Forms Reference Manual

Release 4.5

Volume 1
Forms™ Reference Manual

Release 4.5
Part No. A32509–2

ORACLE®
Preface

The Forms Reference Manual, Volume 1, provides information necessary to help you use Forms 4.5. This preface includes the following topics:

- Forms Documentation Set
- Audience
- Related Publications
- Typographic Conventions
- Your Comments Are Welcome
Forms Documentation Set

The documentation set for Forms Version 4.5 consists of the following documents:

<table>
<thead>
<tr>
<th>Document</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Forms Documentation Set</em>, Version 4.5</td>
<td>A32503</td>
</tr>
<tr>
<td><em>Getting Started with Forms</em>, Version 4.5</td>
<td>A32504</td>
</tr>
<tr>
<td><em>Forms Developer’s Guide</em>, Version 4.5</td>
<td>A32505</td>
</tr>
<tr>
<td><em>Forms Advanced Techniques</em>, Version 4.5</td>
<td>A32506</td>
</tr>
<tr>
<td><em>Forms Messages and Codes</em>, Version 4.5</td>
<td>A32508</td>
</tr>
</tbody>
</table>

Audience

All the manuals in the Forms Version 4.5 documentation set are written for the application developer.

Related Publications

As an application designer using Version 4.5 of Forms, you should also be familiar with the following documents:

<table>
<thead>
<tr>
<th>Document</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Procedure Builder Developer’s Guide</em></td>
<td>A32485</td>
</tr>
<tr>
<td><em>Oracle7 Server Messages and Codes Manual</em></td>
<td>A12379</td>
</tr>
<tr>
<td>Forms documentation for your operating system</td>
<td></td>
</tr>
</tbody>
</table>
Typographic Conventions

This manual uses the following typographic conventions to distinguish important elements from the body of the manual.

Function Keys

Forms function keys are represented by the key name enclosed in square brackets: [Next Item].

For key mappings for your particular keyboard type, refer to the following sources:

- online Forms Show Keys screen (for most keyboards, [Ctrl–K])
- the keypad diagram

For more information on the keypad diagram, refer to the Forms documentation for your operating system.

Screen Messages

Hint messages and error messages appear in a monotype font:

This is a monotype font.
Command and Example Syntax

Commands and examples appear in a monotype font, as follows:

```
Syntax:  
SET_CANVAS_PROPERTY(canvas_name, property, value);
/**
** Built-in: SET_CANVAS_PROPERTY
** Example: Change the "background color" by setting the
** canvas color dynamically at runtime to the name
** of a visual attribute you created.
*/
BEGIN
Set_Canvas_Property('my_main_cnv',VISUAL_ATTRIBUTE,'blue_text');
END;
```

Command and example syntax adhere to the following conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Indicates a PL/SQL keyword or, if used within the parameter list for a built-in routine, a constant that you must enter exactly as spelled.</td>
</tr>
<tr>
<td>MONOTYPE</td>
<td>Used for code fragments and examples.</td>
</tr>
<tr>
<td>plain monotype</td>
<td>Indicates user-supplied items such as variables, exceptions, and actual parameters.</td>
</tr>
<tr>
<td>italic monotype</td>
<td>Indicates a default parameter. If you indicate no parameter in a parameter set, Forms applies the default parameter.</td>
</tr>
<tr>
<td>underlined monotype</td>
<td>An ellipsis shows that statements or clauses were left out. The ellipsis can appear horizontally as shown, or in vertical format.</td>
</tr>
<tr>
<td>/*</td>
<td>A slash and asterisk begin a C-style comment.</td>
</tr>
<tr>
<td>*/</td>
<td>An asterisk and slash end a C-style comment.</td>
</tr>
<tr>
<td>—</td>
<td>Two consecutive hyphens begin an ANSI-style comment, which extends to the end of the line.</td>
</tr>
<tr>
<td>indentation</td>
<td>Indentation helps show structure within code examples, but is not required.</td>
</tr>
</tbody>
</table>
Case Sensitivity

Although neither PL/SQL nor Forms commands are case sensitive (that is, you can enter text in upper or lower case without restriction), in the documentation both upper and lower case are used for ease in reading.

In syntax examples, built-in names are shown in all caps; user-defined values are shown in lower case.

**Syntax:**

```
SET_CANVAS_PROPERTY(canvas_name, property, value);
```

**All caps.** In code examples, PL/SQL reserved words (such as BEGIN, IF, THEN, ELSE, and END) and SQL commands (such as SELECT, WHERE, ORDERBY, and INTO) are shown in all caps.

Properties, such as VISUAL_ATTRIBUTE, are also shown in upper case.

**Upper and lower case.** Names of built-in procedures (Set_Canvas_Property) and system variables (System.Suppress_Working) are shown in upper and lower case.

**Lower case.** User-defined values (‘my_main_cnv’) are shown in lower case.

**Example:**

```
/*
** Built-in: SET_CANVAS_PROPERTY
** Example: Change the "background color" by setting the canvas color dynamically at runtime to the name of a visual attribute you created.
*/
BEGIN
Set_Canvas_Property('my_main_cnv',VISUAL_ATTRIBUTE,'blue_text');
END;
```

Syntax Examples

This example illustrates first how the syntax is presented in this manual, followed by an example of how you actually enter a built-in procedure into your triggers.

**Example Syntax:**

```
SET_FORM_PROPERTY(formmodule_name, property, value);
```

**Example Syntax:**

```
Set_Form_Property('my_form', savepoint_mode, PROPERTY_ON);
```

**Example Syntax:**

```
SET_TIMER(timer_name, milliseconds, iterate);
```

**Example Syntax:**

```
Set_Timer('my_timer', 12000, REPEAT);
```
Your Comments Are Welcome

We value and appreciate your comments as an Oracle user and reader of the manuals. As we write, revise, and evaluate our documentation, your opinions are the most important input we receive. At the back of our printed manuals is a Reader’s Comment Form, which we encourage you to use to tell us what you like and dislike about this manual or other Oracle manuals. If the form is not available, please use the following address or FAX number.

Forms Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood City, CA 94065
U.S.A.
FAX: 415–506–7200
Contents

Chapter 1

Components and Options ........................................ 1 – 1
  About Oracle Forms Components .......................... 1 – 2
  Starting Oracle Forms Components ....................... 1 – 3
  Logging In to the Database ................................. 1 – 8
    USERID .................................................. 1 – 8
  Runform Options ........................................... 1 – 9
  Array (Runform) ............................................ 1 – 11
  Block_Menu (Runform) ...................................... 1 – 11
  Buffer_Records (Runform) .................................. 1 – 12
  Debug (Runform) ........................................... 1 – 12
  Debug_Messages (Runform) ................................ 1 – 12
  Help (Runform) ............................................. 1 – 13
  Interactive (Runform) ..................................... 1 – 13
  Keyin (Runform) ............................................ 1 – 13
  Keyout (Runform) .......................................... 1 – 14
  Logon_Screen (Runform) .................................... 1 – 14
  OptimizeSQL (Runform) .................................... 1 – 15
  OptimizeTP (Runform) ...................................... 1 – 16
  Options_Screen (Runform) .................................. 1 – 17
  Output_File (Runform) ..................................... 1 – 17
  PECS (Runform) ............................................. 1 – 18
  Query_Only (Runform) ..................................... 1 – 19
  Quiet (Runform) ............................................ 1 – 19
  Session (Runform) .......................................... 1 – 19
  Statistics (Runform) ....................................... 1 – 20
  Term (Runform) ............................................. 1 – 20
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window_State (Runform)</td>
<td>1–21</td>
</tr>
<tr>
<td>Setting Generate Options</td>
<td>1–21</td>
</tr>
<tr>
<td>Add_Triggers (Generate)</td>
<td>1–23</td>
</tr>
<tr>
<td>Batch (Generate)</td>
<td>1–23</td>
</tr>
<tr>
<td>CRT_File (Generate)</td>
<td>1–23</td>
</tr>
<tr>
<td>Debug (Generate)</td>
<td>1–24</td>
</tr>
<tr>
<td>Delete (Generate)</td>
<td>1–24</td>
</tr>
<tr>
<td>Extract (Generate)</td>
<td>1–24</td>
</tr>
<tr>
<td>Generate_on_Upgrade (Generate)</td>
<td>1–25</td>
</tr>
<tr>
<td>Help (Generate)</td>
<td>1–25</td>
</tr>
<tr>
<td>Insert (Generate)</td>
<td>1–25</td>
</tr>
<tr>
<td>Logon (Generate)</td>
<td>1–26</td>
</tr>
<tr>
<td>Module_Access (Generate)</td>
<td>1–26</td>
</tr>
<tr>
<td>Module_Type (Generate)</td>
<td>1–26</td>
</tr>
<tr>
<td>Nofail (Generate)</td>
<td>1–27</td>
</tr>
<tr>
<td>Options_Screen (Generate)</td>
<td>1–27</td>
</tr>
<tr>
<td>Output_File (Generate)</td>
<td>1–27</td>
</tr>
<tr>
<td>Parse (Generate)</td>
<td>1–28</td>
</tr>
<tr>
<td>Script (Generate)</td>
<td>1–28</td>
</tr>
<tr>
<td>Statistics (Generate)</td>
<td>1–29</td>
</tr>
<tr>
<td>Upgrade (Generate)</td>
<td>1–29</td>
</tr>
<tr>
<td>Upgrade_Roles (Generate)</td>
<td>1–30</td>
</tr>
<tr>
<td>Version (Generate)</td>
<td>1–30</td>
</tr>
<tr>
<td>Widen_Fields (Generate)</td>
<td>1–31</td>
</tr>
<tr>
<td>Setting Designer Options</td>
<td>1–31</td>
</tr>
<tr>
<td>Designer Options</td>
<td>1–31</td>
</tr>
<tr>
<td>Runtime Options</td>
<td>1–32</td>
</tr>
<tr>
<td>Keyword Parameters</td>
<td>1–32</td>
</tr>
<tr>
<td>Color Mode</td>
<td>1–33</td>
</tr>
<tr>
<td>Color Palette</td>
<td>1–34</td>
</tr>
<tr>
<td>Generate Before Run</td>
<td>1–34</td>
</tr>
<tr>
<td>Help (Designer)</td>
<td>1–35</td>
</tr>
<tr>
<td>Module_Access (Designer)</td>
<td>1–35</td>
</tr>
<tr>
<td>Module_Type (Designer)</td>
<td>1–36</td>
</tr>
<tr>
<td>Printer</td>
<td>1–36</td>
</tr>
<tr>
<td>Run Modules Asynchronously</td>
<td>1–37</td>
</tr>
<tr>
<td>Save Before Generate</td>
<td>1–37</td>
</tr>
<tr>
<td>Suppress Hints</td>
<td>1–37</td>
</tr>
<tr>
<td>Term (Designer)</td>
<td>1–38</td>
</tr>
<tr>
<td>Use System Editor</td>
<td>1–38</td>
</tr>
<tr>
<td>User Preference File</td>
<td>1–39</td>
</tr>
<tr>
<td>Syntax for Options</td>
<td>1–39</td>
</tr>
</tbody>
</table>
Chapter 2

Triggers .............................................. 2 – 1
About Triggers and Processes ...................... 2 – 2
  SQL Statements in Trigger Text ...................... 2 – 2
Trigger Tables ............................................. 2 – 3
Function Key Triggers .................................. 2 – 6
Key–Fn Trigger .......................................... 2 – 8
Key–Others Trigger ...................................... 2 – 9
On–Check–Delete–Master Trigger ................. 2 – 10
On–Check–Unique Trigger ......................... 2 – 11
On–Clear–Details Trigger ......................... 2 – 12
On–Close Trigger ....................................... 2 – 13
On–Column–Security Trigger ..................... 2 – 14
On–Commit Trigger ................................... 2 – 15
On–Count Trigger ...................................... 2 – 16
On–Delete Trigger ...................................... 2 – 17
On–Error Trigger ...................................... 2 – 18
On–Fetch Trigger ...................................... 2 – 19
On–Insert Trigger ...................................... 2 – 21
On–Lock Trigger ....................................... 2 – 22
On–Logon Trigger ...................................... 2 – 23
On–Logout Trigger ..................................... 2 – 24
On–Message Trigger ................................... 2 – 25
On–Populate–Details Trigger .................... 2 – 26
On–Rollback Trigger .................................. 2 – 27
On–Savepoint Trigger ................................. 2 – 28
On–Select Trigger ...................................... 2 – 29
On–Sequence–Number Trigger ................... 2 – 30
On–Update Trigger ..................................... 2 – 31
Post–Block Trigger ..................................... 2 – 32
Post–Change Trigger ................................... 2 – 33
Post–Database–Commit Trigger ................... 2 – 35
Post–Delete Trigger ..................................... 2 – 36
Post–Form Trigger ...................................... 2 – 37
Post–Forms–Commit Trigger ....................... 2 – 38
Post–Insert Trigger ..................................... 2 – 39
Post–Logon Trigger ..................................... 2 – 40
Post–Logout Trigger .................................... 2 – 41
Post–Query Trigger ...................................... 2 – 42
Post–Record Trigger .................................... 2 – 44
Post–Select Trigger ..................................... 2 – 45
Post–Text–Item Trigger ............................... 2 – 46
Post–Update Trigger .................................... 2 – 47
Pre–Block Trigger ........................................ 2 – 48
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre–Commit Trigger</td>
<td>2–49</td>
</tr>
<tr>
<td>Pre–Delete Trigger</td>
<td>2–50</td>
</tr>
<tr>
<td>Pre–Form Trigger</td>
<td>2–51</td>
</tr>
<tr>
<td>Pre–Insert Trigger</td>
<td>2–52</td>
</tr>
<tr>
<td>Pre–Logon Trigger</td>
<td>2–53</td>
</tr>
<tr>
<td>Pre–Logout Trigger</td>
<td>2–54</td>
</tr>
<tr>
<td>Pre–Query Trigger</td>
<td>2–55</td>
</tr>
<tr>
<td>Pre–Record Trigger</td>
<td>2–57</td>
</tr>
<tr>
<td>Pre–Select Trigger</td>
<td>2–58</td>
</tr>
<tr>
<td>Pre–Text–Item Trigger</td>
<td>2–59</td>
</tr>
<tr>
<td>Pre–Update Trigger</td>
<td>2–60</td>
</tr>
<tr>
<td>User–Named Trigger</td>
<td>2–62</td>
</tr>
<tr>
<td>When–Button–Pressed Trigger</td>
<td>2–63</td>
</tr>
<tr>
<td>When–Checkbox–Changed Trigger</td>
<td>2–64</td>
</tr>
<tr>
<td>When–Clear–Block Trigger</td>
<td>2–65</td>
</tr>
<tr>
<td>When–Create–Record Trigger</td>
<td>2–66</td>
</tr>
<tr>
<td>When–Custom–Item–Event Trigger</td>
<td>2–67</td>
</tr>
<tr>
<td>When–Database–Record Trigger</td>
<td>2–68</td>
</tr>
<tr>
<td>When–Form–Navigate</td>
<td>2–69</td>
</tr>
<tr>
<td>When–Image–Activated Trigger</td>
<td>2–69</td>
</tr>
<tr>
<td>When–Image–Pressed Trigger</td>
<td>2–70</td>
</tr>
<tr>
<td>When–List–Activated Trigger</td>
<td>2–70</td>
</tr>
<tr>
<td>When–List–Changed Trigger</td>
<td>2–71</td>
</tr>
<tr>
<td>When–Mouse–Click Trigger</td>
<td>2–71</td>
</tr>
<tr>
<td>When–Mouse–DoubleClick Trigger</td>
<td>2–72</td>
</tr>
<tr>
<td>When–Mouse–Down Trigger</td>
<td>2–74</td>
</tr>
<tr>
<td>When–Mouse–Enter Trigger</td>
<td>2–75</td>
</tr>
<tr>
<td>When–Mouse–Leave Trigger</td>
<td>2–76</td>
</tr>
<tr>
<td>When–Mouse–Move Trigger</td>
<td>2–77</td>
</tr>
<tr>
<td>When–Mouse–Up Trigger</td>
<td>2–78</td>
</tr>
<tr>
<td>When–New–Block–Instance Trigger</td>
<td>2–79</td>
</tr>
<tr>
<td>When–New–Form–Instance Trigger</td>
<td>2–80</td>
</tr>
<tr>
<td>When–New–Item–Instance Trigger</td>
<td>2–81</td>
</tr>
<tr>
<td>When–New–Record–Instance Trigger</td>
<td>2–82</td>
</tr>
<tr>
<td>When–Radio–Changed Trigger</td>
<td>2–83</td>
</tr>
<tr>
<td>When–Remove–Record Trigger</td>
<td>2–84</td>
</tr>
<tr>
<td>When–Timer–Expired Trigger</td>
<td>2–85</td>
</tr>
<tr>
<td>When–Validate–Item Trigger</td>
<td>2–87</td>
</tr>
<tr>
<td>When–Validate–Record Trigger</td>
<td>2–89</td>
</tr>
<tr>
<td>When–Window–Activated Trigger</td>
<td>2–91</td>
</tr>
<tr>
<td>When–Window–Closed Trigger</td>
<td>2–92</td>
</tr>
<tr>
<td>When–Window–Deactivated Trigger</td>
<td>2–92</td>
</tr>
<tr>
<td>When–Window–Resized Trigger</td>
<td>2–93</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>FIND_TIMER</td>
<td>3 – 98</td>
</tr>
<tr>
<td>FIND_VIEW</td>
<td>3 – 99</td>
</tr>
<tr>
<td>FIND_WINDOW</td>
<td>3 – 100</td>
</tr>
<tr>
<td>FIRST_RECORD</td>
<td>3 – 101</td>
</tr>
<tr>
<td>FORM_FAILURE</td>
<td>3 – 101</td>
</tr>
<tr>
<td>FORM_FATAL</td>
<td>3 – 103</td>
</tr>
<tr>
<td>FORM_SUCCESS</td>
<td>3 – 104</td>
</tr>
<tr>
<td>FORMS_DDL</td>
<td>3 – 105</td>
</tr>
<tr>
<td>FORMS_OLE.ACTIVATE_SERVER</td>
<td>3 – 109</td>
</tr>
<tr>
<td>FORMS_OLE.CLOSE_SERVER</td>
<td>3 – 110</td>
</tr>
<tr>
<td>FORMS_OLE.EXEC_VERB</td>
<td>3 – 111</td>
</tr>
<tr>
<td>FORMS_OLE.FIND_OLE_VERB</td>
<td>3 – 112</td>
</tr>
<tr>
<td>FORMS_OLE.GET_INTERFACE_POINTER</td>
<td>3 – 113</td>
</tr>
<tr>
<td>FORMS_OLE.GET_VERB_COUNT</td>
<td>3 – 114</td>
</tr>
<tr>
<td>FORMS_OLE.GET_VERB_NAME</td>
<td>3 – 115</td>
</tr>
<tr>
<td>FORMS_OLE.INITIALIZE_CONTAINER</td>
<td>3 – 116</td>
</tr>
<tr>
<td>FORMS_OLE.SERVER_ACTIVE</td>
<td>3 – 117</td>
</tr>
<tr>
<td>GENERATE_SEQUENCE_NUMBER</td>
<td>3 – 118</td>
</tr>
<tr>
<td>GET_APPLICATION_PROPERTY</td>
<td>3 – 119</td>
</tr>
<tr>
<td>GET_BLOCK_PROPERTY</td>
<td>3 – 123</td>
</tr>
<tr>
<td>GET_CANVAS_PROPERTY</td>
<td>3 – 128</td>
</tr>
<tr>
<td>GET_FORM_PROPERTY</td>
<td>3 – 129</td>
</tr>
<tr>
<td>GET_GROUP_CHAR_CELL</td>
<td>3 – 133</td>
</tr>
<tr>
<td>GET_GROUP_DATE_CELL</td>
<td>3 – 135</td>
</tr>
<tr>
<td>GET_GROUP_NUMBER_CELL</td>
<td>3 – 136</td>
</tr>
<tr>
<td>GET_GROUP_RECORD_NUMBER</td>
<td>3 – 138</td>
</tr>
<tr>
<td>GET_GROUP_ROW_COUNT</td>
<td>3 – 139</td>
</tr>
<tr>
<td>GET_GROUP_SELECTION</td>
<td>3 – 140</td>
</tr>
<tr>
<td>GET_GROUP_SELECTION_COUNT</td>
<td>3 – 142</td>
</tr>
<tr>
<td>GET_ITEM_PROPERTY</td>
<td>3 – 143</td>
</tr>
<tr>
<td>GET_LIST_ELEMENT_COUNT</td>
<td>3 – 151</td>
</tr>
<tr>
<td>GET_LIST_ELEMENT_LABEL</td>
<td>3 – 153</td>
</tr>
<tr>
<td>GET_LIST_ELEMENT_VALUE</td>
<td>3 – 154</td>
</tr>
<tr>
<td>GET_LOV_PROPERTY</td>
<td>3 – 155</td>
</tr>
<tr>
<td>GET_MENU_ITEM_PROPERTY</td>
<td>3 – 156</td>
</tr>
<tr>
<td>GET_MESSAGE</td>
<td>3 – 158</td>
</tr>
<tr>
<td>GET_PARAMETER_ATTR</td>
<td>3 – 159</td>
</tr>
<tr>
<td>GET_PARAMETER_LIST</td>
<td>3 – 160</td>
</tr>
<tr>
<td>GET_RADIO_BUTTON_PROPERTY</td>
<td>3 – 160</td>
</tr>
<tr>
<td>GET_RECORD_PROPERTY</td>
<td>3 – 163</td>
</tr>
<tr>
<td>GET_RELATION_PROPERTY</td>
<td>3 – 165</td>
</tr>
<tr>
<td>GET_VIEW_PROPERTY</td>
<td>3 – 167</td>
</tr>
<tr>
<td>GET_WINDOW_PROPERTY</td>
<td>3 – 170</td>
</tr>
</tbody>
</table>
GO_BLOCK .......................................................... 3 – 171
GO_FORM .......................................................... 3 – 172
GO_ITEM ........................................................... 3 – 173
GO_RECORD ........................................................ 3 – 174
HELP ................................................................. 3 – 175
HIDE_MENU ......................................................... 3 – 175
HIDE_VIEW ......................................................... 3 – 176
HIDE_WINDOW ...................................................... 3 – 176
HOST ................................................................. 3 – 178
ID_NULL .............................................................. 3 – 180
IMAGE_ZOOM ....................................................... 3 – 181
INSERT_RECORD .................................................... 3 – 183
ISSUE_ROLLBACK .................................................. 3 – 184
ISSUE_SAVEPOINT .................................................. 3 – 185
ITEM_ENABLED ...................................................... 3 – 186
LAST_RECORD ......................................................... 3 – 186
LIST_VALUES ......................................................... 3 – 187
LOCK_RECORD ....................................................... 3 – 187
LOGON ............................................................... 3 – 188
LOGON_SCREEN ..................................................... 3 – 189
LOGOUT .............................................................. 3 – 190
MAIN_MENU ......................................................... 3 – 192
MENU_CLEAR_FIELD ................................................ 3 – 193
MENU_NEXT_FIELD .................................................. 3 – 193
MENU_PARAMETER .................................................. 3 – 194
MENU_PREVIOUS_FIELD ........................................... 3 – 195
MENU_REDISPLAY .................................................. 3 – 195
MENU_SHOW_KEYS ................................................. 3 – 195
MESSAGE .............................................................. 3 – 196
MESSAGE_CODE ..................................................... 3 – 197
MESSAGE_TEXT ....................................................... 3 – 198
MESSAGE_TYPE ..................................................... 3 – 199
MOVE_WINDOW ....................................................... 3 – 200
NAME_IN ............................................................... 3 – 202
NEW_FORM ............................................................. 3 – 205
NEXT_BLOCK ......................................................... 3 – 208
NEXT_FORM .......................................................... 3 – 209
NEXT_ITEM ........................................................... 3 – 210
NEXT_KEY ............................................................. 3 – 210
NEXT_MENU_ITEM .................................................... 3 – 211
NEXT_RECORD ......................................................... 3 – 211
NEXT_SET ............................................................. 3 – 212
OPEN_FORM .......................................................... 3 – 213

xlv Forms Reference Manual
SET_RADIO_BUTTON_PROPERTY ........................................ 3 – 274
SET_RECORD_PROPERTY .................................................. 3 – 276
SET_RELATION_PROPERTY ................................................. 3 – 278
SET_TIMER ................................................................ 3 – 280
SET_VIEW_PROPERTY ........................................................ 3 – 282
SET_WINDOW_PROPERTY ..................................................... 3 – 284
SHOW_ALERT ................................................................... 3 – 286
SHOW_BACKGROUND_MENU .................................................. 3 – 287
SHOWEDITOR .................................................................. 3 – 288
SHOW_KEYS .................................................................... 3 – 290
SHOW_LOV ...................................................................... 3 – 290
SHOW_MENU ..................................................................... 3 – 291
SHOW_VIEW ..................................................................... 3 – 292
SHOW_WINDOW ................................................................. 3 – 293
Synchronize .................................................................... 3 – 294
TERMINATE ...................................................................... 3 – 295
UNSET_GROUP_SELECTION ................................................... 3 – 295
UP ....................................................................................... 3 – 296
UPDATE_RECORD ................................................................. 3 – 296
USER_EXIT ........................................................................ 3 – 297
VALIDATE .......................................................................... 3 – 299
VBX.FIRE_EVENT ................................................................. 3 – 300
VBX.GET_PROPERTY ............................................................ 3 – 301
VBX.GET_VALUE_PROPERTY .................................................. 3 – 302
VBX.INVOKE_METHOD ........................................................... 3 – 303
VBX.SET_PROPERTY ............................................................... 3 – 304
VBX.SET_VALUE_PROPERTY .................................................... 3 – 305
WHERE_DISPLAY ................................................................. 3 – 306
WRITE_IMAGE_FILE .............................................................. 3 – 306

Chapter 4

System Variables ......................................................... 4 – 1
About System Variables .................................................. 4 – 2
Date and Time System Default Values ................................. 4 – 3
Local Variables .............................................................. 4 – 4
Uppercase Return Values .................................................. 4 – 4
$DATE$ ............................................................................. 4 – 5
$DATETIME$ ..................................................................... 4 – 6
$DBDATE$ ........................................................................ 4 – 7
$DBDATETIME$ ................................................................. 4 – 8
$DBTIME$ ......................................................................... 4 – 9
$TIME$ .............................................................................. 4 – 9
SYSTEM.BLOCK_STATUS .................................................. 4 – 10
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction (Radio Group)</td>
<td>5–71</td>
</tr>
<tr>
<td>Direction (Windows)</td>
<td>5–71</td>
</tr>
<tr>
<td>Directory</td>
<td>5–72</td>
</tr>
<tr>
<td>Display Width (LOV)</td>
<td>5–72</td>
</tr>
<tr>
<td>Display w/o Privilege</td>
<td>5–73</td>
</tr>
<tr>
<td>Display X Position, Display Y Position</td>
<td>5–73</td>
</tr>
<tr>
<td>Displayed (Item)</td>
<td>5–74</td>
</tr>
<tr>
<td>Displayed (Canvas-view)</td>
<td>5–74</td>
</tr>
<tr>
<td>Displayed (Menu Item)</td>
<td>5–75</td>
</tr>
<tr>
<td>Display Height</td>
<td>5–75</td>
</tr>
<tr>
<td>Display Width</td>
<td>5–76</td>
</tr>
<tr>
<td>Editor</td>
<td>5–76</td>
</tr>
<tr>
<td>Editor X Position, Editor Y Position</td>
<td>5–77</td>
</tr>
<tr>
<td>Enabled (Item)</td>
<td>5–77</td>
</tr>
<tr>
<td>Enabled (Menu Item)</td>
<td>5–78</td>
</tr>
<tr>
<td>Enterable</td>
<td>5–79</td>
</tr>
<tr>
<td>Execution Style</td>
<td>5–79</td>
</tr>
<tr>
<td>File</td>
<td>5–80</td>
</tr>
<tr>
<td>File_Name</td>
<td>5–81</td>
</tr>
<tr>
<td>Fire in Enter Query Mode</td>
<td>5–82</td>
</tr>
<tr>
<td>First Navigation Block</td>
<td>5–83</td>
</tr>
<tr>
<td>First_Block</td>
<td>5–83</td>
</tr>
<tr>
<td>First_Detail_Relation</td>
<td>5–84</td>
</tr>
<tr>
<td>First_Item</td>
<td>5–84</td>
</tr>
<tr>
<td>First_Master_Relation</td>
<td>5–85</td>
</tr>
<tr>
<td>Fixed Length(Item)</td>
<td>5–86</td>
</tr>
<tr>
<td>Fixed Length (Menu Substitution Parameter)</td>
<td>5–86</td>
</tr>
<tr>
<td>Fixed Size</td>
<td>5–87</td>
</tr>
<tr>
<td>Format Mask</td>
<td>5–88</td>
</tr>
<tr>
<td>Form_Name</td>
<td>5–94</td>
</tr>
<tr>
<td>Group_Name</td>
<td>5–94</td>
</tr>
<tr>
<td>Help</td>
<td>5–95</td>
</tr>
<tr>
<td>Hint (Item)</td>
<td>5–95</td>
</tr>
<tr>
<td>Hint (Menu Item)</td>
<td>5–96</td>
</tr>
<tr>
<td>Hint (Menu Substitution Parameter)</td>
<td>5–96</td>
</tr>
<tr>
<td>Horizontal MDI Toolbar</td>
<td>5–97</td>
</tr>
<tr>
<td>Horizontal Scroll Bar</td>
<td>5–97</td>
</tr>
<tr>
<td>Horizontal Toolbar</td>
<td>5–98</td>
</tr>
<tr>
<td>Icon Name</td>
<td>5–99</td>
</tr>
<tr>
<td>Icon Title</td>
<td>5–100</td>
</tr>
<tr>
<td>Iconic</td>
<td>5–100</td>
</tr>
<tr>
<td>Iconifiable</td>
<td>5–101</td>
</tr>
<tr>
<td>Identification</td>
<td>5–101</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>In Menu/Block Description</td>
<td>5 – 102</td>
</tr>
<tr>
<td>Inherit Menu</td>
<td>5 – 102</td>
</tr>
<tr>
<td>Initial Keyboard State</td>
<td>5 – 103</td>
</tr>
<tr>
<td>Insert Allowed (Block)</td>
<td>5 – 103</td>
</tr>
<tr>
<td>Insert Allowed (Item)</td>
<td>5 – 104</td>
</tr>
<tr>
<td>Item Type</td>
<td>5 – 105</td>
</tr>
<tr>
<td>Items Displayed</td>
<td>5 – 106</td>
</tr>
<tr>
<td>Item_Is_Valid</td>
<td>5 – 106</td>
</tr>
<tr>
<td>Join Condition</td>
<td>5 – 107</td>
</tr>
<tr>
<td>Keep Position</td>
<td>5 – 108</td>
</tr>
<tr>
<td>Key Mode</td>
<td>5 – 109</td>
</tr>
<tr>
<td>Label (Item)</td>
<td>5 – 110</td>
</tr>
<tr>
<td>Label (Menu Item)</td>
<td>5 – 110</td>
</tr>
<tr>
<td>Label (Menu Substitution Parameter)</td>
<td>5 – 111</td>
</tr>
<tr>
<td>Last_Block</td>
<td>5 – 111</td>
</tr>
<tr>
<td>Last_Item</td>
<td>5 – 112</td>
</tr>
<tr>
<td>Length (Record Group)</td>
<td>5 – 112</td>
</tr>
<tr>
<td>List Elements</td>
<td>5 – 112</td>
</tr>
<tr>
<td>List Item</td>
<td>5 – 112</td>
</tr>
<tr>
<td>List Item Value</td>
<td>5 – 112</td>
</tr>
<tr>
<td>List Style</td>
<td>5 – 113</td>
</tr>
<tr>
<td>Lock Record</td>
<td>5 – 113</td>
</tr>
<tr>
<td>Locking Mode</td>
<td>5 – 114</td>
</tr>
<tr>
<td>Long List</td>
<td>5 – 115</td>
</tr>
<tr>
<td>LOV</td>
<td>5 – 116</td>
</tr>
<tr>
<td>LOV for Validation</td>
<td>5 – 116</td>
</tr>
<tr>
<td>LOV Position</td>
<td>5 – 118</td>
</tr>
<tr>
<td>LOV Type</td>
<td>5 – 119</td>
</tr>
<tr>
<td>Magic Item</td>
<td>5 – 120</td>
</tr>
<tr>
<td>Main Menu</td>
<td>5 – 121</td>
</tr>
<tr>
<td>Master Deletes</td>
<td>5 – 122</td>
</tr>
<tr>
<td>Maximum Length</td>
<td>5 – 123</td>
</tr>
<tr>
<td>Maximum Length (Form Parameter)</td>
<td>5 – 124</td>
</tr>
<tr>
<td>Maximum Length (Menu Substitution Parameter)</td>
<td>5 – 124</td>
</tr>
<tr>
<td>Menu Item Radio Group</td>
<td>5 – 125</td>
</tr>
<tr>
<td>Menu Item Roles</td>
<td>5 – 125</td>
</tr>
<tr>
<td>Menu Item Type</td>
<td>5 – 126</td>
</tr>
<tr>
<td>Menu Module</td>
<td>5 – 128</td>
</tr>
<tr>
<td>Menu Module Roles</td>
<td>5 – 129</td>
</tr>
<tr>
<td>Menu Role</td>
<td>5 – 129</td>
</tr>
<tr>
<td>Menu Style</td>
<td>5 – 130</td>
</tr>
<tr>
<td>Message</td>
<td>5 – 130</td>
</tr>
<tr>
<td>Mirror Item</td>
<td>5 – 131</td>
</tr>
</tbody>
</table>
Modal ................................................. 5 – 132
Module_NLS_Lang ............................... 5 – 133
Mouse Navigate ................................. 5 – 134
Mouse Navigation Limit ..................... 5 – 135
Moveable ......................................... 5 – 135
Multi–Line ........................................ 5 – 136
Name ............................................... 5 – 136
Navigable ........................................ 5 – 138
Navigation Style ................................. 5 – 139
Next Navigation Block ....................... 5 – 140
Next Navigation Item ......................... 5 – 140
NextBlock ........................................ 5 – 141
NextItem .......................................... 5 – 142
Next_Detail_Relation ......................... 5 – 142
Next_Master_Relation ......................... 5 – 142
OLE Activation Style ......................... 5 – 143
OLE Class ......................................... 5 – 144
OLE Do In Out ................................... 5 – 145
OLE In–place Activation ...................... 5 – 146
OLE Popup Menu Items ....................... 5 – 147
OLE Resize Style ................................ 5 – 150
OLE Tenant Aspect ............................. 5 – 150
OLE Tenant Types ............................... 5 – 151
Operating_System ................................ 5 – 152
Optimizer_Hint .................................. 5 – 152
Order By .......................................... 5 – 153
Other Values ..................................... 5 – 154
Parameter Menus ................................ 5 – 154
Password ......................................... 5 – 155
Prevent Masterless Operations ............. 5 – 155
Previous Navigation Block ................. 5 – 156
Previous Navigation Item ................. 5 – 157
PreviousBlock .................................... 5 – 158
PreviousItem ...................................... 5 – 158
Primary Key (Block) ......................... 5 – 159
Primary Key (Item) ............................. 5 – 159
Quality ............................................ 5 – 160
Query Allowed (Block) ...................... 5 – 160
Query Allowed (Item) ......................... 5 – 161
Query Length .................................... 5 – 162
Query Only ....................................... 5 – 162
Query_Hits ....................................... 5 – 163
Query_Options .................................... 5 – 164
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title (Menu)</td>
<td>5 – 189</td>
</tr>
<tr>
<td>Title (Window)</td>
<td>5 – 189</td>
</tr>
<tr>
<td>Top Title</td>
<td>5 – 189</td>
</tr>
<tr>
<td>Top_Record</td>
<td>5 – 190</td>
</tr>
<tr>
<td>Transactional Triggers</td>
<td>5 – 190</td>
</tr>
<tr>
<td>Trigger Style</td>
<td>5 – 191</td>
</tr>
<tr>
<td>Trigger Text</td>
<td>5 – 191</td>
</tr>
<tr>
<td>Trigger Type</td>
<td>5 – 192</td>
</tr>
<tr>
<td>Unchecked Value</td>
<td>5 – 192</td>
</tr>
<tr>
<td>Update Allowed (Block)</td>
<td>5 – 193</td>
</tr>
<tr>
<td>Update Allowed (Item)</td>
<td>5 – 194</td>
</tr>
<tr>
<td>Update Changed Columns</td>
<td>5 – 195</td>
</tr>
<tr>
<td>Update_Column</td>
<td>5 – 196</td>
</tr>
<tr>
<td>Update Only if NULL</td>
<td>5 – 197</td>
</tr>
<tr>
<td>Update_Permission</td>
<td>5 – 198</td>
</tr>
<tr>
<td>Use File</td>
<td>5 – 199</td>
</tr>
<tr>
<td>Use Security</td>
<td>5 – 201</td>
</tr>
<tr>
<td>Use 3D Controls</td>
<td>5 – 201</td>
</tr>
<tr>
<td>Username</td>
<td>5 – 202</td>
</tr>
<tr>
<td>User_Interface</td>
<td>5 – 202</td>
</tr>
<tr>
<td>User_NLS_Lang</td>
<td>5 – 203</td>
</tr>
<tr>
<td>Validation</td>
<td>5 – 203</td>
</tr>
<tr>
<td>Validation Unit</td>
<td>5 – 204</td>
</tr>
<tr>
<td>Value</td>
<td>5 – 205</td>
</tr>
<tr>
<td>VBX Control File</td>
<td>5 – 205</td>
</tr>
<tr>
<td>VBX Control Name</td>
<td>5 – 206</td>
</tr>
<tr>
<td>VBX Control Value Property</td>
<td>5 – 206</td>
</tr>
<tr>
<td>Vertical MDI Toolbar</td>
<td>5 – 207</td>
</tr>
<tr>
<td>Vertical Scroll Bar</td>
<td>5 – 208</td>
</tr>
<tr>
<td>Vertical Toolbar</td>
<td>5 – 208</td>
</tr>
<tr>
<td>View</td>
<td>5 – 209</td>
</tr>
<tr>
<td>View Height, View Width</td>
<td>5 – 210</td>
</tr>
<tr>
<td>View Horizontal Scroll Bar</td>
<td>5 – 210</td>
</tr>
<tr>
<td>View Vertical Scroll Bar</td>
<td>5 – 211</td>
</tr>
<tr>
<td>Visible</td>
<td>5 – 211</td>
</tr>
<tr>
<td>Visual Attribute Name</td>
<td>5 – 213</td>
</tr>
<tr>
<td>Visual_Attribute</td>
<td>5 – 216</td>
</tr>
<tr>
<td>WHERE Clause/ORDER BY Clause</td>
<td>5 – 217</td>
</tr>
<tr>
<td>Width/Height (WD, HT)</td>
<td>5 – 218</td>
</tr>
<tr>
<td>Window</td>
<td>5 – 218</td>
</tr>
<tr>
<td>Window_Handle</td>
<td>5 – 219</td>
</tr>
<tr>
<td>Window_State</td>
<td>5 – 220</td>
</tr>
<tr>
<td>Window_Style</td>
<td>5 – 221</td>
</tr>
</tbody>
</table>
Replacing Default Commit Processing ........................................ 7 – 15
Locking .................................................................................. 7 – 16
Methods for Locking Rows .................................................... 7 – 17
Failing to Obtain a Lock .......................................................... 7 – 17
When Locks are Released ....................................................... 7 – 18
Replacing Default Locking ...................................................... 7 – 18

