE_ACCOUNT_DAYS_LEFT Function" on page 21-158
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 21–108 describes the parameters available in the UNLOCK_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 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 21-103 and  "GET_ACCOUNT_LOCKED_STATUS Function" on page 21-50
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 21–109 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 21–110 describes the parameters available in the WORKSPACE_ACCOUNT_DAYS_LEFT procedure.

Table 21–110 WORKSPACE_ACCOUNT_DAYS_LEFT 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 WORKSPACE_ACCOUNT_DAYS_LEFT function. It can be used in to find the number of days remaining before an Application Express administrator or developer account in the current workspace expires.

DECLARE
  l_days_left NUMBER;
BEGIN
  FOR c1 IN (SELECT user_name from wwv_flow_users) LOOP
    l_days_left := APEX_UTIL.WORKSPACE_ACCOUNT_DAYS_LEFT(p_user_name => c1.user_name);
    htp.p('Workspace Account:'||c1.user_name||' expires in '|l_days_left||' days.');
  END LOOP;
END;

See Also: "EXPIRE_WORKSPACE_ACCOUNT Procedure" on page 21-37 and "UNEXPIRE_WORKSPACE_ACCOUNT Procedure" on page 21-154
The APEX_WEB_SERVICE API enables you to integrate other systems with Application Express by allowing you to interact with Web services anywhere you can use PL/SQL in your application. The API contains procedures and functions to call both SOAP and RESTful style Web services. It contains functions to parse the responses from Web services and to encode/decode into SOAP friendly base64 encoding.

This API also contains package globals for managing cookies and HTTP headers when calling Web services whether from the API or by using standard processes of type Web service. Cookies and HTTP headers can be set before invoking a call to a Web service by populating the globals and the cookies and HTTP headers returned from the Web service response can be read from other globals.

Topics:
- About the APEX_WEB_SERVICE API
- BLOB2CLOBBASE64 Function
- CLOBBASE642BLOB Function
- MAKE_REQUEST Procedure
- MAKE_REQUEST Function
- MAKE_REST_REQUEST Function
- PARSE_RESPONSE Function
- PARSE_RESPONSE_CLOB Function
- PARSE_XML Function
- PARSE_XML_CLOB Function
About the APEX_WEB_SERVICE API

Use the APEX_WEB_SERVICE API to invoke a Web service and examine the response anywhere you can use PL/SQL in Application Express.

The following are examples of when you might use the APEX_WEB_SERVICE API:

- When you want to invoke a Web service by using an On Demand Process using AJAX.
- When you want to invoke a Web service as part of an Authentication Scheme.
- When you need to pass a large binary parameter to a Web service that is base64 encoded.
- When you want to invoke a Web service as part of a validation.

Topics:
- Invoking a SOAP Style Web Service
- Invoking a RESTful Style Web Service
- Retrieving Cookies and HTTP Headers
- Setting Cookies and HTTP Headers
Invoking a SOAP Style Web Service

There is a procedure and a function to invoke a SOAP style Web service. The procedure stores the response in the collection specified by the parameter p_collection_name. The function returns the results as an XMLTYPE. To retrieve a specific value from the response, you use either the PARSE_RESPONSE function if the result is stored in a collection or the PARSE_XML function if the response is returned as an XMLTYPE.

To pass a binary parameter to the Web service as base64 encoded character data, use the function BLOB2CLOBBASE64. Conversely, to transform a response that contains a binary parameter that is base64 encoded use the function CLOBBASE642BLOB.

The following is an example of using the BLOB2CLOBBASE64 function to encode a parameter, MAKE_REQUEST procedure to call a Web service, and the PARSE_RESPONSE function to extract a specific value from the response.

```
declare
    l_filename varchar2(255);
    l_BLOB BLOB;
    l_CLOB CLOB;
    l_envelope CLOB;
    l_response_msg varchar2(32767);
BEGIN
    IF :P1_FILE IS NOT NULL THEN
        SELECT filename, BLOB_CONTENT
        INTO l_filename, l_BLOB
        FROM APEX_APPLICATION_FILES
        WHERE name = :P1_FILE;
        l_CLOB := apex_web_service.blob2clobbase64(l_BLOB);
        l_envelope := q'!<?xml version='1.0' encoding='UTF-8'?>!';
        l_envelope := l_envelope '<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmns:chec="http://www.stellent.com/CheckIn/"
<soapenv:Header/>
<soapenv:Body>
    <chec:CheckInUniversal>
        <chec:DocName>'||l_filename||'</chec:DocName>
        <chec:DocTitle>'||l_filename||'</chec:DocTitle>
        <chec:DocType>Document</chec:DocType>
        <chec:DocAuthor>GM</chec:DocAuthor>
        <chec:dSecurityGroup>Public</chec:dSecurityGroup>
        <chec:DocAccount></chec:DocAccount>
        <chec:CustomDocMetaData>
            <chec:property>
                <chec:name></chec:name>
                <chec:value></chec:value>
            </chec:property>
        </chec:CustomDocMetaData>
        <chec:primaryFile>
            <chec:fileName>'||l_filename||'</chec:fileName>
            <chec:fileContent>'||l_CLOB||'</chec:fileContent>
        </chec:primaryFile>
        <chec:alternateFile>
            <chec:fileName></chec:fileName>
            <chec:fileContent></chec:fileContent>
        </chec:alternateFile>
```
<chec:extraProps>
  <chec:property>
    <chec:name></chec:name>
    <chec:value></chec:value>
  </chec:property>
</chec:extraProps>
</soapenv:Body>
</soapenv:Envelope>'

apex_web_service.make_request{
  p_url => 'http://127.0.0.1/idcplg',
  p_action => 'http://www.stellent.com/CheckIn/',
  p_collection_name => 'STELLENT_CHECKIN',
  p_envelope => l_envelope,
  p_username => 'sysadmin',
  p_password => 'welcome1'};

l_response_msg := apex_web_service.parse_response{
  p_collection_name=>'STELLENT_CHECKIN',
  p_xpath=>'//idc:CheckInUniversalResponse/idc:CheckInUniversalResult/idc:StatusInfo/idc:statusMessage/text()',
  p_ns =>'xmlns:idc="http://www.stellent.com/CheckIn/''');

:P1_RES_MSG := l_response_msg;

END IF;
END;
Invoking a RESTful Style Web Service

RESTful style Web services use a simpler architecture than SOAP. Typically the input to a RESTful style Web service is a collection of name/value pairs. The response can be an XML document or simply text such as a comma separated response or JSON.

The following is an example of MAKE_REST_REQUEST being used in an application process that is callable by AJAX.

```sql
declare
    l_clob clob;
    l_buffer         varchar2(32767);
    l_amount         number;
    l_offset         number;
begin
    l_clob := apex_web_service.make_rest_request(
        p_url => 'http://us.music.yahooapis.com/
                video/v1/list/published/popular',
        p_http_method => 'GET',
        p_parm_name => apex_util.string_to_table('appid:format'),
        p_parm_value => apex_util.string_to_table(apex_application.g_x01||':'||apex_application.g_x02));
    l_amount := 32000;
    l_offset := 1;
begin
    loop
        dbms_lob.read( l_clob, l_amount, l_offset, l_buffer );
        htp.p(l_buffer);
        l_offset := l_offset + l_amount;
        l_amount := 32000;
    end loop;
exception
    when no_data_found then
        null;
end;
end;
```
Retrieving Cookies and HTTP Headers

When you invoke a Web service using any of the supported methods in Application Express, the \texttt{g\_response\_cookies} and \texttt{g\_headers} globals are populated if the Web service response included any cookies or HTTP headers. You can interrogate these globals and store the information in collections.

The following are examples of interrogating the \texttt{APEX\_WEB\_SERVICE} globals to store cookie and HTTP header responses in collections.

```sql
declare
  i number;
  secure varchar2(1);
begin
  apex_collection.create_or_truncate_collection('P31\_RESP\_COOKIES');
  for i in 1.. apex_web_service.g_response_cookies.count loop
    IF (apex_web_service.g_response_cookies(i).secure) THEN
      secure := 'Y';
    ELSE
      secure := 'N';
    END IF;
    apex_collection.add_member(p_collection_name => 'P31\_RESP\_COOKIES',
               p_c001 => apex_web_service.g_response_cookies(i).name,
               p_c002 => apex_web_service.g_response_cookies(i).value,
               p_c003 => apex_web_service.g_response_cookies(i).domain,
               p_c004 => apex_web_service.g_response_cookies(i).expire,
               p_c005 => apex_web_service.g_response_cookies(i).path,
               p_c006 => secure,
               p_c007 => apex_web_service.g_response_cookies(i).version);
  end loop;
end;

declare
  i number;
begin
  apex_collection.create_or_truncate_collection('P31\_RESP\_HEADERS');
  for i in 1.. apex_web_service.g_headers.count loop
    apex_collection.add_member(p_collection_name => 'P31\_RESP\_HEADERS',
               p_c001 => apex_web_service.g_headers(i).name,
               p_c002 => apex_web_service.g_headers(i).value,
               p_c003 => apex_web_service.g_status_code);
  end loop;
end;
```
Setting Cookies and HTTP Headers

You set cookies and HTTP headers that should be sent along with a Web service request by populating the globals `g_request_cookies` and `g_request_headers` before the process that invokes the Web service.

The following examples show populating the globals to send cookies and HTTP headers with a request.

```sql
for c1 in (select seq_id, c001, c002, c003, c004, c005, c006, c007
            from apex_collections
            where collection_name = 'P31_RESP_COOKIES' ) loop
    apex_web_service.g_request_cookies(c1.seq_id).name := c1.c001;
    apex_web_service.g_request_cookies(c1.seq_id).value := c1.c002;
    apex_web_service.g_request_cookies(c1.seq_id).domain := c1.c003;
    apex_web_service.g_request_cookies(c1.seq_id).expire := c1.c004;
    apex_web_service.g_request_cookies(c1.seq_id).path := c1.c005;
    if c1.c006 = 'Y' then
        apex_web_service.g_request_cookies(c1.seq_id).secure := true;
    else
        apex_web_service.g_request_cookies(c1.seq_id).secure := false;
    end if;
    apex_web_service.g_request_cookies(c1.seq_id).version := c1.c007;
end loop;

for c1 in (select seq_id, c001, c002
            from apex_collections
            where collection_name = 'P31_RESP_HEADERS' ) loop
    apex_web_service.g_request_headers(c1.seq_id).name := c1.c001;
    apex_web_service.g_request_headers(c1.seq_id).value := c1.c002;
end loop;
```
BLOB2CLOBBASE64 Function

Use this function to convert a BLOB datatype into a CLOB that is base64 encoded. This is often used when sending a binary as an input to a Web service.

**Syntax**

```sql
APEX_WEB_SERVICE.BLOB2CLOBBASE64 (p_blob IN BLOB)
RETURN CLOB;
```

**Parameters**

Table 22–1 describes the parameters available in the BLOB2CLOBBASE64 function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_blob</td>
<td>The BLOB to convert into base64 encoded CLOB.</td>
</tr>
</tbody>
</table>

**Example**

The following example gets a file that was uploaded from the apex_application_files view and converts the BLOB into a CLOB that is base64 encoded.

```sql
declare
  l_clobCLOB;
  l_blobBLOB;
begint
  SELECT BLOB_CONTENT
    INTO l_BLOB
    FROM APEX_APPLICATION_FILES
    WHERE name = :P1_FILE;

  l_CLOB := apex_web_service.blob2clobbase64(l_BLOB);
end;
```
CLOBBASE642BLOB Function

Use this function to convert a CLOB datatype that is base64 encoded into a BLOB. This is often used when receiving output from a Web service that contains a binary parameter.

Syntax
APEX_WEB_SERVICE.CLOBBASE642BLOB {
    p_clob IN CLOB
} RETURN BLOB;

Parameters
Table 22–2 describes the parameters available in the CLOBBASE642BLOB function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_clob</td>
<td>The base64 encoded CLOB to convert into a BLOB.</td>
</tr>
</tbody>
</table>

Example
The following example retrieves a base64 encoded node from an XML document as a CLOB and converts it into a BLOB.

declare
    l_base64CLOB;
    l_blobBLOB;
    l_xml   XMLTYPE;
begin
    l_base64 := apex_web_service.parse_xml_clob(l_xml, '//runReportReturn/reportBytes/text()');
    l_blob := apex_web_service.clobbase642blob(l_base64);
end;
MAKE_REQUEST Procedure

Use this procedure to invoke a SOAP style Web service with the supplied SOAP envelope and store the results in a collection.

Syntax

APEX_WEB_SERVICE.MAKE_REQUEST (
    p_url               IN VARCHAR2,
    p_action            IN VARCHAR2 default null,
    p_version           IN VARCHAR2 default '1.1',
    p_collection_name   IN VARCHAR2 default null,
    p_envelope          IN CLOB,
    p_username          IN VARCHAR2 default null,
    p_password          IN VARCHAR2 default null,
    p_proxy_override    IN VARCHAR2 default null,
    p_transfer_timeout  IN NUMBER   default 180,
    p_wallet_path       IN VARCHAR2 default null,
    p_wallet_pwd        IN VARCHAR2 default null );

Parameters

Table 22-3 describes the parameters available in the MAKE_REQUEST procedure.