Chapter 8

Processing Flowcharts .............................................................. 8 – 1
ABORT_QUERY ....................................................................... 8 – 2
BLOCK_MENU ........................................................................ 8 – 3
CALL_FORM ........................................................................... 8 – 4
CALL_INPUT .......................................................................... 8 – 5
Check Block for Query ............................................................ 8 – 6
Check Block for Update ........................................................... 8 – 7
Check Item for Edit .................................................................. 8 – 8
Check Record Uniqueness ....................................................... 8 – 10
Check Record for Update ........................................................ 8 – 11
CLEAR_BLOCK ...................................................................... 8 – 12
CLEAR_EOL ............................................................................ 8 – 13
CLEAR_FORM ......................................................................... 8 – 14
CLEAR_ITEM ........................................................................... 8 – 15
CLEAR_RECORD ...................................................................... 8 – 16
Close the Query ................................................................. 8 – 17
COMMIT_FORM ...................................................................... 8 – 18
COPY ...................................................................................... 8 – 19
COUNT_QUERY ........................................................................ 8 – 20
CREATE_RECORD ................................................................... 8 – 21
DEFAULT_VALUE .................................................................... 8 – 22
DELETE_RECORD .................................................................... 8 – 23
DO_KEY .................................................................................. 8 – 25
DOWN .................................................................................... 8 – 26
DUPLICATE_ITEM ................................................................... 8 – 27
DUPLICATE_RECORD ............................................................... 8 – 28
ENTER ................................................................................... 8 – 29
Enter the Block ................................................................. 8 – 31
Enter the Form ................................................................. 8 – 32
Enter the Item ................................................................. 8 – 33
ENTER_QUERY ................................................................. 8 – 34
Enter the Record .............................................................. 8 – 35
Enter the Value into an Item .................................................. 8 – 36
EXECUTE_QUERY ............................................................. 8 – 37
Execute the Query .............................................................. 8 – 38
Savepoint ................................. 8 – 100
SCROLL_DOWN .......................... 8 – 101
SCROLL_UP .............................. 8 – 103
SHOW_LOV .............................. 8 – 105
UP ........................................ 8 – 107
Validate the Block ...................... 8 – 108
Validate the Form ....................... 8 – 109
Validate the Item ....................... 8 – 110
Validate the Record .................... 8 – 112

Appendix A

Compatibility with Prior Versions ......................... A – 1
About Upgrading Forms and Menus to Version 4.5 .... A – 2
Upgrading a Form ................................ A – 2
  Upgrading from Version 4.0 to Version 4.5 ........ A – 3
  Upgrading from Version 3.0 to Version 4.5 ........ A – 4
  Upgrading from Version 2.3 to Version 4.5 ........ A – 5
  Running an Application After Upgrading ........... A – 5
  GUI Platforms ................................ A – 5
Upgrading Forms Containing Referenced Objects ...... A – 6
Resolving Naming Conflicts in Form–Level Procedures A – 8
Upgrading File Formats ............................. A – 9
  Version 3.0 .INP Files .................... A – 10
Converting File Formats ............................ A – 10
Upgrading a Menu ................................ A – 11
  Upgrading from SQL*Menu 5.0 ................. A – 11
  Creating an .MMB File Only .................... A – 12
Upgrading Menu Security Roles ...................... A – 13
Output When Upgrading from Version 3.0 to Version 4.5 A – 14
  Triggers ................................ A – 14
  Fields ................................ A – 15
  Pages ................................ A – 15
  LOVs ................................ A – 15
  Master–Detail Block Relationships ............... A – 16
  Properties ................................ A – 16
  Packaged Procedures and Functions ............. A – 17
  Character–mode Applications .................... A – 17
Enhanced Version 3.0 Functionality .................... A – 18
  Integrated Forms and Menus .................... A – 18
  New Block Window .......................... A – 19
  Master–Detail Block Relationship ............... A – 19
  List of Values (LOV) ....................... A – 20
  Pages ................................ A – 20
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/SQL Libraries</td>
<td>A – 21</td>
</tr>
<tr>
<td>User Exits</td>
<td>A – 22</td>
</tr>
<tr>
<td>Calling Oracle Forms from a C Program</td>
<td>A – 22</td>
</tr>
<tr>
<td>New Features in Version 4.5</td>
<td>A – 23</td>
</tr>
<tr>
<td>Terminology Changes</td>
<td>A – 24</td>
</tr>
<tr>
<td>Terminology Related to Properties</td>
<td>A – 25</td>
</tr>
<tr>
<td>Terminology Related to System Variables</td>
<td>A – 25</td>
</tr>
<tr>
<td>Command Line Options</td>
<td>A – 26</td>
</tr>
<tr>
<td>Triggers, Built-ins, and Properties</td>
<td>A – 27</td>
</tr>
<tr>
<td>Triggers</td>
<td>A – 27</td>
</tr>
<tr>
<td>Built-In Subprograms</td>
<td>A – 29</td>
</tr>
<tr>
<td>Properties</td>
<td>A – 33</td>
</tr>
<tr>
<td>3.0 Packaged Procedures and Functions</td>
<td>A – 36</td>
</tr>
<tr>
<td>SQL*Menu Version 5.0 Packaged Procedures and Functions</td>
<td>A – 38</td>
</tr>
<tr>
<td>Moving from Character Mode to GUI</td>
<td>A – 39</td>
</tr>
<tr>
<td>User Expectations</td>
<td>A – 41</td>
</tr>
<tr>
<td>Migration Strategies</td>
<td>A – 41</td>
</tr>
<tr>
<td>Conversion Sequence</td>
<td>A – 43</td>
</tr>
<tr>
<td>Adding GUI Functionality</td>
<td>A – 44</td>
</tr>
<tr>
<td>Converting Key Triggers</td>
<td>A – 45</td>
</tr>
<tr>
<td>Widening Fields</td>
<td>A – 46</td>
</tr>
<tr>
<td>Creating a Checkbox from a Version 3.0 Checkbox Field</td>
<td>A – 47</td>
</tr>
<tr>
<td>Converting Pop-Up Pages</td>
<td>A – 48</td>
</tr>
</tbody>
</table>

**Appendix B**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module Storage</td>
<td>B – 1</td>
</tr>
<tr>
<td>Managing Modules</td>
<td>B – 2</td>
</tr>
<tr>
<td>Form Storage Formats</td>
<td>B – 3</td>
</tr>
<tr>
<td>Database Format</td>
<td>B – 3</td>
</tr>
<tr>
<td>.FMB Format</td>
<td>B – 4</td>
</tr>
<tr>
<td>.FMX Format</td>
<td>B – 4</td>
</tr>
<tr>
<td>.FMT Format</td>
<td>B – 5</td>
</tr>
<tr>
<td>Menu Storage Formats</td>
<td>B – 5</td>
</tr>
<tr>
<td>Database Format</td>
<td>B – 6</td>
</tr>
<tr>
<td>.MMB Format</td>
<td>B – 6</td>
</tr>
<tr>
<td>.MMX Format</td>
<td>B – 7</td>
</tr>
<tr>
<td>.MMT Format</td>
<td>B – 7</td>
</tr>
<tr>
<td>Library Storage Formats</td>
<td>B – 8</td>
</tr>
<tr>
<td>Database Format</td>
<td>B – 8</td>
</tr>
<tr>
<td>.PLL Format</td>
<td>B – 8</td>
</tr>
<tr>
<td>.PLL Format Stripped of Source Code</td>
<td>B – 9</td>
</tr>
<tr>
<td>.PLD Format</td>
<td>B – 9</td>
</tr>
<tr>
<td>Portability Among Systems</td>
<td>B – 10</td>
</tr>
<tr>
<td>Appendix C</td>
<td>Reserved Words ................................................. C – 1</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>Appendix D</td>
<td>Constraints ...................................................... D – 1</td>
</tr>
<tr>
<td></td>
<td>Object Constraints ............................................... D – 2</td>
</tr>
<tr>
<td></td>
<td>Cursor Constraints ............................................... D – 3</td>
</tr>
</tbody>
</table>
This chapter discusses the three components that make up Oracle Forms and the options that let you modify the behavior of each component. The chapter covers the following topics:

- About Oracle Forms Components 1–2
- Starting Oracle Forms Components 1–3
- Logging In to the Database 1–8
- Setting Runform Options 1–9
- Setting Generate Options 1–21
- Setting Designer Options 1–31

As a designer, you most often use Oracle Forms Designer, from which you can access both the Generate and Runform components. However, you will sometimes find it useful to run the other components individually from the command line.
### About Oracle Forms Components

Oracle Forms consists of the following programs, or components, which you can execute independently from the command line or by clicking on an icon:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Designer</strong></td>
<td>The Designer is the main component of Oracle Forms, and you can call Generate or Runform from the Designer. You use the Designer to create, generate, and run Oracle Forms applications. Using the Designer, you can create three types of modules: forms, menus, and libraries.</td>
</tr>
<tr>
<td><strong>Runform</strong></td>
<td>Form operators use the Runform component to run the completed application. You can also use Runform to test and debug forms during the design stage. Runform reads the machine-readable file created by Generate, and executes the form.</td>
</tr>
</tbody>
</table>
| **Generate** | Most often, you use Generate to create a machine-readable file that Runform can execute. Generate also allows you to convert various representations of a form. Using Generate, you can:  
  - Convert files between binary, text, and database module storage formats.  
  - Insert module definitions into database tables.  
  - Delete module definitions from the database.  
  - Regenerate application modules when porting to different platforms.  
  - Upgrade applications created with previous versions of Oracle Forms, SQL*Forms, and SQL*Menu. |
Starting Oracle Forms Components

Some platforms support icons, and some support command lines. You can start the Designer, Generate, or Runform components in one of two ways, depending on your computer platform:

icon  You will see a different icon for each component: Designer, Runform, and Generate. When you activate an icon, Oracle Forms starts the corresponding component. In the Designer, you go immediately to the Designer window, where you can enter options for the session by choosing Tools–>Options. In Runform or Generate, Oracle Forms displays a dialog. Use the dialog to enter the options you want to use for this session.

command line  When you start an Oracle Forms component by entering a command on the command line, you can indicate the options you want to use for this session by entering keyword parameters on the command line.

For more information on starting Oracle Forms components, refer to the Oracle Forms documentation for your operating system.
Starting Oracle Forms Components from the Command Line

To start any Oracle Forms component from the command line, enter this statement at the system prompt:

```
component_name [module_name] [userid/password] [parameters]
```

where:

- `component_name` Specifies the Oracle Forms component you want to use:
  - Runform
  - Generate
  - Designer

**Example:**

`f45run`—Starts Oracle Forms Runform component on Microsoft Windows, with no calls to the user exit interface.

To indicate that foreign functions accessible through the user exit interface have been linked into the executable, add an `x` to `component_name`.

**Example:**

`f45runmx`—Starts Oracle Forms Runform component on Motif, with calls to the user exit interface:

- `f45`—Indicates Oracle Forms Version 4.5
- `run, gen, or des`—Indicates the specific Oracle Forms component
- `m`—Indicates the window manager (m=Motif)
- `x`—Indicates that foreign functions accessible through the user exit interface have been linked into the executable

For more information on valid component names, refer to the Oracle Forms documentation for your operating system.
module_name

Specifies the module you want to load: a form, menu, or library name. If you omit the module name, Oracle Forms displays a dialog allowing you to choose the module to open.

userid/password

Specifies your ORACLE username and password.

parameters

Specifies any optional command line parameters you want to activate for this session. Optional parameters are entered in this format:

keyword1=value1 keyword2=value2...

Example:

f45run custform scott/tiger statistics=yes

Note: In this chapter, the examples assume that you’re running Oracle Forms on Microsoft Windows, with no calls to the user exit interface, so the Runform component name is shown as “f45run.” You should substitute the correct value of component_name for your platform and application.

Keyword Usage

There are three categories of parameters in Oracle Forms:

- MODULE and USERID
- options (command line parameters for setting options)
- form parameters

The first two parameters, MODULE and USERID, are unique because you can use either positional or keyword notation to enter them. Use keyword notation to enter optional parameters, on the command line. (Many optional parameters can also be set using dialogs.) Form parameters are optional input variables that are defined at design time for a specific form.
MODULE and USERID

If you enter the first two parameters, MODULE and USERID, in the specified order, you may omit the keywords and enter only values, as shown in the following example:

`f45run custform scott/tiger`

**Invalid Example:**

`f45run scott/tiger`

This sequence is invalid because the value for username/password is out of sequence, so it must be preceded by the USERID keyword. To use positional notation instead of keywords would require inserting the value of the MODULE parameter immediately after the component name, as in the previous example.

**Valid Examples:**

`f45run module=custform userid=scott/tiger`
`f45run userid=scott/tiger`
`f45run`

If you indicate only the module name, Oracle Forms will prompt you for module name and username/password.
Options

Use keyword notation for setting options on the command line. For information on options, see:

- Setting Runform Options: page 1 – 9
- Setting Generate Options: page 1 – 21
- Setting Designer Options: page 1 – 31

The following syntax rules apply to all keyword parameters, including options and form parameters:

- No spaces should be placed before or after the equal sign of an argument.
- Separate arguments with one or more spaces; do not use commas to separate arguments.

Invalid Example:

f45run custform scott/tiger statistics = yes
f45run custform scott/tiger statistics=yes,debug=yes

Valid Examples:

f45run custform scott/tiger statistics=yes
f45run custform scott/tiger statistics=yes debug=yes

Form Parameters

Form parameters are variables that you define at design time. Form parameters provide a simple mechanism for defining and setting the value of inputs that are required by a form at startup. Operators can specify values for form parameters by entering them on the command line using standard command line syntax.

The default value for a form parameter is taken from the Default Value field of the Properties window. The operator can override the default value when starting Runform by specifying a new value for the form parameter on the command line.

In the following example, myname_param is a user-defined form parameter that was defined in the Designer.

Note: If a form parameter value includes a space or punctuation, enclose the value in double quotes.

Example:

f45run empform scott/tiger myname_param="Msr. Dubois"
Displaying Hint Text on Command Line Options

To receive help on syntax and parameters, type the component name followed by “help=yes” at the system prompt.

Example:

f45run help=yes

Logging In to the Database

To explicitly log in to the database, use the USERID keyword or, in the Designer, choose File→Connect.

USERID

USERID is your ORACLE username and password with an optional SQL*Net connect string. The maximum length for the connect string is 255 characters.

If you omit your password, Oracle Forms displays the Login dialog, which prompts you for a username and password.

To log in, use one of the following forms:

username
username/password
username@node
username/password@node

Example:

You might specify the following command to run the ORDER_ENTRY form on the default database of the LONDON node:

f45run order_entry scott/tiger@D:london

For information on SQL*Net, refer to the SQL*Net User’s Guide. For help with your ORACLE username, see your Database Administrator.
Runform Options

Runform options specify Oracle Forms default behavior during a Runform session. You can set Runform options in two ways:

- Set options in the Runform Options dialog.
- Pass parameters to Oracle Forms on the command line when you invoke Runform.

In addition, you can set Runform options to specify the defaults for forms you run from the Designer in the Options dialog. To display the Options dialog, choose Tools–>Options.

**Note:** Runform options set in the Designer apply only to forms run from within the Designer.

The following chart lists the Runform options from the Options window and their corresponding keyword parameters. For information on a specific Runform option, see the corresponding parameter in the alphabetical list that follows the chart.

In the alphabetical list of Runform parameters, the following information is shown for each parameter:

- example, showing the parameter set to a value other than its default
- description
- default

If you enter these keyword parameters as command line options, you can enter more than one at a time, in any order:

```bash
f45run module=myform userid=scott/tiger debug=YES statistics=YES
```
<table>
<thead>
<tr>
<th>Runform Options and Keyword Parameters</th>
<th>Option Name</th>
<th>Keyword Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Oracle terminal resource file</td>
<td>Term</td>
</tr>
<tr>
<td></td>
<td>Run in debug mode</td>
<td>Debug</td>
</tr>
<tr>
<td></td>
<td>Debug messages</td>
<td>Debug_Messages*</td>
</tr>
<tr>
<td></td>
<td>Write input keystrokes to file</td>
<td>Keyout</td>
</tr>
<tr>
<td></td>
<td>Read input keystrokes from file</td>
<td>Keyin</td>
</tr>
<tr>
<td></td>
<td>Write output to file</td>
<td>Output_File</td>
</tr>
<tr>
<td></td>
<td>Write output to display</td>
<td>Interactive</td>
</tr>
<tr>
<td></td>
<td>Array SQL processing</td>
<td>Array</td>
</tr>
<tr>
<td></td>
<td>Buffer records to temporary file</td>
<td>Buffer_Records</td>
</tr>
<tr>
<td></td>
<td>Display screen to specify logon</td>
<td>Logon_Screen</td>
</tr>
<tr>
<td></td>
<td>Display block menu on startup</td>
<td>Block_Menu</td>
</tr>
<tr>
<td></td>
<td>Optimize V2-style trigger step SQL processing</td>
<td>OptimizeSQL</td>
</tr>
<tr>
<td></td>
<td>Optimize transaction mode processing</td>
<td>OptimizeTP</td>
</tr>
<tr>
<td></td>
<td>Run in quiet mode</td>
<td>Quiet</td>
</tr>
<tr>
<td></td>
<td>Show statistics</td>
<td>Statistics</td>
</tr>
<tr>
<td></td>
<td>Run in query only mode</td>
<td>Query_Only</td>
</tr>
<tr>
<td></td>
<td>Show help information</td>
<td>Help</td>
</tr>
<tr>
<td></td>
<td>Window state</td>
<td>Window_State</td>
</tr>
<tr>
<td></td>
<td>Collect PECS data?</td>
<td>PECS</td>
</tr>
<tr>
<td></td>
<td>Options screen</td>
<td>Options_Screen*</td>
</tr>
<tr>
<td></td>
<td>Session</td>
<td>Session*</td>
</tr>
</tbody>
</table>

*Use from command line only; not available from the Runform Options dialog.
Array (Runform)

Example: f45run module=myform userid=scott/tiger array=NO

Description: Use array processing during a Runform session.

When you suppress array processing, Oracle Forms requests that the database only return a single row of query results at a time from server to client.

Suppressing array processing usually results in the first retrieved record displaying faster than it would if you fetched a number of records with array processing. However, the total time required to fetch and display a number of records is shorter with array processing because network overhead can be reduced.

Option Name: Array SQL Processing
Default: YES

Block_Menu (Runform)

Example: f45run module=myform userid=scott/tiger block_menu=YES

Description: Automatically displays the block menu as the first screen (after the login screen, if it displays) instead of the form.

Option Name: Display Block Menu
Default: NO
Buffer_Records (Runform)

Example:    f45run module=myform userid=scott/tiger buffer_records=YES

Description:  Sets the number of records buffered in memory to the minimum allowable number of rows displayed plus 3 (for each block). If a block retrieves any records by a query beyond this minimum, Oracle Forms buffers these additional records to a temporary file on disk.

Setting this option saves Runform memory, but may slow down processing because of disk I/O.

Buffer_Records=NO tells Oracle Forms to set the minimum to the number specified using the Buffered property from each block.

Option Name:  Buffer Records to Temporary File

Default:    NO

Debug (Runform)

Example:    f45run module=myform userid=scott/tiger debug=YES

Description:  Invokes the debug mode for the Runform session. Debug mode invokes break processing if the BREAK built-in is used in any trigger or if you use the Help->Debug command from the Oracle Forms menu.

To invoke debug mode on non–Windows platforms, you must use the debug runform executable:

45runmd module=myform userid=scott/tiger debug=YES

Option Name:  Run in Debug Mode

Default:    NO

Debug_Messages (Runform)

Example:    f45run module=myform userid=scott/tiger debug_messages=YES

Description:  Debug_Messages displays ongoing messages about trigger execution while the form runs.

Default:    NO
Help (Runform)

Example: `f45run help=YES`

Description: Invokes the Oracle Forms help screen.

Option Name: Show Help Information
Default: NO

Interactive (Runform)

Example: `f45run module=myform userid=scott/tiger keyin=myfile.key output_file=mydisplay.out interactive=NO`

Description: Interactive specifies that, when you are using a keyscript file as input, Oracle Forms will display the output on the terminal screen (i.e., run interactively) as well as print the output to a file. Use Interactive=NO to suppress screen output when running forms in batch mode.

This parameter applies to character-mode terminals only.

Note: You must use the Keyin and Output_File parameters whenever you use Interactive. The Keyin file specifies the input, or keyscript, file; Output_File specifies the output, or display log, file.

Option Name: Write Output to Display
Default: YES

Keyin (Runform)

Example: `f45run module=myform userid=scott/tiger keyin=myfile.key`

Description: Allows you to read a keyscript file into a form as input. The keyscript file starts, executes, and ends the Runform session.

The file name specified is the input, or keyscript, file.

By default, Oracle Forms performs all actions on the terminal screen. If you want to suppress screen output, specify Interactive=NO and use Output_File to specify the output file.

This parameter applies to character-mode terminals only.

Option Name: Read Input Keystrokes from File
Keyout (Runform)

Example:  f45run module=myform userid=scott/tiger keyout=newfile.key

Description: Captures in a keysheet file the keystrokes involved during the Runform session. The keysheet file includes the keystrokes involved in navigating within a form, invoking functions, and performing transactions.

The file name specifies the output, or keysheet, file.

This parameter applies to character-mode terminals only.

Option Name: Write Input Keystrokes to File

Logon_Screen (Runform)

Example:  f45run module=myform userid=scott/tiger logon_screen=YES

Description: Forces the logon screen to display if you have not entered the password. Do not specify a username and password when you use Logon_Screen (Oracle Forms will ignore it if you do).

Use Logon_Screen when you do not want to type your password on the command line (where it is visible).

Option Name: Display Screen to Specify Logon

Default: NO
**OptimizeSQL (Runform)**

**Example:**
```bash
f45run module=myform userid=scott/tiger optimizesql=NO
```

**Description:** Specifies that Oracle Forms is to optimize SQL statement processing in V2-style triggers by sharing database cursors.

By default, Oracle Forms assigns a separate database cursor for each SQL statement that a form executes explicitly in a V2 trigger. This behavior enhances processing because the statements in each cursor need to be parsed only the first time they are executed in a Runform session—not every time.

When you specify OptimizeSQL=NO, Oracle Forms assigns a single cursor for all SQL statements in V2 triggers. These statements share, or reuse, that cursor. This behavior saves memory, but slows processing because the SQL statements must be parsed every time they are executed.

You can fine-tune this behavior through the New Cursor Area trigger step characteristic. If a trigger step that contains a SQL statement has this characteristic turned on, Oracle Forms assigns a separate cursor to the statement, in effect overriding the OptimizeSQL parameter for that statement.

**Note:** OptimizeSQL has no effect on statements in PL/SQL triggers.

**Option Name:** Optimize V2-Style Trigger Step SQL Processing

**Default:** YES
**OptimizeTP (Runform)**

**Example:**  
f45run module=myform userid=scott/tiger optimizetp=NO

**Description:** Optimizes transaction mode processing.

By default, Oracle Forms assigns a separate database cursor for each SQL statement that a form executes implicitly as part of posting or querying data. This behavior enhances processing because the statements in each cursor are parsed only the first time they are executed in a Runform session, not every time.

Note that the cursors that are assigned to query SELECT statements must be parsed every time they are executed. This exception exists because queries can vary from execution to execution.

When you specify OptimizeTP=NO, Oracle Forms assigns a separate cursor only for each query SELECT statement. All other implicit SQL statements share, or reuse, cursors. This behavior saves memory but slows processing because all INSERT, UPDATE, DELETE, and SELECT FOR UPDATE statements must be parsed every time they are executed.

**Restrictions:** The OptimizeTP parameter has no effect if you replace standard Oracle Forms processing with On–Insert, On–Update, and On–Delete triggers because these triggers replace the implicit issuance of INSERT, UPDATE, and DELETE statements.

**Option Name:** Optimize Transaction Mode Processing

**Default:** YES
Options_Screen (Runform)

Example:  
```
f45run module=myform userid=scott/tiger options_screen=YES
```

Description: Displays the Options window.
This parameter applies on GUI displays only.

Default: NO

Output_File (Runform)

Example:  
```
f45run module=myform userid=scott/tiger keyin=myfile.key 
output_file=mydisplay.out
```

Description: Captures the terminal output for a form in a display log file, as well as displaying it on the screen. If you want to suppress screen output, use Interactive=NO and then specify an Output_File.

This parameter applies to character-mode terminals only.

Note: You must use the Keyin parameter whenever you use Output_File. The Keyin file specifies the input, or keyscript, file; Output_File specifies the output, or display log, file.

Option Name: Write Output to File
PECS (Runform)

Example:  
f45run module=myform userid=scott/tiger PECS=ON

Description:  
Runs a form with Performance Event Collection Services (PECS) enabled.

PECS is a performance measurement tool you can use to perform the following tasks:

- Measure resource usage (CPU time per event or transactions processed per hour) of Oracle Forms or application-specific events
- Locate performance problems (elapsed time per event)
- Measure object coverage (whether a specific object, such as a trigger, alert, or window, is visited during test execution)
- Measure line-by-line coverage (for PL/SQL code in triggers and procedures)

The PECS option can be set to ON, OFF, or FULL:

- For object coverage, set \texttt{PECS=ON}
- For object coverage \textit{and} line coverage:
  - Generate with \texttt{Debug=ON}
  - Run with \texttt{PECS=FULL}
- The default is \texttt{PECS=OFF}

To use PECS on non-Windows platforms, you must use the debug runform executable:

45runmd module=myform userid=scott/tiger PECS=ON

For more information, see \textit{Oracle Forms Advanced Techniques}, Appendix C, "PECS."

Option Name:  
Collect PECS data?

Default:  
OFF
Query Only (Runform)

Example:  
```
f45run module=myform userid=scott/tiger query_only=YES
```

Description: Invokes the form in query-only mode. Setting this option to On is equivalent to using the CALL_FORM(query_only) built-in.

Option Name: Run in Query Only Mode

Default: NO

Quiet (Runform)

Example:  
```
f45run module=myform userid=scott/tiger quiet=YES
```

Description: Invokes the quiet mode for the Runform session. In quiet mode, messages do not produce an audible beep. You can explicitly ring the bell from a trigger by way of a call to the BELL built-in. The default of quiet=NO means that the bell rings. To turn off the bell, set quiet=YES.

Option Name: Run in Quiet Mode

Default: NO

Session (Runform)

Example:  
```
f45run module=myform userid=scott/tiger SESSION=YES
```

Description: Specifies whether Runform should allow forms invoked with OPEN_FORM to start a separate session. When one form opens another with OPEN_FORM('form', ACTIVATE, SESSION), Oracle Forms will open a separate session for the new form only if Runform is running with Session On. If Session is Off, Oracle Forms issues an error and does not open the form.

You can also set session On for all Runform invocations by setting the FORMS45_SESSION environment variable to TRUE. When you set the FORMS45_SESSION variable, all Runform invocations inherit its setting, unless you override it by setting the Session option on the command line.

Default: NO
Statistics (Runform)

Example:  
f45run module=myform userid=scott/tiger statistics=YES

Description:  
Displays a message at the end of the session that states the maximum number of simultaneous cursors that were used during the session. This message appears on the terminal screen, not on the message line.

This option also issues the following command to the database:

```
ALTER SESSION SET SQL_TRACE TRUE
```

This command enables the SQL trace facility for the current session, displaying the trace file directory on the server. For more information on this facility—which gathers database performance information—refer to the Oracle RDBMS Performance Tuning Guide.

If you are running a form within the Designer and you want to use this feature, activate the Statistics Runform option.

Option Name:  
Show Statistics

Default:  
NO

Term (Runform)

Example:  
f45run module=myform userid=scott/tiger term=resfile:mymapping

Description:  
Specifies a mapping other than the default mapping for the current device and product:

- **resfile**  
The file name specified is the name of your Oracle Terminal resource file. If you do not specify resfile, Oracle Forms defaults to a file name that is platform–specific, but begins with “FMR” on most platforms. For example, the MS Windows default file is FMRUSW.

- **mymapping**  
The mapping name specified is the mapping you want to use for this Oracle Forms session.

Note:  
You or the DBA define mappings with Oracle Terminal. For more information on resource files, refer to the Oracle forms documentation for your operating system.

Option Name:  
Oracle Terminal Resource File
Window_State (Runform)

Example:  

```plaintext
f45run module=myform userid=scott/tiger window_state=MAXIMIZE
```

Description: Sets the size of the MDI application window at the beginning of Runform.

When set to **MAXIMIZE**, the MDI application window is maximized at the beginning of a Runform session. When set to **MINIMIZE**, the MDI application window is minimized at the beginning of a Runform session. The **NORMAL** setting starts up an MDI application window that is normal size.

Option Name: Window State

Default: **NORMAL**

Restrictions: Valid only on MS Windows.

Setting Generate Options

Generate options specify Oracle Forms default behavior during a Generate session. Some of these options apply to file generation during development, for running and testing forms; other options apply only when you are converting files from earlier versions to Version 4.5.

You can set Generate options in two ways:

- Set options in the "Generate Options" dialog.
- Pass parameters to Oracle Forms on the command line when you invoke Generate.

The following chart lists the Generate options from the "Generate Options" window and their corresponding keyword parameters. For information on a specific Generate option, see the corresponding parameter in the alphabetical list that follows the chart.

In the alphabetical list of Generate parameters, the following information is shown for each parameter:

- example, showing the parameter set to a value other than its default
- relevant module type: Form, Menu, Library, or All
- description
- default
If you enter these keyword parameters as command line options, you can enter more than one at a time, in any order:

```plaintext
f45gen module=myform userid=scott/tiger batch=YES statistics=YES
```

<table>
<thead>
<tr>
<th><strong>Option Name</strong></th>
<th><strong>Keyword Parameter</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Module</td>
</tr>
<tr>
<td>Userid/Password</td>
<td>Userid</td>
</tr>
<tr>
<td>Module type is Form, Menu, or Library</td>
<td>Module_Type</td>
</tr>
<tr>
<td>Module access is File or Database</td>
<td>Module_Access</td>
</tr>
<tr>
<td>Generate in Debug mode</td>
<td>Debug</td>
</tr>
<tr>
<td>Show statistics</td>
<td>Statistics</td>
</tr>
<tr>
<td>Logon to the database</td>
<td>Logon</td>
</tr>
<tr>
<td>Write output to file</td>
<td>Output_File</td>
</tr>
<tr>
<td>Write script file</td>
<td>Script</td>
</tr>
<tr>
<td>Delete module from database</td>
<td>Delete</td>
</tr>
<tr>
<td>Insert module into database</td>
<td>Insert</td>
</tr>
<tr>
<td>Extract module from database into file</td>
<td>Extract</td>
</tr>
<tr>
<td>Upgrade 3.0 Form or 5.0 Menu to 4.5 Module</td>
<td>Upgrade</td>
</tr>
<tr>
<td>Upgrade SQL*Menu 5.0 table privileges</td>
<td>Upgrade_Roles</td>
</tr>
<tr>
<td>Version to upgrade</td>
<td>Version</td>
</tr>
<tr>
<td>CRT file to use when upgrading</td>
<td>CRT_File</td>
</tr>
<tr>
<td>Generate a runform/runmenu file when upgrading</td>
<td>Generate_on_Upsgrade</td>
</tr>
<tr>
<td>Add key–up and down triggers when upgrading</td>
<td>Add_Triggers</td>
</tr>
<tr>
<td>Add NOFAIL to exemacro steps when upgrading</td>
<td>Nofail</td>
</tr>
<tr>
<td>Show help information</td>
<td>Help</td>
</tr>
<tr>
<td>Options_Screen</td>
<td>Options_Screen*</td>
</tr>
<tr>
<td>Batch</td>
<td>Batch*</td>
</tr>
</tbody>
</table>

*Use from command line only; not available from the Generate Options dialog.*
Add_Triggers (Generate)

Example:  
```
f45gen module=myform userid=scott/tiger upgrade=yes version=23  
add_triggers=YES
```

Module:  Form

Description:  Indicates whether to add key–up and key–down triggers when  
upgrading from Forms 2.0 or 2.3 to 4.0 wherever KEY–PRVREC and  
KEY–NXTREC triggers existed.

Default:  NO

Batch (Generate)

Example:  
```
f45gen module=myform userid=scott/tiger batch=YES
```

Module:  Form

Description:  Suppresses interactive messages; use when performing a batch  
generation.

Default:  NO

CRT_File (Generate)

Example:  
```
f45gen module=myform userid=scott/tiger upgrade=yes version=20  
crt_file=myfile.crt
```

Module:  Form

Description:  Indicates CRT file to use when upgrading from SQL*Forms Version 2.0  
or 2.3.
**Debug (Generate)**

- **Example:** `f45gen module=myform userid=scott/tiger debug=yes`
- **Description:** Generates a debug–capable form.
  The debug Generate option creates entries in your .FMX file used by the runtime source-level debugger, so set `debug=yes` for Generate whenever you plan to set `debug=yes` for runtime.
- **Option Name:** Generate in Debug Mode
- **Default:** NO

---

**Delete (Generate)**

- **Example:** `f45gen module=myform userid=scott/tiger delete=YES`
- **Module:** All
- **Description:** Deletes the module directly from the database.
- **Default:** NO

---

**Extract (Generate)**

- **Example:** `f45gen module=myform userid=scott/tiger extract=YES`
- **Module:** All
- **Description:** Extracts the module from the database into a file with the same module name.
- **Default:** NO
### Generate_on_Upgrade (Generate)

**Example:**
```
f45gen module=myform userid=scott/tiger upgrade=YES
  generate_on_upgrade=NO
```

**Module:** Form, Menu

**Description:** Use the Generate option in conjunction with Upgrade. Oracle Forms creates two files when you specify `upgrade=YES` and omit `generate_on_upgrade`, thus accepting the default of `generate_on_upgrade=YES`:
- an upgraded binary design module (.FMB or .MMB file)
- an upgraded Runform executable module (.FMX or .MMX file)

If you do not want to automatically generate the Runform module, specify `generate_on_upgrade=NO`.

**Default:** YES

### Help (Generate)

**Example:**
```
f45gen help=YES
```

**Module:** All

**Description:** Invokes the Oracle Forms help screen.

**Default:** NO

### Insert (Generate)

**Example:**
```
f45gen module=myform userid=scott/tiger insert=YES
```

**Module:** All

**Description:** Inserts a module directly into the database from the Generate command line.

**Default:** NO
Logon (Generate)

Example:  
```
f45gen module=myform userid=scott/tiger logon=NO
```

Module:  
Form

Description:  Specifies whether Generate should log on to the database. If the module contains any PL/SQL code with table references, a connection will be required for generation.

Default:  YES

Module_Access (Generate)

Example:  
```
f45gen module=myform userid=scott/tiger module_access=database
```

Module:  
All

Description:  Specifies whether you want to open and save modules to the file system or to the database.

Default:  FILE

Module_Type (Generate)

Example:  
```
f45gen module=orders userid=scott/tiger module_type=menu
```

Module:  
All

Description:  Specifies module type for current module. By specifying Module_Type, you can have form, menu and library modules with the same name.

Default:  FORM
Nofail (Generate)

Example: `f45gen module=myform userid=scott/tiger upgrade=yes version=20
nofail=YES`

Module: Form

Description: Indicates whether to add the NOFAIL keyword to exemacro steps when upgrading from Forms 2.0 only.

Default: NO

Options_Screen (Generate)

Example: `f45gen module=myform userid=scott/tiger options_screen=YES`

Module: All

Description: Invokes the Options window. This parameter applies to GUI displays only.

Default: NO

Output_File (Generate)

Example: `f45gen module=myform userid=scott/tiger upgrade=yes
output_file=myform.fmb`

Module: All

Description: Specifies the file name for the generated file.

When used with `upgrade=yes`, `output_file` specifies:

- the complete name of the upgraded binary design module (.FMB,.MMB, or .PLL file)

  Note: To specify the name of the generated library file, you must use Strip_Source in conjunction with Output_File.

- the root name (without extension) of the upgraded Runform executable module (.FMX or .MMX file)

When used with `upgrade=yes` and `generate_on_upgrade=no`, the file extension is ignored.
Parse (Generate)

Example:  f45gen module=myform parse=YES
Module:  All
Description:  Converts the text file format of a module (.FMT, .MMT, .PLD) to a binary format (.FMB, .MMB, .PLL).
This operation can also be done from the Designer using the Convert command. For more information, refer to online Help in the Designer.
Default:  NO

Script (Generate)

Example:  f45gen module=myform script=YES
Module:  All
Description:  Converts a binary file format (.FMB, .MMB, or .PLL) to a text format (.FMT, .MMT, or .PLD).
This operation can also be done from the Designer using the Convert command. For more information, refer to online Help in the Designer.
Default:  NO
Statistics (Generate)

Example: `f45gen module=myform userid=scott/tiger statistics=YES`

Module: Form

Description: Displays a message at the end of the session listing the following statistics:

- form statistics (shareable memory size, dynamic memory size, total objects)
- trigger statistics
- block statistics
- item statistics

These statistics are useful for determining memory utilization and the effect of shared terminals.

Default: NO

Upgrade (Generate)

Example: `f45gen module=myform userid=scott/tiger upgrade=YES`

Module: Form, Menu

Description: Upgrades modules from SQL*Forms 2.0, 2.3, or 3.0 to Oracle Forms 4.5, or from SQL*Menu 5.0 to an Oracle Forms 4.5 menu module:

- To upgrade from SQL*Forms 3.0 or SQL*Menu 5.0 to Oracle Forms 4.5, specify `upgrade=yes` and omit `version`.
- To upgrade from SQL*Forms 2.0, specify `upgrade=yes` and `version=20`.
- To upgrade from SQL*Forms 2.3, specify `upgrade=yes` and `version=23`.