Table 22-3  MAKE_REQUEST Procedure Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_url</td>
<td>The URL endpoint of the Web service.</td>
</tr>
<tr>
<td>p_action</td>
<td>The SOAP Action corresponding to the operation to be invoked.</td>
</tr>
<tr>
<td>p_version</td>
<td>The SOAP version, 1.1 or 1.2. The default is 1.1.</td>
</tr>
<tr>
<td>p_collection_name</td>
<td>The name of the collection to store the response.</td>
</tr>
<tr>
<td>p_envelope</td>
<td>The SOAP envelope to post to the service.</td>
</tr>
<tr>
<td>p_username</td>
<td>The username if basic authentication is required for this service.</td>
</tr>
<tr>
<td>p_password</td>
<td>The password if basic authentication is required for this service.</td>
</tr>
<tr>
<td>p_proxy_override</td>
<td>The proxy to use for the request. The proxy supplied overrides the proxy defined in the application attributes.</td>
</tr>
<tr>
<td>p_transfer_timeout</td>
<td>The amount of time in seconds to wait for a response.</td>
</tr>
<tr>
<td>p_wallet_path</td>
<td>The file system path to a wallet if the URL endpoint is https. For example, file:/usr/home/oracle/WALLETS. The wallet path provided overrides the wallet defined in the instance settings.</td>
</tr>
<tr>
<td>p_wallet_pwd</td>
<td>The password to access the wallet.</td>
</tr>
</tbody>
</table>

Example

The following example uses the make_request procedure to retrieve a list of movies from a SOAP style Web service. The response is stored in an Application Express collection named MOVIE_LISTINGS.

declare
    l_envelope CLOB;
BEGIN
    l_envelope := '<?xml version="1.0" encoding="UTF-8">
    <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
APEX_WEB_SERVICE 22-11

xmlns:tns='http://www.ignyte.com/whatsshowing'
xmlns:xs='http://www.w3.org/2001/XMLSchema'>
    <soap:Body>
        <tns:GetTheatersAndMovies>
            <tns:zipCode>43221</tns:zipCode>
            <tns:radius>5</tns:radius>
        </tns:GetTheatersAndMovies>
    </soap:Body>
</soap:Envelope'>;

apex_web_service.make_request(
    p_url => 'http://www.ignyte.com/webservices/ignyte.whatsshowing.webservice/moviefunctions.asmx',
    p_action => 'http://www.ignyte.com/whatsshowing/GetTheatersAndMovies',
    p_collection_name => 'MOVIE_LISTINGS',
    p_envelope => l_envelope
);
END;
MAKE_REQUEST Function

Use this function to invoke a SOAP style Web service with the supplied SOAP envelope returning the results in an XMLTYPE.

Syntax

APEX_WEB_SERVICE.MAKE_REQUEST (  
  p_url               IN VARCHAR2,  
  p_action            IN VARCHAR2 default null,  
  p_version           IN VARCHAR2 default '1.1',  
  p_envelope          IN CLOB,  
  p_username          IN VARCHAR2 default null,  
  p_password          IN VARCHAR2 default null,  
  p_proxy_override    IN VARCHAR2 default null,  
  p_transfer_timeout  IN NUMBER   default 180,  
  p_wallet_path       IN VARCHAR2 default null,  
  p_wallet_pwd        IN VARCHAR2 default null )  
RETURN XMLTYPE;

Parameters

Table 22–4 describes the parameters available in the MAKE_REQUEST function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_url</td>
<td>The URL endpoint of the Web service.</td>
</tr>
<tr>
<td>p_action</td>
<td>The SOAP Action corresponding to the operation to be invoked.</td>
</tr>
<tr>
<td>p_version</td>
<td>The SOAP version, 1.1 or 1.2. The default is 1.1.</td>
</tr>
<tr>
<td>p_envelope</td>
<td>The SOAP envelope to post to the service.</td>
</tr>
<tr>
<td>p_username</td>
<td>The username if basic authentication is required for this service.</td>
</tr>
<tr>
<td>p_password</td>
<td>The password if basic authentication is required for this service</td>
</tr>
<tr>
<td>p_proxy_override</td>
<td>The proxy to use for the request. The proxy supplied overrides the proxy defined in the application attributes.</td>
</tr>
<tr>
<td>p_transfer_timeout</td>
<td>The amount of time in seconds to wait for a response.</td>
</tr>
<tr>
<td>p_wallet_path</td>
<td>The file system path to a wallet if the URL endpoint is https. For example, file:/usr/home/oracle/WALLETS. The wallet path provided overrides the wallet defined in the instance settings.</td>
</tr>
<tr>
<td>p_wallet_pwd</td>
<td>The password to access the wallet.</td>
</tr>
</tbody>
</table>

Example

The following example uses the make_request function to invoke a SOAP style Web service that returns movie listings. The result is stored in an XMLTYPE.

```
declare  
  l_envelope CLOB;  
  l_xmlXMLTYPE;  
BEGIN  
  l_envelope := '  
```

22-12 Oracle Application Express API Reference
CREATE OR REPLACE FUNCTION make_request 
AS l_xml VARCHAR2;
BEGIN

-- Request XML
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <tns:GetTheatersAndMovies>
      <tns:zipCode>43221</tns:zipCode>
      <tns:radius>5</tns:radius>
    </tns:GetTheatersAndMovies>
  </soap:Body>
</soap:Envelope>';

  l_xml := apex_web_service.make_request(
    p_url => 'http://www.ignyte.com/webservices/ignyte.whatsshowing.webservice/moviefunctions.asmx',
    p_action => 'http://www.ignyte.com/whatsshowing/GetTheatersAndMovies',
    p_envelope => l_envelope
  );
END;
MAKE_REST_REQUEST Function

Use this function to invoke a RESTful style Web service supplying either name value pairs, a character based payload or a binary payload and returning the response in a CLOB.

Syntax

APEX_WEB_SERVICE.MAKE_REST_REQUEST(
    p_url               IN VARCHAR2,
    p_http_method       IN VARCHAR2,
    p_username          IN VARCHAR2 default null,
    p_password          IN VARCHAR2 default null,
    p_proxy_override    IN VARCHAR2 default null,
    p_transfer_timeout  IN NUMBER   default 180,
    p_body              IN CLOB default empty_clob(),
    p_body_blob         IN BLOB default empty_blob(),
    p_parm_name         IN WWV_FLOW_GLOBAL.VC_ARR2 default empty_vc_arr,
    p_parm_value        IN WWV_FLOW_GLOBAL.VC_ARR2 default empty_vc_arr,
    p_wallet_path       IN VARCHAR2 default null,
    p_wallet_pwd        IN VARCHAR2 default null)
RETURN CLOB;

Parameters

Table 22–5 describes the parameters available in the MAKE_REST_REQUEST function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_url</td>
<td>The URL endpoint of the Web service.</td>
</tr>
<tr>
<td>p_http_method</td>
<td>The HTTP method to use, PUT, POST, GET, HEAD, or DELETE.</td>
</tr>
<tr>
<td>p_username</td>
<td>The username if basic authentication is required for this service.</td>
</tr>
<tr>
<td>p_password</td>
<td>The password if basic authentication is required for this service.</td>
</tr>
<tr>
<td>p_proxy_override</td>
<td>The proxy to use for the request. The proxy supplied overrides the proxy defined in the application attributes.</td>
</tr>
<tr>
<td>p_transfer_timeout</td>
<td>The amount of time in seconds to wait for a response.</td>
</tr>
<tr>
<td>p_body</td>
<td>The HTTP payload to be sent as CLOB.</td>
</tr>
<tr>
<td>p_body_blob</td>
<td>The HTTP payload to be sent as binary BLOB. For example, posting a file.</td>
</tr>
<tr>
<td>p_parm_name</td>
<td>The name of the parameters to be used in name/value pairs.</td>
</tr>
<tr>
<td>p_parm_value</td>
<td>The value of the parameters to be used in name/value pairs.</td>
</tr>
<tr>
<td>p_wallet_path</td>
<td>The file system path to a wallet if the URL endpoint is https. For example, file:/usr/home/oracle/WALLETS. The wallet path provided overrides the wallet defined in the instance settings.</td>
</tr>
<tr>
<td>p_wallet_pwd</td>
<td>The password to access the wallet.</td>
</tr>
</tbody>
</table>

Example

The following example calls a RESTful style Web service using the make_rest_request function passing the parameters to the service as name/value pairs. The response from the service is stored in a locally declared CLOB.
declare
    l_clob CLOB;
BEGIN

    l_clob := apex_web_service.make_rest_request(
        p_http_method => 'GET',
        p_parm_name => apex_util.string_to_table('appid:format'),
        p_parm_value => apex_util.string_to_table('xyz:xml'));

END
PARSE_RESPONSE Function

Use this function to parse the response from a Web service that is stored in a collection and return the result as a VARCHAR2 type.

Syntax
APEX_WEB_SERVICE.PARSE_RESPONSE (  
    p_collection_name IN VARCHAR2,  
    p_xpath IN VARCHAR2,  
    p_ns IN VARCHAR2 default null )  
RETURN VARCHAR2;

Parameters
Table 22–6 describes the parameters available in the PARSE_RESPONSE function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_collection_name</td>
<td>The name of the collection where the Web service response is stored.</td>
</tr>
<tr>
<td>p_xpath</td>
<td>The XPath expression to the desired node.</td>
</tr>
<tr>
<td>p_ns</td>
<td>The namespace to the desired node.</td>
</tr>
</tbody>
</table>

Example
The following example parses a response stored in a collection called STELLENT_CHECKIN and stores the value in a locally declared VARCHAR2 variable.

declarer  
    l_response_msg  VARCHAR2(4000);  
BEGIN  
    l_response_msg := apex_web_service.parse_response(  
        p_collection_name=>'STELLENT_CHECKIN',  
        p_xpath =>OURS://idc:CheckInUniversalResponse/idc:CheckInUniversalResult/idc:StatusInfo/idc:statusMessage/text()',  
        p_ns=>'xmlns:idc="http://www.stellent.com/CheckIn/"');  
END;

Table 22–6 PARSE_RESPONSE Function Parameters
PARSE_RESPONSE_CLOB Function

Use this function to parse the response from a Web service that is stored in a collection and return the result as a CLOB type.

Syntax

APEX_WEB_SERVICE.PARSE_RESPONSE_CLOB (  
p_collection_name   IN VARCHAR2,  
p_xpath             IN VARCHAR2,  
p_ns                IN VARCHAR2 default null )  
RETURN CLOB;

Parameters

Table 22–7 describes the parameters available in the PARSE_RESPONSE_CLOB function.

Table 22–7  PARSE_RESPONSE_CLOB Function Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_collection_name</td>
<td>The name of the collection where the Web service response is stored.</td>
</tr>
<tr>
<td>p_xpath</td>
<td>The XPath expression to the desired node.</td>
</tr>
<tr>
<td>p_ns</td>
<td>The namespace to the desired node.</td>
</tr>
</tbody>
</table>

Example

The following example parses a response stored in a collection called STELLENT_CHECKIN and stores the value in a locally declared CLOB variable.

```sql
declare  
  l_response_msg  CLOB;  
BEGIN  
  l_response_msg := apex_web_service.parse_response_clob(  
    p_collection_name=>'STELLENT_CHECKIN',  
    p_xpath=>  
      '//idc:CheckInUniversalResponse/idc:CheckInUniversalResult/idc:StatusInfo/idc:statusMessage/text()',  
    p_ns=>'xmlns:idc="http://www.stellent.com/CheckIn/"');  
END;
```
PARSE_XML Function

Use this function to parse the response from a Web service returned as an XMLTYPE and return the value requested as a VARCHAR2.

Syntax

```sql
APEX_WEB_SERVICE.PARSE_XML (  
  p_xml               IN XMLTYPE,  
  p_xpath             IN VARCHAR2,  
  p_ns                IN VARCHAR2 default null )  
RETURN VARCHAR2;
```

Parameters

Table 22–8 describes the parameters available in the PARSE_XML function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_xml</td>
<td>The XML document as an XMLTYPE to parse.</td>
</tr>
<tr>
<td>p_xpath</td>
<td>The XPath expression to the desired node.</td>
</tr>
<tr>
<td>p_ns</td>
<td>The namespace to the desired node.</td>
</tr>
</tbody>
</table>

Example

The following example uses the make_request function to call a Web service and store the results in a local XMLTYPE variable. The parse_xml function is then used to pull out a specific node of the XML document stored in the XMLTYPE and stores it in a locally declared VARCHAR2 variable.

```sql
declare  
  l_envelope CLOB;  
  l_xml XMLTYPE;  
  l_movie VARCHAR2(4000);  
BEGIN  
  l_envelope := ' <?xml version="1.0" encoding="UTF-8"?>  
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  
xmlns:tns="http://www.ignyte.com/whatsshowing"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema">  
<soap:Body>  
<tns:GetTheatersAndMovies>  
<tns:zipCode>43221</tns:zipCode>  
<tns:radius>5</tns:radius>  
</tns:GetTheatersAndMovies>  
</soap:Body>  
</soap:Envelope>';  
  l_xml := apex_web_service.make_request(  
  p_action => 'http://www.ignyte.com/whatsshowing/GetTheatersAndMovies',  
  p_envelope => l_envelope );  
  l_movie := apex_web_service.parse_xml(  
  p_xml => l_xml,
```
p_xpath => '  
//GetTheatersAndMoviesResponse/GetTheatersAndMoviesResult/Theater/Movies/Movie/Name[1]',
  p_ns => ' xmlns="http://www.ignyte.com/whatsshowing"');

END;
PARSE_XML_CLOB Function

Use this function to parse the response from a Web service returned as an XMLTYPE and return the value requested as a CLOB.