Default: NO
Upgrade_Roles (Generate)

**Example:**  
```
f45gen userid=system/manager upgrade_roles=YES
```

**Module:**  
none

**Description:**  
Upgrades SQL*Menu 5.0 table privileges to Oracle7 database roles.  

**Note:** Menu roles are independent of any specific menu application (no module name is specified). You cannot specify `upgrade=yes` and `upgrade_roles=yes` in one run.

**Default:**  
NO

Version (Generate)

**Example:**  
```
f45gen module=myform userid=scott/tiger upgrade=yes version=23
```

**Module:**  
Form

**Description:**  
Indicates version from which to upgrade. Use in conjunction with `upgrade=yes` to upgrade from version 2.3 (`version=23`) or version 2.0 (`version=20`).  
To upgrade from version 3.0, specify `upgrade=yes` and omit the `version` parameter.

**Default:**  
`version=30`
Widen_Fields (Generate)

Example:  
```
f45gen module=myform userid=scott/tiger upgrade=yes
widen_fields=YES
```

Module:  Form

Description:  Use the Widen_Fields option in conjunction with Upgrade. When upgrading to Version 4.5, the bevels on each field can cause the loss of up to one character per field. Specify this option when upgrading to automatically add one character to the Display Width of each field.

Note:  This has no effect on the maximum allowable data length.

This option is most useful for upgrading Oracle Forms 3.0 character-mode applications with a large number of 1–6 character fields. The effects of the Widen_Fields option will depend on your interface design, and should be tested carefully. Effects can include:

- Text items may overlap boilerplate text if space between fields is limited.
- If two fields are currently flush against each other, the Widen_Fields option will cause the fields to overlap.

Default:  NO

Setting Designer Options

Designer options specify Oracle Forms default behavior during a Designer session. Choose Tools–>Options in the Designer to invoke the Options dialog. To set your options, click on the check boxes or fill in file names for the options you choose.

The Options dialog includes both Designer options and Runtime options.

Designer Options

You can set the following design options to specify the defaults for the current Designer session:

- Save Before Generate
- Generate Before Run
- Suppress Hints
- Run Module Asynchronously
• Use System Editor
• Module Access (File, Database, File/Database)
• Module Filter (Forms, Menus, Libraries, All)
• Printer
• Color Palette
• Color Mode

For information on a specific design option, see the alphabetical list that follows.

Runtime Options

You can set the following Runtime options to specify the defaults for forms that you run from the Designer:

• Buffer Records in File
• Debug Mode
• Array Processing
• Optimize SQL Processing
• Optimize Transaction Mode Processing
• Statistics
• Display Block Menu
• Query Only Mode
• Quiet Mode

Runtime options are listed earlier in this chapter.

Keyword Parameters

In addition to the options listed in the Options dialog, you can set these keyword parameters on the Designer command line:

• Module_Type
• Module_Access
• Help
• Term

Example:

f45des module=orders userid=scott/tiger module_type=menu
Determine how an Oracle Forms color palette will be loaded on your system. Each time you load, open, or create a form, Oracle Forms loads the Oracle Forms color palette into your current system color table. Because this system color table can handle only a limited number of colors at once, Oracle Forms may not be able to accurately modify multiple forms simultaneously if they use different color palettes. For this reason, use the Read Only – Shared option except when you are actively modifying the Oracle Forms color palette.

**Color Mode options:**

*Editable* Select Editable mode only when you want to change the Oracle Forms color palette. Once you have changed the color palette, return to Read Only – Shared mode. In Editable mode, each color has its own unique entry in the current system color table, and if there is no more room in the table, the color palette may refuse to load.

To change the Oracle Forms color palette:

- Change Color Mode to Editable and save your options.
- Restart the Designer.
- Use Edit, Colors to make changes to the color palette.
- Use Edit, Export, Color Palette to save the Oracle Forms color palette to a file.
- Change your options to use the new color file.
- Change Color Mode back to Read Only – Shared and save your options.
- Restart the Designer.

*Read Only–Shared* In Read Only – Shared mode, Oracle Forms maps duplicate colors to the same entry in the current system color table before appending new entries from your Oracle Forms color palette. Read Only – Shared will help you avoid the color flicker that can result when you switch between Oracle Forms color palettes and is the recommended setting for Color Mode unless you are modifying the palette.
Read Only–Private  This option is provided for consistency with Oracle Graphics, and is not relevant for Oracle Forms. In Oracle Forms, it maps to Read Only – Shared.

For information about setting color mode, refer to online Help.

Default:  Read Only – Shared

Color Palette

Specifies the name of the Oracle Forms color palette that is automatically loaded when you create a new form. If this field is left blank, the Oracle Forms default color palette is loaded.

For information about creating a custom color palette, refer to online Help.

Generate Before Run

Determines whether Oracle Forms automatically generates the active module when you run a form from the Designer. When Generate Before Run is On, Oracle Forms does the following when you issue the File–>Run command and then run a specified form:

- generates the active form, menu, or library module to create an executable runfile having the same name as the module
- runs the .FMX file (form runfile) you specify in the Run dialog

This option lets you avoid issuing separate Generate and Run commands each time you modify and then run a form. However, this option does not save the module. You must issue the File–>Save command to save the module.

Also, when the Generate Before Run option is On, Oracle Forms does not automatically generate any menu or library modules attached to that form. You must generate menu and library modules separately before running a form that references them.

Default:  off
Help (Designer)

Example:  f45des help=YES
Module:  All
Description:  Invokes the Oracle Forms help screen.
Default:  NO

Module_Access (Designer)

Example:  f45des module=myform userid=scott/tiger module_access=database
Module:  All
Description:  Specifies whether you want to open and save modules to the file system or to the database.

This option can be set on a command line or on the Designer Options tab of the Options window.

The command line parameter establishes module access on a one–time basis for the current Designer session. On the command line, the Module_Access option can be set to file or database.

To set this option for future Designer sessions, use the Module Access option (Tools–>Options) to change your Preferences file.

In the Module Access option, you can specify one of the following storage preferences for opening and saving modules:

**File**  Modules are loaded from and saved to the file system.

**Database**  Modules are loaded from and saved to the database.

**File/Database**  Modules can be loaded from and saved to either the file system or the database. Oracle Forms prompts you each time you perform these operations to indicate an module access preference of File or Database.

If you specify Database storage for modules, Module Access also lets you set the filter conditions for the Database dialog. The filter conditions determine which modules you see in the Database dialog.
Specifies that the filter display all types of modules stored in the database.

Specifies that Oracle Forms display only forms, menus, or library modules in the Database dialog.

Default: FILE

---

**Module_Type (Designer)**

**Example:**

```
f45des module=orders userid=scott/tiger module_type=menu
```

**Module:** All

**Description:** Specifies module type for current module. By specifying Module_Type, you can have form, menu and library modules with the same name.

Default: FORM

---

**Printer**

The name of the default printer. This name is operating–system dependent.

For more information about printers, refer to the Oracle Forms documentation for your operating system.
Run Modules Asynchronously

Determines whether forms that you run from the Designer are executed synchronously or asynchronously with respect to the Designer itself:

- When Run Modules Asynchronously is Off, forms you run from the Designer are synchronous. That is, you cannot work in the Designer until you exit the form.
- When Run Modules Asynchronously is On, forms you run from the Designer are asynchronous, so you can move back and forth between the Designer and the form.

When you run a form synchronously, Oracle Forms notifies you of any Runform startup errors that occur by displaying an alert in the Designer. When you run a form asynchronously, no such communication between Runform and the Designer occurs, and Runform startup errors are not reported in the Designer.

Default: off

Save Before Generate

Determines whether Oracle Forms saves the current module automatically when you choose File–>Administration–>Generate.

Default: off

Suppress Hints

Determines whether hints are suppressed from the message line as you work in the Designer.

Default: off
Term (Designer)

Example:  
```
f45des module=myform userid=scott/tiger term=resfile:mymapping
```

Description: Specifies a mapping other than the default mapping for the current device and product:

- **resfile**: The file name specified is the name of your Oracle Terminal resource file. If you do not specify `resfile`, Oracle Forms defaults to a file name that is platform-specific, but begins with “FMR” on most platforms. For example, the MS Windows default file is FMRUSW.

- **mymapping**: The mapping name specified is the mapping you want to use for this Oracle Forms session.

For more information on resource files, refer to the Oracle forms documentation for your operating system.

**Note:** You or the DBA define mappings with Oracle Terminal.

---

Use System Editor

Determines which editor Oracle Forms uses when you invoke an editor from a multi-line text item in the Designer. When System Editor is Off, Oracle Forms displays the default Oracle Forms editor. When System Editor is On, Oracle Forms displays the default system editor defined on your system.

**Note:** If System Editor is On and you are using an editor with a native document format, you must save the document as ASCII text (with line breaks), instead of saving the document in that editor’s format.

For more information about defining the default system editors, refer to the Oracle Forms documentation for your operating system.

**Default:** Off
User Preference File

Although the Options dialog is the most convenient way to set your options, you can also set them directly in the preference file (usually called PREFS.ORA).

The preference file that enforces Oracle Forms options is automatically updated every time you change your preferences. Oracle Forms reads the updated preference file when you start the Designer. This file contains keywords and settings that allow you to preset each of the Oracle Forms Designer and Runform options.

You can use any of the Designer or Runform keyword parameters listed in this chapter in a user preference file. For example, to ensure that any form that you run from the Designer runs in quiet mode, you would include the following line in the user preference file:

```
FORMS.QUIT=ON
```

The preference file also allows you to preset a mapping for the Designer. On most platforms, the preference file must be named PREFS.ORA and must reside in the login directory.

If you start the Designer with a command line parameter that specifies a preference setting or mapping, the command line parameter overrides the setting in the preference file. Also, if a line in the preference file contains an error, Oracle Forms ignores that line when it reads the file.

Syntax for Options

To preset a Designer or Runform option, include the appropriate keyword and setting in the preference file, just as you would on the command line. Use the following syntax:

```
keyword = {on | off | string}
```

For a list of keywords and appropriate values, save your preferences, then examine the current contents of your PREFS.ORA file.
This chapter provides a full description of the triggers available in Oracle Forms.
About Triggers and Processes

Triggers are blocks of PL/SQL code that you write to perform specific tasks. There are pre-defined runtime events for which you can create triggers. Trigger names correspond to these events. For ease of description, the terms *events* and *triggers* are synonymous in this chapter. In effect, an Oracle Forms trigger is an event-handler written in PL/SQL to augment (or occasionally replace) the default processing behavior.

A *process* is a series of individual, related events that occurs during a specific Oracle Forms Runform operation. Oracle Forms includes navigational, validation, and database transaction processes. To see a visual representation of Oracle Forms processes showing where each trigger fires, refer to the flowchart for the process named in the “Fires In” section for each trigger. All the flowcharts are in the *Oracle Forms Reference Manual*, Vol. 2, Chapter 8, “Processing Flow Charts”.

SQL Statements in Trigger Text

The trigger descriptions in this chapter include a section called “Legal Commands.” This section lists the types of statements that are valid in the indicated trigger type, including:

- restricted built-in subprograms
- unrestricted built-in subprograms
- SELECT statements
- Data Manipulation Language (DML) statements

Restricted built-in subprograms initiate navigation. They include built-ins that move the input focus from one item to another, such as NEXT_ITEM, and those that involve database transactions, such as CREATE_RECORD. Restricted built-ins are illegal in triggers that fire in response to navigation, such as Pre- and Post- navigational triggers. Each built-in description includes a “Built-in Type” section that indicates whether the built-in is restricted or unrestricted. For more information, refer to the *Oracle Forms Developers Guide*, Chapter 7, “Writing Event Triggers.”

While you can write a trigger that uses any DML statement, Oracle Corporation advises that you follow the recommendations that are stated for each trigger type. Using DML statements in certain triggers can desynchronize the state of records in Oracle Forms and rows in the database, and can cause unexpected results.
Trigger Tables

The following cross references are included to help you locate the triggers you need in each category.

<table>
<thead>
<tr>
<th>Block Processing Triggers</th>
<th>Trigger Name</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>When–Clear–Block</td>
<td>2 – 65</td>
</tr>
<tr>
<td></td>
<td>When–Create–Record</td>
<td>2 – 66</td>
</tr>
<tr>
<td></td>
<td>When–Database–Record</td>
<td>2 – 68</td>
</tr>
<tr>
<td></td>
<td>When–Remove–Record</td>
<td>2 – 84</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interface Event Triggers</th>
<th>Trigger Name</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>When–Button–Pressed</td>
<td>2 – 63</td>
</tr>
<tr>
<td></td>
<td>When–Checkbox–Changed</td>
<td>2 – 64</td>
</tr>
<tr>
<td></td>
<td>When–Custom–Item–Event</td>
<td>2 – 67</td>
</tr>
<tr>
<td></td>
<td>When–Image–Activated</td>
<td>2 – 69</td>
</tr>
<tr>
<td></td>
<td>When–Image–Pressed</td>
<td>2 – 70</td>
</tr>
<tr>
<td></td>
<td>When–List–Changed</td>
<td>2 – 71</td>
</tr>
<tr>
<td></td>
<td>When–Mouse–Click</td>
<td>2 – 71</td>
</tr>
<tr>
<td></td>
<td>When–Mouse–DoubleClick</td>
<td>2 – 72</td>
</tr>
<tr>
<td></td>
<td>When–Mouse–Down</td>
<td>2 – 74</td>
</tr>
<tr>
<td></td>
<td>When–Mouse–Enter</td>
<td>2 – 75</td>
</tr>
<tr>
<td></td>
<td>When–Mouse–Leave</td>
<td>2 – 76</td>
</tr>
<tr>
<td></td>
<td>When–Mouse–Move</td>
<td>2 – 77</td>
</tr>
<tr>
<td></td>
<td>When–Mouse–Up</td>
<td>2 – 78</td>
</tr>
<tr>
<td></td>
<td>When–Radio–Changed</td>
<td>2 – 83</td>
</tr>
<tr>
<td></td>
<td>When–Timer–Expired</td>
<td>2 – 85</td>
</tr>
<tr>
<td></td>
<td>When–Window–Activated</td>
<td>2 – 91</td>
</tr>
<tr>
<td></td>
<td>When–Window–Closed</td>
<td>2 – 92</td>
</tr>
<tr>
<td></td>
<td>When–Window–Deactivated</td>
<td>2 – 92</td>
</tr>
<tr>
<td></td>
<td>When–Window–Resized</td>
<td>2 – 93</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key Triggers</th>
<th>Trigger Name</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Function Key</td>
<td>2 – 6</td>
</tr>
<tr>
<td></td>
<td>Key–Fn</td>
<td>2 – 8</td>
</tr>
<tr>
<td></td>
<td>Key–Others</td>
<td>2 – 9</td>
</tr>
<tr>
<td>Master–Detail Triggers</td>
<td>Trigger Name</td>
<td>Page Number</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>On–Check–Delete–Master</td>
<td>2 – 10</td>
</tr>
<tr>
<td></td>
<td>On–Clear–Details</td>
<td>2 – 12</td>
</tr>
<tr>
<td></td>
<td>On–Populate–Details</td>
<td>2 – 26</td>
</tr>
<tr>
<td>Message–Handling Triggers</td>
<td>Trigger Name</td>
<td>Page Number</td>
</tr>
<tr>
<td></td>
<td>On–Error</td>
<td>2 – 18</td>
</tr>
<tr>
<td></td>
<td>On–Message</td>
<td>2 – 25</td>
</tr>
<tr>
<td>Navigational Triggers</td>
<td>Trigger Name</td>
<td>Page Number</td>
</tr>
<tr>
<td></td>
<td>Post–Block</td>
<td>2 – 32</td>
</tr>
<tr>
<td></td>
<td>Post–Form</td>
<td>2 – 37</td>
</tr>
<tr>
<td></td>
<td>Post–Record</td>
<td>2 – 44</td>
</tr>
<tr>
<td></td>
<td>Post–Text–Item</td>
<td>2 – 46</td>
</tr>
<tr>
<td></td>
<td>Pre–Block</td>
<td>2 – 48</td>
</tr>
<tr>
<td></td>
<td>Pre–Form</td>
<td>2 – 51</td>
</tr>
<tr>
<td></td>
<td>Pre–Record</td>
<td>2 – 57</td>
</tr>
<tr>
<td></td>
<td>Pre–Text–Item</td>
<td>2 – 59</td>
</tr>
<tr>
<td></td>
<td>When–Form–Navigate</td>
<td>2 – 69</td>
</tr>
<tr>
<td></td>
<td>When–New–Block–Instance</td>
<td>2 – 79</td>
</tr>
<tr>
<td></td>
<td>When–New–Form–Instance</td>
<td>2 – 80</td>
</tr>
<tr>
<td></td>
<td>When–New–Item–Instance</td>
<td>2 – 81</td>
</tr>
<tr>
<td></td>
<td>When–New–Record–Instance</td>
<td>2 – 82</td>
</tr>
<tr>
<td></td>
<td>User–Named</td>
<td>2 – 62</td>
</tr>
<tr>
<td>Query–Time Triggers</td>
<td>Trigger Name</td>
<td>Page Number</td>
</tr>
<tr>
<td></td>
<td>Post–Query</td>
<td>2 – 42</td>
</tr>
<tr>
<td></td>
<td>Pre–Query</td>
<td>2 – 55</td>
</tr>
<tr>
<td>Transactional Triggers</td>
<td>Trigger Name</td>
<td>Page Number</td>
</tr>
<tr>
<td></td>
<td>On–Check–Delete–Master</td>
<td>2 – 10</td>
</tr>
<tr>
<td></td>
<td>On–Check–Unique</td>
<td>2 – 11</td>
</tr>
<tr>
<td></td>
<td>On–Close</td>
<td>2 – 13</td>
</tr>
<tr>
<td></td>
<td>On–Column–Security</td>
<td>2 – 14</td>
</tr>
<tr>
<td></td>
<td>On–Commit</td>
<td>2 – 15</td>
</tr>
<tr>
<td></td>
<td>On–Count</td>
<td>2 – 16</td>
</tr>
<tr>
<td></td>
<td>On–Delete</td>
<td>2 – 17</td>
</tr>
<tr>
<td>Trigger Name</td>
<td>Page Number</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>On–Fetch</td>
<td>2 – 19</td>
<td></td>
</tr>
<tr>
<td>On–Insert</td>
<td>2 – 21</td>
<td></td>
</tr>
<tr>
<td>On–Lock</td>
<td>2 – 22</td>
<td></td>
</tr>
<tr>
<td>On–Logon</td>
<td>2 – 23</td>
<td></td>
</tr>
<tr>
<td>On–Logout</td>
<td>2 – 24</td>
<td></td>
</tr>
<tr>
<td>On–Rollback</td>
<td>2 – 27</td>
<td></td>
</tr>
<tr>
<td>On–Savepoint</td>
<td>2 – 28</td>
<td></td>
</tr>
<tr>
<td>On–Select</td>
<td>2 – 29</td>
<td></td>
</tr>
<tr>
<td>On–Update</td>
<td>2 – 31</td>
<td></td>
</tr>
<tr>
<td>Post–Change</td>
<td>2 – 33</td>
<td></td>
</tr>
<tr>
<td>Post–Database–Commit</td>
<td>2 – 35</td>
<td></td>
</tr>
<tr>
<td>Post–Delete</td>
<td>2 – 36</td>
<td></td>
</tr>
<tr>
<td>Post–Forms–Commit</td>
<td>2 – 38</td>
<td></td>
</tr>
<tr>
<td>Post–Insert</td>
<td>2 – 39</td>
<td></td>
</tr>
<tr>
<td>Post–Logon</td>
<td>2 – 40</td>
<td></td>
</tr>
<tr>
<td>Post–Logout</td>
<td>2 – 41</td>
<td></td>
</tr>
<tr>
<td>Post–Select</td>
<td>2 – 45</td>
<td></td>
</tr>
<tr>
<td>Post–Update</td>
<td>2 – 47</td>
<td></td>
</tr>
<tr>
<td>Pre–Commit</td>
<td>2 – 49</td>
<td></td>
</tr>
<tr>
<td>Pre–Delete</td>
<td>2 – 50</td>
<td></td>
</tr>
<tr>
<td>Pre–Insert</td>
<td>2 – 52</td>
<td></td>
</tr>
<tr>
<td>Pre–Logon</td>
<td>2 – 53</td>
<td></td>
</tr>
<tr>
<td>Pre–Logout</td>
<td>2 – 54</td>
<td></td>
</tr>
<tr>
<td>Pre–Select</td>
<td>2 – 58</td>
<td></td>
</tr>
<tr>
<td>Pre–Update</td>
<td>2 – 60</td>
<td></td>
</tr>
</tbody>
</table>

**Validation Triggers**

<table>
<thead>
<tr>
<th>Trigger Name</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>When–Validate–Item</td>
<td>2 – 87</td>
</tr>
<tr>
<td>When–Validate–Record</td>
<td>2 – 89</td>
</tr>
</tbody>
</table>
Function Key Triggers

Function key triggers are associated with individual Runform function keys. A function key trigger fires only when an operator presses the associated function key. The actions you define in a function key trigger replace the default action that the function key would normally perform.

The following table shows all function key triggers and the corresponding Runform function keys.

<table>
<thead>
<tr>
<th>Key Trigger</th>
<th>Associated Function Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key–CLRBLK</td>
<td>[Clear Block]</td>
</tr>
<tr>
<td>Key–CLFRM</td>
<td>[Clear Form]</td>
</tr>
<tr>
<td>Key–CLRREC</td>
<td>[Clear Record]</td>
</tr>
<tr>
<td>Key–COMMIT</td>
<td>[Accept]</td>
</tr>
<tr>
<td>Key–CQUERY</td>
<td>[Count Query Hits]</td>
</tr>
<tr>
<td>Key–CEREC</td>
<td>[Insert Record]</td>
</tr>
<tr>
<td>Key–DELREC</td>
<td>[Delete Record]</td>
</tr>
<tr>
<td>Key–DOWN</td>
<td>[Down]</td>
</tr>
<tr>
<td>Key–DUP–ITEM</td>
<td>[Duplicate Item]</td>
</tr>
<tr>
<td>Key–DUPREC</td>
<td>[Duplicate Record]</td>
</tr>
<tr>
<td>Key–EDIT</td>
<td>[Edit]</td>
</tr>
<tr>
<td>Key–ENTQRY</td>
<td>[Enter Query]</td>
</tr>
<tr>
<td>Key–EXEQRY</td>
<td>[Execute Query]</td>
</tr>
<tr>
<td>Key–EXIT</td>
<td>[Exit]</td>
</tr>
<tr>
<td>Key–HELP</td>
<td>[Help]</td>
</tr>
<tr>
<td>Key–LISTVAL</td>
<td>[List of Values]</td>
</tr>
<tr>
<td>Key–MENU</td>
<td>[Block Menu]</td>
</tr>
<tr>
<td>Key–NXTBLK</td>
<td>[Next Block]</td>
</tr>
<tr>
<td>Key–NXT–ITEM</td>
<td>[Next Item]</td>
</tr>
<tr>
<td>Key–NXTKEY</td>
<td>[Next Primary Key]</td>
</tr>
<tr>
<td>Key–NXTREC</td>
<td>[Next Record]</td>
</tr>
<tr>
<td>Key–NXTSET</td>
<td>[Next Set of Records]</td>
</tr>
<tr>
<td>Key–PRINT</td>
<td>[Print]</td>
</tr>
<tr>
<td>Key–PRVBLK</td>
<td>[Previous Block]</td>
</tr>
<tr>
<td>Key Trigger</td>
<td>Associated Function Key</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>Key–PRV–ITEM</td>
<td>[Previous Item]</td>
</tr>
<tr>
<td>Key–PRVREC</td>
<td>[Previous Record]</td>
</tr>
<tr>
<td>Key–SCRDOWN</td>
<td>[Scroll Down]</td>
</tr>
<tr>
<td>Key–SCRUP</td>
<td>[Scroll Up]</td>
</tr>
<tr>
<td>Key–UP</td>
<td>[Up]</td>
</tr>
<tr>
<td>Key–UPDREC</td>
<td>Equivalent to Record, Lock command on the default menu</td>
</tr>
</tbody>
</table>

Note that you cannot redefine all Runform function keys with function key triggers. Specifically, you cannot ever redefine the following static function keys because they are often performed by the terminal or user interface management system and not by Oracle Forms.

- [Clear Item]
- [First Line]
- [Scroll Left]
- [Copy]
- [Insert Line]
- [Scroll Right]
- [Cut]
- [Last Line]
- [Search]
- [Delete Character]
- [Left]
- [Select]
- [Delete Line]
- [Paste]
- [Show Keys]
- [Display Error]
- [Refresh]
- [Toggle Insert/Replace]
- [End of Line]
- [Right]
- [Transmit]

**Definition Level:** form, block, or item

**Legal Commands:** SELECT statements, restricted built-ins, unrestricted built-ins

**Enter Query Mode:** yes

**Usage Notes:** The default functionality performed by the following keys is not allowed in Enter Query mode:

- [Clear Block]
- [Down]
- [Next Record]
- [Clear Form]
- [Duplicate Item]
- [Next Set of Records]
- [Clear Record]
- [Duplicate Record]
- [Previous Block]
- [Accept]
- [Block Menu]
- [Previous Record]
- [Insert Record]
- [Next Block]
- [Up]
- [Delete Record]
- [Next Primary Key]
- [Lock Record]
Common Uses: Use function key triggers to perform the following tasks:

- Disable function keys dynamically.
- Replace the default behavior of function keys.
- Dynamically remap function keys.
- Perform complex or multiple functions with a single key or key sequence.

Restrictions: 

- Oracle Forms ignores some function key triggers in Edit mode.
- Oracle Forms ignores the Key–Commit trigger when an operator presses [Commit/Accept] in a dialog box.

Key–Fn Trigger

A Key–Fn trigger fires when an operator presses the associated key.

You can attach Key–Fn triggers to 10 keys or key sequences that normally do not perform any Oracle Forms operations. These keys are referred to as Key–F0 through Key–F9. Before you can attach key triggers to these keys, you or the DBA must use Oracle Terminal to map the keys to the appropriate functions.

Definition Level: form, block, or item

Legal Commands: SELECT statements, restricted built–ins, unrestricted built–ins

Enter Query Mode: yes

Usage Notes: Use Key–Fn triggers to create additional function keys for custom functions.

Restrictions: Oracle Forms ignores Key–Fn triggers in Edit mode.
Key–Others Trigger

A Key–Others trigger fires when an operator presses the associated key.

A Key–Others trigger is associated with all keys that can have key triggers associated with them but are not currently defined by function key triggers (at any level).

A Key–Others trigger overrides the default behavior of a Runform function key (unless one of the following restrictions apply). When this occurs, however, Oracle Forms still displays the function key’s default entry in the Show Keys screen.

**Trigger Type:** key

**Definition Level:** form, block, or item

**Legal Commands:** SELECT statements, restricted built-ins, unrestricted built-ins

**Enter Query Mode:** yes

**Usage Notes:** Use Key–Others triggers to limit an operator’s possible actions. Specifically, use Key–Others triggers to perform the following tasks:

- Disable all keys that are not relevant in a particular situation.
- Perform one specific action whenever an operator presses any key.

**Restrictions:** Oracle Forms ignores a Key–Others trigger under the following conditions:

- The form is in Enter Query mode and Fire in Enter Query Mode is Off.
- A list of values, the Show Keys screen, a help screen, or an error screen is displayed.
- The operator is responding to a Runform prompt.
- The operator presses a static function key.
On-Check-Delete-Master Trigger

Oracle Forms creates this trigger automatically when you define a master-detail relation and set the Master Deletes property to Non-Isolated. It fires when there is an attempt to delete a record in the master block of a master-detail relation.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, DML statements (DELETE, INSERT, UPDATE), unrestricted built-ins

**Enter Query Mode:** no

**On Failure:** Prevents the deletion of the current master record

**Fires In:** Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:

Master-Detail Coordination

**Example:** The following example replaces the On-Check-Delete-Master that is generated by default for a master/detail relation with a trigger that will fail if the sum of the distributions does not equal the purchase order total.

```sql
DECLARE
    the_sum NUMBER;
BEGIN
    SELECT SUM(dollar_amt)
    INTO the_sum
    FROM po_distribution
    WHERE po_number = :purchase_order.number;

    IF the_sum <> :purchase_order.total THEN
        Message('PO Distributions do not reconcile.');
        RAISE Form_Trigger_Failure;
    END IF;
END;
```
On–Check–Unique Trigger

During a commit operation, the On–Check–Unique trigger fires when Oracle Forms normally checks that primary key values are unique before inserting or updating a record in a base table. It fires once for each record that has been inserted or updated.

Replaces: The default processing for checking record uniqueness. When a block has the Primary Key property set to True, Oracle Forms, by default, checks the uniqueness of a record by constructing and executing the appropriate SQL statement to select for rows that match the current record’s primary key values. If a duplicate row is found, Oracle Forms displays message FRM–40600: Record has already been inserted.

For a record that has been marked for insert, Oracle Forms always checks for unique primary key values. In the case of an update, Oracle Forms checks for unique primary key values only if one or more items that have the Primary Key item property have been modified.

Definition Level: form, block

Legal Commands: SELECT statements, PL/SQL, unrestricted built–ins

Enter Query Mode: no

Usage Notes: To perform the default processing from this trigger, call the CHECK_RECORD_UNIQUENESS built–in.

On Failure: no effect

Fires In: Refer to the following process flowcharts in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, ”Processing Flow Charts”:

Check Record Uniqueness
Post and Commit Transactions

Example: The following example verifies that the current record in question does not already exist in the DEPT table.
DECLARE
   CURSOR chk_unique IS SELECT 'x'
       FROM dept
       WHERE deptno = :dept.deptno;
   tmp VARCHAR2(1);
BEGIN
   OPEN chk_unique;
   FETCH chk_unique INTO tmp;
   CLOSE chk_unique;
   IF tmp IS NOT NULL THEN
      Message('This department already exists.');
      RAISE Form_Trigger_Failure;
   END IF;
END;

On–Clear–Details Trigger

Fires when a coordination–causing event occurs in a block that is a master block in a master–detail relation. A coordination–causing event is any event that makes a different record the current record in the master block.

Definition Level: form, block

Legal Commands: Any command, including restricted built–ins

Enter Query Mode: no

Usage Notes: Oracle Forms creates the On–Clear–Details trigger automatically when you define a master–detail block relation.

On Failure: Causes the coordination–causing operation and any scheduled coordination triggers to abort.

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts":
Master Detail Coordination
On–Close Trigger

Fires when an operator or the application causes a query to close. By default, Oracle Forms closes a query when all of the records identified by the query criteria have been fetched, or when the operator or the application aborts the query.

The On–Close trigger augments the normal Oracle Forms “close cursor” phase of a query.

Definition Level: form
Legal Commands: SELECT statements, PL/SQL, unrestricted built–ins
Enter Query Mode: no
Usage Notes:
• Use an On–Close trigger after using the On–Select or On–Fetch triggers, specifically, to close files, close cursors, and free memory.
• The On–Close trigger fires automatically when the ABORT_QUERY built–in is called from an On–Select trigger.
On Failure: no effect
Fires In: Refer to the following process flowcharts in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:
ABORT_QUERY
Close The Query
Example: The following example releases memory being used by a user–defined data access method via the transactional triggers.

```
BEGIN
  IF NOT my_data source_open('DX110_DEPT') THEN
    my_data source_close('DX110_DEPT');
  END IF;
END;
```
On–Column–Security Trigger

Fires when Oracle Forms would normally enforce column–level security for each block that has the Column Security block property set On.

Replaces: The normal Oracle Forms processing for checking column security. By default, Oracle Forms enforces column security by querying the database to determine the base table columns to which the current form operator has update privileges. For columns to which the operator does not have update privileges, Oracle Forms makes the corresponding base table items in the form non–updateable by setting the Update Allowed item property Off dynamically. Oracle Forms performs this operation at form startup, processing each block in sequence.

Definition Level: form, block

Legal Commands: SELECT statements, PL/SQL, unrestricted built–ins

Enter Query Mode: no

Usage Notes: To perform the default processing from this trigger, call the ENFORCE_COLUMN_SECURITY built–in.

On Failure: no effect

Example: The following example sets salary and commission text items in the current block to disabled and non–updateable, unless the SUPERUSER role is enabled. Only users with the user–defined SUPERUSER role can change these number fields.

```
DECLARE
  itm_id Item;
  on_or_off NUMBER;
BEGIN
  IF NOT role_is_set('SUPERUSER') THEN
    on_or_off := PROPERTY_OFF;
  ELSE
    on_or_off := PROPERTY_ON;
  END IF;
  itm_id := Find_Item('Emp.Sal');
  Set_Item_Property(itm_id,ENABLED,on_or_off);
  Set_Item_Property(itm_id,UPDATEABLE,on_or_off);
  itm_id := Find_Item('Emp.Comm');
```
On–Commit Trigger

Fires whenever Oracle Forms would normally issue a database commit statement to finalize a transaction. By default, this operation occurs after all records that have been marked as updates, inserts, and deletes have been posted to the database.

**Replaces:** The default COMMIT statement that Oracle Forms issues to finalize a transaction during the Post and Commit Transactions process.

**Definition Level:** form

**Legal Commands:** SELECT statements, PL/SQL, unrestricted built–ins

**Enter Query Mode:** no

**Usage Notes:**
- Use an On–Commit trigger to change the conditions of normal Oracle Forms commit processing to fit the particular requirements of a commit to a non–ORACLE database.
- To perform the default processing from this trigger, call to the COMMIT_FORM built–in.

**On Failure:** Aborts Post and Commit processing

**Fires In:** Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:

Post and Commit Transactions

**Example:** This example disables the commit operation when running against a datasource that does not support transaction control. If the application is running against ORACLE, the commit operation behaves normally.

```sql
BEGIN
  IF Get_Application_Property(DATA_SOURCE) = 'ORACLE' THEN
    Commit_Form;
  END IF;
  /*
   ** Otherwise, no action is performed
   */
END;
```
On–Count Trigger

Fires when Oracle Forms would normally perform default Count Query processing to determine the number of rows in the database that match the current query criteria. When the On–Count trigger completes execution, Oracle Forms issues the standard query hits message: FRM–40355: Query will retrieve <n> records.

Replaces: The default Oracle Forms selection processing for identifying the rows that match the query criteria.

Definition Level: form, block

Legal Commands: SELECT statements, PL/SQL, unrestricted built–ins

Enter Query Mode: yes

Usage Notes:

• Use an On–Count trigger to replace default Count Query processing in an application running against a non–ORACLE data source.

• To perform the default Oracle Forms processing from this trigger, include a call to the COUNT_QUERY built–in.

• If you are replacing default processing, you can set the value of the Query_Hits block property to indicate the number of records in the non–ORACLE data source that match the query criteria.

• Oracle Forms will display the query hits message (FRM–40355) even if the On–Count trigger fails to set the value of the Query_Hits block property. In such a case, the message reports 0 records identified.

On Failure: no effect

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:

COUNT_QUERY

Example: This example calls a user–named subprogram to count the number of records to be retrieved by the current query criteria, and sets the QUERY_HITS property appropriately.

```sql
DECLARE
    j NUMBER;
BEGIN
    j := Recs_Returned('DEPT', Name_In('DEPT.DNAME'));
    Set_Block_Property('DEPT', QUERY_HITS, j);
END;
```
On–Delete Trigger

Fires during the Post and Commit Transactions process. Specifically, it fires after the Pre–Delete trigger fires and before the Post–Delete trigger fires, replacing the actual database delete of a given row. The trigger fires once for each row that is marked for deletion from the database.

Replaces: The default Oracle Forms processing for handling deleted records during transaction posting.

Definition Level: form or block

Legal Commands: SELECT statements, DML statements (DELETE, INSERT, UPDATE), unrestricted built–ins

Enter Query Mode: no

Usage Notes:
- Use an On–Delete trigger to replace the default Oracle Forms processing for handling deleted records during transaction posting.
- To perform the default Oracle Forms processing from this trigger, that is, to delete a record from your form or from the database, include a call to the DELETE_RECORD built–in.

On Failure: Oracle Forms rolls back to the most recent savepoint

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:

Post and Commit Transactions

Example: This example updates the employee table to set the Termination_Date, rather than actually deleting the employee from the database.

BEGIN
  UPDATE emp
    SET termination_date = SYSDATE
    WHERE empno = :Emp.Empno;
END;
On–Error Trigger

An On–Error trigger fires whenever Oracle Forms would normally cause an error message to display.

**Replaces:** The writing of an error message to the message line.

**Definition Level:** form, block, or item

**Legal Commands:** SELECT statements, unrestricted built-ins

**Enter Query Mode:** yes

**Usage Notes**

- Use an On–Error trigger for the following purposes:
  - to trap and recover from an error
  - to replace a standard error message with a custom message

Use the ERROR_CODE, ERROR_TEXT, ERROR_TYPE, DBMS_ERROR_TEXT, or DBMS_ERROR_CODE built–in function in an On–Error trigger to identify a specific error condition.

- In most cases, On–Error triggers should be attached to the form, rather than to a block or item. Trapping certain errors at the block or item level can be difficult if these errors occur while Oracle Forms is performing internal navigation, such as during a Commit process.

**On Failure:** no effect

**Example:** The following example checks specific error message codes and responds appropriately.

```
DECLARE
  lv_errcod  NUMBER  := ERROR_CODE;
  lv_errtyp  VARCHAR2(3)  := ERROR_TYPE;
  lv_errtxt  VARCHAR2(80) := ERROR_TEXT;
BEGIN
  IF (lv_errcod = 40nnn) THEN
    /*
     ** Perform some tasks here
     */
  ELSIF (lv_errcod = 40mmm) THEN
    /*
     ** More tasks here
     */
  ...  
  ELSIF (lv_errcod = 40zzz) THEN
    /*
```
On–Fetch Trigger

When a query is first opened, fires immediately after the On–Select trigger fires, when the first records are fetched into the block. While the query remains open, fires again each time a set of rows must be fetched into the block.

**Replaces:** The normal Oracle Forms processing for fetching records into a block during a query operation.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, PL/SQL, unrestricted built–ins

**Enter Query Mode:** no

**Usage Notes:**
- When you write an On–Fetch trigger to replace default fetch processing, the trigger must do the following:
  - Retrieve the appropriate number of records from the non–ORACLE data source, as indicated by the setting of the Records_To_Fetch property.
  - Create that number of queried records in the current block.
  - Populate the records with the retrieved data.
- You can create queried records from an On–Fetch trigger by calling the CREATE_QUERIED_RECORD built–in subprogram.
- While the query remains open, the On–Fetch trigger continues to fire as more records are needed in the block. This behavior continues
  - until no queried records are created in a single execution of the trigger. Failing to create any records signal an end–of–fetch to Oracle Forms, indicating that there are no more records to be retrieved.
  - until the query is closed, either by the operator or programmatically through a call to ABORT_QUERY.
On Failure:

- until the trigger raises the built-in exception FORM_TRIGGER_FAILURE.