Syntax

APEX_WEB_SERVICE.PARSE_XML_CLOB (  
  p_xml               IN XMLTYPE,  
  p_xpath             IN VARCHAR2,  
  p_ns                IN VARCHAR2 default null ) 
RETURN VARCHAR2;

Parameters

Table 22–9 describes the parameters available in the PARSE_XML_CLOB function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_xml</td>
<td>The XML document as an XMLTYPE to parse.</td>
</tr>
<tr>
<td>p_xpath</td>
<td>The XPath expression to the desired node.</td>
</tr>
<tr>
<td>p_ns</td>
<td>The namespace to the desired node.</td>
</tr>
</tbody>
</table>

Example

The following example uses the make_request function to call a Web service and store the results in a local XMLTYPE variable. The parse_xml function is then used to pull out a specific node of the XML document stored in the XMLTYPE and stores it in a locally declared VARCHAR2 variable

```
begin
  declare
    l_envelope CLOB;
    l_xml XMLTYPE;
    l_movie CLOB;

  l_envelope := ' <?xml version="1.0" encoding="UTF-8"?>
  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:tns="http://www.ignyte.com/whatsshowing"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <tns:GetTheatersAndMovies>
      <tns:zipCode>43221</tns:zipCode>
      <tns:radius>5</tns:radius>
    </tns:GetTheatersAndMovies>
  </soap:Body>
</soap:Envelope>';

  l_xml := apex_web_service.make_request( 
    p_url => 'http://www.ignyte.com/webservices/ignyte.whatsshowing.webservice/moviefunctions.asmx', 
    p_action => 'http://www.ignyte.com/whatsshowing/GetTheatersAndMovies', 
    pEnvelope => l_envelope );

  l_movie := apex_web_service.parse_xml_clob( 
    p_xml => l_xml, 
  );
end;
```
p_xpath => '
//GetTheatersAndMoviesResponse/GetTheatersAndMoviesResult/Theater/Movies/Movie/Name[1]',
    p_ns => ' xmlns="http://www.ignyte.com/whatsshowing" ');

END;
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:
- **apex namespace**
- **apex.event namespace**
- **apex.item namespace**
- **apex.navigation namespace**
- **apex.server namespace**

---

**Note:** Legacy JavaScript. Work has commenced in attempting to reduce the overall size of JavaScript that is loaded by Application Express when rendering a page. JavaScript functions that are no longer served on every page are gradually being moved to a legacy JavaScript file, which can be found in `/i/libraries/apex/legacy.js`.

When developing applications, a developer has the option to either include, or not include the legacy JavaScript functions. This is achieved by using the Include Legacy JavaScript property on the User Interface Attributes page under the application’s Shared Components.

Existing applications are migrated with this option enabled, for backward compatibility. To not include this legacy file, you need to go through the functions listed in the legacy file, and search your application and associated JavaScript files for any references to those files. If you are happy that there are no references to these functions, you can switch off including the legacy file and benefit from the slightly smaller library.

When developing new applications, the legacy file is included by default in all applications that use a Desktop User Interface Type. New applications that use a jQuery Mobile Smartphone User Interface Type do not include this file.

For both new and existing application development, Oracle recommends that you do not continue to use any of the functions in legacy.js, to reduce your dependency to this legacy JavaScript.
- apex.storage namespace
- apex.widget namespace
- Miscellaneous Javascript APIs
apex namespace

Use the apex namespace to store global variables and highly used functions in Application Express.

Global Variables

- **apex.gPageContext$** - Application Express variable that stores the current page context. The current page context is different depending on whether the page is a Desktop, or jQuery Mobile page. For Desktop, this is set to the document level. For jQuery Mobile, where pages are actually represented as DIV elements in the Browser DOM and multiple page DIVs can be loaded in the Browser DOM at one time, this is set to the DIV element representing the current page.

This is used to set the context for your jQuery selectors, to ensure that the selector is executing within the context of the correct page.

For example:

```javascript
jQuery( ".my_class", apex.gPageContext$ );
```

This selects all elements with a CSS class of **my_class**, in the context of the current page.

Topics:

- **apex.confirm**
- **apex.submit**
apex.confirm

The apex.confirm function displays a confirmation and depending on the user's choice either submits the page, or cancels a page submit. This function has 2 signatures, as described below.

**Topics:**
- apex.confirm(pMessage, pRequest)
- apex.confirm(pMessage, pOptions)

**apex.confirm(pMessage, pRequest)**
Displays a confirmation showing a message, pMessage, and depending on user's choice, submits a page setting request value, pRequest, or cancels page submit.

**Parameters**
- pMessage (string)
- pRequest (string)

**Example**
This example shows a confirmation dialog with the text 'Delete Department'. If the user chooses to proceed with the delete, the current page is submitted with a REQUEST value of 'DELETE'

```javascript
apex.confirm('Delete Department', 'DELETE');
```

**apex.confirm(pMessage, pOptions)**
Displays a confirmation showing a message (pMessage) and depending on user's choice, submits a page setting request values specified by (pOptions) or cancels page submit.

**Parameters**
- pMessage (string)
- pOptions (Object)
  where pOptions can contain the following properties:
  - **submitIfEnter** - If you only want to confirm when the ENTER key has been pressed, call apex.confirm in the event callback and pass the event object as this parameter.
  - **request** - The request value to set (defaults to null)
  - **set** - Object containing name/value pairs of items to be set on the page prior to submission (defaults to null).
  - **showWait** - Flag to control if a 'Wait Indicator' icon is displayed, which can be useful when running long page operations (Defaults to false).

**Return Values**
Boolean - If the submitIfEnter option is specified, a boolean value is returned. True is returned if the ENTER key was not pressed and false if the ENTER key was pressed. If submitIfEnter is not been specified, nothing is returned.
Example

This example shows a confirmation message with the 'Save Department?' text. If the user chooses to proceed with the save, the page is submitted with a REQUEST value of 'SAVE' and 2 page item values are set, P1_DEPTNO to 10 and P1_EMPNO to 5433.

```javascript
apex.confirm('Save Department?', {
    request: 'SAVE',
    set: {'P1_DEPTNO': 10, 'P1_EMPNO': 5433}
});
```
The `apex.submit` function submits the current page. This function has 2 signatures, as described below.

**Topics:**
- `apex.submit(pOptions)`
- `apex.submit(pRequest)`

### apex.submit(pOptions)

This function submits the page using the options specified in `pOptions`.

**Parameters**

- `pOptions (Object)`
  where `pOptions` can contain the following properties:
  - `submitIfEnter` - If you only want to submit when the ENTER key has been pressed, call `apex.submit` in the event callback and pass the event object as this parameter.
  - `request` - The request value to set (defaults to null)
  - `set` - Object containing name/value pairs of items to be set on the page prior to submission (defaults to null).
  - `showWait` - Flag to control if a 'Wait Indicator' icon is displayed, which can be useful when running long page operations (Defaults to false).

**Return Values**

Boolean - If the `submitIfEnter` option is specified, a boolean value is returned. True is returned if the ENTER key was not pressed and false if the ENTER key was pressed. If `submitIfEnter` is not been specified, nothing is returned.

### Example

This example submits the page with a REQUEST value of 'DELETE' and 2 page item values are set, `P1_DEPTNO` to 10 and `P1_EMPNO` to 5433. During submit a wait icon is displayed as visual indicator for the user as well.

```javascript
apex.submit({
  request: 'DELETE',
  set: {'P1_DEPTNO': 10, 'P1_EMPNO': 5433});
```

### apex.submit(pRequest)

This function submits the page setting the Application Express Request value `pRequest`.

**Parameters**

- `pRequest (String)`

### Example

Submits the current page with a REQUEST value of 'DELETE'.

```javascript
apex.submit( 'DELETE' );
```
Use the `apex.event` namespace to store all event related functions of Oracle Application Express.

**Topics:**

- `apex.event.trigger(pSelector,pEvent,pData)`
**apex.event.trigger(pSelector,pEvent,pData)**

Given a jQuery selector, jQuery object or DOM Node the specified pEvent is triggered. pEvent can be a browser event like "click" or "change" but also a custom event like "slidechange". This function should only be used to trigger events that are handled by the dynamic action framework. Otherwise, custom events registered by plug-ins installed in your application or any event that is already exposed in dynamic actions can be compromised.

**Return Value**

Boolean

**Parameters**

- **pSelector** (jQuery selector | jQuery object | DOM Node)
- **pEvent** (String)
- **pData** (Object)
This is the APEX page item namespace. This namespace holds all single item functions. These functions assume that these are APEX generated page items.

**Topics:**
- `apex.item( pNd )`
- `apex.item( pNd ).addValue( pValue )`
- `apex.item( pNd ).disable()`
- `apex.item( pNd ).enable()`
- `apex.item( pNd ).getValue()`
- `apex.item( pNd ).hide( pHideRow )`
- `apex.item( pNd ).isEmpty()`
- `apex.item( pNd ).setFocus()`
- `apex.item( pNd ).setStyle( pPropertyName, pPropertyValue )`
- `apex.item( pNd ).setValue(pValue, pDisplayValue, p suppressChangeEvent)`
The `apex.item` API provides a single interface for item related functionality of Application Express. The API returns an Application Express item object, which can then be used to access item related functions and properties.

Plug-in developers can override much of the behavior defined in the `apex.item` namespace, by calling `apex.widget.initPageItem` with their overrides. See the documentation on "`apex.widget.initPageItem( pName, pOptions)`" on page 23-40 for more details.

**Return Values**

Table 23–1, "Return Values for `apex.item( pNd )`" describes the return values for this function.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Object)</td>
<td>Returns the Application Express item object, which is used to access item specific functions. For example <code>getValue</code>, <code>setValue</code>, and so on.</td>
</tr>
</tbody>
</table>

**Parameters**

Table 23–2, "Parameters for `apex.item( pNd )`" describes the parameters available for this function.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Optional/Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pNd</td>
<td>(DOM Node</td>
<td>String)</td>
<td>Required</td>
<td>Application Express item name or DOM node.</td>
</tr>
</tbody>
</table>

**Examples**

This will not be used by itself, rather it is used to access item specific functions and properties, as documented in the proceeding APIs.
**apex.item( pNd ).addValue( pValue )**

Adds a value to an Application Express item that supports multiple values.

**Return Values**
None.

**Parameters**
Table 23–3, "Parameters for apex.item( pNd ).addValue( pValue )" describes parameters available for this function.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Optional/Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pValue</td>
<td>(String)</td>
<td>Required</td>
<td></td>
<td>The value to be set.</td>
</tr>
</tbody>
</table>

**Examples**
In this example, the page item called 'P1_ITEM' will have the value '100' added to the values currently selected.

```javascript
apex.item( 'P1_ITEM' ).addValue('100');
```
Disables the Application Express item value, taking into account the item type, making it unavailable for edit.

**Return Values**
None.

**Parameters**
None.

**Examples**
In this example, the page item called 'P1_ITEM' will be disabled and unavailable for edit.

```javascript
apex.item( "P1_ITEM" ).disable();
```
apex.item(pNd).enable()

Enables the Application Express item value, taking into account the item type, making it available for edit.

**Return Values**
None.

**Parameters**
None.

**Examples**
In this example, the page item called 'P1_ITEM' will be enabled and available for edit.

```javascript
apex.item('P1_ITEM').enable();
```
apex.item(pNd).getValue()

Returns the current value of an Application Express item on a page, taking into account the current item type. This does not return the item's current value from session state (although that could be the same), rather it will return the value as it is on the current page.

There are 2 related functions to .getValue(). $v(pNd)$ which returns an item's value, but in the format it will be posted. This will either be a single value, or if the item supports multiple values, will be a ':' colon separated list of values. There is also the $v2(pNd)$ function, which is just a shortcut to .getValue() and returns either a single value, or array of values.

Return Values
Table 23–4, "Return Values for apex.item(pNd).getValue()" describes the return values for this function.

Table 23–4 Return Values for apex.item(pNd).getValue()
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(String</td>
<td>Array)</td>
</tr>
</tbody>
</table>

Parameters
None.

Examples
In this example, the current value of the page item called 'P1_ITEM' will be shown in an alert.

```javascript
alert( "P1_ITEM value = " + apex.item( "P1_ITEM" ).getValue()  );
```
apex.item( pNd ).hide( pHideRow )

Hides the Application Express item value, taking into account the item type. When using the .hide() function, it is important to understand the following:

- If the item being hidden is rendered on a page using table layout (meaning the page references a page template with Grid Layout Type set to 'HTML Table'), and the call to hide has specified to hide the entire table row (pHideRow = TRUE), then it is assumed that everything pertaining to the item is contained in that row, and the entire row will be hidden.

- If the item being hidden is rendered on a page using table layout, and the call to hide has specified not to hide the entire table row (pHideRow = FALSE, or not passed), then the function will attempt to hide the item's label, where the FOR attribute matches the ID of the item.

- If the item being hidden is rendered on a page using grid layout (meaning the page references a page template with Grid Layout Type set to either 'Fixed Number of Columns', or 'Variable Number of Columns'), and the item references a Label template that includes a Field Container element with a known ID (so where the Field Container > Before Label and Item attribute includes an HTML element with id="#CURRENT_ITEM_CONTAINER_ID#"), then it is assumed that everything pertaining to the item is contained in the Field Container, and this will be hidden.

Return Values
None.

Parameters
Table 23–5, "Parameters for apex.item( pDN ).hide( pHideRow )" describes the parameters available for this function.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Optional/Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pHideRow</td>
<td>(String</td>
<td>Array)</td>
<td>Optional</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Examples
In this example, the page item called P1_ITEM will be hidden. If P1_ITEM is on a page using grid layout and the item references a Label template that includes a Field Container element with a known ID (as detailed above), then that container element will be hidden. Otherwise just the item and its corresponding label will be hidden.

apex.item( 'P1_ITEM' ).hide();

In this example, the page item called P1_ITEM's nearest containing table row (TR) will be hidden (as pHideRow = TRUE). Hiding the entire table row should only be used on a page using table layout. If P1_ITEM is on a page using grid layout, then passing pHideRow = TRUE will not work and could result in adverse consequence for the page layout, where an incorrect table row is wrongly hidden.

apex.item( 'P1_ITEM' ).hide(TRUE);
Returns true or false if an Application Express item is empty and will consider any whitespace including a space, a tab or a form-feed, as empty. This will also respect if the item type uses a List of Values, and a 'Null Return Value' has been defined in the List of Values. In that case, the 'Null Return Value' will be used to assert if the item is empty. In this case, the DOM node returned is the nearest ancestor of \$\text{pNd}\$ that has a node name of \$\text{pToTag}\$ and optionally a matching class. Also it returns false if \$\text{pNd}\$ is not found or if there is no \$\text{pToTag}\$ ancestor.

### Return Values

Table 23–6, "Parameters for `apex.item( pNd ).isEmpty()`" describes the return values for this function.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Boolean)</td>
<td>Returns true or false if an Application Express item is empty.</td>
</tr>
</tbody>
</table>

### Parameters

None.

### Examples

In this example, the call to .isEmpty() determines if the page item called 'P1_ITEM' is null, and if so displays an alert.

```javascript
if( apex.item( "P1_ITEM" ).isEmpty() ) {
    alert( "P1_ITEM empty!" );
}
```
apex.item( pNd ).setFocus()

Places user focus on the Application Express item, taking into account how specific items are designed to receive focus.

**Return Values**
None.

**Parameters**
None.

**Examples**
In this example, user focus is set to the page item called 'P1_ITEM'.

```javascript
apex.item( 'P1_ITEM' ).setFocus();
```
apex.item( pNd ).setStyle( pPropertyName, pPropertyValue )

Sets a style for the Application Express item, taking into account how specific items are designed to be styled.

**Return Values**
None.

**Parameters**
Table 23–7, "Parameters for apex.item( pNd ).setStyle( pPropertyName, pPropertyValue )" describes the parameters available for this function.

Table 23–7 Parameters for apex.item( pNd ).setStyle( pPropertyName, pPropertyValue )

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Optional/Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pPropertyName</td>
<td>(CSS Property Name)</td>
<td>Required</td>
<td></td>
<td>The CSS property name that will be set.</td>
</tr>
<tr>
<td>pPropertyValue</td>
<td>(CSS Property Value)</td>
<td>Required</td>
<td></td>
<td>The value used to set the CSS property.</td>
</tr>
</tbody>
</table>

**Examples**
In this example, the CSS property 'color' will be set to 'red' for the page item called 'P1_ITEM'.

apex.item( "P1_ITEM" ).setStyle( "color", "red" );
apex.item( pNd ).setValue(pValue, pDisplayValue, pSuppressChangeEvent)

Sets the Application Express item value, taking into account the item type. This function sets the current value of an Application Express item on the page, not the item's current value in session state. It also allows for the caller to suppress the 'change' event for the item being set, if desired.

See the $s( pNd, pValue, pDisplayValue, pSuppressChangeEvent ) function for a shortcut to .setValue().

Return Values
None.

Parameters
Table 23-8, "Parameters for apex.item (pNd ).setValue( pValue, pDisplayValue, pSuppressChangeEvent)" describes the parameters available for this function.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Optional/Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pValue</td>
<td>(String</td>
<td>Array)</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>pDisplayValue</td>
<td>(String)</td>
<td>Optional</td>
<td></td>
<td>Optional parameter used to set the page item's display value, in the case where the return value is different. For example for the item type &quot;Popup LOV&quot;, with the attribute &quot;Input Field&quot; = &quot;Not Enterable, Show Display Value and Store Return Value&quot;, this value sets the &quot;Input Field&quot;. The value of pValue is then used to set the item's hidden return field.</td>
</tr>
<tr>
<td>pSuppressChangeEvent</td>
<td>(Boolean)</td>
<td>Optional</td>
<td>FALSE</td>
<td>Pass TRUE to prevent the 'change' event from being triggered, for the item being set.</td>
</tr>
</tbody>
</table>

Examples
In this example, the value of the page item called P1_ITEM will be set to "10". As pSuppressChangeEvent has not been passed, the default behavior of the 'change' event triggering for P1_ITEM will occur.

apex.item( 'P1_ITEM' ).setValue( "10" );

In this example P1_ITEM is a "Popup LOV" page item with the attribute "Input Field" = "Not Enterable, Show Display Value and Store Return Value", set to "Input Field". The display value of P1_ITEM will be set to "SALES" and the hidden return value will be set to "10". As 'true' has been passed for the pSuppressChangeEvent parameter, the 'change' event will not trigger for the P1_ITEM item.
apex.item( "P1_ITEM" ).setValue( "10", "SALES", true );
**apex.item( pNd ).show( pShowRow )**

Shows the Application Express item value, taking into account the item type. When using the `show()` function, it is important to understand the following:

- If the item being shown is rendered on a page using table layout (meaning the page references a page template with Grid Layout Type set to 'HTML Table'), and the call to show has specified to show the entire table row (`pShowRow = TRUE`), then it is assumed that everything pertaining to the item is contained in that row, and the entire row will be shown.

- If the item being shown is rendered on a page using table layout, and the call to show has specified not to show the entire table row (`pShowRow = FALSE`, or not passed), then the function will attempt to show the item's label, where the FOR attribute matches the ID of the item.

- If the item being shown is rendered on a page using grid layout (meaning the page references a page template with Grid Layout Type set to either 'Fixed Number of Columns', or 'Variable Number of Columns'), and the item references a Label template that includes a Field Container element with a known ID (so where the Field Container > Before Label and Item attribute includes an HTML element with id="#CURRENT_ITEM_CONTAINER_ID"), then it is assumed that everything pertaining to the item is contained in the Field Container, and this will be shown.

**Return Values**

None.

**Parameters**

Table 23–9, "Parameters for apex.item( pNd ).show( pShowRow )" describes the parameters for this function.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Optional/Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pShowRow</td>
<td>(String</td>
<td>Array)</td>
<td>Optional</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

**Examples**

In this example, the page item called P1_ITEM will be shown. If P1_ITEM is on a page using grid layout and the item references a Label template that includes a Field Container element with a known ID (as detailed above), then that container element will be shown. Otherwise just the item and its corresponding label will be shown.

```javascript
apex.item( 'P1_ITEM' ).show();
```

In this example, the page item called P1_ITEM's nearest containing table row (TR) will be shown (as `pShowRow = TRUE`). Showing the entire table row should only be used on a page using table layout. If P1_ITEM is on a page using grid layout, then passing `pShowRow = TRUE` will not work and could result in adverse consequence for the page layout, where an incorrect table row is wrongly shown.

```javascript
apex.item( 'P1_ITEM' ).show(TRUE);
```
Use the `apex.navigation` namespace to store popup and redirect related functions of Oracle Application Express.

**Topics:**
- `apex.navigation.popup.close(pThat,pValue)`
apex.navigation.popup.close(pThat,pValue)

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

Return Value
Not applicable.

Parameters
pValue (string)
pThat (DOM node | string ID)
Use the `apex.server` namespace to store all AJAX functions to communicate with the server part of Oracle Application Express.

Topics:
- `apex.server.plugin(pAjaxIdentifier,pData,pOptions)`
- `apex.server.pluginUrl( pAjaxIdentifier, pData )`
- `apex.server.process( pAjaxIdentifier, pData, pOptions )`
apex.server.plugin(pAjaxIdentifier,pData,pOptions)

This function calls the PL/SQL AJAX function which has been defined for a plug-in. This function is a wrapper of the jQuery.ajax function and supports all the settings the jQuery function provides, with additional Application Express specific features.
Parameters
### Table 23–10  apex.server.plugin(pAjaxIdentifier,pData,pOptions) Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Optional/Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pAjaxIdentifier</td>
<td>(String)</td>
<td>Required</td>
<td>Use the value returned by the PL/SQL package apex_plugin.get_ajax_identifier to identify your plug-in.</td>
</tr>
<tr>
<td>pData</td>
<td>{Object}</td>
<td>Optional</td>
<td>Object which can optionally be used to send additional values to be sent with the AJAX request. The special attribute pageItems which can be of type jQuery selector, jQuery or DOM object or array of item names identifies the page items which should be included in the URL. But you can also set additional parameters that the wwv_flow.show procedure provides (for example you can set the scalar parameters x01 - x10 and the arrays f01 - f20).</td>
</tr>
<tr>
<td>pOptions</td>
<td>{Object}</td>
<td>Optional</td>
<td>Object which can optionally be used to set additional options used by the AJAX. It supports the following optional Application Express specific attributes: refreshObject - jQuery selector, jQuery- or DOM object which identifies the DOM element for which the apexbeforerefresh and apexafterrefresh events are fired. refreshObjectData - Specify data that is internally passed by the apexbeforerefresh and apexafterrefresh event triggering code, so that any handlers defined for these events can access this data. In Dynamic Actions defined for the Before Refresh or After Refresh events, this can be accessed from JavaScript via the this.data property. For custom jQuery event handlers, this can be accessed via the pData parameter of the event handler. clear - JavaScript function used to clear the DOM after the apexbeforerefresh event has fired and before the actual AJAX call is triggered. loadingIndicator - jQuery selector, jQuery- or DOM object which identifies the DOM element where the loading indicator should be displayed next to it. loadingIndicator can also be a function which gets the loading Indicator as jQuery object and has to return the jQuery reference to the created loading indicator. For example: function( pLoadingIndicator ) { return pLoadingIndicator.prependTo( apex.jQuery( &quot;td.shuttleControl&quot;, gShuttle )) } loadingIndicatorPosition - 4 options to define the position of the loading indicator displayed. Only considered if the value passed to loadingIndicator is not a function. ■ before: Displays before the DOM element(s) defined by loadingIndicator. ■ after: Displays after the DOM element(s) defined by loadingIndicator. ■ prepend: Displays inside at the beginning of the DOM element(s) defined by loadingIndicator. ■ append: Displays inside at the end of the DOM element(s) defined by loadingIndicator. See Also: See jQuery documentation of jQuery.ajax for all other available attributes. The attribute dataType is defaulted to json.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><a href="http://docs.jquery.com/">http://docs.jquery.com/</a></td>
</tr>
</tbody>
</table>
Return Values

Table 23–11  Return Value

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Returns a jqXHR object.</td>
</tr>
<tr>
<td></td>
<td>See Also: See the jQuery documentation for more details on this object:</td>
</tr>
<tr>
<td></td>
<td><a href="http://docs.jquery.com/">http://docs.jquery.com/</a></td>
</tr>
</tbody>
</table>

Example

This call to `apex.server.plugin` sets the scalar value `x01` to `test` (which can be accessed from PL/SQL using `apex_application.g_x01`) and sets the page item’s `P1_DEPTNO` and `P1_EMPNO` values in session state (using jQuery selector syntax). The `P1_MY_LIST` item is used as the element for which the `apexbeforerefresh` and `apexafterrefresh` events are fired. `P1_MY_LIST` is used as the element for which to display the loading indicator next to. The success callback is stubbed out and is used for developers to add their own code that fires when the call successfully returns.

The `pData` parameter to the success callback will contain any response sent from the call.

```javascript
apex.server.plugin('lAjaxIdentifier', {
  x01: "test",
  pageItems: '#P1_DEPTNO,#P1_EMPNO'
}, {
  refreshObject: '#P1_MY_LIST',
  loadingIndicator: '#P1_MY_LIST',
  success: function(pData) { ... do something here ... }
});
```
apex.server.pluginUrl( pAjaxIdentifier, pData )

This function returns the URL to issue a GET request to the PL/SQL AJAX function which has been defined for a plug-in.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Optional/Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pAjaxIdentifier</td>
<td>(String)</td>
<td>Required</td>
<td></td>
<td>Use the value returned by the PL/SQL package apex_plugin.get_ajax_identifier to identify your plug-in.</td>
</tr>
<tr>
<td>pData</td>
<td>[Object]</td>
<td>Optional</td>
<td></td>
<td>Object which can optionally be used to set additional values which are included into the URL. The special attribute pageItems which can be of type jQuery selector, jQuery or DOM object or array of item names identifies the page items which are included in the URL. You can also set additional parameters that the wwv_flow.show procedure provides (for example you can set the scalar parameters x01 - x10 and the arrays f01 - f20).</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(String)</td>
<td>The URL to issue the GET request.</td>
</tr>
</tbody>
</table>