- To perform the default Oracle Forms processing from this trigger, include a call to the FETCH_RECORDS built-in.

- Do not use an ABORT_QUERY built-in in an On-Fetch trigger. ABORT_QUERY is not valid in an On-Fetch trigger, and produces inconsistent results.

On Failure: no effect

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts":

- Fetch Records

Example 1: The trigger will fire once for each record that is to be fetched.

On-Fetch:

DECLARE
  next_row mypackage.mycursor%rowtype;
BEGIN
  FETCH mypackage.mycursor INTO next_row;
  IF %notfound THEN
    RETURN;
  ENDIF;
  Create_Queried_Record;
  :emp.rowid := next_row.rowid;
  :emp.empno := next_row.empno;
  :emp.ename := next_row.ename;
  :emp.mgr := next_row.mgr;
  ...
END;

Example 2: This example calls a client-side package function to retrieve the proper number of rows from a package cursor.

DECLARE
  j NUMBER := Get_Block_Property(RECORDS_TO_FETCH);
  emprow emp%ROWTYPE;
BEGIN
  FOR ctr IN 1..j LOOP
    /*
     ** Try to get the next row.
     */
    EXIT WHEN NOT MyPackage.Get_Next_Row(emprow);
    Create_Queried_Record;
    :Emp.rowid := emprow.ROWID;
On–Insert Trigger

Fires during the Post and Commit Transactions process. Specifically, it fires after the Pre–Insert trigger fires and before the Post–Insert trigger fires, when Oracle Forms would normally insert a record in the database. It fires once for each row that is marked for insertion into the database.

Replaces: The default Oracle Forms processing for inserting a record during transaction posting.

Definition Level: form or block

Legal Commands: SELECT statements, DML statements (DELETE, INSERT, UPDATE), unrestricted built–ins

Enter Query Mode: no

Usage Notes:
- Use an On–Insert trigger to replace the default Oracle Forms processing for handling inserted records during transaction posting.
- To perform the default Oracle Forms processing from this trigger, include a call to the INSERT_RECORD built–in.

On Failure: Oracle Forms performs the following steps when the On–Insert trigger fails:
- sets the error location
- rolls back to the most recent savepoint

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:
Post and Commit Transactions
On–Lock Trigger

Fires whenever Oracle Forms would normally attempt to lock a row, such as when an operator presses a key to modify data in an item. The trigger fires between the keypress and the display of the modified data.

Replaces: The default Oracle Forms processing for locking rows.

Definition Level: form or block

Legal Commands: SELECT statements, unrestricted built-ins

Enter Query Mode: no

Usage Notes:

- Use an On–Lock trigger to replace the default Oracle Forms processing for locking rows. For example, if you are designing an application for use on a single–user system, you can use the On–Lock trigger to speed processing by bypassing all lock processing. Also, use On–Lock if you are accessing a non–ORACLE data source directly, not by way of Open Gateway.

- When the On–Lock trigger fires as a result of an operator trying to modify data, the trigger fires only the first time the operator tries to modify an item in the record. The trigger does not fire during subsequent modifications to items in the same record. In other words, for every row that is to be locked, the trigger fires once.

- To perform the default Oracle Forms processing from this trigger, include a call to the LOCK_RECORD built–in.

- Use this trigger to lock underlying tables for non–updateable views.

Caution: In special circumstances, you may use the LOCK TABLE DML statement in an On–Lock trigger. However, as this could result in other users being locked out of the table, please exercise caution and refer to the ORACLE RDMS Database Administrator’s Guide before using LOCK TABLE.

On Failure: When the On–Lock trigger fails, the target record is not locked and Oracle Forms attempts to put the input focus in the current item. If the current item cannot be entered for some reason, Oracle Forms attempts to put the input focus in the previous navigable item.

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, ”Processing Flow Charts”:

Lock the Row
On–Logon Trigger

Fires once per logon when Oracle Forms normally initiates the logon sequence.

Replaces: The default Oracle Forms logon processing.
Definition Level: form
Legal Commands: unrestricted built–ins
Enter Query Mode: no

Usage Notes:

- Use an On–Logon trigger to initiate a logon procedure to a non–ORACLE data source.
- Pre–Logon and Post–Logon triggers fire as part of the logon procedure.
- You can supply a NULL command to this trigger to bypass the connection to a data source, if you want to create an application that does not require a data source.
- To perform the default Oracle Forms processing from this trigger, include a call to the LOGON built–in.

On Failure: Oracle Forms attempts to exit the form gracefully, and does not fire the Post–Logon trigger.

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, ”Processing Flow Charts“:
LOGON
On-Logout Trigger

Fires when Oracle Forms normally initiates a logout procedure.

**Replaces:** The default Oracle Forms processing for logout of Oracle Forms and from the RDBMS.

**Definition Level:** form

**Legal Commands:** SELECT statements, unrestricted built-ins

**Enter Query Mode:** no

**Usage Notes:**
- Use an On-Logout trigger to replace the default logout processing either from the RDBMS or from a non-ORACLE data source.
- To perform the default Oracle Forms processing from this trigger, include a call to the LOGOUT built-in.
- If you call certain built-ins from within one of the Logout triggers, the results are undefined. For example, you cannot call the COPY built-in from a Pre-Logout trigger because Pre-Logout fires after the Leave the Form event. Because the form is no longer accessible, a COPY operation is not possible.

**On Failure:** If an exception is raised in an On-Logout trigger and the current Runform session is exited, Oracle Forms will not fire other triggers, such as Post-Logout.

**Fires In:** Refer to the following process flowchart in the *Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:* LOGOUT
On–Message Trigger

Fires whenever Oracle Forms would normally cause a message to display.

Replaces: The writing of a message to the message line.

Definition Level: form, block, or item

Legal Commands: SELECT statements, unrestricted built–ins

Enter Query Mode: yes

Usage Notes:

• Use an On–Message trigger for the following purposes:
  • to trap and respond to an informative message
  • to replace a standard informative message with a custom message
  • to exclude an inappropriate message

• Use the MESSAGE_CODE, MESSAGE_TEXT, MESSAGE_TYPE, DBMS_MESSAGE_CODE, or DBMS_MESSAGE_TEXT built–ins in an On–Message trigger to identify the occurrence of a specific message condition.

• If you use the On–Message trigger to trap a message so that it does not display on the message line, the GET_MESSAGE built–in does not return a value. To display the current message from this trigger, you must trap the message and explicitly write it to the display device.

• In most cases, On–Message triggers should be attached to the form, rather than to a block or item. Trapping certain errors at the block or item level can be difficult if these errors occur while Oracle Forms is performing internal navigation, such as during a Commit process.

On Failure: no effect
Example: The following example responds to an error message by displaying an alert that gives the user a message and gives the user the choice to continue or to stop:

```
DECLARE
    alert_button NUMBER;
    lv_errtype VARCHAR2(3) := MESSAGE_TYPE;
    lv_errcod NUMBER := MESSAGE_CODE;
    lv_errtxt VARCHAR2(80) := MESSAGE_TEXT;
BEGIN
    IF lv_errcod = 40350 THEN
        alert_button := Show_Alert('continue_alert');
        IF alert_button = ALERT_BUTTON1 THEN
            ...;
        ELSE
            ...;
        END IF;
    ELSE
        Message(lv_errtyp||'–'||to_char(lv_errcod)||':  '||lv_errtxt);
        RAISE Form_Trigger_Failure;
    END IF;
END;
```

On–Populate–Details Trigger

Oracle Forms creates this trigger automatically when you define a master–detail relation. It fires when Oracle Forms would normally need to populate the detail block in a master–detail relation.

Definition Level: form, block

Legal Commands: SELECT statements, PL/SQL, unrestricted built–ins, restricted built–ins

Enter Query Mode: no

Usage Notes:

- Use an On–Populate–Details trigger when you have established a master–detail relationship and you want to replace the default populate phase of a query.

Note: The On–Populate–Details trigger does not fire unless an On–Clear–Details trigger is present. If you are using the default master–detail functionality, Oracle Forms creates the necessary triggers automatically. However, if you are writing your own master–detail logic, be aware that the On–Clear–Details trigger must be present, even if it contains only the NULL statement.
On Failure:

- When Immediate coordination is set, this causes the details of the instantiated master to be populated immediately. Immediate coordination is the default.
- When Deferred coordination is set and this trigger fires, Oracle Forms marks the blocks as needing to be coordinated.
- If you intend to manage block coordination yourself, you can call the `SET_BLOCK_PROPERTY(COORDINATION_STATUS)` built-in. Can cause an inconsistent state in the form.

Fires In:

- Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts": Master–Detail Coordination

---

On–Rollback Trigger

Fires when Oracle Forms would normally issue a ROLLBACK statement, to roll back a transaction to the last savepoint that was issued.

Replaces:
The default Oracle Forms processing for issuing a rollback.

Definition Level:
form

Legal Commands:
SELECT statements, PL/SQL

Enter Query Mode:
no

Usage Notes:
- Use an On–Rollback trigger to replace standard Oracle Forms rollback processing.
- To perform default Oracle Forms processing from this trigger, include a call to the ISSUE_ROLLBACK built-in.

Fires In:

- Refer to the following process flowcharts in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts":
  - CLEAR_FORM
  - Post and Commit Transactions
  - ROLLBACK_FORM
On–Savepoint Trigger

Fires when Oracle Forms would normally issue a Savepoint statement. By default, Oracle Forms issues savepoints at form startup, and at the start of each Post and Commit Transaction process.

Replaces: The default Oracle Forms processing for issuing a savepoint.

Definition Level: form

Legal Commands: SELECT statements, PL/SQL, unrestricted built–ins

Enter Query Mode: no

Usage Notes:

• To perform default Oracle Forms processing from this trigger, include a call to the ISSUE_SAVEPOINT built–in.

• In an On–Savepoint trigger, the Savepoint_Name application property returns the name of the next savepoint that Oracle Forms would issue by default, if no On–Savepoint trigger were present. In an On–Rollback trigger, Savepoint_Name returns the name of the savepoint to which Oracle Forms would roll back.

• You can suppress default savepoint processing by setting the Savepoint Mode form module property Off. When Savepoint Mode is Off, Oracle Forms does not issue savepoints and, consequently, the On–Savepoint trigger never fires.

On Failure: no effect

Fires In: Refer to the following process flowcharts in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, ”Processing Flow Charts”:

CALL_FORM
Post and Commit Transactions
SAVEPOINT
On–Select Trigger

Fires when Oracle Forms would normally execute the selection phase of a query, to identify the records in the database that match the current query criteria.

Replaces: The open cursor, parse, and execute phases of a query.

Definition Level: form or block

Legal Commands: SELECT statements, PL/SQL, unrestricted built–ins

Enter Query Mode: no

Usage Notes:
- Use an On–Select trigger to open and execute the database cursor. Specifically, use this trigger when you are retrieving data from a non–ORACLE data source. The On–Select trigger can be used in conjunction with the On–Fetch trigger to replace the processing that normally occurs in the EXECUTE_QUERY built–in subprogram.
- To perform the default Oracle Forms processing from this trigger, include a call to the SELECT_RECORDS built–in.

On Failure: no effect

Fires In: Refer to the following process flowcharts in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, ”Processing Flow Charts”:
EXECUTE_QUERY
Open The Query

Example: In the following example, the On–Select trigger is used to call a user exit, ‘Query,’ and a built–in subprogram, SELECT_RECORDS, to perform a query against a database.

```
IF Get_Application_Property(DATASOURCE) = 'DB2' THEN
  User.Exit ( 'Query' );
  IF Form.Failure OR Form.Fatal THEN
    ABORT_QUERY;
  ELSE
    /*
    ** Perform the default Oracle Forms task of opening the query.
    */
    Select.Records;
  END IF;
END IF;
```
On–Sequence–Number Trigger

Fires when Oracle Forms would normally perform the default processing for generating sequence numbers for default item values.

Replaces: The default series of events that occurs when Oracle Forms interacts with the database to get the next value from a SEQUENCE object defined in the database.

Definition Level: form, block, or item

Legal Commands: SELECT statements, unrestricted built-ins

Enter Query Mode: no

Usage Notes:

• When a SEQUENCE is used as a default item value, Oracle Forms queries the database to get the next value from the SEQUENCE whenever the Create Record event occurs. You can suppress or override this functionality with an On–Sequence–Number trigger.

• To perform the default Oracle Forms processing from this trigger, call the GENERATE_SEQUENCE_NUMBER built-in.

On Failure: no effect

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, ”Processing Flow Charts”:

Generate Sequence Number
On–Update Trigger

Fires during the Post and Commit Transactions process. Specifically, it fires after the Pre–Update trigger fires and before the Post–Update trigger fires, when Oracle Forms would normally update a record in the database. It fires once for each row that is marked for update in the form.

Replaces: The default Oracle Forms processing for updating a record during transaction posting.

Definition Level: form or block

Legal Commands: SELECT statements, DML statements (DELETE, INSERT, UPDATE), unrestricted built–ins

Enter Query Mode: no

Usage Notes:
- Use an On–Update trigger to replace the default Oracle Forms processing for handling updated records during transaction posting.
- To perform the default Oracle Forms processing from this trigger, include a call to the UPDATE_RECORD built–in.

On Failure: Oracle Forms performs the following steps when the On–Update trigger fails:
- sets the error location
- rolls back to the most recently issued savepoint

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:

Post and Commit Transactions
Post–Block Trigger

Fires during the Leave the Block process.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, unrestricted built-ins

**Enter Query Mode:** no

**Restrictions:** A Post–Block trigger does not fire when the Validation Unit form module property is set to Form.

**Usage Notes:**
- Use a Post–Block trigger to validate the block’s current record; that is, the record that had input focus when the Leave the Block event occurred.
- You might also use this trigger to test a condition and prevent the user from leaving a block based on that condition.

**On Failure:** If the trigger fails while trying to make the form the navigation unit, Oracle Forms tries to set the target to a particular block, record or item. Failing that, Oracle Forms attempts to put the cursor at a target location, but, if the target is outside of the current unit or if the operator indicates an end to the process, Oracle Forms aborts the form.

**Fires In:** Refer to the following process flowchart in the *Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts”:

Leave the Block
**Post–Change Trigger**

Fires when any of the following conditions exist:

- The Validate the Item process determines that an item is marked as Changed and is not NULL.
- An operator returns a value into an item by making a selection from a list of values, and the item is not NULL.
- Oracle Forms fetches a non–NULL value into an item. In this case, the When–Validate–Item trigger does not fire. If you want to circumvent this situation and effectively get rid of the Post–Change trigger, you must include a Post–Query trigger in addition to your When–Validate–Item trigger. See “Usage Notes” below.

**Definition Level:** form, block, or item

**Legal Commands:** SELECT statements, unrestricted built–ins

**Enter Query Mode:** no

**Usage Notes:**
- The Post–Change trigger is included only for compatibility with previous versions of Oracle Forms. Its use is not recommended in new applications.
- The Post–Query trigger does not have the restrictions of the Post–Change trigger. You can use Post–Query to make changes to the fetched database values. Given such changes, Oracle Forms marks the corresponding items and records as changed.

**On Failure:** If fired as part of validation initiated by navigation, navigation fails, and the focus remains in the original item.
• If you have V2-style triggers in your form and Oracle Forms is populating a record with fetched values, the following restrictions apply:
  • Oracle Forms ignores any attempt to change the value of a database item in the record.
  • Oracle Forms does not execute any SELECT statement that only affects database items in the record.
  • Oracle Forms does not execute a SELECT statement that does not have an INTO clause.

If Oracle Forms does not execute a SELECT statement in a V2-style trigger step, it treats the trigger step as though the step succeeded, even when the Reverse Return Code trigger step property is set.

During fetch processing, Post–Change triggers defined as PL/SQL triggers do not operate with these restrictions. Regardless of trigger style, during a record fetch, Oracle Forms does not perform validation checks, but marks the record and its items as Valid, after firing the Post–Change trigger for each item.

Restrictions: Note that it is possible to write a Post–Change trigger that changes the value of an item that Oracle Forms is validating. If validation succeeds, Oracle Forms marks the changed item as Valid and does not re–validate it. While this behavior is necessary to avoid validation loops, it does allow you to commit an invalid value to the database.

On Failure: no effect

Fires In: Refer to the following process flowcharts in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts":
  Validate the Item
  Fetch Records
Post–Database–Commit Trigger

Fires once during the Post and Commit Transactions process, after the database commit occurs. Note that the Post–Forms–Commit trigger fires after inserts, updates, and deletes have been posted to the database, but before the transaction has been finalized by issuing the Commit. The Post–Database–Commit Trigger fires after Oracle Forms issues the Commit to finalize the transaction.

Definition Level: form

Legal Commands: SELECT statements, DML statements (DELETE, INSERT, UPDATE), unrestricted built-ins

Enter Query Mode: no

Usage Notes: Use a Post–Database–Commit trigger to perform an action anytime a database commit has occurred.

On Failure: There is no rollback, because at the point at which this trigger might fail, Oracle Forms has already moved past the point at which a successful rollback operation can be initiated as part of a failure response.

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, ”Processing Flow Charts”:

Post and Commit Transactions

Example:

```/*
** FUNCTION recs_posted_and_not_committed
** RETURN BOOLEAN IS
** BEGIN
**  Default_Value(‘TRUE’,’Global.Did_DB_Commit’);
**  RETURN (:System.Form_Status = ‘QUERY’
**     AND :Global.Did_DB_Commit = ‘FALSE’);
** END;
*/
BEGIN
  :Global.Did_DB_Commit := ‘FALSE’;
END;```
Post–Delete Trigger

Fires during the Post and Commit Transactions process, after a row is deleted. It fires once for each row that is deleted from the database during the commit process.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, DML statements (DELETE, INSERT, UPDATE), unrestricted built-ins

**Enter Query Mode:** no

**Usage Notes:** Use a Post–Delete trigger to audit transactions.

**On Failure:** Oracle Forms performs the following steps when the Post–Delete trigger fails:

- sets the error location
- rolls back to the most recent savepoint

**Fires In:** Refer to the following process flowchart in the *Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts":*

Post and Commit Transactions
Post–Form Trigger

Fires during the Leave the Form process, when a form is exited.

**Definition Level:** form

**Legal Commands:** SELECT statements, unrestricted built-ins

**Enter Query Mode:** no

**Usage Notes:** You can use a Post–Form trigger for the following tasks:

- To *clean up* the form before exiting. For example, use a Post–Form trigger to erase any global variables that the form no longer requires.

- To display a message to the operator upon form exit.

This trigger does not fire when the form is exited abnormally, for example, if validation fails in the form.

**On Failure:** processing halts

**Fires In:** Refer to the following process flowchart in the *Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts”*:

Leave the Form
Post–Forms–Commit Trigger

Fires once during the Post and Commit Transactions process. If there are records in the form that have been marked as inserts, updates, or deletes, the Post–Forms–Commit trigger fires after these changes have been written to the database but before Oracle Forms issues the database Commit to finalize the transaction.

If the operator or the application initiates a Commit when there are no records in the form have been marked as inserts, updates, or deletes, Oracle Forms fires the Post–Forms–Commit trigger immediately, without posting changes to the database.

**Definition Level:** form

**Legal Commands:** SELECT statements, DML statements (DELETE, INSERT, UPDATE), unrestricted built-ins

**Enter Query Mode:** no

**Usage Notes:** Use a Post–Forms–Commit trigger to perform an action, such as updating an audit trail, anytime a database commit is about to occur.

**On Failure:** Aborts post and commit processing; Oracle Forms issues a ROLLBACK and decrements the internal Savepoint counter.

**Fires In:** Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts":

Post and Commit Transactions

**Example:** This example can be used in concert with the Post–Database–Commit trigger to detect if records have been posted but not yet committed.

```sql
FUNCTION recs_posted_and_not_committed
RETURN BOOLEAN IS
BEGIN
  Default_Value('TRUE','Global.Did_DB_Commit');
  RETURN (:System.Form_Status = 'QUERY'
           AND :Global.Did_DB_Commit = 'FALSE');
END;
*/
BEGIN
  :Global.Did_DB_Commit := 'FALSE';
END;
```
Post–Insert Trigger

Fires during the Post and Commit Transactions process, just after a record is inserted. It fires once for each record that is inserted into the database during the commit process.

Definition Level: form or block

Legal Commands: SELECT statements, DML statements (DELETE, INSERT, UPDATE), unrestricted built-ins

Enter Query Mode: no

Usage Notes: Use a Post–Insert trigger to audit transactions.

On Failure: Oracle Forms performs the following steps when the Post–Insert trigger fails:
- sets the error location
- rolls back to the most recent savepoint

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”.

Post and Commit Transactions
Post–Logon Trigger

Fires after either of the following events:

- The successful completion of Oracle Forms default logon processing.
- The successful execution of the On–Logon trigger.

**Definition Level:** form

**Legal Commands:** SELECT statements, unrestricted built–ins

**Enter Query Mode:** no

**Usage Notes:** Use a Post–Logon trigger to provide an event point for a task such as setting up a custom environment with special variables—to initialize on an application level rather than a form–by–form basis. You might accomplish this by initializing special global variables from this trigger.

**On Failure:** Varies based on the following conditions:

- If the trigger fails during the first logon process, Oracle Forms exits the form, and returns to the operating system.
- If the trigger fails after a successful logon, Oracle Forms raises the built–in exception FORM_TRIGGER_FAILURE.

**Fires In:** Refer to the following process flowchart in the *Oracle Forms Reference Manual, Vol. 2, Chapter 8, ”Processing Flow Charts”:

LOGON

**Example:** This example calls a user exit to log the current username and time to an encrypted audit trail file on the file system, which for security reasons is outside the database.

```sql
BEGIN
  User_Exit('LogCrypt ' || USER || ' ' || TO_CHAR(SYSDATE, 'YYYYMMDDHH24MISS'));
END;
```
Post-Logout Trigger

Fires after either of the following events:

- Oracle Forms successfully logs out of ORACLE.
- The successful execution of the On-Logout trigger.

Definition Level: form

Legal Commands: SELECT statements, unrestricted built-ins

Enter Query Mode: no

Usage Notes:

- Use a Post-Logout trigger to audit or to perform tasks on an Oracle Forms application that does not require or affect the RDBMS or other data source.
- If you call certain built-ins from within one of the Logout triggers, the results are undefined. For example, you cannot call COPY from a Pre-Logout trigger because Pre-Logout fires after the Leave the Form event. Because the form is no longer accessible, a COPY operation is not possible.

On Failure:

- If this trigger fails while leaving the form, there is no effect.
- If this trigger fails and you have initiated a call to the LOGOUT built-in from within the trigger, FORM_FAILURE is set to TRUE.

Fires In:

Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:

LOGOUT
Post–Query Trigger

When a query is open in the block, the Post–Query trigger fires each time Oracle Forms fetches a record into a block. The trigger fires once for each record placed on the block’s list of records.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, unrestricted built-ins

**Enter Query Mode:** no

**Usage Notes:**
- Use a Post–Query trigger to perform the following tasks:
  - populate control items or items in other blocks
  - calculate statistics about the records retrieved by a query
  - calculate a running total
- When you use a Post–Query trigger to SELECT non–base table values into control items, Oracle Forms marks each record as CHANGED, and so fires the When–Validate–Item trigger by default. You can avoid the execution of the When–Validate–Item trigger by explicitly setting the status of each record to QUERY_STATUS in the Post–Query trigger. To set record status programmatically, use SET_RECORD_PROPERTY.

**On Failure:** Oracle Forms flushes the record from the block and attempts to fetch the next record from the database. If there are no other records in the database, Oracle Forms closes the query and waits for the next operator action.

**Fires In:** Refer to the following process flowchart in the *Oracle Forms Reference Manual, Vol. 2, Chapter 8, ”Processing Flow Charts”*:

Fetch Records
Example: This example retrieves descriptions for code fields, for display in non–database items in the current block.

DECLARE

    CURSOR lookup_payplan IS SELECT Payplan_Desc
                      FROM Payplan
                     WHERE Payplan_Id =
                          :Employee.Payplan_Id;

    CURSOR lookup_area IS SELECT Area_Name
                       FROM Zip_Code
                      WHERE Zip = :Employee.Zip;

BEGIN
    /*
     ** Lookup the Payment Plan Description given the
     ** Payplan_Id in the Employee Record just fetched.
     ** Use Explicit Cursor for highest efficiency.
     */
    OPEN lookup_payplan;
    FETCH lookup_payplan INTO :Employee.Payplan_Desc_Nondb;
    CLOSE lookup_payplan;

    /*
     ** Lookup Area Descript given the Zipcode in
     ** the Employee Record just fetched. Use Explicit
     ** Cursor for highest efficiency.
     */
    OPEN lookup_area;
    FETCH lookup_area INTO :Employee.Area_Desc_Nondb;
    CLOSE lookup_area;
END;
Post–Record Trigger

Fires during the Leave the Record process. Specifically, the Post–Record trigger fires whenever the operator or the application moves the input focus from one record to another. The Leave the Record process can occur as a result of numerous operations, including INSERT_RECORD, DELETE_RECORD, NEXT_RECORD, NEXT_BLOCK, CREATE_RECORD, PREVIOUS_BLOCK, etc.

Definition Level: form or block
Legal Commands: SELECT statements, unrestricted built-ins
Enter Query Mode: no

Usage Notes: Use a Post–Record trigger when you want to perform an action whenever the operator or the application moves the input focus from one record to another. For example, if you want to set a visual attribute for an item as the operator scrolls down through a set of records, you can do so from within this trigger.

Restrictions: A Post–Record trigger fires only when the form is run with a validation unit of the item or record, as specified by the Validation Unit form property.

On Failure: The input focus stays in the current record.

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:
Leave the Record
Post–Select Trigger

The Post–Select trigger fires after the default selection phase of query processing, or after the successful execution of the On–Select trigger. It fires before any records are actually retrieved through fetch processing.

Definition Level: form or block

Legal Commands: SELECT statements, unrestricted built-ins

Enter Query Mode: no

Usage Note: Use the Post–Select trigger to perform an action based on the outcome of the Select phase of query processing, for example, an action based on the number of records that match the query criteria.

On Failure: no effect

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts":

Execute the Query
Open the Query
Post–Text–Item Trigger

Fires during the Leave the Item process for a text item. Specifically, this trigger fires when the input focus moves from a text item to any other item.

Definition Level: form, block, or item

Legal Commands: SELECT statements, unrestricted built-ins

Enter Query Mode: no

Usage Notes: Use a Post–Text–Item trigger to calculate or change item values.

Restrictions:

• A Post–Text–Item trigger fires only when the form is run with the item validation unit, as specified by the Validation Unit form property.

• The Post–Text–Item trigger does not fire when the input focus is in a text item and the operator uses the mouse to click on a button, check box, list item, or radio group item that has the Mouse Navigate property Off. When Mouse Navigate is Off for these items, clicking them with the mouse is a non-navigational event, and the input focus remains in the current item (in this example, a text item).

On Failure: Navigation fails and focus remains in the text item.

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:

Leave the Item
Post–Update Trigger

Fires during the Post and Commit Transactions process, after a row is updated. It fires once for each row that is updated in the database during the commit process.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, DML statements (DELETE, INSERT, UPDATE), unrestricted function codes

**Enter Query Mode:** no

**Usage Notes:** Use a Post–Update trigger to audit transactions.

**On Failure:** Oracle Forms performs the following steps when the Post–Update trigger fails:
- sets the error location
- rolls back to the most recent savepoint

**Fires In:** Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”.

Post and Commit Transactions
Pre–Block Trigger

Fires during the Enter the Block process, during navigation from one block to another.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, unrestricted built–ins

**Enter Query Mode:** no

**Usage Notes:** Use a Pre–Block trigger to:
- allow or disallow access to a block
- set variable values

**Restrictions:** A Pre–Block trigger fires only when the form is run with a validation unit of the item, record, or block, as specified by the Validation Unit form property.

**On Failure:** Navigation fails and focus remains in the source item.

**Fires In:** Refer to the following process flowchart in the *Oracle Forms Reference Manual, Vol. 2, Chapter 8, ”Processing Flow Charts“*:

Enter the Block
**Pre–Commit Trigger**

Fires once during the Post and Commit Transactions process, before Oracle Forms processes any records to change. Specifically, it fires after Oracle Forms determines that there are inserts, updates, or deletes in the form to post or commit. The trigger does not fire when there is an attempt to commit, but validation determines that there are no changed records in the form.

- **Definition Level:** form
- **Legal Commands:** SELECT statements, DML statements (DELETE, INSERT, UPDATE), unrestricted built–ins
- **Enter Query Mode:** no
- **Usage Notes:** Use a Pre–Commit trigger to perform an action, such as setting up special locking requirements, anytime a database commit is going to occur.
- **On Failure:** The Post and Commit process fails: No records are written to the database and focus remains in the current item.
  
  **Note:** If you perform DML in a Pre–Commit trigger and the trigger fails, you must perform a manual rollback, because Oracle Forms does not perform an automatic rollback. To prepare for a possible manual rollback, save the savepoint name in an On–Savepoint trigger, using GET_APPLICATION_PROPERTY(Savepoint_Name). Then you can roll back using ISSUE_ROLLBACK(Savepoint_Name).
- **Fires In:** Refer to the following process flowchart in the *Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”*
  
  Post and Commit Transactions
Pre-Delete Trigger

Fires during the Post and Commit Transactions process, before a row is deleted. It fires once for each record that is marked for delete.

**Note:** Oracle Forms creates a Pre-Delete trigger automatically for any master-detail relation that has the Master Deletes property set to Cascading.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, Data Manipulation Language (DML) statements (i.e., DELETE, INSERT, UPDATE), unrestricted built-ins

**Enter Query Mode:** no

**Usage Notes:**
- Use a Pre-Delete trigger to delete the detail record of a master record.
- Use a Pre-Delete trigger to prevent the deletion of a record if that record is the master record for detail records that still exist.

**On Failure:** Oracle Forms performs the following steps when the Pre-Delete trigger fails:
- sets the error location
- rolls back to the most recent savepoint

**Fires In:** Refer to the following process flowchart in the *Oracle Forms Reference Manual, Vol. 2*, Chapter 8, “Processing Flow Charts”:

Post and Commit Transactions
Pre-Form Trigger

Fires during the Enter the Form event, at form startup.

**Definition Level:** form

**Legal Commands:** SELECT statements, unrestricted built-ins

**Enter Query Mode:** no

**Usage Notes:** Use a Pre-Form trigger to perform the following tasks:
- assign unique primary key from sequence
- restrict access to a form
- initialize global variables

**On Failure:** Oracle Forms leaves the current form and fires no other triggers.

**Fires In:** Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, ”Processing Flow Charts”:

Enter the Form
Pre–Insert Trigger

Fires during the Post and Commit Transactions process, before a row is inserted. It fires once for each record that is marked for insert.

Definition Level: form or block
Legal Commands: SELECT statements, DML statements (DELETE, INSERT, UPDATE), unrestricted built–ins
Enter Query Mode: no
Usage Notes: Use a Pre–Insert trigger to perform the following tasks:
• change item values
• keep track of the date a record is created and store that in the record prior to committing
On Failure: Oracle Forms performs the following steps when the Pre–Insert trigger fails:
• sets the error location
• rolls back to the most recent savepoint
Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:
Post and Commit Transactions
Example: This example assigns a primary key field based on a sequence number, and then writes a row into an auditing table, flagging creation of a new order.

DECLARE
CURSOR next_ord IS SELECT orderid_seq.NEXTVAL FROM dual;
BEGIN
/*
** Fetch the next sequence number from the explicit cursor directly into the item in the Order record. Could use SELECT...INTO, but explicit cursor is more efficient.
*/
OPEN next_ord;
FETCH next_ord INTO :Order.OrderId;
CLOSE next_ord;
/*
** Make sure we populated a new order id ok... */
IF :Order.OrderId IS NULL THEN
    Message('Error Generating Next Order Id');
    RAISE Form_Trigger_Failure;
END IF;

/*/ ** Insert a row into the audit table */
INSERT INTO ord_audit( orderid, operation, username, timestamp )
VALUES ( :Order.OrderId,
    'New Order',
    USER,
    SYSDATE );
END;

Pre–Logon Trigger

Fires just before Oracle Forms initiates a logon procedure to the data source.

Definition Level: form

Legal Commands: SELECT statements, unrestricted built–ins

Enter Query Mode: no

Usage Notes: Use a Pre–Logon trigger to prepare the form for the logon procedure, particularly to a non–ORACLE data source.

On Failure: The results of a failure depend on which of the following conditions applies:

- If Oracle Forms is entering the form for the first time and the trigger fails, the form is exited gracefully, but no other triggers are fired.
- If the trigger fails while Oracle Forms is attempting to execute the LOGON built–in from within the trigger, Oracle Forms raises the FORM_TRIGGER_FAILURE exception.

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts":

LOGON
Pre-Logout Trigger

Fires once before Oracle Forms initiates a logout procedure.

Definition Level: form
Legal Commands: SELECT statements, unrestricted built-ins
Enter Query Mode: no

Usage Notes:
- Use a Pre-Logout trigger to prepare the form for logging out from the data source, particularly a non-ORACLE data source.
- If you call certain built-ins from within one of the Logout triggers, the results are undefined. For example, you cannot call the COPY built-in from a Pre-Logout trigger because Pre-Logout fires after the Leave the Form event. Because the form is no longer accessible, the COPY operation is not possible.

On Failure: The results of a failure depend on which of the following conditions applies:
- If Oracle Forms is exiting the form and the trigger fails, the form is exited gracefully, but no other triggers are fired.
- If the trigger fails while Oracle Forms is attempting to execute the LOGOUT built-in from within the trigger, Oracle Forms raises the FORM_TRIGGER_FAILURE exception.

If an exception is raised in a Pre-Logout trigger, Oracle Forms does not fire other triggers, such as On-Logout and Post-Logout.

Fires In: Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”.

LOGOUT
Pre–Query Trigger

Fires during Execute Query or Count Query processing, just before Oracle Forms constructs and issues the SELECT statement to identify rows that match the query criteria.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, unrestricted built–ins

**Enter Query Mode:** no

**Usage Notes:** Use a Pre–Query trigger to modify the example record that determines which rows will be identified by the query.

**On Failure:** The query is canceled. If the operator or the application had placed the form in Enter Query mode, the form remains in Enter Query mode.

**Fires In:** Refer to the following process flowcharts in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, ”Processing Flow Charts”:

- COUNT_QUERY
- EXECUTE_QUERY
- Open the Query
- Prepare the Query

**Example:** This example validates or modifies query criteria for a database block query.

```sql
BEGIN
    /*
    ** Set the ORDER BY clause for the current block
    ** being queried, based on a radio group
    ** called 'Sort_Column' in a control block named
    ** 'Switches'. The Radio Group has three buttons
    ** with character values giving the names of
    ** three different columns in the table this
    ** block is based on:
    **
    **     SAL
    **     MGR, ENAME
    **     ENAME
    */
    Set_Block_Property('EMP', ORDER_BY, :Switches.Sort_Column);
    /*
    ** Make sure the user has given one of the two
    ** Columns which we have indexed in their search
    */
```

Triggers 2 – 55
** criteria, otherwise fail the query with a helpful
** message
*/
IF :Employee.Ename IS NULL AND :Employee.Mgr IS NULL THEN
  Message('Supply Employee Name and/or Manager Id '||
    'for Query.');
  RAISE Form_Trigger_Failure;
END IF;
/*
** Change the default where clause to either show “Current
** Employees Only” or “Terminated Employees” based on the
** setting of a checkbox named ‘Show_Term’ in a control
** block named ‘Switches’.
*/
IF Checkbox_Checked('Switches.Show_Term') THEN
  Set_Block_Property('EMP',DEFAULT_WHERE,'TERM_DATE IS NOT
    NULL');
ELSE
  Set_Block_Property('EMP',DEFAULT_WHERE,'TERM_DATE IS NULL');
END IF;
END;
Pre–Record Trigger

Fires during the Enter the Record process, during navigation to a different record.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, unrestricted built–ins

**Enter Query Mode:** no

**Usage Notes:** Use a Pre–Record trigger to keep a running total.

**Restrictions:** A Pre–Record trigger fires only when the form is run with a validation unit of the item or record, as specified by the Validation Unit form property.

**On Failure:** Navigation fails and focus remains in the current item.

**Fires In:** Refer to the following process flowchart in the *Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts":*

**Example:** The following trigger prevents the user from entering a new record given some dynamic condition and the status of SYSTEM.RECORD_STATUS evaluating to NEW.

```sql
IF (dynamic-condition)
    AND :System.Record_Status = 'NEW') THEN
    RAISE Form_Trigger_Failure;
END IF;
```
Pre-Select Trigger

Fires during Execute Query and Count Query processing, after Oracle Forms constructs the SELECT statement to be issued, but before the statement is actually issued. Note that the SELECT statement can be examined in a Pre-Select trigger by reading the value of the system variable SYSTEM.LAST_QUERY.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, unrestricted built-ins

**Enter Query Mode:** no

**Usage Notes:** Use a Pre-Select trigger to prepare a query prior to execution against a non-ORACLE data source.

**On Failure:** no effect

**Fires In:** Refer to the following process flowcharts in the *Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”*:

- EXECUTE_QUERY
- Open the Query
- Prepare the Query
Pre–Text–Item Trigger

Fires during the Enter the Item process, during navigation from an item to a text item.

**Definition Level:** form, block, or item

**Legal Commands:** SELECT statements, unrestricted built–ins

**Enter Query Mode:** no

**Usage Notes:** Use a Pre–Text–Item trigger to perform the following types of tasks:
- derive a complex default value, based on other items previously entered into the same record.
- record the current value of the text item for future reference, and store that value in a global variable or form parameter.

**Restrictions:** A Pre–Text–Item trigger fires only when the form is run with a validation unit of the item, as specified by the Validation Unit form property.

**On Failure:** Navigation fails and focus remains in the current item.

**Fires In:** Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts":

Enter the Item
Pre–Update Trigger

Fires during the Post and Commit Transactions process, before a row is updated. It fires once for each record that is marked for update.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, DML statements (DELETE, INSERT, UPDATE), unrestricted built-ins

**Enter Query Mode:** no

**Usage Notes:** Use a Pre–Update trigger to audit transactions.

**On Failure:** Oracle Forms performs the following steps when the Pre–Update trigger fails:
- sets the error location
- rolls back to the most recent savepoint

**Fires In:** Refer to the following process flowchart in the *Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts":*

**Post and Commit Transactions**

**Example:** The following example writes a row into an Audit Table showing old discount and new discount for a given customer, including timestamp and username making the change.