Example

This call to apex.server.pluginUrl returns a URL to issue a GET request to the PL/SQL AJAX function which has been defined for a plug-in, where the URL sets the scalar value x01 to test (which can be accessed from PL/SQL using apex_application.g_x01) and will also set the page item's P1_DEPTNO and P1_EMPNO values in session state (using jQuery selector syntax).

```javascript
var lUrl;
lUrl = apex.server.pluginUrl( { pAjaxIdentifier, x01: 'test', pageItems: '#P1_DEPTNO,#P1_EMPNO' });
```
This function calls a PL/SQL on-demand process defined on page or application level. This function is a wrapper of the jQuery.ajax function and supports all the settings the jQuery function provides but provides additional Application Express features.
Parameters
apex.server.process( pAjaxIdentifier, pData, pOptions )
### Table 23–14  `apex.server.process` Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Optional/Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pAjaxIdentifier</td>
<td>(String)</td>
<td>Required</td>
<td></td>
<td>Use the value returned by the PL/SQL package <code>apex_plugin.get_ajax_identifier</code> to identify your plug-in.</td>
</tr>
<tr>
<td>pData</td>
<td>{Object}</td>
<td>Optional</td>
<td></td>
<td>Object which can optionally be used to send additional values to be sent with the AJAX request. The special attribute <code>pageItems</code> which can be of type jQuery selector, jQuery or DOM object or array of item names identifies the page items which are included in the URL. You can also set additional parameters that the <code>wwv_flow.show</code> procedure provides (for example you can set the scalar parameters x01 - x10 and the arrays f01 - f20).</td>
</tr>
</tbody>
</table>
Table 23–14  (Cont.) apex.server.process Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Optional/Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pOptions</td>
<td>{Object}</td>
<td>Optional</td>
<td></td>
<td>Object which can optionally be used to set additional options used by the AJAX.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>It supports the following optional Application Express specific attributes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>refreshObject - jQuery selector, jQuery- or DOM object which identifies the DOM element for which the apexbeforerefresh and apexafterrefresh events are fired.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>refreshObjectData - Specify data that is internally passed by the apexbeforerefresh and apexafterrefresh event triggering code, so that any handlers defined for these events can access this data. In Dynamic Actions defined on the Before Refresh or After Refresh events, this can be accessed from JavaScript via the this.data property. For custom jQuery event handlers, this can be accessed via the pData parameter of the event handler.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>clear - JavaScript function used to clear the DOM after the apexbeforerefresh event has fired and before the actual AJAX call is triggered.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>loadingIndicator - jQuery selector, jQuery- or DOM object which identifies the DOM element where the loading indicator should be displayed next to it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>loadingIndicator can also be a function which gets the loading Indicator as jQuery object and has to return the jQuery reference to the created loading indicator. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>function( pLoadingIndicator ) {</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>return lLoadingIndicator.prependTo( apex.jQuery(</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'td.shuttleControl', gShuttle ))</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>loadingIndicatorPosition - 4 options to define the position of the loading indicator displayed. Only considered if the value passed to loadingIndicator is not a function.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>before: Displays before the DOM element(s) defined by loadingIndicator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>after: Displays after the DOM element(s) defined by loadingIndicator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>prepend: Displays inside at the beginning of the DOM element(s) defined by loadingIndicator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>append: Displays inside at the end of the DOM element(s) defined by loadingIndicator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>See Also: See jQuery documentation of $.ajax for all other available attributes. The attribute dataType is defaulted to json.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>See the jQuery documentation for more details on this object:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><a href="http://docs.jquery.com/">http://docs.jquery.com/</a></td>
</tr>
</tbody>
</table>
Return Values

Table 23–15  Return Value

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Object]</td>
<td>Returns a jqXHR object. See the jQuery documentation for more details on this object: <a href="http://docs.jquery.com/">http://docs.jquery.com/</a></td>
</tr>
</tbody>
</table>

Example

This call to `apex.server.process` calls an on-demand process called `MY_PROCESS` and sets the scalar value `x01` to `test` (which can be accessed from PL/SQL using `apex_application.g_x01`) and sets the page item's `P1_DEPTNO` and `P1_EMPNO` values in session state (using jQuery selector syntax). The success callback is stubbed out so that developers can add their own code that fires when the call successfully returns.

Note: The `pData` parameter to the success callback contains any response sent from the call.

```javascript
apex.server.process ( "MY_PROCESS", {
    x01: 'test',
    pageItems: '#P1_DEPTNO,#P1_EMPNO'
}, {
    success: function( pData ) { ... do something here ... }
} );
```
Use the `apex.storage` namespace to store storage related functions of Oracle Application Express.

**Topics:**
- `apex.storage.getCookie(pName)`
- `apex.storage.setCookie(pName,pValue)`
apex.storage.getCookie(pName)

Returns the value of cookie name (pName).

Return Value
Not applicable.

Parameters
pName {String}
apex.storage.setCookie(pName,pValue)

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

**Return Value**
Not applicable.

**Parameters**
- pName (String)
- pValue (String)
Use the `apex.widget` namespace to store all the general purpose widget related functions of Oracle Application Express.

Topics:
- `apex.widget.initPageItem(pName, pOptions)`
apex.widget.initPageItem( pName, pOptions)

Given the Application Express page item name or the DOM node, different callbacks and properties can be registered for a page item. This is necessary to seamlessly integrate a plug-in item type with the built-in page item related client-side functionality of Application Express.

For more information about implementing plug-ins, see "Implementing Plug-ins" in Oracle Application Express Application Builder User’s Guide:

For samples authored by Oracle, see the plug-in repository, on OTN:

http://apex.oracle.com/plugins

Return Values
None.

Parameters
Table 23–16, "Parameters for apex.widget.initPageItem( pName, pOptions )" describes the available parameters for this function.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Optional/Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pName</td>
<td>(DOM Node</td>
<td>String)</td>
<td>Required</td>
<td></td>
</tr>
</tbody>
</table>
| pOptions  | (Object)              | Required          |         | Supports many properties to specify callbacks and certain item-specific values. Specifying any of these properties will override the default behavior of Application Express for that particular property. See Table 23–17, "Properties for the pOptions parameter" for pOption property details. pOptions can contain one of the following properties:
  ■ getValue()
  ■ setValue( pValue, pDisplayValue )
  ■ enable()
  ■ disable()
  ■ show()
  ■ hide()
  ■ addValue()
  ■ nullValue()
  ■ setFocusTo
  ■ setStyleTo
  ■ afterModify()
  ■ loadingIndicator( pLoadingIndicator$ ) |
Table 23–17  Properties for the pOptions parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| getValue()           | Specify a function for getting the item's value, which overrides the default page item handling. Ensuring the item returns its value correctly means certain item related client-side functionality of Application Express still works, for example in Dynamic Actions to evaluate a When condition on the item, or when calling the JavaScript function $v to get the item’s value.  

See "apex.item( pNd ).getValue()" on page 23-14, for details on how to define this function.  

Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check by calling apex.item( pNd ).getValue(); to see if that returns the item value correctly.  

| setValue( pValue, pDisplayValue ) | Specify a function for setting the item's value, which overrides the default page item handling. Ensuring the item can set its value correctly means certain item related client-side functionality of Application Express still works, for example when using the Set Value action of a Dynamic Action to set the item’s value, or when calling the JavaScript function $s to set the item’s value.  

Note: Even if this function is defined, the default handling always handles the logic associated with the .afterModify() function and the pSuppressChangeEvent parameter, so that is outside the scope of what a plug-in developer is concerned with.  

See the "apex.item( pNd ).setValue(pValue, pDisplayValue, pSuppressChangeEvent)" on page 23-19, for details on how to define this function.  

Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check by calling apex.item( pNd ).setValue( pValue ); to see if that sets the item value correctly.  

| enable()            | Specify a function for enabling the item, which overrides the default page item handling. This could be useful for example where the item consists of compound elements which also need enabling, or if the item is based on a widget that already has its own enable method that you want to reuse. Ensuring the item can enable correctly means certain item related client-side functionality of Application Express still works, for example when using the Enable action of a Dynamic Actions, to enable the item.  

Note: Even if this function is defined, the default handling always handles the logic associated with the .afterModify() function, so that is outside the scope of what a plug-in developer is concerned with.  

See the "apex.item( pNd ).enable()" on page 23-13, for details on how to define this function.  

Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check by calling apex.item( pNd ).enable(); to see if that enables the item satisfactorily.  


disable()

Specify a function for disabling the item, which overrides the default page item handling. This could be useful for example where the item consists of compound elements which also need disabling, or if the item is based on a widget that already has its own disable method that you want to reuse. Ensuring the item can disable correctly means certain item related client-side functionality of Application Express still works, for example when using the Disable action of a Dynamic Action to disable the item.

Note: Even if this function is defined, the default handling always handles the logic associated with the .afterModify() function, so that is outside the scope of what a plug-in developer is concerned with.

See the "apex.item( pNd ).disable()" on page 23-12, for details on how to define this function.

Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check by calling apex.item( pNd ).disable(); to see if that disables the item satisfactorily.

show()

Specify a function for showing the item, which overrides the default page item handling. This is useful for example where the item consists of compound elements which also need showing, or if the item is based on a widget that already has its own show method that you want to reuse. Ensuring the item can show correctly means certain item related client-side functionality of Application Express still works, for example when using the Show action of a Dynamic Action, to show the item.

See the "apex.item( pNd ).show( pShowRow )" on page 23-21, for details on how to define this function.

Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check by calling apex.item( pNd ).show(); to see if that shows the item satisfactorily.

hide()

Specify a function for hiding the item, which overrides the default page item handling. This could be useful for example where the item consists of compound elements which also needs hiding, or if the item is based on a widget that already has its own hide method that you want to reuse. Ensuring the item can hide correctly means certain item related client-side functionality of Application Express still works, for example when using the Hide action of a Dynamic Action, to hide the item.

See the "apex.item( pNd ).hide( pHideRow )" on page 23-15, for details on how this function should be defined.

Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check by calling apex.item( pNd ).hide(); to see if that hides the item satisfactorily.
addValue()

Specify a function for adding a value to the item, where the item supports multiple values. Currently there is no client-side functionality of Application Express dependent on this. There is also no default page item handling.

Note: Even if this function is defined, the default handling always handles the logic associated with the .afterModify() function, so that is outside the scope of what a plug-in developer is concerned with.

See the "apex.item( pNd ).addValue( pValue )" on page 23-11, for details on how this function should be defined.

nullValue

Specify a value that to be used to determine if the item is null. This is used when the item supports definition of a List of Values, where a developer can define a Null Return Value for the item and where the default item handling needs to know this in order to assert if the item is null or empty. This can be done by following these steps:

1. From the Render function in the plug-in definition, emit the value stored in p_item.lov_null_value as part of the item initialization JavaScript code that fires when the page loads. For example:

   /* Assumes that you have some JavaScript function called 'com_your_company_your_item' that accepts 2 parameters, the first being the name of the item and the second being an object storing properties (say pOptions) required by the item's client side code. */

   apex_javascript.add_onload_code {
     p_code => 'com_your_company_your_item('||
       apex_javascript.add_value(
         apex_plugin_util.page_item_
         names_to_jquery(p_item.name)||', {'||
         apex_javascript.add_attribute('lovNullValue', p_item.lov_null_value, false, false)||
         '});' );
   }

2. Then, in the implementation of com_your_company_your_item( pName, pOptions ) you have the value defined for the specific item's Null Return Value in the pOptions.lovNullValue property. This can then be used in your call to apex.widget.initPageItem, to set the nullValue property.

   Ensuring the nullValue property is set means certain item related client-side functionality of Application Express still works, for example, in Dynamic Actions to correctly evaluate an is null or is not null when condition on the item, or when calling the JavaScript function apex.item( pNd ).isEmpty() to determine if the item is null.

   See the "apex.item( pNd ).isEmpty()" on page 23-16, for further details of this API.
setFocusTo

Specify the element to receive focus, when focus is set to the item using the `apex.item(pNd).setFocus()` API. This can be defined as either a jQuery selector, jQuery or DOM object which identifies the DOM element, or a function that returns a jQuery object referencing the element. This can be useful when the item consists of compound elements, and you do not want focus to go to the element that has an ID matching the item name, which is the default behavior. For example, the native item type Popup LOV when the attribute Input Field is set to Not enterable, Show Display Value and Store Return Value renders a disabled input field as the main element with an ID matching the item name and a popup selection icon next to the input. In this case, because you do not want focus to go to the disabled input, use the `setFocusTo` item property and set that to the popup selection icon.

Ensuring the item sets focus correctly means certain item related client-side functionality of Application Express still works, for example when using the Set Focus action of a Dynamic Action to set focus to the item, when users follow the Go to Error link that displays in a validation error message to go straight to the associated item, or when the item is the first item on a page and the developer has the page level attribute Cursor Focus set to First item on page.

See the "`apex.item(pNd).setFocus()`" on page 23-17, for further details of this API.

Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check this by adding the item as the first item on a page, where the page has the page attribute Cursor Focus set to First item on page, and then running the page. The item receives focus.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setFocusTo</td>
<td>Specify the element to receive focus, when focus is set to the item using the <code>apex.item(pNd).setFocus()</code> API. This can be defined as either a jQuery selector, jQuery or DOM object which identifies the DOM element, or a function that returns a jQuery object referencing the element. This can be useful when the item consists of compound elements, and you do not want focus to go to the element that has an ID matching the item name, which is the default behavior. For example, the native item type Popup LOV when the attribute Input Field is set to Not enterable, Show Display Value and Store Return Value renders a disabled input field as the main element with an ID matching the item name and a popup selection icon next to the input. In this case, because you do not want focus to go to the disabled input, use the <code>setFocusTo</code> item property and set that to the popup selection icon. Ensuring the item sets focus correctly means certain item related client-side functionality of Application Express still works, for example when using the Set Focus action of a Dynamic Action to set focus to the item, when users follow the Go to Error link that displays in a validation error message to go straight to the associated item, or when the item is the first item on a page and the developer has the page level attribute Cursor Focus set to First item on page. See the &quot;<code>apex.item(pNd).setFocus()</code>&quot; on page 23-17, for further details of this API. Note: You should first check if the default handling of Application Express works for the item, because in that case you do not need to specify this. You can check this by adding the item as the first item on a page, where the page has the page attribute Cursor Focus set to First item on page, and then running the page. The item receives focus.</td>
</tr>
</tbody>
</table>
Examples
The following example shows a call to `apex.widget.initPageItem` with all the available callbacks and properties passed.

```javascript
apex.widget.initPageItem( "P100_COMPANY_NAME", {  
    getValue:   function() {  
        var lValue;  
        // code to determine lValue based on the item type.  
        return lValue;  
    },  
    setValue:   function( pValue, pDisplayValue ) {  
        // code that sets pValue and pDisplayValue (if required), for the item
```
apex.widget.initPageItem(pName, pOptions)

type
},
  enable:  function() {
    // code that enables the item type
  },
  disable: function() {
    // code that disables the item type
  },
  show:    function() {
    // code that shows the item type
  },
  hide:    function() {
    // code that hides the item type
  },
  addValue: function(pValue) {
    // code that adds pValue to the values already in the item type
  },
  nullValue: "<null return value for the item>",
  setFocusTo: $("<some jQuery selector>"),
  setStyleTo: $("<some jQuery selector>"),
  afterModify: function(){
    // code to always fire after the item has been modified (value set, enabled, etc.)
  },
  loadingIndicator: function(pLoadingIndicator$){
    // code to add the loading indicator in the best place for the item
    return pLoadingIndicator$;
  }
});
Miscellaneous Javascript APIs

This section contains all the miscellaneous, non-namespace APIs of Oracle Application Express, including shortcuts to highly used functions.