```sql
DECLARE
  old_discount NUMBER;
  new_discount NUMBER := :Customer.Discount_Pct;
  oper_desc    VARCHAR2(80);
  CURSOR old_value IS SELECT discount_pct FROM customer
                        WHERE CustId = :Customer.CustId;
BEGIN
  /*
   ** Fetch the old value of discount percentage from the
   ** database by CustomerId. We need to do this since the
   ** value of :Customer.Discount_Pct will be the *new* value
   ** we’re getting ready to commit and we want to record for
   ** posterity the old and new values. We could use
   ** SELECT...INTO but choose an explicit cursor for
   ** efficiency.
   */
  OPEN old_value;
  FETCH old_value INTO old_discount;
  CLOSE old_value;
END;
```

```sql
BEGIN
  -- Insert the new values into the Audit Table
  INSERT INTO Audit Table
  VALUES (
    :Customer.CustId,
    :Customer.Discount_Pct,
    :Customer.Username,
    SYSDATE
  );
END;
```
/*
** If the old and current values are different, then
** we need to write out an audit record
*/
IF old_discount <> new_discount THEN
    /*
    ** Construct a string that shows the operation of
    ** Changing the old value to the new value. e.g.
    ** 'Changed Discount from 13.5% to 20%'
    */
    oper_desc := 'Changed Discount from ' ||
                 TO_CHAR(old_discount) || '% to ' ||
                 TO_CHAR(new_discount) || '%';

    /*
    ** Insert the audit record with timestamp and user
    */
    INSERT INTO cust_audit( custid, operation, username,
                             timestamp )
        VALUES ( :Customer.CustId,
                 oper_desc,
                 USER,
                 SYSDATE );
END IF;
END;
User–Named Trigger

A user–named trigger is a trigger that you define yourself in a form, and then call explicitly from other triggers or user–named subprograms. Each user–named trigger defined at the same definition level must have a unique name.

To execute a user–named trigger, you must call the EXECUTE_TRIGGER built–in procedure, as shown here:

Execute_Trigger('my_user_named_trigger');

Note: You can write user–named PL/SQL subprograms to perform almost any task for which you might use a user–named trigger.

Definition Level:  form, block, or item

Legal Commands:  Any commands that are legal in the parent trigger from which the user–named trigger was called.

Enter Query Mode:  no

- As with all triggers, the scope of a user–named trigger is the definition level and below. When more than one user–named trigger has the same name, the trigger defined at the lowest level has precedence.
- It is most practical to define user–named triggers at the form level.
- Create a user–named trigger when you want to execute user–named subprograms defined in a form module from menu PL/SQL commands and user–named subprograms. (User–named subprograms defined in a form cannot be called directly from menu PL/SQL, which is defined in a different module.) In the menu PL/SQL, call the EXECUTE_TRIGGER built–in to execute a user–named trigger, which in turn calls the user–named subprogram defined in the current form.

On Failure:  Sets the FORM_FAILURE built–in to TRUE. Because the user–named trigger is always called by the EXECUTE_TRIGGER built–in, you can test the outcome of a user–named trigger the same way you test the outcome of a built–in subprogram; that is, by testing for errors with the built–in functions FORM_FAILURE, FORM_SUCCESS, FORM_FATAL.
**When-Button-Pressed Trigger**

Fires when an operator selects a button, either by way of a key, or by clicking with a mouse.

**Definition Level:** form, block, or item

**Legal Commands:** SELECT statements, unrestricted built-ins, restricted built-ins

**Enter Query Mode:** yes

**Usage Notes:** Use a When-Button-Pressed trigger to perform navigation, to calculate text item values, or for other item, block, or form level functionality.

**On Failure:** no effect

**Example:** This example executes a COMMIT_FORM if there are changes in the form.

```
BEGIN
  IF :System.Form_Status = 'CHANGED' THEN
    Commit_Form;
    /*
    ** If the Form_Status is not back to 'QUERY'
    ** following a commit, then the commit was
    ** not successful.
    */
    IF :System.Form_Status <> 'QUERY' THEN
      Message('Unable to commit order to database...');
      RAISE Form_Trigger_Failure;
    END IF;
  END IF;
END;
```
When–Checkbox–Changed Trigger

Fires whenever an operator changes the state of a check box, either by clicking with the mouse, or through keyboard interaction.

**Definition Level:** form, block, or item

**Legal Commands:** SELECT statements, unrestricted built-ins, restricted built-ins

**Enter Query Mode:** yes

**Usage Notes:** Use a When–Checkbox–Changed trigger to initiate a task dependent upon the state of a check box.

When an operator clicks in a check box, the internal value of that item does not change until navigation is completed successfully. Thus, the When–Checkbox–Changed trigger is the first trigger to register the changed value of a check box item. So for all navigation triggers that fire before the When–Checkbox–Changed trigger, the value of the check box item remains as it was before the operator navigated to it using the mouse.

**On Failure:** no effect
When-Clear-Block Trigger

Fires just before Oracle Forms clears the data from the current block.

Note that the When-Clear-Block trigger does not fire when Oracle Forms clears the current block during the CLEAR_FORM event.

Definition Level: form or block
Legal Commands: SELECT statements, unrestricted built-ins
Enter Query Mode: yes

Usage Notes:
- Use a When-Clear-Block trigger to perform an action every time Oracle Forms flushes the current block. For example, you might want to perform an automatic commit whenever this condition occurs.
- In a When-Clear-Block trigger, the value of SYSTEM.RECORD_STATUS is unreliable because there is no current record. An alternative is to use GET_RECORD_PROPERTY to obtain the record status. Because GET_RECORD_PROPERTY requires you to reference a specific record, its value is always accurate.

On Failure: no effect on the clearing of the block

Fires In: Refer to the following process flowcharts in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:
- CLEAR_BLOCK
- COUNT_QUERY
- ENTER_QUERY
- Open the Query
**When–Create–Record Trigger**

Fires whenever Oracle Forms creates a new record. For example, when the operator presses the [Insert] key, or navigates to the last record in a set while scrolling down, Oracle Forms fires this trigger.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, unrestricted built-ins

**Enter Query Mode:** no

**Usage Notes:** Use a When–Create–Record trigger to perform an action every time Oracle Forms attempts to create a new record. This trigger also is useful for setting complex, calculated, or data–driven default values that must be specified at runtime, rather than design–time.

**On Failure:** Prevents the new record from being created. Returns to the previous location, if possible.

**Fires In:** Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:

**CREATE_RECORD**

**Example:** This example assigns data–driven or calculated default values without marking the record as changed.

```java
DECLARE
    CURSOR ship_dflt IS SELECT val
        FROM cust_pref
        WHERE Custid = :Customer.Custid
        AND pref   = 'SHIP';
BEGIN
    /*
    ** Default Invoice Due Date based on Customer’s
    ** Net Days Allowed value from the Customer block.
    */
    :Invoice.Due_Date := SYSDATE + :Customer.Net_Days_Allowed;
    /*
    ** Default the shipping method based on this customers
    ** preference, stored in a preference table. We could
    ** use SELECT...INTO, but explicit cursor is more
    ** efficient.
    */
    OPEN ship_dflt;
    FETCH ship_dflt INTO :Invoice.Ship_Method;
    CLOSE ship_dflt;
END;
```
When–Custom–Item–Event Trigger

Fires whenever a VBX control sends an event to Oracle Forms.

**Definition Level:** form, block, item

**Legal Commands:** unrestricted built-ins

**Enter Query Mode:** yes

**Usage Notes:** Use a When–Custom–Item–Event trigger to respond to a selection or change of value for a VBX control. The system variable SYSTEM.CUSTOM_ITEM_EVENT stores the case-sensitive name of the event that occurred, and the system variable SYSTEM.CUSTOM_ITEM_EVENT_PARAMETERS stores a parameter name that contains the supplementary arguments for an event that is fired by a VBX control.

**On Failure:** no effect

**Example:** This is an example of a procedure that can be called when Oracle Forms fires the When–Custom–Item–Event Trigger.

```
DECLARE
    TabEvent  varchar2(80);
    TabNumber Number;
BEGIN
    TabEvent := : system.custom_item_event;
    /*
     ** After detecting a Click event, identify the
     ** tab selected, and use the user-defined Goto_Tab_Page
     ** procedure to navigate to the selected page.
     */
    IF (UPPER(TabEvent) = 'CLICK') THEN
        TabNumber := VBX.Get_Property('TABCONTROL','CurrTab');
        Goto_Tab_Page(TabNumber);
    END IF;
END;
```
When–Database–Record Trigger

Fires when Oracle Forms first marks a record as an insert or an update. That is, the trigger fires as soon as Oracle Forms determines through validation that the record should be processed by the next post or commit as an insert or update. This generally occurs only when the operator modifies the first item in a record, and after the operator attempts to navigate out of the item.

Definition Level: form or block

Legal Commands: SELECT statements, unrestricted built-ins

Enter Query Mode: no

Usage Notes: Use a When–Database–Record trigger to perform an action every time a record is first marked as an insert or an update.

On Failure: no effect
When–Form–Navigate

Fires whenever any peer form navigation takes place.

Definition Level: form
Legal Commands: restricted built–ins
Enter Query Mode: no
Usage Notes: Use a When–Form–Navigate trigger to perform actions when any cross form navigation takes place without relying on window activate and window deactivate events.
On Failure: no effect
Example: This is an example of a procedure that can be called when Oracle Forms fires the When–Form–Navigate Trigger.

```plaintext
DECLARE
    win_id WINDOW := FIND_WINDOW('WINDOW12');
BEGIN
    if (GET_WINDOW_PROPERTY(win_id,WINDOW_STATE) = 'MAXIMIZE') THEN
        SET_WINDOW_PROPERTY(win_id,WINDOW_STATE,MINIMIZE);
    else
        SET_WINDOW_PROPERTY(win_id,WINDOW_STATE,MAXIMIZE);
    end if;
END;
```

When–Image–Activated Trigger

Fires when an operator double–clicks on an image item with the mouse.
Note that When–Image–Pressed also fires on a double–click.

Definition Level: form, block, or item
Legal Commands: SELECT statements, unrestricted built–ins
Enter Query Mode: no
On Failure: no effect
When–Image–Pressed Trigger

Fires when an operator uses the mouse to:

- single–click on an image item
- double–click on an image item (note that When–Image–Activated also fires on a double–click)

Definition Level: form, block, or item
Legal Commands: SELECT statements, unrestricted built–ins, restricted built–ins
Enter Query Mode: yes
Usage Notes: Use a When–Image–Pressed trigger to perform an action when an operator clicks or double–clicks on an image item.
On Failure: no effect

When–List–Activated Trigger

Fires when an operator double–clicks on an element in a list item that is displayed as a T–list.

Definition Level: form, block, or item
Legal Commands: SELECT statements, unrestricted built–ins, restricted built–ins
Enter Query Mode: yes
Usage Notes: A When–List–Activated trigger fires only for T–list style list items, not for poplist or combo box style list items. The display style of a list item is determined by the List Style property.
On Failure: no effect
When–List–Changed Trigger

Fires when an operator selects a different element in a list item or de-selects the currently selected element. In addition, if a When–List–Changed trigger is attached to a combo box style list item, it fires each time the operator enters or modifies entered text.

Definition Level: form, block, or item
Legal Commands: SELECT statements, unrestricted built-ins, restricted built-ins
Enter Query Mode: yes
Usage Notes: Use a When–List–Changed trigger to initiate an action when the value of the list changes.
On Failure: no effect

When–Mouse–Click Trigger

Fires after the operator clicks the mouse if one of the following events occurs:

- if attached to the form, when the mouse is clicked within any canvas-view or item in the form
- if attached to a block, when the mouse is clicked within any item in the block
- if attached to an item, when the mouse is clicked within the item

Three events must occur before a When–Mouse–Click trigger will fire:

- Mouse down
- Mouse up
- Mouse click

Any trigger that is associated with these events will fire before the When–Mouse–Click trigger fires.

Definition Level: form, block, or item
Legal Commands: SELECT statements, restricted built-ins, unrestricted built-ins
Enter Query Mode: yes
Usage Notes: Use the When–Mouse–Click trigger to perform an action every time the operator clicks the mouse within an item and/or canvas–view.

On Failure: Prevents mouse navigation via a mouse click.

---

When–Mouse–DoubleClick Trigger

Fires after the operator double–clicks the mouse if one of the following events occurs:

- if attached to the form, when the mouse is double–clicked within any canvas–view or item in the form
- if attached to a block, when the mouse is double–clicked within any item in the block
- if attached to an item, when the mouse is double–clicked within the item

Six events must occur before a When–Mouse–DoubleClick trigger will fire:

- Mouse down
- Mouse up
- Mouse click
- Mouse down
- Mouse up
- Mouse double–click

Any trigger that is associated with these events will fire before the When–Mouse–DoubleClick trigger fires.

Definition Level: form, block, or item

Legal Commands: SELECT statements, restricted built–ins, unrestricted built–ins

Enter Query Mode: yes

Usage Notes: Use a When–Mouse–DoubleClick trigger to perform an action every time the operator double–clicks the mouse within an item and/or canvas–view.

On Failure: no effect
Example: Assume that your application requires Behavior A when the operator clicks the mouse and Behavior B when the operator double-clicks the mouse. For example, if the operator clicks the mouse, a product information window must appear. If the operator double-clicks the mouse, an online help window must appear.

Three triggers are used in this example, a When-Mouse-Click trigger, a When-Timer-Expired trigger, and a When-Mouse-DoubleClick trigger.

```sql
/*
** Trigger: When-Mouse-Click
** Example: When the operator clicks the mouse, create a timer that will expire within .5 seconds.
*/
DECLARE
    timer_id        TIMER;
    timer_duration  NUMBER(5) := 500;
BEGIN
    timer_id := Create_Timer('doubleclick_timer', timer_duration, NO_REPEAT);
END;

/*
** Trigger: When-Timer-Expired
** Example: When the timer expires display the online help window if the operator has double-clicked the mouse within .5 seconds, otherwise display the product information window.
*/
BEGIN
    IF :Global.double_click_flag = 'TRUE' THEN
        Show_Window('online_help');
        :Global.double_click := 'FALSE';
    ELSE
        Show_Window('product_information');
    END IF;
END;

/*
** Trigger: When-Mouse-DoubleClick
** Example: If the operator double-clicks the mouse, set a flag that indicates that a double-click event occurred.
*/
BEGIN
    :Global.double_click_flag := 'TRUE';
END;
```
When–Mouse–Down Trigger

Fires after the operator presses down the mouse button if one of the following events occurs:

- if attached to the form, when the mouse is pressed down within any canvas–view or item in the form
- if attached to a block, when the mouse is pressed down within any item in the block
- if attached to an item, when the mouse is pressed within the item

Definition Level: form, block, or item

Legal Commands: SELECT statements, restricted built–ins, unrestricted built–ins

Enter Query Mode: yes

Usage Notes: Use a When–Mouse–Down trigger to perform an action every time the operator presses down the mouse button within an item and/or canvas–view.

Note: The mouse down event is always followed by a mouse up event.

Restrictions: Depending on the window manager, navigational code within a When–Mouse–Down trigger may fail. For example on MS Windows, if the operator clicks the mouse button within a field (Item_One), a When–Mouse–Down trigger that calls GO_ITEM('item_two') will fail because Windows will return focus to Item_One, not Item_Two since the When–Mouse–Up event occurred within Item_Two.

On Failure: no effect
When–Mouse–Enter Trigger

Fires when the mouse enters an item or canvas–view if one of the following events occurs:

- if attached to the form, when the mouse enters any canvas–view or item in the form
- if attached to a block, when the mouse enters any item in the block
- if attached to an item, when the mouse enters the item

**Definition Level:** form, block, or item

**Legal Commands:** SELECT statements, restricted built–ins, unrestricted built–ins

**Enter Query Mode:** yes

**Usage Notes:**
- Use a When–Mouse–Enter trigger to perform an action every time the mouse enters an item or canvas–view.
- Be careful when calling a modal window from a When–Mouse–Enter trigger. Doing so may cause the modal window to appear unnecessarily.

For example, assume that your When–Mouse–Enter trigger causes Alert_One to appear whenever the mouse enters Canvas_One. Assume also that your application contains two canvases, Canvas_One and Canvas_Two. Canvas_One and Canvas_Two do not overlap each other, but appear side by side on the screen. Further, assume that Alert_One displays within Canvas_Two’s border.

Finally, assume that the mouse has entered Canvas_One causing the When–Mouse–Enter trigger to fire which, in turn, causes Alert_One to appear.

When the operator dismisses the alert, Alert_One will appear again unnecessarily if the operator subsequently enters Canvas_One with the mouse. In addition, when the operator moves the mouse out of Canvas_Two, any When–Mouse–Leave triggers associated with this event will fire. This may not be the desired behavior.

**On Failure:** no effect
When–Mouse–Leave Trigger

Fires after the mouse leaves an item or canvas–view if one of the following events occurs:

• if attached to the form, when the mouse leaves any canvas–view or item in the form
• if attached to a block, when the mouse leaves any item in the block
• if attached to an item, when the mouse leaves the item

**Definition Level:** form, block, or item

**Legal Commands:** SELECT statements, restricted built–ins, unrestricted built–ins

**Enter Query Mode:** yes

**Usage Notes:** Use a When–Mouse–Leave trigger to perform an action every time the mouse leaves an item and/or canvas–view.

**On Failure:** no effect
When–Mouse–Move Trigger

Fires each time the mouse moves if one of the following events occurs:

- if attached to the form, when the mouse moves within any canvas–view or item in the form
- if attached to a block, when the mouse moves within any item in the block
- if attached to an item, when the mouse moves within the item

Definition Level: form, block, or item

Legal Commands: SELECT statements, restricted built–ins, unrestricted built–ins

Enter Query Mode: yes

Usage Notes: Use the When–Mouse–Move trigger to perform an action every time the operator moves the mouse.

The When–Mouse–Move trigger may have performance implications because of the number of times this trigger can potentially fire.

On Failure: no effect
When–Mouse–Up Trigger

Fires each time the operator presses down and releases the mouse button if one of the following events occurs:

- if attached to the form, when the mouse up event is received within any canvas–view or item in a form
- if attached to a block, when the mouse up event is received within any item in a block
- if attached to an item, when the mouse up event is received within an item

Two events must occur before a When–Mouse–Up trigger will fire:

- Mouse down
- Mouse up

**Definition Level:** form, block, or item

**Legal Commands:** SELECT statements, restricted built–ins, unrestricted built–ins

**Enter Query Mode:** yes

**Usage Notes:** Use the When–Mouse–Up trigger to perform an action every time the operator presses and releases the mouse.

The mouse up event is always associated with the item that received the mouse down event. For example, assume that there is a When–Mouse–Up trigger attached to Item_One. If the operator presses down the mouse on Item_One, but then releases the mouse on Item_Two, the mouse up trigger will fire for Item_One, rather than for Item_Two.

**On Failure:** no effect
When–New–Block–Instance Trigger

Fires when the input focus moves to an item in a block that is different than the block that previously had input focus. Specifically, it fires after navigation to an item, when Oracle Forms is ready to accept input in a block that is different than the block that previously had input focus.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, unrestricted built–ins, restricted built–ins

**Enter Query Mode:** no

**Usage Notes:** Use a When–New–Block–Instance trigger to perform an action every time Oracle Forms instantiates a new block.

**On Failure:** no effect

**Fires In:** Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, ”Processing Flow Charts”:

Return for Input
When–New–Form–Instance Trigger

At form start-up, Oracle Forms navigates to the first navigable item in the first navigable block. A When–New–Form–Instance trigger fires after the successful completion of any navigational triggers that fire during the initial navigation sequence.

This trigger does not fire when control returns to a calling form from a called form.

In a multiple–form application, this trigger does not fire when focus changes from one form to another.

**Definition Level:** form

**Legal Commands:** SELECT statements, restricted built-ins, unrestricted built-ins

**Enter Query Mode:** no

**Restrictions:** When you call FORMS_OLE.GET_INTERFACE_POINTER from the When–New–Form–Instance trigger, an exception is raised because Oracle Forms has not initialized an OLE item at this point.

**On Failure:** no effect

**Fires In:** Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts":

Run the Form

**Example:** This example sets up all necessary global variables, writes dynamic boilerplate, starts a timer to refresh the on–screen clock, and queries the first block.

```
BEGIN
   Setup_My_Global_Variables;
   Populate_Dynamic_Boilerplate;
   Start_OnScreen_Clock_Timer;
   Go_Block('Primary_Ord_Info');

   /*
    ** Query the block without showing
    ** the working message.
    */
   :System.Suppress_Working := 'TRUE';
   Execute_Query;
   :System.Suppress_Working := 'FALSE';
END;
```
**When–New–Item–Instance Trigger**

Fires when the input focus moves to an item. Specifically, it fires after navigation to an item, when Oracle Forms is ready to accept input in an item that is different than the item that previously had input focus.

- **Definition Level:** form, block, or item
- **Legal Commands:** SELECT statements, restricted built-ins, unrestricted built-ins.
- **Enter Query Mode:** yes
- **Usage Notes:** Use a When–New–Item–Instance trigger to perform an action whenever an item gets input focus. The When–New–Item–Instance trigger is especially useful for calling restricted (navigational) built-ins.
- **Restrictions:** The conditions for firing this trigger are *not* met under the following circumstances:
  - Oracle Forms navigates through an item, without stopping to accept input
  - the input focus moves to a field in an alert window, or to any part of an Oracle Forms menu
- **On Failure:** no effect
- **Fires In:** Refer to the following process flowchart in the *Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”*:
  - Return for Input
When–New–Record–Instance Trigger

Fires when the input focus moves to an item in a record that is different than the record that previously had input focus. Specifically, it fires after navigation to an item in a record, when Oracle Forms is ready to accept input in a record that is different than the record that previously had input focus.

Fires whenever Oracle Forms instantiates a new record.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, unrestricted built-ins, restricted built-ins

**Enter Query Mode:** yes

**Usage Notes:** Use a When–New–Record–Instance trigger to perform an action every time Oracle Forms instantiates a new record. For example, when an operator presses [Down] to scroll through a set of records, Oracle Forms fires this trigger each time the input focus moves to the next record, in other words, each time Oracle Forms instantiates a new record in the block.

**On Failure:** no effect

**Fires In:** Refer to the following process flowchart in the *Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:*

- Return for Input
When–Radio–Changed Trigger

Fires when an operator selects a different radio button in a radio group, or de–selects the currently selected radio button, either by clicking with the mouse, or by way of keyboard selection commands.

**Definition Level:** form, block, or item

**Legal Commands:** SELECT statements, unrestricted built–ins, restricted built–ins

**Enter Query Mode:** yes

**Usage Notes:** Use a When–Radio–Changed trigger to perform an action depending on the state of a radio group. (De–selecting a radio button in a radio group sets the radio group value to NULL; operators use this technique in Enter Query mode to exclude a radio group from a query.)

When an operator clicks in a radio group, the internal value of that item does not change until navigation is completed successfully. Thus, the When–Radio–Changed trigger is the first trigger to register the changed value of a radio group. So for all navigation triggers that fire before the When–Radio–Changed trigger, the value of the radio group remains as it was before the operator navigated to it using the mouse.

**On Failure:** no effect
**When–Remove–Record Trigger**

Fires whenever the operator or the application clears or deletes a record.

**Definition Level:** form, block, or item

**Legal Commands:** SELECT statements, unrestricted built-ins

**Enter Query Mode:** no

**Usage Notes:** Use a When–Remove–Record trigger to perform an action every time Oracle Forms clears or deletes a record.

**On Failure:** Oracle Forms navigates to the block level with or without validation depending on the current operation, and puts the cursor at the target block.

**Fires In:** Refer to the following process flowcharts in the *Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:*

- CLEAR_RECORD
- DELETE_RECORD
- PREVIOUS_RECORD
When–Timer–Expired Trigger

Fires when a timer expires.

**Note:** Timers are created programmatically by calling the CREATE_TIMER built-in procedure.

**Definition Level:** form

**Legal Commands:** SELECT statements, unrestricted built-ins, restricted built-ins

**Enter Query Mode:** yes

**Usage Notes:**

- The When–Timer–Expired trigger can not fire during trigger, navigation, or transaction processing.
- Use a When–Timer–Expired trigger to initiate an event, update item values, or perform any task that should occur after a specified interval.
- You can call GET_APPLICATION_PROPERTY(TIMER_NAME) in a When–Timer–Expired trigger to determine the name of the most recently expired timer.

**Restrictions:** A When–Timer–Expired trigger will not fire when the user is currently navigating a menu.

**On Failure:** no effect

**Fires In:** Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts":

Process Timer Expired

**Example:** The following example displays an alert each time a repeating timer expires. The following example is from a telemarketing application, in which sales calls are timed, and alerts are displayed to prompt the salesperson through each stage of the call. The alert is displayed each time a repeating timer expires.

```sql
DECLARE
    timer_id  TIMER;
    alert_id  ALERT;
    call_status NUMBER;
    msg_1       VARCHAR2(80) := 'Wrap up the first phase of your presentation';
    msg_2       VARCHAR2(80) := 'Move into your close.';
    msg_3       VARCHAR2(80) := 'Ask for the order or repeat the close.';
```
two_minutes NUMBER(6) := (120 * 1000);
one_and_half NUMBER(5) := (90 * 1000);
BEGIN
:GLOBAL.timer_count := 1
BEGIN
  timer_id := FIND_TIMER('tele_timer');
  alert_id := FIND_ALERT('tele_alert');
  IF :GLOBAL.timer_count = 1 THEN
    Change_Alert_Message(alert_id, msg_1);
    call_status := Show_Alert(alert_id);
    IF call_status = ALERT_BUTTON1 THEN
      Delete_Timer(timer_id);
      Next_Record;
    ELSIF call_status = ALERT_BUTTON2 THEN
      :GLOBAL.timer_count := 0;
    ELSE
      Set_Timer(timer_id, two_minutes, NO_CHANGE);
    END IF;
  ELSIF :GLOBAL.timer_count = 2 THEN
    Change_Alert_Message(alert_id, msg_2);
    call_status := Show_Alert(alert_id);
    IF call_status = ALERT_BUTTON1 THEN
      Delete_Timer(timer_id);
      Next_Record;
    ELSIF call_status = ALERT_BUTTON2 THEN
      :GLOBAL.timer_count := 0;
    ELSE
      Set_Timer(timer_id, one_and_half, NO_CHANGE);
    END IF;
  ELSE
    Change_Alert_Message(alert_id, msg_3);
    call_status := Show_Alert(alert_id);
    IF call_status = ALERT_BUTTON1 THEN
      Delete_Timer(timer_id);
      Next_Record;
    ELSIF call_status = ALERT_BUTTON2 THEN
      :GLOBAL.timer_count := 0;
    ELSE
      Set_Timer(timer_id, NO_CHANGE, NO_REPEAT);
    END IF;
  END IF;
END IF;
:GLOBAL.timer_count := 2;
END;
END;
When–Validate–Item Trigger

Fires during the Validate the Item process. Specifically, it fires as the last part of item validation for items with the New or Changed validation status.

**Definition Level:** form, block, or item

**Legal Commands:** SELECT statements, unrestricted built-ins

**Enter Query Mode:** no

**Usage Notes:**
- Use a When–Validate–Item trigger to supplement Oracle Forms default item validation processing.
- It is possible to write a When–Validate–Item trigger that changes the value of an item that Oracle Forms is validating. If validation succeeds, Oracle Forms marks the changed item as Valid and does not re-validate it. While this behavior is necessary to avoid validation loops, it does make it possible for your application to commit an invalid value to the database.
- The Defer_Required_Enforcement property postpones enforcement of the Required property from item validation to record validation. When an item has the Required property set to True, by default Oracle Forms will not allow navigation out of the item until a valid value is entered. Setting the Defer_Required_Enforcement property to True allows the operator to move freely among the items in the record.

**Note:** When Defer_Required_Enforcement is On, all other item validation takes place as usual: only checking the Required property is postponed until the operator leaves the record.

**On Failure:** no effect

**Fires In:** Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, "Processing Flow Charts":

Validate the Item

**Example:** The following example finds the commission plan in the COMPLAN table, based on the current value of the commcode item in the EMPLOYEE block in the form, to verify that the code is valid. If the code in the COMPLAN table is located, the description of the COMPLAN is obtained and deposited in the non-database Description item. Otherwise, an error is raised.
** Method 1: Using a SELECT...INTO statement, the trigger looks more readable but can be less efficient than Method 2 because for ANSI Standard compliance, the SELECT...INTO statement must return an error if more than one row is retrieved that matches the criteria. This implies PL/SQL may attempt to fetch data twice from the table in question to insure that there aren’t two matching rows. */

```
BEGIN
SELECT description
  INTO :Employee.Commplan_Desc
FROM commplan
WHERE commcode = :Employee.Commcode;
EXCEPTION
WHEN No.Data_Found THEN
  Message('Invalid Commission Plan, Use <List> for help');
  RAISE Form_Trigger_Failure;
WHEN Too_Many_Rows THEN
  Message('Error. Duplicate entries in COMMPLAN table!');
  RAISE Form_Trigger_Failure;
END;
```

/*
** Method 2: Using an Explicit Cursor looks a bit more daunting but is actually quite simple. The SELECT statement is declared as a named cursor in the DECLARE section and then is OPENed, FETCHed, and CLOSED in the code explicitly (hence the name). Here we guarantee that only a single FETCH will be performed against the database. */

```
DECLARE
  noneFound BOOLEAN;
  CURSOR cp IS SELECT description
    FROM commplan
    WHERE commcode = :Employee.Commcode;
BEGIN
  OPEN cp;
  FETCH cp INTO :Employee.Commplan_Desc;
  noneFound := cp%NOTFOUND;
  CLOSE cp;
  IF noneFound THEN
    Message('Invalid Commission Plan, Use <List> for help');
    RAISE Form_Trigger_Failure;
  END IF;
END;
```
When–Validate–Record Trigger

Fires during the Validate the Record process. Specifically, it fires as the last part of record validation for records with the New or Changed validation status.

**Definition Level:** form or block

**Legal Commands:** SELECT statements, unrestricted built-ins

**Enter Query Mode:** no

**Usage Notes:** Use a When–Validate–Record trigger to supplement Oracle Forms default record validation processing.

**Caution:** Note that it is possible to write a When–Validate–Record trigger that changes the value of an item in the record that Oracle Forms is validating. If validation succeeds, Oracle Forms marks the record and all of the fields as Valid and does not re-validate. While this behavior is necessary to avoid validation loops, it does make it possible for your application to commit an invalid value to the database.

**On Failure:** no effect

**Fires In:** Refer to the following process flowchart in the Oracle Forms Reference Manual, Vol. 2, Chapter 8, “Processing Flow Charts”:

Validate the Record

**Example:** The following example verifies that Start_Date is less than End_Date. Since these two text items have values that are related, it’s more convenient to check the combination of them once at the record level, rather than check each item separately. This code presumes both date items are mandatory and that neither will be NULL.

```plaintext
/* Method 1: Hardcode the item names into the trigger.
** Structured this way, the chance this code will be reusable in other forms we write is pretty low because of dependency on block and item names.
*/
BEGIN
  IF :Experiment.Start_Date > :Experiment.End_Date THEN
    Message('Your date range ends before it starts!');
    RAISE Form_Trigger_Failure;
  END IF;
END;
```
/* Method 2: Call a generic procedure to check the date range. This way our date check can be used in any validation trigger where we want to check that a starting date in a range comes before the ending date. Another bonus is that with the error message in one standard place, i.e. the procedure, the user will always get a consistent failure message, regardless of the form they’re currently in. */
BEGIN
  Check_Date_Range(:Experiment.Start_Date,:Experiment.End_Date);
END;

/*
** The procedure looks like this
*/
PROCEDURE Check_Date_Range( d1 DATE, d2 DATE ) IS
BEGIN
  IF d1 > d2 THEN
    Message('Your date range ends before it starts!');
    RAISE Form_Trigger_Failure;
  END IF;
END;
When–Window–Activated Trigger

Fires when a window is made the active window. This occurs at form startup and whenever a different window is given focus. Note that on some window managers, a window can be activated by, say, clicking on its title bar. This operation is independent of navigation to an item in the window. Thus, navigating to an item in a different window always activates that window, but window activation can also occur independently of navigation.

Definition Level: form
Legal Commands: SELECT statements, unrestricted built-ins, restricted built-ins
Enter Query Mode: yes
Usage Notes: Use this trigger to perform the following types of tasks:

- Capture initial settings of window properties, by way of the GET_WINDOW_PROPERTY built-in.
- Enforce navigation to a particular item whenever a window is activated.
- Keep track of the most recently fired window trigger by assigning the value from SYSTEM.EVENT_WINDOW to a variable or global variable.

On Failure: no effect
When–Window–Closed Trigger

Fires when an operator closes a window using a window–manager specific Close command.

Definition Level: form

Legal Commands: SELECT statements, unrestricted built–ins, restricted built–ins

Enter Query Mode: yes

Usage Notes: Use this trigger to programmatically close a window when the operator issues the window–manager Close command. You can close a window with the HIDE_WINDOW, SET_WINDOW_PROPERTY, and EXIT_FORM built–in subprograms.

You can hide the window that contains the current item.

On Failure: no effect

Example: The following example of a call to SET_WINDOW_PROPERTY from this trigger closes a window whenever the operator closes it by way of the window manager operation:

```
Set_Window_Property('window_name', VISIBLE, PROPERTY_OFF);
```

When–Window–Deactivated Trigger

Fires when an operator deactivates a window by setting the input focus to another window.

Definition Level: form

Legal Commands: SELECT statements, unrestricted built–ins, restricted built–ins

Enter Query Mode: yes

Usage Notes: Use this trigger to audit the state of a window whenever the operator deactivates the window by setting the input focus in another window.

On Failure: no effect
When–Window–Resized Trigger

Fires when a window is resized, either by the operator or programmatically through a call to RESIZE_WINDOW or SET_WINDOWPROPERTY. (Even if the window is not currently displayed, resizing the window programmatically fires the When–Window–Resized trigger.) This trigger also fires at form startup, when the root window is first drawn. It does not fire when a window is iconified.

Definition Level: form
Legal Commands: SELECT statements, unrestricted built–ins, restricted built–ins
Enter Query Mode: yes

Usage Notes: Use this trigger to perform any one of the following types of tasks:
- Capture the changed window properties, such as width, height, x coordinate, or y coordinate.
- Audit the actions of an operator.
- Set input focus in an item on the target window.
- Maintain certain visual standards by resetting window size if the window was improperly resized.

On Failure: no effect
Built-in Subprograms

This chapter includes the following information about Oracle Forms built-in subprograms included in the STANDARD Extensions, FORMS_OLE, and VBX packages:

- syntax
- named parameters
- code examples
- object IDs
- restricted versus unrestricted built-ins
- constants
- built-in functions
- individual descriptions of built-in subprograms

For information on the following packages, refer to the Online Help system or the Oracle Procedure Builder User’s Guide and Reference: OLE2, TOOL_ENV, ORA_NLS, TOOL_RES, ORA_FFI, ORA_DE, STPROC, TEXT_IO.
Overview

Oracle Forms provides built-in subprograms that you can call from triggers and user-named subprograms that you write yourself. Built-ins provide programmatic control over standard application functions, including navigation, interface control, and transaction processing.

Syntax

Refer to the Preface in this book for typographic conventions. Named parameters are shown in an italic monospaced font. You can replace any named parameter with the actual parameter, which can be a constant, a literal, a bind variable, or a number.

```
SET_TIMER(timer_name, milliseconds, iterate);
```

In this example, the timer name you supply must be enclosed in single quotes, because the timer_name is a CHAR value. The milliseconds parameter is passed as a number and, as such, does not require single quotes. The iterate parameter is passed as a constant, and, as such, must be entered exactly as shown in the parameter description, without single quotes. Capitalization is unimportant.

In those cases where a number of optional elements are available, various alternate syntax statements are presented. These alternatives are presented to preclude having to decipher various complicated syntactical conventions.

Note that you sometimes use variables instead of including a specific object name. In those cases, do not enclose the variable within single quotes. The following example illustrates a When–Timer–Expired trigger that calls the SET_TIMER built-in and references a variable that contains a valid timer name:

```
DECLARE
  the_timer CHAR := GET_APPLICATION_PROPERTY(TIMER_NAME);
BEGIN
  SET_TIMER(the_timer, 60000, REPEAT);
END;
```
Named Parameters

You can directly use any of the named parameters shown in the syntax diagrams in this chapter. The named parameter should be followed with the equal/greater than signs (=>), which point to the actual parameter that follows the named parameter. For example, if you intend to change the milliseconds in the SET_TIMER Built-in you can directly use that parameter with the following syntax:

```
SET_TIMER(timer_name => 'my_timer', milliseconds => 12000, iterate => NO_REPEAT);
```

Also, you can continue to call the built-in with the following syntax:

```
SET_TIMER('my_timer', 12000, NO_REPEAT);
```

Code Examples

Examples have been included for the built-in subprograms. Some examples are simple illustrations of the syntax. Others are more complex illustrations of how to use the Built-in either alone or in conjunction with other built-ins. A few points to keep in mind regarding the syntax of examples:

- Examples are shown exactly as they can be entered.
- Casing and use of italics can be ignored and is included for readability.
- Built-in names and other PL/SQL reserved words, such as IF, THEN, ELSE, BEGIN, and END are shown in capital letters for easier readability.
- Named parameters, when illustrated, are shown in an italic typeface. If you choose to use named parameters, enter these parameter names exactly as shown, without quotes and follow them with the equal/greater than symbols (=>).
- CHAR type arguments must be enclosed in single quotes.
- Any other data type argument should not be enclosed in quotes.
- Special characters other than single quotes (’), commas (,), parentheses, underscores (_), and semicolons(;) should be ignored.
Object IDs

Some built-in subprograms accept *object IDs* as actual parameters. An object ID is an internal, opaque handle that is assigned to each object when created in the Designer. Object IDs are internally managed and cannot be externally viewed by the user. The only method you can use to retrieve the ID is to define a local or global variable and assign the return value of the object to the variable.

You make the assignment by way of the FIND_ built-in functions. Once you have used FIND_ within a PL/SQL block, you can use the variable as an object ID while still in that block. The valid PL/SQL type for each object is included in the syntax descriptions for each parameter. The description for the FIND_BLOCK built-in provides an example of how to obtain an object ID.

Form Coordinate Units

Many built-in subprograms allow you to specify size and position coordinates, using properties such as:

- HEIGHT
- WIDTH
- DISPLAY_POSITION
- DISPLAY_X_POS
- DISPLAY_Y_POS
- VIEW_SIZE
- X_POS_ON_CANVAS
- Y_POS_ON_CANVAS
When you specify coordinates or width and height, you express these measurements in units of the current form coordinate system, set on the Form Module property sheet. The form coordinate system defines the units for specifying size and position coordinates of objects in the Designer. Use the form module property, Coordinate Information, to set the form’s coordinate units:

- character cells or
- real units:
  - inches
  - centimeters
  - pixels
  - points

When you design in the character cell coordinate system, all object dimensions and position coordinates are expressed in character cells, so Oracle Forms accepts only whole numbers for size and position properties.

When you design using real units (inches, centimeters, or points), all object dimensions and position coordinates are expressed in the units you specify, so Oracle Forms will accept decimals as well as whole numbers for size and position properties. The precision of real units is three digits, so you can specify coordinates to thousandths. If you use pixels or character cells, coordinates are truncated to whole numbers.

**Uppercase Return Values**

The GET_X_PROPERTY built-ins, such as GET_FORM_PROPERTY, return CHAR arguments as uppercase values. This will affect the way you compare results in IF statements.

**Restricted Built-In Subprograms**

Restricted built-ins affect navigation in your form, either external screen navigation, or internal navigation. You can call these built-ins only from triggers while no internal navigation is occurring.

Restricted built-ins cannot be called from the Pre and Post triggers, which fire when Oracle Forms is navigating from object to another.

Restricted built-ins can be called from the When triggers that are specific to interface items, such as When–Button–Pressed or When–Checkbox–Changed. Restricted built-ins can also be called from
any of the When–New–“object”–Instance triggers and from key triggers.

Unrestricted built-ins do not affect logical or physical navigation and can be called from any trigger.

The descriptions of built-in subprograms in this chapter include a heading, Built-In Type, that indicates if the built-in is restricted or unrestricted.

### Constants

Many of the built-in subprograms take numeric values as arguments. Often, constants have been defined for these numeric arguments. A constant is a named numeric value. When passing a constant to a built-in do not enclose the constant value in quotation marks.

Constants can only appear on the right side of an operator in an expression.

In some cases, a built-in can take a number of possible constants as arguments. Possible constants are listed in the descriptions for each parameter.

In the following example, BLOCK_SCOPE is a constant that can be supplied for the parameter constant VALIDATION_UNIT. Other constants listed in the description are FORM, RECORD, and ITEM.

```plaintext
SET_FORM_PROPERTY('my_form', VALIDATION_UNIT, BLOCK_SCOPE);
```
Built-in Subprograms Tables

The following cross reference list organizes the built-in subprograms by object type. In some cases, these objects are externalized interface objects, such as windows or canvases. In other cases, they are data structures such as records, record groups, or parameter lists.

<table>
<thead>
<tr>
<th>Alert Built-ins</th>
<th>Built-in</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND_ALERT</td>
<td></td>
<td>3 – 89</td>
</tr>
<tr>
<td>ID_NULL</td>
<td></td>
<td>3 – 180</td>
</tr>
<tr>
<td>SET_ALERT_BUTTON_PROPERTY</td>
<td></td>
<td>3 – 243</td>
</tr>
<tr>
<td>SET_ALERT_PROPERTY</td>
<td></td>
<td>3 – 244</td>
</tr>
<tr>
<td>SHOW_ALERT</td>
<td></td>
<td>3 – 286</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application Built-ins</th>
<th>Built-in</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>DO_KEY</td>
<td></td>
<td>3 – 72</td>
</tr>
<tr>
<td>GET_APPLICATION_PROPERTY</td>
<td></td>
<td>3 – 119</td>
</tr>
<tr>
<td>HOST</td>
<td></td>
<td>3 – 178</td>
</tr>
<tr>
<td>PAUSE</td>
<td></td>
<td>3 – 215</td>
</tr>
<tr>
<td>SET_APPLICATION_PROPERTY</td>
<td></td>
<td>3 – 245</td>
</tr>
<tr>
<td>USER_EXIT</td>
<td></td>
<td>3 – 297</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block Built-ins</th>
<th>Built-in</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOCK_MENU</td>
<td></td>
<td>3 – 26</td>
</tr>
<tr>
<td>CLEAR_BLOCK</td>
<td></td>
<td>3 – 34</td>
</tr>
<tr>
<td>FIND_BLOCK</td>
<td></td>
<td>3 – 90</td>
</tr>
<tr>
<td>GET_BLOCK_PROPERTY</td>
<td></td>
<td>3 – 123</td>
</tr>
<tr>
<td>GO_BLOCK</td>
<td></td>
<td>3 – 171</td>
</tr>
<tr>
<td>ID_NULL</td>
<td></td>
<td>3 – 180</td>
</tr>
<tr>
<td>NEXT_BLOCK</td>
<td></td>
<td>3 – 208</td>
</tr>
<tr>
<td>PREVIOUS_BLOCK</td>
<td></td>
<td>3 – 220</td>
</tr>
<tr>
<td>SET_BLOCK_PROPERTY</td>
<td></td>
<td>3 – 246</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Canvas and View Built-ins</th>
<th>Built-in</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND_CANVAS</td>
<td></td>
<td>3 – 91</td>
</tr>
<tr>
<td>FIND_VIEW</td>
<td></td>
<td>3 – 99</td>
</tr>
<tr>
<td>GET_CANVAS_PROPERTY</td>
<td></td>
<td>3 – 128</td>
</tr>
<tr>
<td>GET_VIEW_PROPERTY</td>
<td></td>
<td>3 – 167</td>
</tr>
<tr>
<td>Built-in</td>
<td>Page Number</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>HIDE_VIEW</td>
<td>3 – 176</td>
<td></td>
</tr>
<tr>
<td>ID_NULL</td>
<td>3 – 180</td>
<td></td>
</tr>
<tr>
<td>PRINT</td>
<td>3 – 224</td>
<td></td>
</tr>
<tr>
<td>SCROLL_VIEW</td>
<td>3 – 239</td>
<td></td>
</tr>
<tr>
<td>SET_CANVAS_PROPERTY</td>
<td>3 – 250</td>
<td></td>
</tr>
<tr>
<td>SET_VIEW_PROPERTY</td>
<td>3 – 282</td>
<td></td>
</tr>
<tr>
<td>SHOW_VIEW</td>
<td>3 – 292</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Built-in</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>BELL</td>
<td>3 – 26</td>
</tr>
<tr>
<td>BREAK</td>
<td>3 – 27</td>
</tr>
<tr>
<td>CALL_FORM</td>
<td>3 – 28</td>
</tr>
<tr>
<td>CALL_INPUT</td>
<td>3 – 31</td>
</tr>
<tr>
<td>CLEAR_FORM</td>
<td>3 – 36</td>
</tr>
<tr>
<td>COMMIT_FORM</td>
<td>3 – 41</td>
</tr>
<tr>
<td>DEBUG_MODE</td>
<td>3 – 59</td>
</tr>
<tr>
<td>ENTER</td>
<td>3 – 77</td>
</tr>
<tr>
<td>ERASE</td>
<td>3 – 79</td>
</tr>
<tr>
<td>EXECUTE_TRIGGER</td>
<td>3 – 84</td>
</tr>
<tr>
<td>EXIT_FORM</td>
<td>3 – 85</td>
</tr>
<tr>
<td>FIND_FORM</td>
<td>3 – 93</td>
</tr>
<tr>
<td>FORM_FAILURE</td>
<td>3 – 101</td>
</tr>
<tr>
<td>FORM_FATAL</td>
<td>3 – 103</td>
</tr>
<tr>
<td>FORM_SUCCESS</td>
<td>3 – 104</td>
</tr>
<tr>
<td>Built-in</td>
<td>Page Number</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>GET_FORMPROPERTY</td>
<td>3 – 129</td>
</tr>
<tr>
<td>HELP</td>
<td>3 – 175</td>
</tr>
<tr>
<td>ID_NULL</td>
<td>3 – 180</td>
</tr>
<tr>
<td>NEW_FORM</td>
<td>3 – 205</td>
</tr>
<tr>
<td>OPEN_FORM</td>
<td>3 – 213</td>
</tr>
<tr>
<td>POST</td>
<td>3 – 220</td>
</tr>
<tr>
<td>REDISPLAY</td>
<td>3 – 228</td>
</tr>
<tr>
<td>REPLACE_MENU</td>
<td>3 – 230</td>
</tr>
<tr>
<td>SET_FORMPROPERTY</td>
<td>3 – 290</td>
</tr>
<tr>
<td>SHOW_KEYS</td>
<td>3 – 290</td>
</tr>
<tr>
<td>SHOW_MENU</td>
<td>3 – 291</td>
</tr>
<tr>
<td>SYNCHRONIZE</td>
<td>3 – 294</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item Built-ins</th>
<th>Built-in</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECKBOX_CHECKED</td>
<td>3 – 31</td>
<td></td>
</tr>
<tr>
<td>CLEAR_EOL</td>
<td>3 – 35</td>
<td></td>
</tr>
<tr>
<td>CLEAR_ITEM</td>
<td>3 – 37</td>
<td></td>
</tr>
<tr>
<td>CONVERT_OTHER_VALUE</td>
<td>3 – 43</td>
<td></td>
</tr>
<tr>
<td>COPY</td>
<td>3 – 45</td>
<td></td>
</tr>
<tr>
<td>COPY_REGION</td>
<td>3 – 45</td>
<td></td>
</tr>
<tr>
<td>CUT_REGION</td>
<td>3 – 56</td>
<td></td>
</tr>
<tr>
<td>DEFAULT_VALUE</td>
<td>3 – 60</td>
<td></td>
</tr>
<tr>
<td>DISPLAY_ITEM</td>
<td>3 – 70</td>
<td></td>
</tr>
<tr>
<td>DUPLICATE_ITEM</td>
<td>3 – 73</td>
<td></td>
</tr>
<tr>
<td>EDIT_TEXTITEM</td>
<td>3 – 75</td>
<td></td>
</tr>
<tr>
<td>FIND_ITEM</td>
<td>3 – 94</td>
<td></td>
</tr>
<tr>
<td>GET_ITEM_PROPERTY</td>
<td>3 – 143</td>
<td></td>
</tr>
<tr>
<td>GET_RADIO_BUTTON_PROPERTY</td>
<td>3 – 160</td>
<td></td>
</tr>
<tr>
<td>GO_ITEM</td>
<td>3 – 173</td>
<td></td>
</tr>
<tr>
<td>ID_NULL</td>
<td>3 – 180</td>
<td></td>
</tr>
<tr>
<td>IMAGE_ZOOM</td>
<td>3 – 181</td>
<td></td>
</tr>
<tr>
<td>NAME_IN</td>
<td>3 – 202</td>
<td></td>
</tr>
<tr>
<td>NEXT_ITEM</td>
<td>3 – 210</td>
<td></td>
</tr>
<tr>
<td>NEXT_KEY</td>
<td>3 – 211</td>
<td></td>
</tr>
</tbody>
</table>
### List Item Built-ins

<table>
<thead>
<tr>
<th>Built-in</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASTE_REGION</td>
<td>3 – 215</td>
</tr>
<tr>
<td>PREVIOUS_ITEM</td>
<td>3 – 227</td>
</tr>
<tr>
<td>READ_IMAGE_FILE</td>
<td>3 – 227</td>
</tr>
<tr>
<td>SELECT_ALL</td>
<td>3 – 241</td>
</tr>
<tr>
<td>SET_ITEM_PROPERTY</td>
<td>3 – 260</td>
</tr>
<tr>
<td>SET_RADIO_BUTTON_PROPERTY</td>
<td>3 – 274</td>
</tr>
<tr>
<td>WRITE_IMAGE_FILE</td>
<td>3 – 306</td>
</tr>
</tbody>
</table>

### Built-in

<table>
<thead>
<tr>
<th>Built-in</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_LIST_ELEMENT</td>
<td>3 – 22</td>
</tr>
<tr>
<td>CLEAR_LIST</td>
<td>3 – 38</td>
</tr>
<tr>
<td>DELETE_LIST_ELEMENT</td>
<td>3 – 64</td>
</tr>
<tr>
<td>GET_LIST_ELEMENT_COUNT</td>
<td>3 – 151</td>
</tr>
<tr>
<td>GET_LIST_ELEMENT_LABEL</td>
<td>3 – 153</td>
</tr>
<tr>
<td>GET_LIST_ELEMENT_VALUE</td>
<td>3 – 154</td>
</tr>
<tr>
<td>POPULATE_LIST</td>
<td>3 – 218</td>
</tr>
<tr>
<td>RETRIEVE_LIST</td>
<td>3 – 234</td>
</tr>
</tbody>
</table>

### Menu Built-ins

<table>
<thead>
<tr>
<th>Built-in</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION_PARAMETER</td>
<td>3 – 24</td>
</tr>
<tr>
<td>BACKGROUND_MENU</td>
<td>3 – 25</td>
</tr>
<tr>
<td>FIND_MENU_ITEM</td>
<td>3 – 96</td>
</tr>
<tr>
<td>GET_MENU_ITEM_PROPERTY</td>
<td>3 – 156</td>
</tr>
<tr>
<td>HIDE_MENU</td>
<td>3 – 175</td>
</tr>
<tr>
<td>ITEM_ENABLED</td>
<td>3 – 186</td>
</tr>
<tr>
<td>MAIN_MENU</td>
<td>3 – 193</td>
</tr>
<tr>
<td>MENU_CLEAR_FIELD</td>
<td>3 – 193</td>
</tr>
<tr>
<td>MENU_NEXT_FIELD</td>
<td>3 – 194</td>
</tr>
<tr>
<td>MENU_PARAMETER</td>
<td>3 – 194</td>
</tr>
<tr>
<td>MENU_PREVIOUS_FIELD</td>
<td>3 – 195</td>
</tr>
<tr>
<td>MENU_REDISPLAY</td>
<td>3 – 195</td>
</tr>
<tr>
<td>MENU_SHOW_KEYS</td>
<td>3 – 195</td>
</tr>
<tr>
<td>NEXT_MENU_ITEM</td>
<td>3 – 211</td>
</tr>
<tr>
<td>PREVIOUS_MENU</td>
<td>3 – 222</td>
</tr>
<tr>
<td>PREVIOUS_MENU_ITEM</td>
<td>3 – 223</td>
</tr>
<tr>
<td>Messages Built-ins</td>
<td>Built-in</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>CLEAR_MESSAGE</td>
<td>3 – 39</td>
</tr>
<tr>
<td>DBMS_ERROR_CODE</td>
<td>3 – 57</td>
</tr>
<tr>
<td>DBMS_ERROR_TEXT</td>
<td>3 – 58</td>
</tr>
<tr>
<td>DISPLAY_ERROR</td>
<td>3 – 70</td>
</tr>
<tr>
<td>ERROR_CODE</td>
<td>3 – 79</td>
</tr>
<tr>
<td>ERROR_TEXT</td>
<td>3 – 80</td>
</tr>
<tr>
<td>ERROR_TYPE</td>
<td>3 – 81</td>
</tr>
<tr>
<td>GET_MESSAGE</td>
<td>3 – 158</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>3 – 196</td>
</tr>
<tr>
<td>MESSAGE_CODE</td>
<td>3 – 197</td>
</tr>
<tr>
<td>MESSAGE_TEXT</td>
<td>3 – 198</td>
</tr>
<tr>
<td>MESSAGE_TYPE</td>
<td>3 – 199</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Miscellaneous Built-ins</th>
<th>Built-in</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_TIMER</td>
<td>3 – 55</td>
<td></td>
</tr>
<tr>
<td>DELETE_TIMER</td>
<td>3 – 67</td>
<td></td>
</tr>
<tr>
<td>FIND_EDITOR</td>
<td>3 – 92</td>
<td></td>
</tr>
<tr>
<td>FIND_LOV</td>
<td>3 – 95</td>
<td></td>
</tr>
<tr>
<td>FIND_TIMER</td>
<td>3 – 98</td>
<td></td>
</tr>
<tr>
<td>GET_LOV_PROPERTY</td>
<td>3 – 155</td>
<td></td>
</tr>
<tr>
<td>ID_NULL</td>
<td>3 – 180</td>
<td></td>
</tr>
<tr>
<td>LIST_VALUES</td>
<td>3 – 187</td>
<td></td>
</tr>
<tr>
<td>SET_LOV_COLUMN_PROPERTY</td>
<td>3 – 270s</td>
<td></td>
</tr>
<tr>
<td>SET_LOV_PROPERTY</td>
<td>3 – 271</td>
<td></td>
</tr>
<tr>
<td>SET_TIMER</td>
<td>3 – 280</td>
<td></td>
</tr>
<tr>
<td>SHOW_EDITOR</td>
<td>3 – 288</td>
<td></td>
</tr>
<tr>
<td>Built-in</td>
<td>Page Number</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>SHOW_LOV</td>
<td>3 – 290</td>
<td></td>
</tr>
<tr>
<td>VALIDATE</td>
<td>3 – 299</td>
<td></td>
</tr>
<tr>
<td><strong>Multiple-form Application Built-ins</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOSE_FORM</td>
<td>3 – 41</td>
<td></td>
</tr>
<tr>
<td>GO_FORM</td>
<td>3 – 172</td>
<td></td>
</tr>
<tr>
<td>NEXT_FORM</td>
<td>3 – 209</td>
<td></td>
</tr>
<tr>
<td>OPEN_FORM</td>
<td>3 – 213</td>
<td></td>
</tr>
<tr>
<td>PREVIOUS_FORM</td>
<td>3 – 221</td>
<td></td>
</tr>
<tr>
<td><strong>OLE Built-ins (Microsoft Windows only)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FORMS_OLE.ACTIVATE_SERVER</td>
<td>3 – 109</td>
<td></td>
</tr>
<tr>
<td>FORMS_OLE.CLOSE_SERVER</td>
<td>3 – 110</td>
<td></td>
</tr>
<tr>
<td>FORMS_OLE.EXEC_VERB</td>
<td>3 – 111</td>
<td></td>
</tr>
<tr>
<td>FORMS_OLE.FIND_OLE_VERB</td>
<td>3 – 112</td>
<td></td>
</tr>
<tr>
<td>FORMS_OLE.GET_INTERFACE_POINTER</td>
<td>3 – 113</td>
<td></td>
</tr>
<tr>
<td>FORMS_OLE.GET_VERB_COUNT</td>
<td>3 – 114</td>
<td></td>
</tr>
<tr>
<td>FORMS_OLE.GET_VERB_NAME</td>
<td>3 – 115</td>
<td></td>
</tr>
<tr>
<td>FORMS_OLE.INITIALIZE_CONTAINER</td>
<td>3 – 116</td>
<td></td>
</tr>
<tr>
<td>FORMS_OLE.SERVER_ACTIVE</td>
<td>3 – 117</td>
<td></td>
</tr>
<tr>
<td><strong>Parameter List Built-ins</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD_PARAMETER</td>
<td>3 – 23</td>
<td></td>
</tr>
<tr>
<td>CREATE_PARAMETER_LIST</td>
<td>3 – 51</td>
<td></td>
</tr>
<tr>
<td>DELETE_PARAMETER</td>
<td>3 – 65</td>
<td></td>
</tr>
<tr>
<td>DESTROY_PARAMETER_LIST</td>
<td>3 – 69</td>
<td></td>
</tr>
<tr>
<td>GET_PARAMETER_ATTR</td>
<td>3 – 159</td>
<td></td>
</tr>
<tr>
<td>GET_PARAMETER_LIST</td>
<td>3 – 160</td>
<td></td>
</tr>
<tr>
<td>ID_NULL</td>
<td>3 – 180</td>
<td></td>
</tr>
<tr>
<td>RUN_PRODUCT</td>
<td>3 – 235</td>
<td></td>
</tr>
<tr>
<td>SET_PARAMETER_ATTR</td>
<td>3 – 274</td>
<td></td>
</tr>
<tr>
<td><strong>Query Built-ins</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABORT_QUERY</td>
<td>3 – 17</td>
<td></td>
</tr>
<tr>
<td>COUNT_QUERY</td>
<td>3 – 46</td>
<td></td>
</tr>
<tr>
<td><strong>Built-in</strong></td>
<td><strong>Page Number</strong></td>
<td></td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------</td>
<td></td>
</tr>
<tr>
<td>ENTER_QUERY</td>
<td>3 – 77</td>
<td></td>
</tr>
<tr>
<td>EXECUTE_QUERY</td>
<td>3 – 82</td>
<td></td>
</tr>
</tbody>
</table>

**Record Built-ins**

<table>
<thead>
<tr>
<th><strong>Built-in</strong></th>
<th><strong>Page Number</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK_RECORD_UNIQUENESS</td>
<td>3 – 33</td>
</tr>
<tr>
<td>CLEAR_RECORD</td>
<td>3 – 40</td>
</tr>
<tr>
<td>CREATE_QUERIED_RECORD</td>
<td>3 – 52</td>
</tr>
<tr>
<td>CREATE_RECORD</td>
<td>3 – 54</td>
</tr>
<tr>
<td>DELETE_RECORD</td>
<td>3 – 66</td>
</tr>
<tr>
<td>DOWN</td>
<td>3 – 71</td>
</tr>
<tr>
<td>DUPLICATE_RECORD</td>
<td>3 – 74</td>
</tr>
<tr>
<td>FIRST_RECORD</td>
<td>3 – 101</td>
</tr>
<tr>
<td>GENERATE_SEQUENCE_NUMBER</td>
<td>3 – 118</td>
</tr>
<tr>
<td>GET_RECORD_PROPERTY</td>
<td>3 – 163</td>
</tr>
<tr>
<td>GO_RECORD</td>
<td>3 – 174</td>
</tr>
<tr>
<td>INSERT_RECORD</td>
<td>3 – 183</td>
</tr>
<tr>
<td>LAST_RECORD</td>
<td>3 – 186</td>
</tr>
<tr>
<td>LOCK_RECORD</td>
<td>3 – 188</td>
</tr>
<tr>
<td>NEXT_RECORD</td>
<td>3 – 212</td>
</tr>
<tr>
<td>NEXT_SET</td>
<td>3 – 213</td>
</tr>
<tr>
<td>PREVIOUS_RECORD</td>
<td>3 – 224</td>
</tr>
<tr>
<td>SCROLL_DOWN</td>
<td>3 – 238</td>
</tr>
<tr>
<td>SCROLL_UP</td>
<td>3 – 238</td>
</tr>
<tr>
<td>SELECT_RECORDS</td>
<td>3 – 242</td>
</tr>
<tr>
<td>SET_RECORD_PROPERTY</td>
<td>3 – 276</td>
</tr>
<tr>
<td>UP</td>
<td>3 – 296</td>
</tr>
<tr>
<td>UPDATE_RECORD</td>
<td>3 – 296</td>
</tr>
</tbody>
</table>

**Record Group Built-ins**

<table>
<thead>
<tr>
<th><strong>Built-in</strong></th>
<th><strong>Page Number</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_GROUP_COLUMN</td>
<td>3 – 18</td>
</tr>
<tr>
<td>ADD_GROUP_ROW</td>
<td>3 – 20</td>
</tr>
<tr>
<td>CREATE_GROUP</td>
<td>3 – 47</td>
</tr>
<tr>
<td>CREATE_GROUP_FROM_QUERY</td>
<td>3 – 47</td>
</tr>
<tr>
<td>DELETE_GROUP</td>
<td>3 – 61</td>
</tr>
<tr>
<td>DELETE_GROUP_ROW</td>
<td>3 – 62</td>
</tr>
<tr>
<td>Built-in</td>
<td>Page Number</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>FIND_COLUMN</td>
<td>3 – 91</td>
</tr>
<tr>
<td>FIND_GROUP</td>
<td>3 – 94</td>
</tr>
<tr>
<td>GET_GROUP_CHAR_CELL</td>
<td>3 – 133</td>
</tr>
<tr>
<td>GET_GROUP_DATE_CELL</td>
<td>3 – 135</td>
</tr>
<tr>
<td>GET_GROUP_NUMBER_CELL</td>
<td>3 – 136</td>
</tr>
<tr>
<td>GET_GROUP_ROW_COUNT</td>
<td>3 – 139</td>
</tr>
<tr>
<td>GET_GROUP_SELECTION</td>
<td>3 – 140</td>
</tr>
<tr>
<td>GET_GROUP_SELECTION_COUNT</td>
<td>3 – 142</td>
</tr>
<tr>
<td>ID_NULL</td>
<td>3 – 180</td>
</tr>
<tr>
<td>POPULATE_GROUP</td>
<td>3 – 216</td>
</tr>
<tr>
<td>POPULATE_GROUP_WITH_QUERY</td>
<td>3 – 217</td>
</tr>
<tr>
<td>RESET_GROUP_SELECTION</td>
<td>3 – 232</td>
</tr>
<tr>
<td>SET_GROUP_CHAR_CELL</td>
<td>3 – 255</td>
</tr>
<tr>
<td>SET_GROUP_DATE_CELL</td>
<td>3 – 256</td>
</tr>
<tr>
<td>SET_GROUP_NUMBER_CELL</td>
<td>3 – 258</td>
</tr>
<tr>
<td>SET_GROUP_SELECTION</td>
<td>3 – 259</td>
</tr>
<tr>
<td>UNSET_GROUP_SELECTION</td>
<td>3 – 295</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relation Built-ins</th>
<th>Built-in</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND_RELATION</td>
<td></td>
<td>3 – 97</td>
</tr>
<tr>
<td>GET_RELATION_PROPERTY</td>
<td></td>
<td>3 – 165</td>
</tr>
<tr>
<td>ID_NULL</td>
<td></td>
<td>3 – 180</td>
</tr>
<tr>
<td>SET_RELATION_PROPERTY</td>
<td></td>
<td>3 – 278</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transactional Built-ins</th>
<th>Built-in</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK_RECORD_UNIQUENESS</td>
<td></td>
<td>3 – 33</td>
</tr>
<tr>
<td>DELETE_RECORD</td>
<td></td>
<td>3 – 66</td>
</tr>
<tr>
<td>ENFORCE_COLUMN_SECURITY</td>
<td></td>
<td>3 – 76</td>
</tr>
<tr>
<td>FETCH_RECORDS</td>
<td></td>
<td>3 – 87</td>
</tr>
<tr>
<td>FORMS_DDL</td>
<td></td>
<td>3 – 105</td>
</tr>
<tr>
<td>GENERATE_SEQUENCE_NUMBER</td>
<td></td>
<td>3 – 118</td>
</tr>
<tr>
<td>INSERT_RECORD</td>
<td></td>
<td>3 – 183</td>
</tr>
<tr>
<td>ISSUE_ROLLBACK</td>
<td></td>
<td>3 – 184</td>
</tr>
<tr>
<td>ISSUE_SAVEPOINT</td>
<td></td>
<td>3 – 185</td>
</tr>
<tr>
<td>LOGON</td>
<td></td>
<td>3 – 189</td>
</tr>
<tr>
<td>Built–in</td>
<td>Page Number</td>
<td></td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>LOGON_SCREEN</td>
<td>3 – 190</td>
<td></td>
</tr>
<tr>
<td>LOGOUT</td>
<td>3 – 192</td>
<td></td>
</tr>
<tr>
<td>SELECT_RECORDS</td>
<td>3 – 242</td>
<td></td>
</tr>
<tr>
<td>UPDATE_RECORD</td>
<td>3 – 296</td>
<td></td>
</tr>
<tr>
<td><strong>VBX Control Built–ins</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Microsoft Windows only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VBX.FIRE_EVENT</td>
<td>3 – 300</td>
<td></td>
</tr>
<tr>
<td>VBX.GETPROPERTY</td>
<td>3 – 301</td>
<td></td>
</tr>
<tr>
<td>VBX.GET_VALUE_PROPERTY</td>
<td>3 – 302</td>
<td></td>
</tr>
<tr>
<td>VBX.INVOKE_METHOD</td>
<td>3 – 303</td>
<td></td>
</tr>
<tr>
<td>VBX.SETPROPERTY</td>
<td>3 – 304</td>
<td></td>
</tr>
<tr>
<td>VBX.SET_VALUEPROPERTY</td>
<td>3 – 305</td>
<td></td>
</tr>
<tr>
<td><strong>Window Built–ins</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIND_WINDOW</td>
<td>3 – 100</td>
<td></td>
</tr>
<tr>
<td>HIDE_WINDOW</td>
<td>3 – 176</td>
<td></td>
</tr>
<tr>
<td>ID_NULL</td>
<td>3 – 180</td>
<td></td>
</tr>
<tr>
<td>MOVE_WINDOW</td>
<td>3 – 200</td>
<td></td>
</tr>
<tr>
<td>REPLACE_CONTENT_VIEW</td>
<td>3 – 229</td>
<td></td>
</tr>
<tr>
<td>RESIZE_WINDOW</td>
<td>3 – 233</td>
<td></td>
</tr>
<tr>
<td>SHOW_WINDOW</td>
<td>3 – 293</td>
<td></td>
</tr>
</tbody>
</table>
Individual Built–in Descriptions

The remainder of this chapter presents individual built–in descriptions. Each built–in is presented in the following format or a subset of the format, as applicable:

**Syntax:** Describes the syntax of the built–in. If there are multiple formats for a built–in then all formats are shown. For example, if the target object of a built–in can be called by name or by object ID, then both forms of syntax are displayed.

**Built–in Type:** Indicates whether the built–in is restricted or unrestricted.

**Returns:** Indicates the return value or data type of a built–in function.

**Enter Query Mode:** Indicates the capability to call the built–in during enter query mode.

**Description:** Indicates the general purpose and use of the built–in.

**Parameters:** Describes the parameters that are included in the syntax diagrams. Underlined parameters usually are the default.

**Restrictions:** Indicates any restrictions.

**Example:** Provides an actual example that can be used in conjunction with the syntax to develop a realistic call to the built–in.
ABORT_QUERY

Syntax: ABORT_QUERY;

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Closes a query that is open in the current block.

A query is open between the time the SELECT statement is issued and the time when all the rows have been fetched from the database. In particular, a query is not open when the form is in Enter Query mode, because the SELECT statement has not yet been issued.

Parameters: none

Usage Notes: ABORT_QUERY is not the equivalent of the Query, Cancel runtime default menu command. It does not prevent the initial fetch from the database, but rather interrupts fetch processing, thus preventing subsequent fetches.

Restrictions: Do not use ABORT_QUERY in the following triggers:

- On–Fetch. The On–Fetch trigger is provided for applications using transactional triggers to replace default Oracle Forms functions when running against non–Oracle data sources. To signal that your On–Fetch trigger is done fetching rows, exit the On–Fetch trigger without issuing the CREATE_QUERIED_RECORD built–in.

- Pre–Query. The Pre–Query trigger fires before the query is open, so there is no open query to close and ABORT_QUERY is ignored. To programmatically cancel Enter Query mode, call the built–in EXIT_FORM, using a When–New–Record–Instance trigger to check a flag as follows:

  IF (:global.cancel_query = 'Y'
    and :system.mode = 'ENTER–QUERY')
  THEN
    Exit Form;
    :global.cancel_query = 'N';
  END IF;

- Then set the flag to ‘TRUE’ either from a Pre–Query trigger or an On–Error trigger that traps for the FRM–40301 error.
ADD_GROUP_COLUMN

Syntax:

ADD_GROUP_COLUMN(recordgroup_id, groupcolumn_name, column_type);
ADD_GROUP_COLUMN(recordgroup_name, groupcolumn_name, column_type);
ADD_GROUP_COLUMN(recordgroup_id, groupcolumn_name, column_type, column_width);
ADD_GROUP_COLUMN(recordgroup_name, groupcolumn_name, column_type, column_width);

Built-in Type: unrestricted function
Enter Query Mode: yes
Returns: GroupColumn
Description: Adds a column of the specified type to the given record group.

Parameters:

recordgroup_id  The unique ID that Oracle Forms assigns when it creates the group. The data type of the ID is RecordGroup.

recordgroup_name  The name you gave to the record group when creating it. The data type of the name is CHAR.

groupcolumn_name  Specifies the name of the column. The data type of the column name is CHAR.

column_type  Specifies the data type of the column. The allowable values are the following constants:

CHAR_COLUMN  Specify if the column can only accept CHAR data.

DATE_COLUMN  Specify if the column can only accept DATE data.

LONG_COLUMN  Specify if the column can only accept LONG data.

NUMBER_COLUMN  Specify if the column can only accept NUMBER data.

column_width  If you specify CHAR_COLUMN as the column_type, you must indicate the maximum length of the data. COLUMN_WIDTH cannot exceed 2000, and must be passed as a whole number.
Restrictions:

- Maximum 255 columns per group.
- You must add columns to a group before adding rows.
- You cannot add a column to a group that already has rows; instead, delete the rows with DELETE_GROUP_ROW, then add the column.
- You can only add columns to a group after it is created with a call to CREATE_GROUP.
- If the column corresponds to a database column, the width of CHAR_COLUMN–typed columns cannot be less than the width of the corresponding database column.
- If the column corresponds to a database column, the width of CHAR_COLUMN–typed columns can be greater than the width of the corresponding database column.
- Only columns of type CHAR_COLUMN require the width parameter.
- Performance is affected if a record group has a large number of columns.
- There can only be one LONG column per record group.

Error Conditions:

An error is returned under the following conditions:

- You enter the name of a non-existent record group.
- You specify the name for a group or a column that does not adhere to standard Oracle naming conventions.
- You enter a column type other than CHAR, NUMBER, DATE, or LONG.

Example:

/*
** Built-in: ADD_GROUP_COLUMN
** Example: Add one Number and one Char column to a new
** record group.
*/
PROCEDURE Create_My_Group IS
  rg_name VARCHAR2(15) := 'My_Group';
  rg_id RecordGroup;
  gc_id GroupColumn;
BEGIN
  /*
  ** Check to see if Record Group already exists
  */
  rg_id := Find_Group( rg_name );
  /*
  ** If Not, then create it with one number column and one
  ** Char column
ADD_GROUP_ROW

Syntax: ADD_GROUP_ROW(recordgroup_id, row_number);
ADD_GROUP_ROW(recordgroup_name, row_number);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Adds a row to the given record group.

Parameters: 

recordgroup_id The unique ID that Oracle Forms assigns when it creates the group. The data type of the ID is RecordGroup.

recordgroup_name The name you gave to the record group when creating it. The data type of the name is CHAR.

row_number A whole number that specifies a row in the group. If you add a row to any but the last position in a group, all rows below that are logically renumbered. To add a row to the end of a group, use the END_OF_GROUP constant.

Restrictions: 

- You cannot add columns to a group (using ADD_GROUP_COLUMN) after adding rows to the group.
- A group can consist of 0 or more rows.
- You can add rows to a group only after it has been created and columns have been added.
- If you specify a row number greater than the number of rows already in the group (or a negative number), the row is inserted at the end of the group.
- You cannot add rows to a static group without a query.
Error Conditions: Oracle Forms returns a runtime error given either of the following conditions:

- If you enter the name of a non-existent record group.
- If you supply a row number that is out of range or is invalid (for example, an alphabetic character).

Example:

/*
** Built-in: ADD_GROUP_ROW
** Example: Add ten rows to a new record group and populate.
*/
PROCEDURE Populate_My_Group IS
    rg_name  VARCHAR2(20) := 'My_Group';
    rg_col1  VARCHAR2(20) := rg_name||'.NumCol';
    rg_col2  VARCHAR2(20) := rg_name||'.CharCol';
    rg_id    RecordGroup;
    gc_id    GroupColumn;
    in_words VARCHAR2(15);
BEGIN
    /*
    ** Check to see if Record Group already exists
    */
    rg_id := Find_Group( rg_name );
    /*
    ** If it does, then clear all the rows from the group and
    ** populate ten rows with the numbers from 1..10 along
    ** with the equivalent number in words.
    **
    **    Row#    NumCol    CharCol
    **     ----    ------    ------
    **      1       1      one
    **      2       2      two
    **      :       :       :
    **     10      10      ten
    */
    IF NOT Id_Null(rg_id) THEN
        Delete_Group_Row( rg_id, ALL_ROWS );
        FOR i IN 1..10 LOOP
            /*
            ** Add the i-th Row to the end (bottom) of the
            ** record group, and set the values of the two cells
            */
            in_words := TO_CHAR(TO_DATE(i,'YYYY'),'year');
            Add_Group_Row( rg_id, END_OF_GROUP );
            Set_Group_Number_Cell( rg_col1, i, i);
            Set_Group_Char_Cell( rg_col2, i, in_words);
        END LOOP;
    END IF;
END;
**ADD_LIST_ELEMENT**

**Syntax:**

```
ADD_LIST_ELEMENT(list_name, list_index, list_label, list_value);
ADD_LIST_ELEMENT(list_id, list_index, list_label, list_value);
```

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Adds a single element to a list item.

**Parameters:**

- `list_id` Specifies the unique ID that Oracle Forms assigns when it creates the list item. Use the `FIND_ITEM` built-in to return the ID to an appropriately typed variable. The data type of the ID is `ITEM`.
- `list_name` The name you gave to the list item when you created it. The data type of the name is `CHAR`.
- `list_index` Specifies the list index value. The list index is 1 based.
- `list_label` Specifies the `CHAR` string that you want displayed as the label of the list element.
- `list_value` The actual list element value you intend to add to the list item.

**Restrictions:** For a base table list with the List Style property set to Poplist or T-list, Oracle Forms does not allow you to add an other values element when the block contains queried or changed records. Doing so causes error c. This situation occurs if you have previously used `DELETE_LIST_ELEMENT` or `CLEAR_LIST` to remove the other values element that was specified at design time by the Other Values list item property setting.

**Note:** The block status is `QUERY` when a block contains queried records. The block status is `CHANGED` when a block contains records that have been either inserted or updated.

**Example:**

```sql
/*
** Built-in: ADD_LIST_ELEMENT
** Example: Deletes index value 1 and adds the value "1994" to the list item called years when a button is pressed.
** Trigger: When-Button-Pressed
*/
BEGIN
    Delete_List_Element('years',1);
    Add_List_Element('years', 1, '1994', '1994');
END;
```
**ADD_PARAMETER**

**Syntax:**

ADD_PARAMETER(list, key, paramtype, value);
ADD_PARAMETER(name, key, paramtype, value);

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Adds parameters to a parameter list. Each parameter consists of a key, its type, and an associated value.

**Parameters:**

- `list or name` Specifies the parameter list to which the parameter is assigned. The actual parameter can be either a parameter list ID of type PARAMLIST, or the CHAR name of the parameter list.

- `key` The name of the parameter. The data type of the key is CHAR.

- `paramtype` Specifies one of the following two types:
  
  - **TEXT_PARAMETER** A CHAR string literal.
  
  - **DATA_PARAMETER** A CHAR string specifying the name of a record group defined in the current form. When Oracle Forms passes a data parameter to Oracle Reports or Oracle Graphics, the data in the specified record group can substitute for a query that Oracle Reports or Oracle Graphics would ordinarily execute to run the report or display.

- `value` The actual value you intend to pass to the called module. If you are passing a text parameter, the maximum length is 255 characters. Data type of the value is CHAR.

**Restrictions:**

- If you pass a number as a parameter, it must be enclosed in single-quotes.

- A parameter list can consist of 0 (zero) or more parameters.

- You cannot create a parameter list if one already exists; to do so will cause an error. To avoid this error, use ID_NULL to check to see if a parameter list already exists before creating one. If a parameter list already exists, delete it with DESTROY_PARAMETER_LIST before creating a new list.
Example:

```c
/*
** Built-in:  ADD_PARAMETER
** Example:   Add a value parameter to an existing Parameter
**            List 'TEMPDATA', then add a data parameter to
**            the list to associate named query 'DEPT_QUERY'
**            with record group 'DEPT_RECORDGROUP'.
*/
DECLARE
    pl_id ParamList;
BEGIN
    pl_id := Get_Parameter_List('tempdata');
    IF NOT Id_Null(pl_id) THEN
        Add_Parameter(pl_id,'number_of_copies',TEXT_PARAMETER,'19');
        Add_Parameter(pl_id, 'dept_query', DATA_PARAMETER,
                        'dept_recordgroup');
    END IF;
END;
```

**APPLICATION_PARAMETER**

**Syntax:** `APPLICATION_PARAMETER;`

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Displays all the parameters associated with the current menu, and their current values, in the Enter Parameter Values dialog box.

**Failure:** If no parameters are defined for the current menu, Oracle Forms issues error message `FRM-10201: No parameters needed.`
BACKGROUND_MENU

Syntax:  BACKGROUND_MENU{1|2|3|4|5|6|7|8|9|10};

Built-in Type:  unrestricted procedure

Enter Query Mode:  yes

Description:  Executes the designer–specified menu item n from the background menu.

Parameters:  menu item n  Specifies a menu item number from 1 to 10, indicating the number of the background menu item whose command you want to execute. The data type of the menu number is NUMBER.

Restrictions:  A custom menu module must be active and have at least n menu items assigned to it. The menu module must have a menu named BGM.

Failure:  If background menu item n does not exist for the current application, Oracle Forms issues error message FRM-10204: No command defined for the selected background item. If background menu item n tries to navigate to a menu, Oracle Forms issues error message FRM-10209: No 'next menu' from background in this context.

Example:  /*
** Built-in:  BACKGROUND_MENU
** Example:  Executes one background menu option, or the other
** /
BEGIN
IF :control.user_choice > 5 THEN
  Background_Menu2;
ELSE
  Background_Menu4;
END IF;
END;
BELL

Syntax:  BELL;

Built–in Type:  unrestricted procedure

Enter Query Mode:  yes

Description:  Sets the terminal bell to ring the next time the terminal screen synchronizes with the internal state of the form. This synchronization can occur as the result of internal processing or as the result of a call to the SYNCHRONIZE built–in subprogram.

Parameters:  none

Example:  The following example rings the bell three times:

```for i in 1..3 loop
    bell;
    synchronize;
end loop;```

BLOCK_MENU

Syntax:  BLOCK_MENU;

Built–in Type:  restricted procedure

Enter Query Mode:  yes; however, it is illegal to navigate out of the current block in Enter Query mode

Description:  Displays a list of values (LOV) containing the sequence number and names of valid blocks in your form. Oracle Forms sets the input focus to the first enterable item in the block you select from the LOV.