Topics:
- $x(pNd)
- $v(pNd)
- $v2(pNd)
- $s(pNd, pValue, pDisplayValue, pSuppressChangeEvent)
- $u_Carray(pNd)
- $u_Narray(pNd)
- $nv1(pTest, pDefault)
- $x_Style(pNd, pStyle, pString)
- $x_Hide(pNd)
- $x_Show(pNd)
- $x_Toggle(pNd)
- $x_Remove(pNd)
- $x_Value(pNd, pValue)
- $x_UpTill(pNd, pToTag)
- $x_ItemRow(pNd, pFunc)
- $x_HideItemRow(pNd)
- $x_ShowItemRow(pNd)
- $x_ToggleItemRow(pNd)
- $x_HideAllExcept(pNd, pNdArray)
- $x_HideSiblings(pNd)
- $x_ShowSiblings(pNd)
- $x_Class(pNd, pClass)
- $x_SetSiblingsClass(pNd, pClass, pNdClass)
- $x_ByClass(pClass, pNd, pTag)
- $x_ShowAllByClass(pNd, pClass, pTag)
- $x_ShowChildren(pNd)
- $x_HideChildren(pNd)
- $x_disableItem(pNd, pTest)
- $f_get_empty(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) [Deprecated]
$x_ToggleWithImage(pThis,pNd)
$x_SwitchImageSrc(pNd, pSearch, pReplace)
$x_CheckImageSrc(pNd, pSearch)
$u_SubString(pText,pMatch)
html_RemoveAllChildren(pNd)
$v_IsEmpty(pThis) [Deprecated]
html_SetSelectValue(pId,pValue)
addLoadEvent(pFunction)
$f_Swap(pThis,pThat)
sSubmitEnter(pNd,e) [Deprecated]
$f_SetValueSequence(pArray,pMultiple)
$dom_AddTag(pThis, pTag, pText)
$tr_AddTD(pThis,pText)
$dom_AddInput(pThis,pType,pId,pName,pValue)
$dom_MakeParent(p_Node,p_Parent)
$x_RowHighlight(pThis, pColor)
$x_RowHighlightOff(pThis)
$v_Upper(pNd)
$d_Find(pThis,pString,pTags,pClass)
setReturn(p_R,p_D) [Deprecated]
$f_First_field(pNd)
- `GetCookie (pName) [Deprecated]`
- `SetCookie (pName,pValue) [Deprecated]`
Given a DOM node or string ID (pNd), this function returns a DOM node if the element is on the page, or returns false if it is not.

**Return Value**

(DOM Node | false)

**Parameters**

pNd (DOM Node | string ID)
$v(pNd)$

Given a DOM node or string ID (pNd), this function returns the value of an Application Express item in the same format as it would be posted.

**Parameters**

pNd (DOM Node | string ID)
Given a DOM node or string ID (pNd), this function returns the value of an Application Express item as a string or an array. If the page item type can contain multiple values like a shuttle, checkboxes or a multi select list an array is returned, otherwise a string.

**Return Value**

(string|array)

**Parameters**

pNd (DOM Node | string ID)
Given a DOM node or string ID (pNd), this function sets the Application Express item value taking into account the item type. The pDisplayValue is optional. If used for a page item of type "Popup LOV" where the attribute "Input Field" = "Not Enterable, Show Display Value and Store Return Value", it sets the "Input Field". The value of pValue is stored in the hidden return field. The pSuppressChangeEvent parameter is optional. Passing either FALSE or not passing this parameter value results in a change event firing for the item being set. Pass TRUE to prevent the change event from firing for the item being set.

**Parameters**

- pNd (DOM Node | string ID)
- pValue (String | Array)
- pDisplayValue (String)
- pSuppressChangeEvent (Boolean)
$u_Narray(pNd)

Given a DOM node or string ID or an array (pNd), this function returns a single value, if an pNd is an array but only has one element the value of that element is returned otherwise the array is returned. Used for creating DOM based functionality that can accept a single or multiple DOM nodes.

**Return Value**
Array (DOM Node | string ID | Array)

**Parameters**
Array or first value
$u_Carray(pNd)

Given a DOM node or string ID or an array (pNd), this function returns an array. Used for creating DOM based functionality that can accept a single or multiple DOM nodes.

Return Value
pNd (DOM Node | string ID | Array)

Parameters
Array
$nvl(pTest, pDefault)

If $pTest$ is empty or false return $pDefault$ otherwise return $pTest$.

Return Value
(string | Array)

Parameters
pTest   (String | Array)
pDefault (String | Array)
$x\_Style(pNd, pStyle, pString)$

Sets a specific style property (pStyle) to given value (pString) of a DOM node or DOM node Array (pNd).

Return Value
(DOM node | DOM Array)

Parameters
pNd (DOM node | string ID | DOM node Array )
pStyle (String)
pString (String)
$x_{\text{Hide}}(pNd)$

Hides a DOM node or array of DOM nodes ($pNd$). This also takes into consideration which type of Application Express item is being hidden.

**Return Value**

(DOM node | Array)

**Parameters**

$pNd$ (DOM node | string ID | DOM node Array)
$x\_Show(pNd)$

Shows a DOM node or array of DOM nodes (pNd). This also takes into consideration which type of Application Express item is being hidden.

**Return Value**

(DOM node | Array)

**Parameters**

pNd (DOM node | string ID | DOM node Array)
$x\_Toggle(pNd)$

Toggles a DOM node or array of DOM nodes (pNd).

**Return Value**

(DOM node | Array)

**Parameters**

pNd (DOM node | string ID | Array)
$x\_Remove(pNd)$

Removes a DOM node or array of DOM nodes.

**Return Value**

(DOM Node | Array)

**Parameters**

pNd (DOM node | string ID | DOM node Array)
$x\_Value(pNd,pValue)$

Sets the value (pValue) of a DOM node or array of DOM nodes (pNd).

**Return Value**
Not applicable.

**Parameters**
- pNd (DOM node | string ID | DOM node Array)
- pValue (String)
$x_{\text{UpTill}}(\text{pNd, pToTag})$

Starting from a DOM node (pNd), this function cascades up the DOM tree until the tag of node name (pToTag) is found. If the optional pToClass is present, the ancestor node must have a node name that equals pToTag and the class must equal pToClass.

**Return Value**

(DOM Node | false)

**Parameters**

- pNd (DOM Node | string ID)
- String (pToTag)
- String (pToClass)
$x_ItemRow(pNd,pFunc)

Given DOM node or array of DOM nodes, this function (shows, hides, or toggles) the entire row that contains the DOM node or array of DOM nodes. This is most useful when using Page Items. This function only works in table layouts since it explicitly looks for a containing \textit{tr} element.

**Return Value**
Not applicable.

**Parameters**
- \texttt{pNd} (DOM Node | string ID | Dom node Array)
- \texttt{pFunc} ['TOGGLE', 'SHOW', 'HIDE'] (String)
$x_HideItemRow(pNd)

Given a page item name, this function hides the entire row that holds the item. In most cases, this is the item and its label. This function only works in table layouts since it explicitly looks for a containing tr element.

**Return Value**
Not applicable.

**Parameters**
pNd (DOM Node | string ID | DON node Array)
$x_ShowItemRow(pNd)

Given a page item name, this function shows the entire row that holds the item. In most cases, this is the item and its label. This function only works in table layouts since it explicitly looks for a containing \texttt{tr} element.

**Return Value**
Not applicable.

**Parameters**
pNd (DOM node | string ID | DOM note Array)
$x_ToggleItemRow(pNd)$

Given a page item name (pNd), this function toggles the entire row that holds the item. In most cases, this is the item and its label. This function only works in table layouts since it explicitly looks for a containing `tr` element.

**Return Value**

Not applicable.

**Parameters**

pNd (DOM node | string ID | DOM node ray)
$x_HideAllExcept(pNd,pNdArray)

Hides all DOM nodes referenced in pNdArray and then shows the DOM node referenced by pNd. This is most useful when pNd is also a node in pNdArray.

Return Value
(DOM node | DOM Array)

Parameters
pNd (DOM node | string ID | DOM node Array)
pNdArray (DOM node | String | Array)
$x\_HideSiblings(pNd)$

Hides all sibling nodes of given pNd.

**Return Value**

(DOM node)

**Parameters**

pNd (DOM node | string ID )
Shows all sibling DOM nodes of given DOM nodes (pNd).

**Return Value**

(DOM node)

**Parameters**

pNd (DOM node | string ID)
$x\_Class(pNd,pClass)$

Sets a DOM node or array of DOM nodes to a single class name.

**Return Value**

Not applicable.

**Parameters**

- `pNd` (DOM node | string ID | DOM node Array)
- `pClass` (String)
$x_SetSiblingsClass(pNd, pClass, pNdClass)

Sets the class (pClass) of all DOM node siblings of a node (pNd). If pNdClass is not null the class of pNd is set to pNdClass.

**Return Value**

(DOM node | false)

**Parameters**

- pNd (DOM Node | string ID)
- pClass (String)
- pThisClass (String)
$x_{\text{OfClass}}(p\text{Class}, p\text{Nd}, p\text{Tag})$

Returns an array of DOM nodes by a given class name ($p\text{Class}$). If the $p\text{Nd}$ parameter is provided, then the returned elements are all children of that DOM node. Including the $p\text{Tag}$ parameter further narrows the list to just return nodes of that tag type.

Return Value
(Array)

Parameters
- $p\text{Class}$ (String)
- $p\text{Nd}$ (DOM node | string ID)
- $p\text{Tag}$ (String)
$x_ShowAllByClass(pNd, pClass, pTag)

Show all the DOM node children of a DOM node (pNd) that have a specific class (pClass) and tag (pTag).

Return Value
Not applicable.

Parameters
pNd (DOM node | string ID)
pClass (String)
pTag (String)
$x_ShowChildren(pNd)

Show all DOM node children of a DOM node (pNd).

Return Value
Not applicable.

Parameters
pNd (DOM node | string ID)
$x\_HideChildren(pNd)$

Hide all DOM node children of a DOM node ($pNd$).

**Return Value**

Not applicable.

**Parameters**

$pNd$ (DOM node | string ID)
$.x_disableItem(pNd, pTest)

Disables or enables an item or array of items based on `pTest`.

**Return Value**
Not applicable.

**Parameters**
- `pNd` (DOM node | string ID | DOM node array)
- `a` (true | false)
Checks an item or an array of items to see if any are empty, set the class of all items that are empty to `pClassFail`, set the class of all items that are not empty to `pClass`.

**Return Value**
false, Array  Array of all items that are empty (false | Array)

**Parameters**
pNd (DOM node | string ID | DOM node Array)
String (pClassFail)
String (pClass)
$v\_Array(pNd)$

Returns an item value as an array. Useful for multiselects and checkboxes.

**Return Value**

(Array)

**Parameters**

pId (DOM Node | string ID)
$f\_ReturnChecked(pNd)$

Returns an item value as an array. Useful for radio items and check boxes.

**Return Value**

(Array)

**Parameters**

pId (DOM node | string ID)
$d_ClearAndHide(pNd)

Clears the content of an DOM node or array of DOM nodes and hides them.

**Return Value**
Not applicable.

**Parameters**
pNd (DOM node | string ID | DOM node array)
$f_SelectedOptions(pNd)$

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

**Return Value**

(DOM Array)

**Parameters**

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

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

**Return Value**

(DOM Array | String)

**Parameters**

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

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

**Return Value**
Not applicable.

**Parameters**
pArray (pArray)
pDelim (String)
\texttt{$x\_CheckImageSrc(pId,pSearch)$}

Checks an image (\texttt{pId}) source attribute for a substring (\texttt{pSearch}). The function returns true if a substring (\texttt{pSearch}) is found. It returns false if a substring (\texttt{pSearch}) is not found.

\textbf{Return Value}

\begin{itemize}
  \item \texttt{(true | false)}
\end{itemize}

\textbf{Parameters}

\begin{itemize}
  \item \texttt{pId} (DOM Node | String)
  \item \texttt{pSearch} (pSearch)
\end{itemize}
$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)
$f_{\text{Hide On Value Item}}(pThis, pThat, pValue)$

Checks page item's (pThis) value against a value (pValue). If it matches, a DOM node (pThat) is set to hidden. If it does not match, then the DOM node (pThat) is set to visible.

Return Value

(true | false)

Parameters

pThis (DOM node | string ID)
pThat (DOM node | string ID | DOM node Array)
pValue (Number | String | Array)
Checks page item’s (pThis) value against a value (pValue). If it matches, a DOM node (pThat) is set to visible. If it does not match, then the DOM node (pThat) is set to hidden.

**Return Value**

(true | false)

**Parameters**

- pThis (DOM node | string ID)
- pThat  (DOM node | string ID | DOM node Array )
- pValue (Number | String | Array)
$f\_Hide\_On\_Value\_Item\_Row(pThis, pThat, pValue)$

Checks the value (pValue) of an item (pThis). If it matches, this function hides the table row that holds (pThat). If it does not match, then the table row is shown.

**Return Value**

(true | false)

**Parameters**

pThis (DOM node | string ID)
pThat  (DOM node | string ID | DOM node Array)
pValue (Number | String | Array)
$f\_Show\_On\_Value\_Item\_Row(pThis, pThat, pValue)$

Checks the value (pValue) of an item (pThis). If it matches, this function shows the table row that holds (pThat). If it does not match, then the table row is hidden.

**Return Value**

(true | false)

**Parameters**

pThis (DOM node | string ID)

pThat (DOM node | string ID | DOM node Array)

pValue (Number | String | Array)
$f_{\text{DisableOnValue}}(pThis, pValue, pThat)$

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)
$f_\text{ValuesToArray}(pThis, pClass, pTag)$

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 ($\text{pThis}$) of a certain type ($\text{pType}$).

**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 check box ($pNd$), useful for tabular forms.

**Return Value**

DOM node Array

**Parameters**

$pNd$ (DOM node | String)
$v_PopupReturn(pValue, pThat) [Deprecated]

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

Note: This function is deprecated. Instead, use:

`apex.navigation.popup.close(pThat,pValue)`

For existing applications, the old function is still available, because of the application including the 'Legacy JavaScript' file (legacy.js). For details on how to control the inclusion of this file, see "About Database Applications" in Oracle Application Express Application Builder User’s Guide.

Return Value

Not applicable.

Parameters

pValue (string)
pThat (DOM node | string ID)
$x_{\text{ToggleWithImage}}(p\text{This},p\text{Nd})$

Given an image element ($p\text{This}$) and a DOM node ($p\text{Nd}$), this function toggles the display of the DOM node ($p\text{Nd}$). The src attribute of the image element ($p\text{This}$) is rewritten. The image src has any plus substrings replaced with minus substrings or minus substrings are replaced with plus substrings.

\textbf{Return Value}

(DOM Node)

\textbf{Parameters}

- $p\text{This}$ (DOM Node | string ID)
- $p\text{Nd}$ (DOM Node | string ID | DOM node Array)
$x\_SwitchImageSrc(pNd, pSearch, pReplace)$

Checks an image (pId) src attribute for a substring (pSearch). If a substring is found, this function replaces the image entire src attribute with (pReplace).

**Return Value**

(DOM node | false)

**Parameters**

- **pNd** (DOM node | string ID)
- **pSearch** (String)
- **pReplace** (String)
$x\_CheckImageSrc(pNd, pSearch)$

Checks an image (pNd) source attribute for a substring (pSearch). The function returns true if a substring (pSearch) is found. It returns false if a substring (pSearch) is not found.

**Return Value**

(true | false)

**Parameters**

pNd (DOM node | string ID)

pSearch (String)
$u\_SubString(pText,pMatch)$

Returns a true or false if a string (pText) contains a substring (pMatch).

**Return Value**

(true | false)

**Parameters**

- pText (String)
- pMatch (String)
html_RemoveAllChildren(pNd)

Use DOM methods to remove all DOM children of DOM node (pNd).

**Return Value**
Not applicable.

**Parameters**
pNd (DOM node | string ID)
$v_IsEmpty(pThis) [Deprecated]

Returns true or false if a form element is empty, this considers any whitespace including a space, a tab, a form-feed, as empty. This also considers any null value that has been specified on the item.

**Note:** This function is deprecated. Instead, use:

```javascript
apex.item( pNd ).isEmpty()
```

For existing applications, the old function is still available, because of the application including the 'Legacy JavaScript' file (legacy.js). For details on how to control the inclusion of this file, see "About Database Applications" in *Oracle Application Express Application Builder User’s Guide*.

**Return Value**

[true | false]

**Parameters**

pThis (DOM Node | String)
html_SetSelectValue(pId,pValue)

Sets the value (pValue) of a select item (pId). If the value is not found, this functions selects the first option (usually the NULL selection).

Return Value
Not applicable.

Parameters
pId (DOM node | String)
pValue (String)
addLoadEvent(pFunction)

Adds an onload function (func) without overwriting any previously specified onload functions.

Return Value
Not applicable.

Parameters
pFunction (Javascript Function)
$f\_Swap(p\text{This},p\text{That})$

Swaps the form values of two form elements ($p\text{This},p\text{That}$).

**Return Value**

Not applicable.

**Parameters**

- $p\text{This}$ (DOM Node | String)
- $p\text{That}$ (DOM Node | String)
submitEnter(pNd,e) [Deprecated]

Submits a page when ENTER is pressed in a text field, setting the request value to the ID of a DOM node (pNd).

Usage is onkeypress="submitEnter(this,event)"

---

**Note:** This function is deprecated. Instead, use:

```
apex.submit( { submitIfEnter : event } )
```

See apex.submit for further details on how to use the 'submitIfEnter' pOptions property.

For existing applications, the old function is still available, because of the application including the 'Legacy JavaScript' file (legacy.js). For details on how to control the inclusion of this file, see "About Database Applications" in Oracle Application Express Application Builder User’s Guide.

---

**Return Value**

Not applicable.

---

**Parameters**

pNd (DOM node | String | Array)
$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)
- **pTest** (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 an table row Dom node (pThis), this function sets the background of all table cells to NULL.

**Return Value**
Not applicable.

**Parameters**

pThis (DOM Element | String)
Sets the value of a form item (pNd) to uppercase.

**Return Value**
Not applicable.

**Parameters**
pNd (DOM Node | String)
$d\_Find(pThis,pString,pTags,pClass)$

Hides child nodes of a Dom node (pThis) where the child node's inner HTML matches any instance of pString. To narrow the child nodes searched by specifying a tag name (pTag) or a class name (pClass). Note that the child node is set to a block level element when set to visible.

**Return Value**
Not applicable.

**Parameters**
- pThis (DOM node | String)
- pString (String)
- pTags (String)
- pClass (String)
setReturn(p_R,p_D) [Deprecated]

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

**Note:** This function is deprecated and due to very limited value there is no alternative.

For existing applications, the old function is still available, because of the application including the 'Legacy JavaScript' file (legacy.js). For details on how to control the inclusion of this file, see "About Database Applications" in *Oracle Application Express Application Builder User’s Guide*.

**Return Value**

Not applicable.

**Parameters**

- p_R
- p_D
$f_{\text{First\_field}}(pNd)$

Places the user focus on a form item (pNd). If pNd is not found then this function places focus on the first found user editable field.

**Return Value**
true (if successful)

**Parameters**
pNd
GetCookie (pName) [Deprecated]

Returns the value of cookie name (pName).

---

**Note:** This function is deprecated. Instead, use:

```
apex.storage.getCookie(pName)
```

For existing applications, the old function is still available, because of the application including the 'Legacy JavaScript' file (legacy.js). For details on how to control the inclusion of this file, see “About Database Applications” in *Oracle Application Express Application Builder User’s Guide*.

---

**Return Value**

Not applicable.

**Parameters**

pName (String)
SetCookie (pName,pValue) [Deprecated]

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

Note: This function is deprecated. Instead, use:

apex.storage.setCookie(pName,pValue)

For existing applications, the old function is still available, because of the application including the 'Legacy JavaScript' file (legacy.js). For details on how to control the inclusion of this file, see "About Database Applications" in Oracle Application Express Application Builder User’s Guide.

Return Value

Not applicable.

Parameters

pName (String)
pValue (String)
SetCookie (pName,pValue) [Deprecated]
## A

<table>
<thead>
<tr>
<th>Procedure/Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_FILTER procedure signature 1</td>
<td>11-2</td>
</tr>
<tr>
<td>ADD_FILTER procedure signature 2</td>
<td>11-4</td>
</tr>
<tr>
<td>APEX_APPLICATION global variables</td>
<td>1-1</td>
</tr>
<tr>
<td>HELP Procedure</td>
<td>1-6</td>
</tr>
<tr>
<td>package</td>
<td>1-1</td>
</tr>
<tr>
<td>Referencing Arrays referencing</td>
<td>1-3</td>
</tr>
<tr>
<td>STOP_APEX_ENGINE Procedure</td>
<td>1-8</td>
</tr>
<tr>
<td>APEX_APPLICATION_INSTALL</td>
<td>2-1</td>
</tr>
<tr>
<td>CLEAR_ALL procedure</td>
<td>2-7</td>
</tr>
<tr>
<td>examples import application into different workspaces using different schema</td>
<td>2-4</td>
</tr>
<tr>
<td>import application with generated application id</td>
<td>2-4</td>
</tr>
<tr>
<td>import application with specified application id</td>
<td>2-4</td>
</tr>
<tr>
<td>import application without modification</td>
<td>2-4</td>
</tr>
<tr>
<td>import into training instance for three different workspaces</td>
<td>2-5</td>
</tr>
<tr>
<td>GENERATE_APPLICATION_ID procedure</td>
<td>2-8</td>
</tr>
<tr>
<td>GENERATE_OFFSET procedure</td>
<td>2-9</td>
</tr>
<tr>
<td>GET_APPLICATION_ALIAS function</td>
<td>2-10</td>
</tr>
<tr>
<td>GET_APPLICATION_ID function</td>
<td>2-11</td>
</tr>
<tr>
<td>GET_APPLICATION_NAME function</td>
<td>2-12</td>
</tr>
<tr>
<td>GET_IMAGE_PREFIX function</td>
<td>2-13</td>
</tr>
<tr>
<td>GET_OFFSET function</td>
<td>2-14</td>
</tr>
<tr>
<td>GET_PROXY function</td>
<td>2-15</td>
</tr>
<tr>
<td>GET_SCHEMA function</td>
<td>2-16</td>
</tr>
<tr>
<td>GET_WORKSPACE_ID function</td>
<td>2-17</td>
</tr>
<tr>
<td>import script examples</td>
<td>2-4</td>
</tr>
<tr>
<td>package overview</td>
<td>2-2</td>
</tr>
<tr>
<td>SET_APPLICATION_ALIAS procedure</td>
<td>2-18</td>
</tr>
<tr>
<td>SET_APPLICATION_ID procedure</td>
<td>2-19</td>
</tr>
<tr>
<td>SET_APPLICATION_NAME procedure</td>
<td>2-20</td>
</tr>
<tr>
<td>SET_IMAGE_PREFIX procedure</td>
<td>2-21</td>
</tr>
<tr>
<td>SET_OFFSET procedure</td>
<td>2-22</td>
</tr>
<tr>
<td>SET_PROXY procedure</td>
<td>2-23</td>
</tr>
<tr>
<td>SET_SCHEMA procedure</td>
<td>2-24</td>
</tr>
<tr>
<td>SET_WORKSPACE_ID procedure</td>
<td>2-25</td>
</tr>
<tr>
<td>APEX_AUTHENTICATION</td>
<td>3-1</td>
</tr>
<tr>
<td>CALLBACK Procedure</td>
<td>3-3</td>
</tr>
<tr>
<td>GET_CALLBACK_URL Procedure</td>
<td>3-5</td>
</tr>
<tr>
<td>GET_LOGIN_USERNAME_COOKIE_LOGIN Function</td>
<td>3-6</td>
</tr>
<tr>
<td>IS_AUTHENTICATED Function</td>
<td>3-7</td>
</tr>
<tr>
<td>IS_PUBLIC_USER Function</td>
<td>3-8</td>
</tr>
<tr>
<td>LOGIN Procedure</td>
<td>3-9</td>
</tr>
<tr>
<td>LOGOUT Procedure</td>
<td>3-10</td>
</tr>
<tr>
<td>POST_LOGIN Procedure</td>
<td>3-11</td>
</tr>
<tr>
<td>SEND_LOGIN_USERNAME_COOKIE Procedure</td>
<td>3-12</td>
</tr>
<tr>
<td>APEX_COLLECTION</td>
<td>4-3</td>
</tr>
<tr>
<td>ADD_MEMBER function</td>
<td>4-14</td>
</tr>
<tr>
<td>ADD_MEMBER procedure</td>
<td>4-12</td>
</tr>
<tr>
<td>ADD_MEMBERS procedure</td>
<td>4-16</td>
</tr>
<tr>
<td>COLLECTION_EXISTS function</td>
<td>4-18</td>
</tr>
<tr>
<td>COLLECTION_HAS_CHANGED function</td>
<td>4-19</td>
</tr>
<tr>
<td>COLLECTION_MEMBER_COUNT function</td>
<td>4-20</td>
</tr>
<tr>
<td>CREATE_COLLECTION procedure</td>
<td>4-21</td>
</tr>
<tr>
<td>CREATE_COLLECTION_FROM_QUERY</td>
<td>4-23</td>
</tr>
<tr>
<td>CREATE_COLLECTION_FROM_QUERY_B procedure</td>
<td>4-25</td>
</tr>
<tr>
<td>CREATE_COLLECTION_FROM_QUERY2 procedure</td>
<td>4-24</td>
</tr>
<tr>
<td>CREATE_COLLECTION_FROM_QUERYB2 procedure</td>
<td>4-27</td>
</tr>
<tr>
<td>CREATE_OR_TRUNCATE_COLLECTION</td>
<td>4-11</td>
</tr>
<tr>
<td>CREATE_OR_TRUNCATE_COLLECTION procedure</td>
<td>4-22</td>
</tr>
<tr>
<td>DELETE_ALL_COLLECTIONS procedure</td>
<td>4-29</td>
</tr>
<tr>
<td>DELETE_ALL_COLLECTIONS_SESSION procedure</td>
<td>4-30</td>
</tr>
<tr>
<td>DELETE_COLLECTION procedure</td>
<td>4-31</td>
</tr>
<tr>
<td>DELETE_MEMBER procedure</td>
<td>4-32</td>
</tr>
<tr>
<td>DELETE_MEMBERS procedure</td>
<td>4-33</td>
</tr>
<tr>
<td>GET_MEMBER_MD5 function</td>
<td>4-34</td>
</tr>
<tr>
<td>MERGE_MEMBERS procedure</td>
<td>4-35</td>
</tr>
<tr>
<td>MOVE_MEMBER_DOWN procedure</td>
<td>4-37</td>
</tr>
<tr>
<td>MOVE_MEMBER_UP procedure</td>
<td>4-38</td>
</tr>
<tr>
<td>RESEQUENCE_COLLECTION procedure</td>
<td>4-39</td>
</tr>
<tr>
<td>RESET_COLLECTION_CHANGED procedure</td>
<td>4-40</td>
</tr>
<tr>
<td>RESET_COLLECTION_CHANGED_ALL procedure</td>
<td>4-41</td>
</tr>
<tr>
<td>SORT_MEMBERS procedure</td>
<td>4-42</td>
</tr>
<tr>
<td>TRUNCATE_COLLECTION procedure</td>
<td>4-43</td>
</tr>
<tr>
<td>UPDATE_MEMBER procedure</td>
<td>4-44</td>
</tr>
</tbody>
</table>
APEX_UTIL, 21-1
CACHE_GET_DATE_OF_PAGE_CACHE function, 21-5, 21-6
CACHE_PURGE_BY_APPLICATION procedure, 21-7
CACHE_PURGE_BY_PAGE procedure, 21-8
CACHE_PURGE_STALE procedure, 21-9
CHANGE_CURRENT_USER_PW procedure, 21-10
CHANGE_PASSWORD_ON_FIRST_USE function, 21-11
CLEAR_APP_CACHE procedure, 21-12
CLEAR_PAGE_CACHE procedure, 21-13
CLEAR_USER_CACHE procedure, 21-14
COUNT_CLICK procedure, 21-15
CREATE_USER procedure, 21-16
CREATE_USER_GROUP procedure, 21-20
CURRENT_USER_IN_GROUP function, 21-21
DELETE_USER_GROUP procedure signature 1, 21-23
DELETE_USER_GROUP procedure signature 2, 21-24
DOWNLOAD_PRINT_DOCUMENT procedure signature 1, 21-25
DOWNLOAD_PRINT_DOCUMENT procedure signature 2, 21-26
DOWNLOAD_PRINT_DOCUMENT procedure signature 3, 21-28
DOWNLOAD_PRINT_DOCUMENT procedure signature 4, 21-30
EDIT_USER procedure, 21-31
END_USER_ACCOUNT_DAYS_LEFT function, 21-35
EXPIRE_END_USER_ACCOUNT procedure, 21-36
EXPIRE_WORKSPACE_ACCOUNT procedure, 21-37
EXPORT_USERS procedure, 21-38
FETCH_APP_ITEM function, 21-39
FETCH_GROUP_ID function, 21-40
FETCH_GROUP_NAME function, 21-43
FETCH_USER procedure signature 1, 21-45
FETCH_USER procedure signature 2, 21-48
FETCH_WORKSPACE function, 21-49
GET_ACCOUNT_LOCKED_STATUS function, 21-50
GET_ATTRIBUTE function, 21-51
GET_AUTHENTICATION_RESULT function, 21-52
GET_BLOB_FILE_SRC function, 21-53
GETBUILD_OPTION_STATUS function signature 1, 21-55
GETBUILD_OPTION_STATUS function signature 2, 21-56
GET_CURRENT_USER_ID function, 21-57
GET_DEFAULT_SCHEMA function, 21-58
GET_EDITION function, 21-59
GET_EMAIL function, 21-60
GET_FEEDBACK_FOLLOW_UP function, 21-61
GET_FILE procedure, 21-62
GET_FILE_ID function, 21-64
GET_FIRST_NAME function, 21-65
GET_GROUP_ID function, 21-67
GET_GROUP_NAME function, 21-68
GETGROUPS_USER_BELONGS_TO function, 21-69
GET_HIGH_CONTRAST_MODE_TOGGLE function, 21-70
GET_LAST_NAME function, 21-70
GET_NUMERIC_SESSION_STATE function, 21-71
GET_PREFERENCE function, 21-72
GET_PRINT_DOCUMENT function signature 1, 21-73
GET_PRINT_DOCUMENT function signature 2, 21-74
GET_PRINT_DOCUMENT function signature 3, 21-75
GET_PRINT_DOCUMENT function signature 4, 21-76
GET_SCREEN_READER_MODE_TOGGLE function, 21-78
GET_SESSION_LANG function, 21-79
GET_SESSION_STATE function, 21-80
GET_SESSION_TERRITORY function, 21-81
GET_SESSION_TIME_ZONE function, 21-82
GET_USER_ID function, 21-83
GET_USER_ROLES function, 21-84
GET_USERNAME function, 21-85
HOST_URL function, 21-86
IR_CLEAR procedure, 21-87
IR_DELETE_REPORT procedure, 21-90
IR_DELETE_SUBSCRIPTION procedure, 21-91
IR_FILTER procedure, 21-92
IR_RESET procedure, 21-94
IS_HIGH_CONTRAST_SESSION function, 21-95
IS_HIGH_CONTRAST_SESSION_YN function, 21-96
IS_LOGIN_PASSWORD_VALID function, 21-97
IS_SCREEN_READER_SESSION function, 21-98
IS_SCREEN_READER_SESSION_YN function, 21-99
IS_USERNAME_UNIQUE function, 21-100
KEYVAL_NUM function, 21-101
KEYVAL_VC2 function, 21-102
LOCK_ACCOUNT procedure, 21-103, 21-155
PASSWORD_FIRST_USE_OCCURRED function, 21-104
PREPARE_URL function, 21-105
PUBLIC_CHECK_AUTHORIZATION function, 21-107
PURGE_REGIONS_BY_APP procedure, 21-108
PURGE_REGIONS_BY_NAME procedure, 21-109
PURGE_REGIONS_BY_PAGE procedure, 21-110
REDIRECT_URL procedure, 21-111
REMOVE_PREFERENCE procedure, 21-112
REMOVE_SORT_PREFERENCES procedure, 21-113
REMOVE_USER procedure, 21-114
RESET_AUTHORIZATIONS procedure, 21-115
RESET_PW procedure, 21-116
SAVEKEY_NUM function, 21-117
SAVEKEY_VC2 function, 21-118
SET_ATTRIBUTE procedure, 21-119
SET_AUTHENTICATION_RESULT procedure, 21-120
SET_BUILD_OPTION_STATUS procedure, 21-121
SET_CUSTOM_AUTH_STATUS procedure, 21-122
SET_EDITION procedure, 21-123
SET_EMAIL procedure, 21-124
SET_FIRST_NAME procedure, 21-125
SET_LAST_NAME procedure, 21-126
SET_PREFERENCE procedure, 21-127
SET_SECURITY_GROUP_ID procedure, 21-128, 21-132
SET_SECURITY_HIGH_CONTRAST_OFF procedure, 21-130
SET_SECURITY_HIGH_CONTRAST_ON procedure, 21-131
SET_SESSION_SCREEN_READER_OFF procedure, 21-135, 21-136
SET_SESSION_STATE procedure, 21-133, 21-134, 21-137
SET_SESSION_TERRITORY procedure, 21-138
SET_SESSION_TIME_ZONE procedure, 21-139
SET_USERNAME procedure, 21-140
SHOW_HIGH_CONTRAST_MODE_TOGGLE procedure, 21-141
SHOW_SCREEN_READER_MODE_TOGGLE function, 21-142
STRING_TO_TABLE function, 21-143
STRONG_PASSWORD_CHECK procedure, 21-144
STRONG_PASSWORD_VALIDATION function, 21-148
SUBMIT_FEEDBACK procedure, 21-149
SUBMIT_FEEDBACK_FOLLOWUP procedure, 21-151
TABLE_TO_STRING function, 21-152
UNEXPIRE_END_USER_ACCOUNT procedure, 21-153
UNEXPIRE_WORKSPACE_ACCOUNT procedure, 21-154
URL_ENCODE function, 21-156
WORKSPACE_ACCOUNT_DAYS_LEFT function, 21-158
APEX_WEB_SERVICE, 22-1
CHANGE_SUBSCRIPTION_LANG procedure, 11-6
check box
creating, 12-2
CLEAR_REPORT procedure signature 1, 11-7
CLEAR_REPORT procedure signature 2, 11-8
clicks
counting, 21-15
collections
accessing, 4-5
APEX_COLLECTION API, 4-3
clearing session state, 4-10
creating, 4-4
deleting members, 4-9
determining status, 4-11
managing, 4-10
merging, 4-7
truncating, 4-7
updating members, 4-8
DELETE_REPORT procedure, 11-9
DELETE_SUBSCRIPTION procedure, 11-10

**E**

email
- sending as an attachment, 16-4
- sending messages in APEX_MAIL_QUEUE, 16-7
- sending outbound, 16-8

export file
- of workspace, 21-38

**F**

F01, 1-3

file repository
- downloading files, 21-62
- obtaining primary key, 21-64

Function, 22-20

**G**

GET_LAST_VIEWED_REPORT_ID function, 11-11
GET_REPORT function, 11-12

**J**

JavaScript API, 23-1
- $d_ClearAndHide(pNd), 23-81
- $d_Find(pThis,pString,pTags,pClass), 23-117
- $dom_addInput(pThis,pType,pId,pName,pValue), 23-112
- $dom_addTag(pThis,pTag,pText), 23-109
- $dom_makeParent(pNode,pParent), 23-113
- $f_CheckAll(pThis,pCheck,pArray), 23-95
- $f_CheckFirstChild(pNd), 23-96
- $f_DisableOnValue(pThis,pValue,pThat), 23-91
- $f_First_field(pNd), 23-119
- $f_get_empty(pNd,pClassFail,pClass), 23-78
- $f_Hide_On_Value_Item_Row(pThis,pThat,pValue), 23-89
- $f_Hide_On_Value_Item(pThis,pThat,pValue), 23-87
- $f_ReturnChecked(pNd), 23-80
- $f_SelectedOptions(pNd), 23-82
- $f_SelectValue(pNd), 23-83
- $f_SetValueSequence(pArray,pMultiple), 23-108
- $f_Show_On_Value_Item_Row(pThis,pThat,pValue), 23-90
- $f_Show_On_Value_Item(pThis,pThat,pValue), 23-88
- $f_Swap(pThis,pThat), 23-106
- $f_ValuesToArray(pThis,pClass,pTag), 23-93
- $nv(pTest,pDefault), 23-56
- $p(pNd,pValue), 23-53
- $tr_addTD(pThis,pText), 23-110
- $tr_addTH(pThis,pText), 23-111
- $u_arrayToString(pArray,pDelim), 23-84
- $u_Carray(pNd), 23-55
- $u_Narray(pNd), 23-54
- $u_SubString(pText,pMatch), 23-101
- $v_Array(pNd), 23-79
- $v_CheckValueAgainst(pThis,pValue), 23-86
- $v_IsEmpty(pThis), 23-103
- $v_PopupReturn(pValue,pThat), 23-97
- $v_Upper(pNd), 23-116
- $v(pNd), 23-51
- $v2(pNd), 23-52
- $x_ByClass(pClass,pNd,pTag), 23-73
- $x_CheckImageSrc(pId,pSearch), 23-85
- $x_CheckImageSrc(pNd,pSearch), 23-100
- $x_Class(pNd,pClass), 23-71
- $x_ClassByClass(pNd,pClass,pTag,pClass2), 23-92
- $x_disableItem(pNd,a), 23-77
- $x_FormItems(pNd,pType), 23-94
- $x_Hide(pNd), 23-58
- $x_HideAllExcept(pNd,pNdArray), 23-68
- $x_HideItemRow(pNd), 23-65
- $x_HideSiblings(pNd), 23-69
- $x_ItemsRow(pNd,pFunc), 23-64
- $x_Remove(pNd), 23-61
- $x_RowHighlight(pThis,pColor), 23-114
- $x_RowHighlightOff(pThis), 23-115
- $x_SetSiblingsClass(pNd,pClass,pNdClass), 23-72
- $x_Show(pNd), 23-59
- $x_ShowAllByClass(pNd,pClass,pTag), 23-74
- $x_ShowChildren(pThis), 23-75, 23-76
- $x_ShowItemRow(pNd), 23-66
- $x_ShowSiblings(pNd), 23-70
- $x_Style(pNd,pStyle,pString), 23-57
- $x_Toggle(pNd), 23-60
- $x_ToggleItemRow(pNd), 23-67
- $x_ToggleWithImage(pThis,pNd), 23-98
- $x_UpTill(pNd,pToTag), 23-63
- $x_Value(pNd,pValue), 23-62
- $x_(pNd), 23-50
- addLoadEvent(pFunction), 23-105
- apex.event.trigger(pSelector,pEvent,dData), 23-8
- apex.submit(pRequest|pOptions), 23-6
- apex.widget.initPageItem(pName,pOptions), 23-40
- confirmDelete(pMessage,pRequest), 23-4
- GetCookie(pName), 23-120
- html_RemoveAllChildren(pNd), 23-102
- html_SetSelectValue(pId,pValue), 23-104
- SetCookie(pName,pValue), 23-121
- submitEnter(pNd,e), 23-107

**L**

LDAP attributes
- obtaining, 6-6

**P**

password
- changing, 21-10
resetting and emailing, 21-116

R
radio group
  generate, 12-21
RESET_REPORT procedure signature 1, 11-13
RESET_REPORT procedure signature 2, 11-14

S
session
  cookies, 6-5
session state
  fetching for current application, 21-39
  removing for current page, 21-13
  removing for current session, 21-12
  setting, 21-133, 21-134, 21-137
special characters
  encoding, 21-156

U
user
  get e-mail address, 21-60
  remove preference, 21-112
user account
  altering, 21-31
  creating new, 21-16
  fetching, 21-40, 21-43, 21-45
  removing, 21-114
  update email address, 21-124
  updating FIRST_NAME, 21-125
  updating LAST_NAME value, 21-126
  updating USER_NAME value, 21-140

V
variables
  global, 1-1

W
workspace
  export file, 21-38
  numeric security group ID, 21-48