Parameters:  none

Example:  /* Built-in:  BLOCK_MENU
** Example:  Calls up the list of blocks in the form when the user clicks a button, and prints a message if the user chooses a new block out of the list to which to navigate. */

```declare
    prev_blk varchar2(40) := :system.cursor_block;
begin```
**BREAK**

**Syntax:**
```
BREAK;
```

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Halts form execution and displays the Debugger, while the current form is running in debug mode. From the Debugger you can make selections to view the values of global and system variables. The BREAK built-in is primarily useful when you need to inspect the state of a form during trigger execution.

**Parameters:** none

**Restrictions:** If the current form is not running in debug mode, issuing a call to the BREAK built-in subprogram has no effect.

**Example:**
```
BEGIN
  IF :Emp.Job = 'CLERK' THEN
    Break;
    Call_Form('clerk_timesheet');
    Break;
  END IF;
END;
```
CALL_FORM

Syntax:

CALL_FORM(formmodule_name);
CALL_FORM(formmodule_name, display);
CALL_FORM(formmodule_name, display, switch_menu);
CALL_FORM(formmodule_name, display, switch_menu, query_mode);
CALL_FORM(formmodule_name, display, switch_menu, query_mode, paramlist_id);
CALL_FORM(formmodule_name, display, switch_menu, query_mode, paramlist_name);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Runs an indicated form while keeping the parent form active. Oracle Forms runs the called form with the same Runform preferences as the parent form. When the called form is exited Oracle Forms processing resumes in the calling form at the point from which you initiated the call to CALL_FORM.

Parameters:

formmodule_name  Specifies the formmodule name of the called form. The name must be enclosed in single quotes. The data type of the name is CHAR.

display  Specify one of the following constants as an argument:

   HIDEX Causes Oracle Forms to clear the calling form from the screen before drawing the called form. HIDE is the default parameter.

   NO_HIDE Causes Oracle Forms to display the called form without clearing the calling form from the screen.

switch_menu  Takes one of the following constants as an argument:

   NO_REPLACE Causes Oracle Forms to keep the default menu application of the calling form active for the called form.

   DO_REPLACE Causes Oracle Forms to replace the default menu application of the calling form with the default menu application of the called form.
Restrictions:

- When you call a form, Oracle Forms issues a savepoint for the called form. If the CLEAR_FORM function causes a rollback when the called form is current, Oracle Forms rolls back uncommitted changes to this savepoint.

- Oracle Forms ignores the query_mode parameter when the calling form is running in QUERY_ONLY mode. Oracle Forms runs any form that is called from a QUERY_ONLY form as a QUERY_ONLY form, even if the CALL_FORM syntax specifies that the called form is to run in NO_QUERY_ONLY (normal) mode.

- A parameter list passed to a form via CALL_FORM cannot contain parameters of type DATA_PARAMETER. Only text parameters can be passed with CALL_FORM.

- Some memory allocated for CALL_FORM is not deallocated until the Runform session ends. Exercise caution when creating a large stack of called forms.

- When you execute CALL_FORM in a Pre–Logon, On–Logon, or Post–Logon trigger, always specify the DO_REPLACE parameter to replace the calling form’s menu with the called form’s menu. Failing to specify DO_REPLACE will result in no menu being displayed for the called form. (An alternative solution is to call the REPLACE_MENU built–in from a When–New–Form–Instance trigger in the called form.)

### query_mode

Takes one of the following constants as an argument:

**NO_QUERY_ONLY** Causes Oracle Forms to run the indicated form in normal mode, allowing the operator to perform inserts, updates, and deletes from within the called form.

**QUERY_ONLY** Causes Oracle Forms to run the indicated form in Query Only mode, allowing the operator to query, but not to insert, update, or delete records.

### paramlist_id

Specifies the unique ID Oracle Forms assigns when it creates the parameter list. You can optionally include a parameter list as initial input to the called form. The data type of the ID is PARAMLIST.

### paramlist_name

The name you gave the parameter list object when you defined it. The data type of the name is CHAR.
Example 1:
/*
** Built-in: CALL_FORM
** Example: Calls a form in query-only mode.
*/
BEGIN
  Call_Form('empbrowser',NO_HIDE,NO_REPLACE,QUERY_ONLY);
END;

Example 2:
/*
** Built-in: CALL_FORM
** Example: Calls a form, passing a parameter list if the parameter list exists.
*/
DECLARE
  pl_id       ParamList;
  theFormName VARCHAR2(20)  := 'addcust';
BEGIN
  /*
  ** Try to lookup the ‘TEMPDATA’ parameter list
  */
  pl_id := Get_Parameter_List('tempdata');
  /*
  IF Id_Null(pl_id) THEN
    Call_Form(theFormName);
  ELSE
    Call_Form(theFormName, HIDE, NO_REPLACE, NO_QUERY_ONLY,
             pl_id);
  END IF;
 */
  Call_Form('lookcust',NO_HIDE,DO_REPLACE,QUERY_ONLY);
END;
CALL_INPUT

Syntax: CALL_INPUT;

Built-in Type: restricted procedure

Enter Query Mode: no

Description: Accepts and processes function key input from the operator. When CALL_INPUT is terminated, Oracle Forms resumes processing from the point at which the call to CALL_INPUT occurred.

Parameters: none

Restrictions: CALL_INPUT is included for compatibility with previous versions. You should not include this built-in in new applications.

CHECKBOX_CHECKED

Syntax: CHECKBOX_CHECKED(item_id);
CHECKBOX_CHECKED(item_name);

Built-in Type: unrestricted function

Returns: BOOLEAN

Enter Query Mode: yes

Description: A call to the CHECKBOX_CHECKED function returns a BOOLEAN value indicating the state of the given check box. If the item is not a check box, Oracle Forms returns the following error:

FRM-41038: Item <item_name> is not a check box.

A call to GET_ITEM_PROPERTY(item_name, ITEM_TYPE) can be used to verify the item type before calling CHECKBOX_CHECKED.

To set the value of a check box programmatically, assign a valid value to the check box using standard bind variable syntax.

Refer to Chapter 5, “Properties” for more information about checked and unchecked properties of check boxes.

Parameters: item_id Specifies the unique ID that Oracle Forms assigns to the item when it creates it. The data type of the ID is ITEM.
item_name  Specifies the string you defined as the name of the item at design time. The data type of the name is CHAR.

Restrictions: The CHECKBOX_CHECKED built-in returns a BOOLEAN value regarding the state of the given check box. It does not return the actual value of the check box nor does it return the value you might have indicated for the Other Values property.

Example: /* */ ** Built-in: CHECKBOX_CHECKED ** Example: Sets the query case-sensitivity of the item whose name is passed as an argument, depending on an indicator checkbox item. */ PROCEDURE Set_Case_Sensitivity( it_name VARCHAR2) IS indicator_name VARCHAR2(80) := 'control.case_indicator'; it_id Item;
BEGIN
   it_id := Find_Item(it_name);
   IF Checkbox_Checked(indicator_name) THEN
      /* ** Set the item whose name was passed in to query case-sensitively (i.e., Case Insensitive is False) */
      Set_Item_Property(it_id, CASE_INSENSITIVE_QUERY, PROPERTY_FALSE );
   ELSE
      /* ** Set the item whose name was passed in to query case-insensitively (ie Case Insensitive True) */
      Set_Item_Property(it_id,CASE_INSENSITIVE_QUERY,PROPERTY_TRUE);
   END IF;
END;
CHECK_RECORD_UNIQUENESS

Syntax: CHECK_RECORD_UNIQUENESS;

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: When called from an On-Check-Unique trigger, initiates the default Oracle Forms processing for checking the primary key uniqueness of a record.

This built-in is included primarily for applications that will run against a non-ORACLE data source.

Parameters: none

Restrictions: Valid only in an On-Check-Unique trigger.

Example:

/*
 ** Built-in: CHECK_RECORD_UNIQUENESS
 ** Example: Perform Oracle Forms record uniqueness checking
 ** from the fields in the block that are marked as
 ** primary keys based on a global flag setup at
 ** startup by the form, perhaps based on a
 ** parameter.
 ** Trigger: On-Check-Unique
 */
BEGIN
  /*
   ** Check the global flag we set during form startup
   */
  IF :Global.Using_Transactional_Triggers = 'TRUE' THEN
    User_Exit('chkuniq block=EMP');
  /*
   ** Otherwise, do the right thing.
   */
  ELSE
    Check_Record_Uniqueness;
  END IF;
END;
CLEAR_BLOCK

Syntax:   CLEAR_BLOCK;
          CLEAR_BLOCK(commit_mode);

Built-in Type:  restricted procedure

Enter Query Mode:  no

Description:  Causes Oracle Forms to remove all records from, or “flush,” the current block.

Parameters:  If the operator has made changes to records in the current block that have not been posted or committed, Oracle Forms processes the records, following the directions indicated by the argument supplied for the commit_mode parameter:

commit_mode  The optional action parameter takes the following possible constants as arguments:

  ASK_COMMIT  Oracle Forms prompts the operator to commit the changes during CLEAR_BLOCK processing.

  DO_COMMIT  Oracle Forms validates the changes, performs a commit, and flushes the current block without prompting the operator.

  NO_COMMIT  Oracle Forms validates the changes and flushes the current block without performing a commit or prompting the operator.

  NO_VALIDATE  Oracle Forms flushes the current block without validating the changes, committing the changes, or prompting the operator.

Example:  /*
  ** Built-in:  CLEAR_BLOCK
  ** Example:  Clears the current block without validation, and deposits the primary key value which the user has typed into a global variable which a Pre-Query trigger will use to include it as a query criterion.
  ** Trigger:  When-New-Item-Instance
  */
BEGIN
  IF :Emp.Empno IS NOT NULL THEN
    :Global.Employee_Id := :Emp.Empno;
    Clear_Block(No_Validate);
  END IF;
CLEAR_EOL

Syntax: 

CLEAR_EOL;

Built-in Type: restricted procedure

Enter Query Mode: yes

Description: Clears the current text item’s value from the current cursor position to the end of the line.

Example: /*
** Built-in:  CLEAR_EOL
** Example:  Clears out the contents of any number field when
**            the operator navigates to it.
** Trigger:  When-New-Item-Instance
*/
BEGIN
  IF Get_Item_Property(:System.Trigger_Item, DATATYPE) = 'NUMBER'
  THEN
    Clear_Eol;
  END IF;
END;
CLEAR_FORM

Syntax:    CLEAR_FORM;
          CLEAR_FORM(commit_mode);
          CLEAR_FORM(commit_mode, rollback_mode);

Built–in Type: restricted procedure

Enter Query Mode: no

Description: Causes Oracle Forms to remove all records from, or flush, the current
form, and puts the input focus in the first item of the first block.

Parameters: If the operator has made changes to records in the current form or any
called form, and those records have not been posted or committed,
Oracle Forms processes the records, following the directions indicated
by the argument supplied for the following parameter:

commit_mode

ASK_COMMIT  Oracle Forms prompts the
operator to commit the changes during
CLEAR_FORM processing.

DO_COMMIT  Oracle Forms validates the
changes, performs a commit, and flushes the
current form without prompting the operator.

NO_COMMIT  Oracle Forms validates the
changes and flushes the current form without
performing a commit or prompting the operator.

NO_VALIDATE  Oracle Forms flushes the current
form without validating the changes, committing
the changes, or prompting the operator.

rollback_mode

TO_SAVEPOINT  Oracle Forms rolls back all
uncommitted changes (including posted changes)
to the current form’s savepoint.

FULL_ROLLBACK  Oracle Forms rolls back all
uncommitted changes (including posted changes)
which were made during the current Runform
session. You cannot specify a FULL_ROLLBACK
from a form that is running in post–only mode.
(Post–only mode can occur when your form issues
a call to another form while unposted records exist
in the calling form. To prevent losing the locks
issued by the calling form, Oracle Forms prevents
any commit processing in the called form.)
Restrictions: If you use a PL/SQL ROLLBACK statement in an anonymous block or a user-defined subprogram, Oracle Forms interprets that statement as a CLEAR_FORM built-in subprogram with no parameters.

Example: /* ** Built-in: CLEAR_FORM
** Example: Clear any changes made in the current form,
** without prompting to commit.
*/
BEGIN
  Clear_Form(No_Validate);
END;

CLEAR_ITEM

Syntax: CLEAR_ITEM;

Built-in Type: restricted procedure

Enter Query Mode: yes

Description: Clears the value from the current text item, regardless of the current cursor position, and changes the text item value to NULL.

Example: /* ** Built-in: CLEAR_ITEM
** Example: Clear the current item if it does not represent the first day of a month.
** Trigger: When-New-Item-Instance
*/
BEGIN
  IF TO_CHAR(:Emp.Hiredate,'DD') <> '01' THEN
    Clear_Item;
    Message('This date must be of the form 01-MON-YY');
  END IF;
END;
CLEAR_LIST

Syntax:  
CLEAR_LIST(list_id);
CLEAR_LIST(list_name);

Built-in Type:  unrestricted procedure

Enter Query Mode:  yes

Description:  Clear all elements from a list item. After Oracle Forms clears the list, the list will contain only one element (the null element), regardless of the item’s Required property.

Parameters:  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list_id</td>
<td>Specifies the unique ID that Oracle Forms assigns when it creates the list item. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.</td>
</tr>
<tr>
<td>list_name</td>
<td>The name you gave to the list item when you created it. The data type of the name is CHAR.</td>
</tr>
</tbody>
</table>

Usage Notes:  

- Do not use the CLEAR_LIST built-in if the Other Values property is defined and there are queried records in the block. Doing so may cause Oracle Forms to be unable to display records that have already been fetched.

For example, assume that a list item contains the values A, B, and C and the Other Values property is defined. Assume also that these values have been fetched from the database (a query is open). At this point, if you clear the list with CLEAR_LIST, an error will occur because Oracle Forms will attempt to display the previously fetched values (A, B, and C), but will be unable to because the list was cleared.

Before clearing a list, close any open queries. Use the ABORT_QUERY built-in to close an open query.

Restrictions:  

- For a base table Poplist or T–list list item, Oracle Forms returns error FRM-41331: Could not delete element from <list_item> if no elements can be deleted because there are queried or changed records in the block and the list does not contain an other values element. Refer to the restrictions on DELETE_LIST_ELEMENT for more information.

Note: The block status is QUERY when a block contains queried records. The block status is CHANGED when a block contains records that have been either inserted or updated (queried records have been modified).
• For a Poplist or T-list–style list item, CLEAR_LIST will not clear the default value element or the other values element from the list if they do not meet the criteria specified for deleting these elements with DELETE_LIST_ELEMENT.

When either the default value or other values element cannot be deleted, CLEAR_LIST leaves these elements in the list and clears all other elements. Refer to the restrictions on DELETE_LIST_ELEMENT for more information.

Example:
/*
** Built-in: CLEAR_LIST
** Example: See POPULATE_LIST
*/

---

**CLEAR_MESSAGE**

Syntax: CLEAR_MESSAGE;

Built-in Type: restricted procedure

Enter Query Mode: yes

Description: Removes the current message from the screen message area.

Example:
/*
** Built-in: CLEAR_MESSAGE
** Example: Clear the message from the message line.
*/
BEGIN
    Message('Working...',No_Acknowledge);
    SELECT current_tax
    INTO :Emp.Tax_Rate
    FROM tax_table
    WHERE state_abbrev = :Emp.State;
    Clear_Message;
END;
CLEAR_RECORD

Syntax: CLEAR_RECORD;

Built-in Type: restricted procedure

Enter Query Mode: yes

Description: Causes Oracle Forms to remove, or flush, the current record from the block, without performing validation. If a query is open in the block, Oracle Forms fetches the next record to refill the block, if the record space is no longer filled after removing the current record.

A database record that has been cleared is not processed as a delete by the next Post and Commit Transactions process.

In a default master-detail block relation, clearing the master record causes all corresponding detail records to be cleared without validation.

Example: /*
** Built-in: CLEAR_RECORD
** Example: Clear the current record if it’s not the last record in the block.
*/
BEGIN
   IF :System.Last_Record = ‘TRUE’ AND :System.Cursor_Record = ‘1’ THEN
      Message(‘You cannot clear the only remaining entry.’);
      Bell;
   ELSE
      Clear_Record;
   END IF;
END;
CLOSE_FORM

CLOSE_FORM(form_name);
CLOSE_FORM(form_id);

Built-in Type: restricted procedure
Enter Query Mode: no
Description: In a multiple-form application, closes the indicated form. When the indicated form is the current form, CLOSE_FORM is equivalent to EXIT_FORM.
Parameters: form_name Specifies the name of the form to close as a CHAR.
form_id The unique ID that is assigned to the form dynamically when it is instantiated at runtime. Use the FIND_FORM built-in to an appropriately typed variable. The data type of the form ID is FORMMODULE.
Restrictions: You cannot close a form that is currently disabled as a result of having issued CALL_FORM to invoke a modal called form.

COMMIT_FORM

Syntax: COMMIT_FORM;
Built-in Type: restricted procedure
Enter Query Mode: no
Description: Causes Oracle Forms to update data in the database to match data in the form. Oracle Forms first validates the form, then, for each block in the form, deletes, inserts, and updates to the database, and performs a database commit. As a result of the database commit, the database releases all row and table locks.

If the operator has posted data to the database during the current Runform session, a call to the COMMIT_FORM built-in commits this data to the database.

Following a commit operation, Oracle Forms treats all records in all base-table blocks as if they are queried records from the database. Oracle Forms does not recognize changes that occur in triggers that fire during commit processing.
Restrictions: If you use a PL/SQL COMMIT statement in an anonymous block or a form-level procedure, Oracle Forms interprets that statement as a call to the COMMIT_FORM built-in.

Example 1:

```sql
/*
** Built-in: COMMIT_FORM
** Example: If there are records in the form to be committed, then do so. Raise an error if the commit was not successful.
*/
BEGIN
  /*
  ** Force validation to happen first
  */
  Enter;
  IF NOT Form_Success THEN
    RAISE Form_Trigger_Failure;
  END IF;
  /*
  ** Commit if anything is changed
  */
  IF :System.Form_Status = 'CHANGED' THEN
    Commit_Form;
    /*
    ** A successful commit operation sets Form_Status back to 'QUERY'.
    */
    IF :System.Form_Status <> 'QUERY' THEN
      Message('An error prevented your changes from being committed.');
      Bell;
      RAISE Form_Trigger_Failure;
    END IF;
  END IF;
END IF;
END;
```

Example 2:

```sql
/*
** Built-in: COMMIT_FORM
** Example: Perform Oracle Forms database commit during commit processing. Decide whether to use this Built-in or a user exit based on a global flag setup at startup by the form, perhaps based on a**
** Trigger: On-Commit
*/
BEGIN
  /*
  ** Check the global flag we set during form startup
  */
  IF :Global.Using_Transactional_Triggers = 'TRUE' THEN
    User_Exit('my_commit');
  END IF;
```
CONVERT_OTHER_VALUE

**Syntax:**

CONVERT_OTHER_VALUE(item_id);
CONVERT_OTHER_VALUE(item_name);

**Built-in Type:** restricted procedure

**Enter Query Mode:** yes

**Description:** Converts the current value of a check box, radio group, or list item to the value associated with the current check box state (Checked/Unchecked), or with the current radio group button or list item element.

**Parameters:**

- `item_id` Specifies the unique ID that Oracle Forms assigns to the item when it creates the item. The data type of the ID is ITEM.

- `item_name` Specifies the CHAR string you defined as the name of the item at design time.

**Restrictions:** If the item is not a check box, radio group, or list item, Oracle Forms returns error FRM-41026: Item does not understand operation. To avoid this error, determine the item type by issuing a call to GET_ITEM_PROPERTY(item_name, ITEM_TYPE) before calling CONVERT_OTHER_VALUE.

**Example:**

```/*
** Otherwise, do the right thing.
*/
ELSE
    Commit_Form;
END IF;
END;

BEGIN
    Convert_Other_Value('Emp.Marital_Status');
END;
```
COPY

Syntax: COPY(source, destination);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Copies a value from one item or variable into another item or global variable. Use specifically to write a value into an item that is referenced through the NAME_IN built-in. COPY exists for two reasons:

- You cannot use standard PL/SQL syntax to set a referenced item equal to a value.
- You might intend to programmatically place characters such as relational operators in NUMBER and DATE fields while a form is in Enter Query mode.

Parameters: source The source can be either a literal value or a reference to a text item or global variable.

destination The destination can be either a text item or another global variable.

Restrictions: No validation is performed on a value copied to a text item. However, for all other types of items, standard validation checks are performed on the copied value.

Example 1:

```sql
/*
** Built-in: COPY
** Example: Force a wildcard search on the EmpNo item during query.
** Trigger: Pre-Query
*/
DECLARE
  cur_val VARCHAR2(40);
BEGIN
  /*
  ** Get the value of EMP.EMPNO as a string
  */
  cur_val := Name_In('Emp.Empno');
  /*
  ** Add a percent to the end of the string.
  */
  cur_val := cur_val || ' %';
  /*
  ** Copy the new value back into the item so Forms will use it as a query criterion.
  */
```
Example 2:

```sql
/*
** Built-in: COPY
** Example: Set the value of a global variable whose name is
**          dynamically constructed.
*/
DECLARE
    global_var_name  VARCHAR2(80);
BEGIN
    IF :Selection.Choice = 5 THEN
        global_var_name := 'Storage_1';
    ELSE
        global_var_name := 'Storage_2';
    END IF;
    /*
    ** Use the name in the 'global_var_name' variable as the
    ** name of the global variable in which to copy the
    ** current 'Yes' value.
    */
    COPY( 'Yes', 'GLOBAL.'||global_var_name );
END;
```

### COPY_REGION

**Syntax:** `COPY_REGION;`

**Built-in Type:** restricted procedure

**Enter Query Mode:** yes

**Description:** Copies the selected text from the screen and stores it in the paste buffer until you cut or copy another selected region.

**Parameters:** none

**Usage Notes:** Use COPY_REGION, as well as the other editing functions, on text only. The cut and copy functions transfer the selected region of text into the system clipboard until you indicate the paste target. At that time, the cut or copied text is pasted onto the target location.
COUNT_QUERY

Syntax: COUNT_QUERY;

Built-in Type: restricted procedure

Enter Query Mode: yes

Description: In an On-Count trigger, performs the default Oracle Forms processing for identifying the number of rows that a query will retrieve for the current block, and clears the current block. If there are changes to commit in the block, Oracle Forms prompts the operator to commit them during COUNT_QUERY processing. Oracle Forms returns the following message as a result of a valid call to COUNT_QUERY:

FRM-40355: Query will retrieve <number> records.

This built-in is included primarily for applications that will run against a non-ORACLE data source.

Parameters: none

Restrictions: Valid only in triggers that allow restricted built-ins.

Example 1:

```/*
** Built-in: COUNT_QUERY
** Example: Display the number of records that will be retrieved
** by the current query.
*/
BEGIN
  Count_Query;
END;
```

Example 2:

```/*
** Built-in: COUNT_QUERY
** Example: Perform Oracle Forms count query hits processing.
** Decide whether to use this Built-in or a user exit based on a global flag setup at startup by the form, perhaps based on a parameter.
** Trigger: On-Count
*/
BEGIN
  /*
** Check the global flag we set during form startup
*/
  IF :Global.Using_Transactional_Triggers = 'TRUE' THEN
    /*
** User exit returns query hits count back into the CONTROL.HITS item.
*/
    User_Exit('my_count');
```
CREATE_GROUP

** Syntax: ** CREATE_GROUP(recordgroup_name);

** Built–in Type: ** unrestricted function

** Returns: ** RecordGroup

** Enter Query Mode: ** yes

** Description: ** Creates a non–query record group with the given name. The new record group has no columns and no rows until you explicitly add them using the ADD_GROUP_COLUMN, the ADD_GROUP_ROW, and the POPULATE_GROUP_WITH_QUERY built–ins.

** Parameters: ** recordgroup_name  The string you defined as the name of the record group at design time. When Oracle Forms creates the record group object it also assigns the object a unique ID of type RecordGroup. You can call the record group by name or by ID in later calls to record group or record group column built–in subprograms.

** Restrictions: ** A record group created with CREATE_GROUP does not have an associated query. For this reason, you cannot populate such a record group with the POPULATE_GROUP built–in. Use POPULATE_GROUP_WITH_QUERY instead.

** Example: ** /*
** Built–in: CREATE_GROUP
** Example:  Creates a record group and populates its values
*/
** from a query.
*/
DECLARE
  rg_name  VARCHAR2(40) := 'Salary_Range';
  rg_id    RecordGroup;
  gc_id    GroupColumn;
  errcode  NUMBER;
BEGIN
  /*
  ** Make sure the record group does not already exist.
  */
  rg_id := Find_Group(rg_name);
  /*
  ** If it does not exist, create it and add the two
  ** necessary columns to it.
  */
  IF Id_Null(rg_id) THEN
    rg_id := Create_Group(rg_name);
    /* Add two number columns to the record group */
    gc_id := Add_Group_Column(rg_id, 'Base_Sal_Range',
                              NUMBER_COLUMN);
    gc_id := Add_Group_Column(rg_id, 'Emps_In_Range',
                              NUMBER_COLUMN);
  END IF;
  /*
  ** Populate group with a query
  */
  errcode := Populate_Group_With_Query( rg_id,
                                        'SELECT SAL-MOD(SAL,1000),COUNT(EMPNO)'
                                        ||'FROM EMP '
                                        ||'GROUP BY SAL-MOD(SAL,1000)'
                                        ||'ORDER BY 1');
END;
CREATE_GROUP_FROM_QUERY

Syntax: CREATE_GROUP_FROM_QUERY(recordgroup_name, query);

Built-in Type: unrestricted function

Returns: RecordGroup

Enter Query Mode: yes

Description: Creates a record group with the given name. The record group has columns representing each column you include in the select list of the query. Add rows to the record group with the POPULATE_GROUP built-in.

Note: If you do not pass a formal column name or alias for a column in the SELECT statement, Oracle Forms creates ICRGGQ with a dummy counter <NUM>. This happens whenever the column name would have been invalid. The first dummy name–counter always takes the number one. For example, the query SELECT 1 + 1 FROM DUAL would result in a column named ICRGGQ_1.

Parameters: recordgroup_name The name of the record group. When Oracle Forms creates the record group object it also assigns the object a unique ID of type RecordGroup. The data type of the name is CHAR.

query A valid SQL SELECT statement, enclosed in single quotes. Any columns retrieved as a result of the query take the data types of the columns in the table. If you restrict the query to a subset of the columns in the table, then Oracle Forms creates only those columns in the record group. The data type of the query is CHAR. (Required parameter.)

Restrictions: You can use the POPULATE_GROUP_WITH_QUERY built-in to populate groups created with the built-in CREATE_GROUP_FROM_QUERY, but the columns you specify in the SELECT statement must match those in the record group, as specified in the CREATE_GROUP_FROM_QUERY.

Note that the query parameter is required.
Example:

```sql
/*
** Built-in: CREATE_GROUP_FROM_QUERY
** Example: Create a record group from a query, and populate it.
*/
DECLARE
    rg_name VARCHAR2(40) := 'Salary_Range';
    rg_id RecordGroup;
    errcode NUMBER;
BEGIN
    /*
    ** Make sure group doesn’t already exist
    */
    rg_id := Find_Group( rg_name );
    /*
    ** If it does not exist, create it and add the two
    ** necessary columns to it.
    */
    IF Id_Null(rg_id) THEN
        rg_id := Create_Group_From_Query( rg_name,
            'SELECT SAL-MOD(SAL,1000) BASE_SAL_RANGE,
            | | COUNT(EMPNO) EMPS_IN_RANGE ' 
            | | FROM EMP ' 
            | | GROUP BY SAL-MOD(SAL,1000) ' 
            | | ORDER BY 1');
    END IF;
    /*
    ** Populate the record group
    */
    errcode := Populate_Group( rg_id );
END;
```
CREATE_PARAMETER_LIST

Syntax: CREATE_PARAMETER_LIST(name);

Built-in Type: unrestricted function

Returns: ParamList

Enter Query Mode: yes

Description: Creates a parameter list with the given name. The parameter list has no parameters when it is created; they must be added using the ADD_PARAMETER built-in subprogram. A parameter list can be passed as an argument to the CALL_FORM, NEW_FORM, OPEN_FORM, and RUN_PRODUCT built-in subprograms.

Parameters: name Specifies the CHAR name of the parameter list object.

When Oracle Forms creates the object, it assigns it a unique ID of type PARAMLIST. You can call the parameter list by name or by ID in later calls to parameter list–related built-in subprograms.

Restrictions: • The value of any parameter must match the type of the target item as defined in the form or called product to which you are passing the list.

• You cannot create a parameter list named DEFAULT. DEFAULT is reserved for the parameter list that Oracle Forms creates at the initiation of a runtime session.

• You cannot create a parameter list if one already exists; to do so will cause an error. To avoid this error, use ID_NULL to check to see if a parameter list already exists before creating one. If a parameter list already exists, delete it before creating a new list.

Example:

```sql
/*
** Built-in: CREATE_PARAMETER_LIST
** Example: Create a parameter list named 'TEMPDATA'. First
** make sure the list does not already exist, then
** attempt to create a new list. Signal an error
** if the list already exists or if creating the
** list fails.
*/
DECLARE
    pl_id ParamList;
    pl_name VARCHAR2(10) := 'tempdata';
BEGIN
```

Built-in Subprograms 3 – 51
CREATE_QUERIED_RECORD

Syntax: CREATE_QUERIED_RECORD;

Built-in Type: restricted procedure

Enter Query Mode: no

Description: When called from an On-Fetch trigger, creates a record on the block’s waiting list. The waiting list is an intermediary record buffer that contains records that have been fetched from the data source but have not yet been placed on the block’s list of active records. This built-in is included primarily for applications using transactional triggers to run against a non-ORACLE data source.

Note that there is no way to remove a record from the waiting list. Consequently, the application must ensure that there is data available to be used for populating the record programmatically.

Parameters: none

Restrictions: • Valid only in the On-Fetch trigger.
• In blocks with a large number of records, this procedure can have side effects on disk I/O, memory allocation, or both.

Example: /*
** Built-in: CREATE_QUERIED_RECORD
** Example: Fetch the next N records into this block. Record count kept in Global.Record_Count.
** Trigger: On-Fetch
*/
DECLARE
    fetch_count NUMBER;
FUNCTION The_Next_Seq
RETURN NUMBER IS
CURSOR next_seq IS SELECT uniq_seq.NEXTVAL FROM DUAL;
tmp NUMBER;
BEGIN
OPEN next_seq;
FETCH next_seq INTO tmp;
CLOSE next_seq;
RETURN tmp;
END;
BEGIN

/*
** Determine how many records Oracle Forms is expecting us to
** fetch
*/
fetch_count := Get_Block_Property('MYBLOCK',RECORDS_TO_FETCH);
FOR i IN 1..fetch_count LOOP

/*
** Create the Queried Record into which we'll deposit
** the values we're about to fetch;
*/
Create_Queried_Record;
:Global.Record_Count := NVL(:Global.Record_Count,0)+1;

/*
** Populate the item in the queried record with a
** sequence function we declared above
*/
:myblock.numbercol := the_next_seq;
END LOOP;
END;
CREATE_RECORD

Syntax: CREATE_RECORD;

Built-in Type: restricted procedure

Enter Query Mode: no

Description: Creates a new record in the current block after the current record. Oracle Forms then navigates to the new record.

Parameters: none

Example: /*
** Built-in: CREATE_RECORD
** Example: Populate new records in a block based on return
** values from a query
*/
PROCEDURE Populate_Rows_Into_Block( projid NUMBER) IS
  CURSOR tempcur( cp_projid NUMBER ) IS
    SELECT milestone_name, due_date
    FROM milestone
    WHERE project_id = cp_projid
    ORDER BY due_date;
BEGIN
  /* Add these records to the bottom of the block */
  Last_Record;
  /* Loop thru the records in the cursor */
  FOR rec IN tempcur( projid ) LOOP
    /*
      ** Create an empty record and set the current row’s
      ** Milestone_Name and Due_Date items.
      */
    Create_Record;
    : Milestone.Milestone_Name := rec.milestone_name;
    : Milestone.Due_Date := rec.due_date;
  END LOOP;
  First_Record;
END;
CREATE_TIMER

Syntax:  CREATE_TIMER(timer_name, milliseconds, iterate);

Built-in Type:  unrestricted function

Returns:  Timer

Enter Query Mode:  yes

Description:  Creates a new timer with the given name. You can indicate the interval and whether the timer should repeat upon expiration or execute once only. When the timer expires, Oracle Forms fires the When–Timer–Expired trigger.

Parameters:

- **timer_name**  Specifies the timer name of up to 30 alphanumeric characters. The name must begin with an alphabetic character. The data type of the name is CHAR.

- **milliseconds**  Specifies the duration of the timer in milliseconds. The range of values allowed for this parameter is 1 to 2147483648 milliseconds. Values > 2147483648 will be rounded down to 2147483648. Note that only positive numbers are allowed. The data type of the parameter is NUMBER. See Restrictions below for more information.

- **iterate**  Specifies whether the timer should repeat or not upon expiration. Takes the following constants as arguments:
  
  - **REPEAT**  Indicates that the timer should repeat upon expiration. Default.
  - **NO_REPEAT**  Indicates that the timer should not repeat upon expiration, but is to be used once only, until explicitly called again.

Restrictions:

- A value less than 1 results in a runtime error.
- Values > 2147483648 will be rounded down to 2147483648.
- Milliseconds cannot be expressed as a decimal.
- No two timers can share the same name in the same form instance, regardless of case.
- If there is no When–Timer–Expired trigger defined at the execution of a timer, Oracle Forms returns an error.
• If there is no When–Timer–Expired trigger defined at the execution of a timer, and the timer is a repeating timer, subsequent repetitions are canceled, but the timer is retained.

• If there is no When–Timer–Expired trigger defined at the execution of a timer, and the timer is not a repeating timer, the timer is deleted.

Example: The following example creates a timer called EMP_TIMER, and sets it to 60 seconds and an iterate value of NO_REPEAT:

```sql
DECLARE
    timer_id Timer;
    one_minute NUMBER(5) := 60000;
BEGIN
    timer_id := CREATE_TIMER('emp_timer', one_minute, NO_REPEAT);
END;
```

---

CUT_REGION

**Syntax:** CUT_REGION;

**Built-in Type:** restricted procedure

**Enter Query Mode:** yes

**Description:** Removes a selected region of text from the screen and stores it in the paste buffer until you cut or copy another selected region.

**Parameters:** none

**Usage Notes:** Use CUT_REGION, as well as the other editing functions, on text only. The cut and copy functions transfer the selected region of text into the system clipboard until you indicate the paste target. At that time, the cut or copied text is pasted onto the target location.
DBMS_ERROR_CODE

Syntax: DBMS_ERROR_CODE;

Built–in Type: unrestricted function

Enter Query Mode: yes

Description: Returns the error number of the last database error that was detected.

Parameters: none

Usage Notes: For recursive errors, this built–in returns the code of the first message in the stack, so the error text must be parsed for numbers of subsequent messages.

Example:

```sql
DECLARE
  errcode     NUMBER     := ERROR_CODE;
  dbmserrcode NUMBER;
  dbmserrtext VARCHAR2(200);
BEGIN
  IF errcode = 40508 THEN
    /*
    ** Forms had a problem INSERTing, so
    ** look at the Database error which
    ** caused the problem.
    */
    dbmserrcode := DBMS_ERROR_CODE;
    dbmserrtext := DBMS_ERROR_TEXT;
    IF dbmserrcode = -1438 THEN
      /*
      ** ORA-01438 is "value too large for column"
      */
      Message('Your number is too large. Try again.');
    ELSIF dbmserrcode = -1400 THEN
      /*
      ** ORA-01400 is "Mandatory column is NULL"
      */
      Message('You forgot to provide a value. Try again.');
  END IF;
END;
```
Syntax: DBMS_ERROR_TEXT;

Built-in Type: unrestricted function

Enter Query Mode: yes

Description: Returns the message number (such as ORA-01438) and message text of the database error.

Parameters: none

Usage Notes: You can use this function to test database error messages during exception handling routines.

DBMS_ERROR_TEXT returns the entire sequence of recursive errors.

Example:

/*
** Built-in: DBMS_ERROR_CODE, DBMS_ERROR_TEXT
** Example: Reword certain Forms error messages by evaluating
** the DBMS error code that caused them
** Trigger: On-Error
*/
DECLARE
  errcode NUMBER := ERROR_CODE;
dbmserrcode NUMBER;
dbmserrtext VARCHAR2(200);
BEGIN
  IF errcode = 40508 THEN
    /*
    ** Forms had a problem INSERTing, so
    ** look at the database error which
    ** caused the problem.
    */
    dbmserrcode := DBMS_ERROR_CODE;
dbmserrtext := DBMS_ERROR_TEXT;
ELSE
  /*
  ** Print out a generic message with the database
  ** error string in it.
  */
  Message('Insert failed because of '||dbmserrtext);
END IF;
END IF;
END;
IF dbmserrcode = -1438 THEN
    /*
     ** ORA-01438 is "value too large for column"
     */
    Message('Your number is too large. Try again.');
ELSIF dbmserrcode = -1400 THEN
    /*
     ** ORA-01400 is "Mandatory column is NULL"
     */
    Message('You forgot to provide a value. Try again.');
ELSE
    /*
     ** Printout a generic message with the database
     ** error string in it.
     */
    Message('Insert failed because of '||dbmserrtext);
END IF;
END IF;
END;

---

**DEBUG_MODE**

**Syntax:** `DEBUG_MODE;`

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Toggles debug mode on and off in a menu. When debug mode is True for a menu, Oracle Forms issues an appropriate message when a menu item command executes.

**Parameters:** none

**Restrictions:** The DEBUG_MODE applies only to a menu module. It does not place the form in Debug Mode.
**DEFAULT_VALUE**

**Syntax:**
```
DEFAULT_VALUE(value_string,variable_name);
```

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Copies an indicated value to an indicated variable if the variable's current value is NULL. If the variable’s current value is not NULL, DEFAULT_VALUE does nothing. Therefore, for text items this built-in works identically to using the COPY built-in on a NULL item. If the variable is an undefined global variable, Oracle Forms creates the variable.

**Parameters:**
- `value_string` A valid CHAR string, variable, or text item containing a valid string.
- `variable_name` A valid variable, global variable, or text item name. The data type of the variable_name is CHAR. Any object passed as an argument to this built-in must be enclosed in single quotes.

**Restrictions:** The DEFAULT_VALUE built-in is not related to the Default Value item property.

**Example:**
```
/*
** Built-in:  DEFAULT_VALUE
** Example:  Make sure a Global variable is defined by
**          assigning some value to it with Default_Value
*/
BEGIN
  /*
  ** Default the value of GLOBAL.Command_Indicator if it is
  ** NULL or does not exist.
  */
  DEFAULT_VALUE('***','global.command_indicator');
  /*
  ** If the global variable equals the string we defaulted
  ** it to above, then it must have not existed before
  */
  IF :Global.Command_Indicator = *** THEN
    Message('You must call this screen from the Main Menu');
    RAISE Form_Trigger_Failure;
  END IF;
END;
```
**DELETE_GROUP**

**Syntax:**
```
DELETE_GROUP(recordgroup_id);
DELETE_GROUP(recordgroup_name);
```

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Deletes a programmatically created record group.

**Parameters:**
- `recordgroup_id` The unique ID that Oracle Forms assigns when it creates the group. The data type of the ID is `RecordGroup`.
- `recordgroup_name` The name you gave to the record group when creating it. The data type of the name is `CHAR`.

**Restrictions:** This built-in cannot be used to delete a record group that was created at design time.

**Example:**
```
PROCEDURE Remove_Record_Group( rg_name VARCHAR2 ) IS
  rg_id RecordGroup;
BEGIN
  /*
  ** Make sure the Record Group exists before trying to delete it.
  */
  rg_id := Find_Group( rg_name );
  IF NOT Id_Null(rg_id) THEN
    Delete_Group( rg_id );
  END IF;
END;
```
DELETE_GROUP_ROW

Syntax:  
DEDELETE_GROUP_ROW(recordgroup_id, row_number);  
DEDELETE_GROUP_ROW(recordgroup_name, row_number);

Built-in Type:  unrestricted procedure

Enter Query Mode:  yes

Description:  Deletes the indicated row or all rows of the given record group. Oracle Forms automatically decrements the row numbers of all rows that follow a deleted row. When rows are deleted, the appropriate memory is freed and available to Oracle Forms.

If you choose to delete all rows of the group by supplying the ALL_ROWS constant, Oracle Forms deletes the rows, but the group still exists until you perform the DELETE_GROUP subprogram.

When a single row is deleted, subsequent rows are renumbered so that row numbers remain contiguous.

Parameters:  
recordgroup_id  The unique ID that Oracle Forms assigns the group when it creates it. The data type of the ID is RecordGroup.
recordgroup_name  The name you gave to the record group when you created it. The data type of the name is CHAR.
row_number  Specifies the row to be deleted from the record group. Rows are automatically numbered from 1 to n. Row number parameter data type is NUMBER.

ALL_ROWS  Specifies that Oracle Forms is to delete all rows without deleting the record group. ALL_ROWS is a constant.

Restrictions:  This built-in cannot be used to delete rows from a static record group.
Example:

```sql
/*
** Built-in:  DELETE_GROUP_ROW
** Example:  Delete certain number of records from the tail of the specified record group.
**
*/
PROCEDURE Delete_Tail_Records( recs_to_del NUMBER,
                                rg_name VARCHAR2 ) IS
  rg_id     RecordGroup;
  rec_count NUMBER;
BEGIN
  /*
  ** Check to see if Record Group exists
  */
  rg_id := Find_Group( rg_name );
  /*
  ** Get a count of the records in the record group
  */
  rec_count := Get_Group_Row_Count( rg_id );
  IF rec_count < recs_to_del THEN
    Message('There are only '||TO_CHAR(rec_count)||' records in the group.');
    RAISE Form_Trigger_Failure;
  END IF;
  /*
  ** Loop thru and delete the last 'recs_to_del' records
  */
  FOR j IN 1..recs_to_del LOOP
    Delete_Group_Row( rg_id, rec_count - j + 1 );
  END LOOP;
END;
```
DELETE_LIST_ELEMENT

Syntax:  DELETE_LIST_ELEMENT(list_name, list_index);
         DELETE_LIST_ELEMENT(list_id, list_index);

Built-in Type:  unrestricted procedure

Enter Query Mode:  yes

Description:  Deletes a specific list element from a list item.

Parameters:  

- list_id  Specifies the unique ID that Oracle Forms assigns when it creates the list item. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.

- list_name  The name you gave to the list item when you created it. The data type of the name is CHAR.

- list_index  Specifies the list index value. The list index is 1 based.

Usage Notes:

- Do not use the DELETE_LIST_ELEMENT built-in if the Other Values property is defined and there are queried records in the block. Doing so may cause Oracle Forms to be unable to display records that have already been fetched.

For example, assume that a list item contains the values A, B, and C and the Other Values property is defined. Assume also that these values have been fetched from the database (a query is open). At this point, if you delete B from the list using DELETE_LIST_ELEMENT, an error will occur because Oracle Forms will attempt to display the previously fetched values (A, B, and C), but will be unable to because B was deleted from the list.

Before deleting a list element, close any open queries. Use the ABORT_QUERY built-in to close an open query.

Note:  A list does not contain an other values element if none was specified at design time or if it was programmaticaly deleted from the list at runtime.

Restrictions:  For a Poplist or T-list–style list item, Oracle Forms returns error FRM-41331: Could not delete element from <list_item> if you attempt to delete the default value element.

The default value element is the element whose label or value was specified at design time for the Default Value property setting.
For a Combobox list item, you can delete the default value element only if the Default Value property was set to an actual value, rather than an element label.

For a base table Poplist or T-list list item, Oracle Forms returns error FRM-41331: Could not delete element from <list_item> if you:

- attempt to delete the other values element when the block contains queried or changed records.
- attempt to delete any element from a list that does not contain an other values element when the block contains queried or changed records.

**Note:** The block status is QUERY when a block contains queried records. The block status is CHANGED when a block contains records that have been either inserted or updated (queried records have been modified).

**Example:**
```c
/*
** Built-in:  DELETE_LIST_ELEMENT
** Example:   See ADD_LIST_ELEMENT
*/
```

---

**DELETE_PARAMETER**

**Syntax:**

```c
DELETE_PARAMETER(list, key);
DELETE_PARAMETER(name, key);
```

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Deletes the parameter with the given key from the parameter list.

**Parameters:**

- `list` or `name` Specifies the parameter list, either by list ID or name. The actual parameter can be either a parameter list ID of type PARAMLIST, or the CHAR name of the parameter list.

- `key` The name of the parameter. The data type of the key is CHAR.

**Restrictions:**

- If you delete the only key from a parameter list and attempt to reference the parameter list, Oracle Forms issues an error.
• Deleting the last parameter from a list does not automatically delete the list itself. To delete the parameter list, issue a call to the DESTROY_PARAMETER_LIST subprogram.

Example:
/*
 ** Built-in: DELETE_PARAMETER
 ** Example: Remove the ‘NUMBER_OF_COPIES’ parameter from the
 ** already existing ‘TEMPDATA’ parameter list.
 */
BEGIN
 Delete_Parameter('tempdata','number_of_copies');
END;

DELETE_RECORD

Syntax: DELETE_RECORD;

Built-in Type: restricted procedure

Enter Query Mode: no

Description: When used outside an On-Delete trigger, removes the current record from the block and marks the record as a delete. Records removed with this built-in are not removed one at a time, but are added to a list of records that are deleted during the next available commit process.

If the record corresponds to a row in the database, Oracle Forms locks the record before removing it and marking it as a delete.

If a query is open in the block, Oracle Forms fetches a record to refill the block if necessary. See also the description for the CLEAR_RECORD built-in subprogram.

In an On-Delete trigger, DELETE_RECORD initiates the default Oracle Forms processing for deleting a record during the Post and Commit Transaction process, as shown in Example 2 below.

Parameters: none

Example 1:
/*
 ** Built-in: DELETE_RECORD
 ** Example: Mark the current record in the current block for deletion.
 */
BEGIN
 Delete_Record;
END;
Example 2: /*
** Built-in: DELETE_RECORD
** Example: Perform Oracle Forms delete record processing
during commit-time. Decide whether to use this
Built-in or a user exit based on a global flag
setup at startup by the form, perhaps based on
a parameter.
** Trigger: On-Delete
*/
BEGIN
  IF :Global.Using_Transactional_Triggers = 'TRUE' THEN
    User_Exit('my_delrec block=EMP');
  ELSE
    Delete_Record;
  END IF;
END;

DELETE_TIMER

Syntax: DELETE_TIMER(timer_id);
        DELETE_TIMER(timer_name);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Deletes the given timer from the form.

Parameters: timer_id Specifies the unique ID that Oracle Forms assigns
when it creates the timer, specifically as a response
to a successful call to the CREATE_TIMER built-in.
Use the FIND_TIMER built-in to return the ID to
an appropriately typed variable. That data type of
the ID is Timer.

timer_name Specifies the name you gave the timer when you
defined it. The data type of the timer_name is
CHAR.
Restrictions:

- You must identify a timer that currently exists in the form, otherwise Oracle Forms generates an error. If you call DELETE_TIMER during execution of the timer, whether a repeating or non-repeating timer, Oracle Forms immediately deletes the timer. A subsequent call to that timer generates a runtime error, unless you recreate the timer first.

- If you delete a timer, you must issue a FIND_TIMER call before attempting to call ID_NULL to check on availability of the timer object. For instance, the following example is incorrect because the call to DELETE_TIMER does not set the value of the ID. In other words, the timer is deleted, but the ID continues to exist, yet points to a non-existent timer, hence, it is not null.

```-- Invalid Example
    timer_id := Find_Timer('my_timer');
    Delete_Timer(timer_id);
    IF (ID_Null(timer_id))...
```

Example:

```/*
** Built-in:  DELETE_TIMER
** Example:   Remove a timer after first checking to see if it exists
**
*/
PROCEDURE Cancel_Timer( tm_name VARCHAR2 ) IS
    tm_id Timer;
BEGIN
    tm_id := Find_Timer( tm_name );

    IF NOT Id_Null(tm_id) THEN
        Delete_Timer(tm_id);
    ELSE
        Message('Timer '||tm_name||' has already been cancelled.');
    END IF;
END;
```
DESTROY_PARAMETER_LIST

Syntax:  
DESTROY_PARAMETER_LIST(list);
DESTROY_PARAMETER_LIST(name);

Built-in Type:  unrestricted procedure

Enter Query Mode:  yes

Description:  Deletes a dynamically created parameter list and all parameters it contains.

Parameters:  list or name

Specifies the parameter list, either by list ID or name. The actual parameter can be either a parameter list ID of type PARAMLIST, or the CHAR name of the parameter list.

Use the GET_PARAMETER_LIST built-in to return the ID to a variable of the type PARAMLIST.

Example:  
/*
** Built-in: DESTROY_PARAMETER_LIST
** Example: Remove the parameter list ‘tempdata’ after first checking to see if it exists
*/
DECLARE
   pl_id ParamList;
BEGIN
   pl_id := Get_Parameter_List(‘tempdata’);
   IF NOT Id_Null(pl_id) THEN
      DESTROY_PARAMETER_LIST(pl_id);
   END IF;
END;
DISPLAY_ERROR

Syntax: DISPLAY_ERROR;
Built-in Type: unrestricted procedure
Enter Query Mode: yes
Description: Displays the Display Error screen if there is a logged error. When the operator presses a function key while viewing the Display Error screen, Oracle Forms redisplays the form. If there is no error to display when you call this built-in, Oracle Forms ignores the call and does not display the DISPLAY ERROR screen.
Parameters: none

DISPLAY_ITEM

Syntax: DISPLAY_ITEM(item_id, attribute);
DISPLAY_ITEM(item_name, attribute);
Built-in Type: unrestricted procedure
Enter Query Mode: yes
Description: Modifies an item’s appearance by assigning a specified display attribute to the item.
You can reference any item in the current form. Note that DISPLAY_ITEM only affects the display of the current instance of the item; other instances of the specified item are not affected. This means that if you specify a display change for an item that exists in a multi-record block, DISPLAY_ITEM only changes the instance of that item that belongs to the block’s current record. If you want to change all instances of an item in a multi-record block, use SET_ITEM_PROPERTY.
Any change made by a DISPLAY_ITEM built-in is effective until another DISPLAY_ITEM references the same item, or that instance of the item is removed (e.g., through a CLEAR_RECORD or a query), or the current form is exited.
Parameters:  

**item_id**  
Specifies the unique ID that Oracle Forms assigns to the item when it creates the item. The data type of the ID is ITEM.

**item_name**  
Specifies the CHAR string you gave to the item when you created it.

**attribute**  
Specifies a named visual attribute that should exist. You can also specify a valid attribute from your Oracle*Terminal resource file. Oracle Forms will search for named visual attribute first.

Example:  
```sql
/*
** Built-in:  DISPLAY_ITEM
** Example:   Change the visual attribute of each item in the current record.
**
* /
DECLARE
  cur_itm   VARCHAR2(80);
  cur_block VARCHAR2(80) := :System.Cursor_Block;
BEGIN
  cur_itm   := Get_Block_Property( cur_block, FIRST_ITEM );
  WHILE ( cur_itm IS NOT NULL ) LOOP
    cur_itm := cur_block||'.'||cur_itm;
    Display_Item( cur_itm, 'My_Favorite_Named_Attribute');
    cur_itm := Get_Item_Property( cur_itm, NEXTITEM );
  END LOOP;
END;
```

DOWN

**Syntax:**  
DOWN;

**Built-in Type:**  
restricted procedure

**Enter Query Mode:**  
no

**Description:**  
Navigates to the instance of the current item in the record with the next higher sequence number. If necessary, Oracle Forms fetches a record. If Oracle Forms has to create a record, DOWN navigates to the first navigable item in the record.

**Parameters:**  
none
DO_KEY

Syntax: \texttt{DO\_KEY(built-in\_subprogram\_name);}

Built-in Type: restricted procedure

Enter Query Mode: yes

Description: Executes the key trigger that corresponds to the specified built-in subprogram. If no such key trigger exists, then the specified subprogram executes. This behavior is analogous to pressing the corresponding function key.

Parameters: \texttt{built-in\_subprogram\_name} Specifies the name of a valid built-in subprogram.

Restrictions: \texttt{DO\_KEY} accepts built-in names only, not key names: \texttt{DO\_KEY(ENTER\_QUERY)}. To accept a specific key name, use the \texttt{EXECUTE\_TRIGGER} built-in: \texttt{EXECUTE\_TRIGGER('KEY\_F11')}.

<table>
<thead>
<tr>
<th>Built-in</th>
<th>Key Trigger</th>
<th>Associated Function Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT_QUERY</td>
<td>Key--EXIT</td>
<td>[Exit/Cancel]</td>
</tr>
<tr>
<td>BLOCK_MENU</td>
<td>Key--MENU</td>
<td>[Block Menu]</td>
</tr>
<tr>
<td>CLEAR_BLOCK</td>
<td>Key--CLRBLK</td>
<td>[Clear Block]</td>
</tr>
<tr>
<td>CLEAR_FORM</td>
<td>Key--CLRFRM</td>
<td>[Clear Form]</td>
</tr>
<tr>
<td>CLEAR_RECORD</td>
<td>Key--CLRREC</td>
<td>[Clear Record]</td>
</tr>
<tr>
<td>COMMIT_FORM</td>
<td>Key--COMMIT</td>
<td>[Commit]</td>
</tr>
<tr>
<td>COUNT_QUERY</td>
<td>Key--CQUERY</td>
<td>[Count Query Hits]</td>
</tr>
<tr>
<td>CREATE_RECORD</td>
<td>Key--CREREC</td>
<td>[Insert Record]</td>
</tr>
<tr>
<td>DELETE_RECORD</td>
<td>Key--DELRREC</td>
<td>[Delete Record]</td>
</tr>
<tr>
<td>DOWN</td>
<td>Key--DOWN</td>
<td>[Down]</td>
</tr>
<tr>
<td>DUPLICATE_ITEM</td>
<td>Key--DUP_ITEM</td>
<td>[Duplicate Item]</td>
</tr>
<tr>
<td>DUPLICATE_RECORD</td>
<td>Key--DUPREC</td>
<td>[Duplicate Record]</td>
</tr>
<tr>
<td>EDIT_TEXTITEM</td>
<td>Key--EDIT</td>
<td>[Edit]</td>
</tr>
<tr>
<td>ENTER</td>
<td>Key--ENTER</td>
<td>[Enter]</td>
</tr>
<tr>
<td>ENTER_QUERY</td>
<td>Key--ENTQRY</td>
<td>[Enter Query]</td>
</tr>
<tr>
<td>EXECUTE_QUERY</td>
<td>Key--EXEQRY</td>
<td>[Execute Query]</td>
</tr>
<tr>
<td>EXIT_FORM</td>
<td>Key--EXIT</td>
<td>[Exit/Cancel]</td>
</tr>
<tr>
<td>HELP</td>
<td>Key--HELP</td>
<td>[Help]</td>
</tr>
<tr>
<td>LIST_VALUES</td>
<td>Key--LISTVAL</td>
<td>[List]</td>
</tr>
<tr>
<td>Built-in</td>
<td>Key Trigger</td>
<td>Associated Function Key</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>LOCK_RECORD</td>
<td>Key–UPDREC</td>
<td>[Lock Record]</td>
</tr>
<tr>
<td>NEXT_BLOCK</td>
<td>Key–NXTBLK</td>
<td>[Next Block]</td>
</tr>
<tr>
<td>NEXT_ITEM</td>
<td>Key–NEXT–ITEM</td>
<td>[Next Item]</td>
</tr>
<tr>
<td>NEXT_KEY</td>
<td>Key–NXTKEY</td>
<td>[Next Primary Key Fld]</td>
</tr>
<tr>
<td>NEXT_RECORD</td>
<td>Key–NXTREC</td>
<td>[Next Record]</td>
</tr>
<tr>
<td>NEXT_SET</td>
<td>Key–NXTSET</td>
<td>[Next Set of Records]</td>
</tr>
<tr>
<td>PREVIOUS_BLOCK</td>
<td>Key–PRVBLK</td>
<td>[Previous Block]</td>
</tr>
<tr>
<td>PREVIOUS_ITEM</td>
<td>Key–PREV–ITEM</td>
<td>[Previous Item]</td>
</tr>
<tr>
<td>PREVIOUS_RECORD</td>
<td>Key–PRVREC</td>
<td>[Previous Record]</td>
</tr>
<tr>
<td>PRINT</td>
<td>Key–PRINT</td>
<td>[Print]</td>
</tr>
<tr>
<td>SCROLL_DOWN</td>
<td>Key–SCRDOWN</td>
<td>[Scroll Down]</td>
</tr>
<tr>
<td>SCROLL_UP</td>
<td>Key–SCRUP</td>
<td>[Scroll Up]</td>
</tr>
<tr>
<td>UP</td>
<td>Key–UP</td>
<td>[Up]</td>
</tr>
</tbody>
</table>

**Example:**
```
/*
** Built-in:  DO_KEY
** Example:  Simulate pressing the [Execute Query] key.
*/
BEGIN
  Do_Key('Execute_Query');
END;
```

---

**DUPLICATE_ITEM**

**Syntax:**

DUPLICATE_ITEM;

**Built-in Type:**

restricted procedure

**Enter Query Mode:**

no

**Description:**

Assigns the current item the same value as the instance of this item in the previous record.

**Parameters:**

none

**Restrictions:**

A previous record must exist in your current session, or Oracle Forms returns error FRM-41803: No previous record to copy value from.
**DUPLICATE_RECORD**

**Syntax:**

DUPLICATE_RECORD;

**Built-in Type:**

restricted procedure

**Enter Query Mode:**

no

**Description:**

Copies the value of each item in the record with the next lower sequence number to the corresponding items in the current record. The current record must not correspond to a row in the database. If it does, an error occurs. The duplicate record inherits the record status (NEW, CHANGED, or QUERY) of the source record.

**Parameters:**

none

**Restrictions:**

A previous record must exist in your current session.

**Example:**

```/*
** Built-in:  DUPLICATE_RECORD;
** Example:   Make a copy of the current record and increment
** the "line_sequence" item by one.
*/
DECLARE
  n NUMBER;
BEGIN
  /*
   ** Remember the value of the 'line_sequence' from the
   ** current record
   */
  n := :my_block.line_sequence;
  /*
   ** Create a new record, and copy all the values from the
   ** previous record into it.
   */
  Create_Record;
  Duplicate_Record;
  /*
   ** Set the new record's 'line_sequence' to one more than
   ** the last record's.
   */
  :my_block.line_sequence := n + 1;
END;```
EDIT_TEXTITEM

Syntax:

```
EDIT_TEXTITEM;
EDIT_TEXTITEM(x, y);
EDIT_TEXTITEM(x, y, width, height);
```

Built-in Type: restricted procedure

Enter Query Mode: yes

Description: Invokes the Runform item editor for the current text item and puts the form in Edit mode.

Parameters:

- `x` Specifies the x coordinate on the screen where you want to place the upper left corner of the pop-up item editor.
- `y` Specifies the y coordinate on the screen where you want to place the upper left corner of the pop-up item editor.
- `width` Specifies the width of the entire editor window, including buttons.
- `height` Specifies the height of the entire editor window, including buttons.

If you specify a height less than 6 character cells, or its equivalent, Oracle Forms sets the height equal to 6.

You can use the optional EDIT_TEXTITEM parameters to specify the location and dimensions of the pop-up window with which the item editor is associated. If you do not use these parameters, Oracle Forms invokes the item editor with its default location and dimensions.

Restrictions:

- The input focus must be in a text item.
- The Width must be at least wide enough to display the buttons at the bottom of the editor window.

Example:

```
/*
 ** Built-in: EDIT_TEXTITEM
 ** Example: Determine the x-position of the current item
 ** then bring up the editor either on the left
 ** side or right side of the screen so as to not
 ** cover the item on the screen.
 */
DECLARE
  itm_x_pos NUMBER;
BEGIN
```
```sql
itm_x_pos := Get_Item_Property(:System.Cursor_Item,X_POS);
IF itm_x_pos > 40 THEN
    Edit_TextItem(1,1,20,8);
ELSE
    Edit_TextItem(60,1,20,8);
END IF;
END;
```

### ENFORCE_COLUMN_SECURITY

**Syntax:**

```
ENFORCE_COLUMN_SECURITY
```

**Built-in Type:**

unrestricted procedure

**Enter Query Mode:**

yes

**Description:**

Executes default processing for checking column security on a database column. This built-in is included primarily for applications that run against a non-ORACLE data source, and use transactional triggers to replace default Oracle Forms transaction processing.

Default Check Column Security processing refers to the sequence of events that occurs when Oracle Forms enforces column-level security for each block that has the Column Security block property set True. To enforce column security, Oracle Forms queries the database to determine the base table columns to which the current form operator has update privileges. For columns to which the operator does not have update privileges, Oracle Forms makes the corresponding base table items in the form non-updateable by setting the Update Allowed item property to False dynamically. By default, Oracle Forms performs this operation at form startup, processing each block in sequence.

For more information, refer to *Oracle Forms Advanced Techniques*, Chapter 4, “Connecting to Non-Oracle Data Sources.”

**Restrictions:**

Valid only in an On-Column-Security trigger.

**Usage Notes:**

You can include this built-in subprogram in the On-Column-Security trigger if you intend to augment the behavior of that trigger rather than completely replace the behavior. For more information, refer to Chapter 2, “Triggers,” in this manual.
**ENTER**

Syntax: \[ENTER;\]

Built-in Type: restricted procedure

Enter Query Mode: yes

Description: Validates data in the current validation unit. (The default validation unit is Item.)

Parameters: none

Example:
```
/*
** Built-in: ENTER
** Example: Force Validation to occur before calling another form
*/
BEGIN
  Enter;
  IF NOT Form_Success THEN
    RAISE Form_Trigger_Failure;
  END IF;
  Call_Form('newcust');
END;
```

**ENTER_QUERY**

Syntax: \[ENTER_QUERY;
ENTER_QUERY(keyword_one);
ENTER_QUERY(keyword_two);
ENTER_QUERY(keyword_one, keyword_two);
ENTER_QUERY(keyword_one, keyword_two, locking);

Built-in Type: restricted procedure

Enter Query Mode: yes (to redisplay the example record from the last query executed in the block)

Description: The behavior of ENTER_QUERY varies depending on any parameters you supply.

Parameters: no parameters

ENTER_QUERY flushes the current block and puts the form in Enter Query mode. If there are changes to commit, Oracle Forms prompts the operator to commit them during the ENTER_QUERY process.
**Examples:**

```
BEGIN
  Enter_Query;
  /*
   ** Check to see if the record status of the first record
   ** is 'NEW' immediately after returning from enter-query
   ** mode. It should be 'QUERY' if at least one row was
   ** returned.
   */
```

**Restrictions:**

Use the ALL_RECORDS and FOR_UPDATE parameters with caution. Locking and fetching a large number of rows can result in long delays due to the many resources that must be acquired to accomplish the task.

```
keyword_one ENTER_QUERY(ALL_RECORDS) performs the same
actions as ENTER_QUERY except that when
EXECUTE_QUERY is invoked, Oracle Forms fetches all
of the selected records.

keyword_two ENTER_QUERY(FOR_UPDATE) performs the same actions
as ENTER_QUERY except that when EXECUTE_QUERY is
invoked, Oracle Forms attempts to lock all of the
selected records immediately.

keyword_one/
keyword_two ENTER_QUERY(ALL_RECORDS, FOR_UPDATE) performs
the same actions as ENTER_QUERY except that when
EXECUTE_QUERY is invoked, Oracle Forms attempts to
lock all of the selected records immediately and
fetches all of the selected records.

locking Can be set to NO_WAIT anytime that you use the
FOR_UPDATE parameter. When you use
NO_WAIT, Oracle Forms displays a dialog to
notify the operator if a record cannot be reserved
for update immediately.

Without the NO_WAIT parameter, Oracle Forms
keeps trying to obtain a lock without letting the
operator cancel the process.

Use the NO_WAIT parameter only when running
against a data source that supports this
functionality.
```
IF :System.Record_Status = 'NEW' THEN
  Exit_Form(No_Validate);
END IF;
END;

---

**ERASE**

**Syntax:**

\[
\text{ERASE} ( \text{global\_variable\_name} ) ;
\]

**Built-in Type:**

unrestricted procedure

**Enter Query Mode:**

yes

**Description:**

Removes an indicated global variable, so that it no longer exists, and releases the memory associated with the global variable.Globals always allocate 255 bytes of storage. To ensure that performance is not impacted more than necessary, always erase any global variable when it is no longer needed.

**Parameters:**

- \( \text{global\_variable\_name} \)  
  Specifies the name of a valid global variable.

**Example:**

\[
\text{ERASE} ( ' \text{global.var} ' ) ;
\]

---

**ERROR_CODE**

**Syntax:**

\[
\text{ERROR\_CODE} ;
\]

**Built-in Type:**

unrestricted function

**Enter Query Mode:**

yes

**Description:**

Returns the error number of the Oracle Forms error.

**Parameters:**

none

**Example:**

```sql
/*
** Built-in:  ERROR_CODE,ERROR_TEXT,ERROR_TYPE
** Example:   Reword certain FRM error messages by checking
**            the Error_Code in an ON-ERROR trigger
** Trigger:  On-Error
*/
DECLARE
  errnum NUMBER       := ERROR_CODE;
  errtxt VARCHAR2(80) := ERROR_TEXT;
```
ERROR_TEXT

Syntax: ERROR_TEXT;

Built-in Type: unrestricted function

Enter Query Mode: yes

Description: Returns the message text of the Oracle Forms error.

Parameters: none

Usage Notes: You can use this function to test error messages during exception handling subprograms.

Example:

```sql
DECLARE
  errnum NUMBER := ERROR_CODE;
  errtxt VARCHAR2(80) := ERROR_TEXT;
  errtyp VARCHAR2(3) := ERROR_TYPE;
BEGIN
  IF errnum = 40301 THEN
    Message('Your search criteria identified no matches...
     Try Again.');
  ELSIF errnum = 40350 THEN
    Message('Your selection does not correspond to an employee.');
  ELSE
    /*
     ** Print the Normal Message that would have appeared
     **
     ** Default Error Message Text Goes Here
     */
    Message(errtyp||'–'||TO_CHAR(errnum)||': '||errtxt);
    RAISE Form_Trigger_Failure;
  END IF;
END;
```

```sql
errtyp VARCHAR2(3) := ERROR_TYPE;
BEGIN
  IF errnum = 40301 THEN
    Message('Your search criteria identified no matches...
     Try Again.');
  ELSIF errnum = 40350 THEN
    Message('Your selection does not correspond to an employee.');
  ELSE
    /*
     ** Print the Normal Message that would have appeared
     **
     ** Default Error Message Text Goes Here
     */
    Message(errtyp||'–'||TO_CHAR(errnum)||': '||errtxt);
    RAISE Form_Trigger_Failure;
  END IF;
END;
```
ERROR_TYPE

Syntax: ERROR_TYPE;

Built-in Type: unrestricted function

Returns: ERROR_TYPE returns one of the following values for the error message type:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRM</td>
<td>Indicates an Oracle Forms error.</td>
</tr>
<tr>
<td>ORA</td>
<td>Indicates an ORACLE error.</td>
</tr>
</tbody>
</table>

Enter Query Mode: yes

Description: Returns the error message type for the action most recently performed during the current Runform session.

Parameters: none

Usage Notes: You can use this function to do one of the following:

- test the outcome of a user action, such as pressing a key, to determine processing within an On–Error trigger
- test the outcome of a built–in to determine further processing within any trigger

To get the correct results in either type of test, you must perform the test immediately after the action executes, before any other action occurs.

Example:
/*
** Built-in: ERROR_CODE,ERROR_TEXT,ERROR_TYPE
** Example: Reword certain FRM error messages by checking the Error_Code in an ON-ERROR trigger
*/
** Trigger: On-Error
*/

DECLARE
  errnum NUMBER := ERROR_CODE;
  errtxt VARCHAR2(80) := ERROR_TEXT;
  errtyp VARCHAR2(3) := ERROR_TYPE;
BEGIN
  IF errnum = 40107 THEN
    Message('You cannot navigate to this non-displayed item...
    Try again.');
  ELSIF errnum = 40109 THEN
    Message('If you want to leave this block,
    you must first cancel Enter Query mode.');
  ELSE
    /*
    ** Print the Normal Message that would have appeared
    **
    ** Default Error Message Text Goes Here
    */
    Message(errtyp||'-'||TO_CHAR(errnum)||': '||errtxt);
    RAISE Form_Trigger_Failure;
  END IF;
END IF;
END;

EXECUTE_QUERY

Syntax:  
EXECUTE_QUERY;
EXECUTE_QUERY(keyword_one);
EXECUTE_QUERY(keyword_two);
EXECUTE_QUERY(keyword_one, keyword_two);
EXECUTE_QUERY(keyword_one, keyword_two, locking);

Built-in Type:  
restricted procedure

Enter Query Mode:  
yes

Description:  
Flushes the current block, opens a query, and fetches a number of selected records. If there are changes to commit, Oracle Forms prompts the operator to commit them before continuing EXECUTE_QUERY processing.

Parameters:  
no parameters  
EXECUTE_QUERY flushes the current block, opens a query, and fetches a number of selected records.
EXECUTE_QUERY(ALL_RECORDS) performs the same actions as EXECUTE_QUERY except that Oracle Forms fetches all of the selected records.

EXECUTE_QUERY(FOR_UPDATE) performs the same actions as EXECUTE_QUERY except that Oracle Forms attempts to lock all of the selected records immediately.

EXECUTE_QUERY(ALL_RECORDS, FOR_UPDATE) performs the same actions as EXECUTE_QUERY except that Oracle Forms attempts to lock all of the selected records immediately and fetches all of the selected records.

Can be set to NO_WAIT anytime that you use the FOR_UPDATE parameter. When you use NO_WAIT, Oracle Forms displays a dialog to notify the operator if a record cannot be reserved for update immediately.

Without the NO_WAIT parameter, Oracle Forms keeps trying to obtain a lock without letting the operator cancel the process.

Use the NO_WAIT parameter only when running against a data source that supports this functionality.

Oracle Corporation recommends that you use the ALL_RECORDS and FOR_UPDATE parameters with caution. Fetching a large number of rows could cause a long delay. Locking a large number of rows at once requires many resources.

Example:

```sql
DECLARE
    block_before VARCHAR2(80) := :System.Cursor_Block;
BEGIN
    Go_Block('Exceptions_List');
    Execute_Query;
    Go_Block('User_Profile');
    Execute_Query;
    Go_Block('Tasks_Competed');
    Execute_Query;
    Go_Block( block_before );
END;
```
EXECUTE_TRIGGER

Syntax: EXECUTE_TRIGGER(trigger_name);

Built-in Type: restricted procedure (if the user-defined trigger calls any restricted built-in subprograms)

Enter Query Mode: yes

Description: EXECUTE_TRIGGER executes an indicated trigger.

Note: EXECUTE_TRIGGER is not the preferred method for executing a user-named trigger: writing a user-named subprogram is the preferred method.

Parameters: trigger_name Specifies the name of a valid user-named trigger.

Restrictions: Although you can use EXECUTE_TRIGGER to execute a built-in trigger as well as a user-named trigger, this usage is not recommended, because the default fail behavior follows a different rule than when invoked automatically by Oracle Forms as part of default processing. For example, in default processing, if the When-Validate-Item trigger fails, it raises an exception and stops the processing of the form. However, if the When-Validate-Item trigger fails when it is invoked by EXECUTE_TRIGGER, that failure does not stop the processing of the form, but only sets Form_Failure to FALSE on return from the EXECUTE_TRIGGER built-in.

Usage Notes: Because you cannot specify scope for this built-in, Oracle Forms always looks for the trigger starting at the lowest level, then working up.

To execute a built-in associated with a key, use the DO_KEY built-in instead of EXECUTE_TRIGGER. For example, rather than:

```
Execute_Trigger ('KEY-NEXT-ITEM');
```

Use instead:

```
Do_Key('NEXT_ITEM');
```

Example:

```
/**
  ** Built-in: EXECUTE_TRIGGER
  ** Example: Execute a trigger dynamically from the PL/SQL
  ** code of a Menu item, depending on a menu
  ** checkbox.
  */

DECLARE
  Cur_Setting VARCHAR2(5);
  Advanced_Mode BOOLEAN;
```
BEGIN
  /*
   ** Check whether the 'Advanced' menu item under the
   ** 'Preferences' submenu is checked on or not.
   */
  Cur_Setting := Get_Menu_Item_Property
      ('Preferences.Advanced',CHECKED);
  /*
   ** If it is checked on, then Advanced_Mode is boolean
   ** true.
   */
  Advanced_Mode := (Cur_Setting = 'TRUE');
  /*
   ** Run the appropriate trigger from the underlying form
   */
  IF Advanced_Mode THEN
    Execute_Trigger('Launch_Advanced_Help');
  ELSE
    Execute_Trigger('Launch_Beginner_Help');
  END IF;
END;

EXIT_FORM

Syntax:    EXIT_FORM;
EXIT_FORM(commit_mode);
EXIT_FORM(commit_mode, rollback_mode);

Built-in Type: restricted procedure

Enter Query Mode: yes

Description: The behavior of EXIT_FORM depends on the current mode:

- In most contexts, EXIT_FORM navigates “outside” the form. If there are changes in the current form that have not been posted or committed, Oracle Forms prompts the operator to commit before continuing EXIT_FORM processing.

- If the operator is in Enter Query mode, EXIT_FORM navigates out of Enter Query mode, not out of the form.

- During a CALL_INPUT, EXIT_FORM terminates the CALL_INPUT function.
Parameters:  

**commit_mode**

**ASK_COMMIT** Oracle Forms prompts the operator to commit the changes during EXIT_FORM processing.

**DO_COMMIT** Oracle Forms validates the changes, performs a commit, and exits the current form without prompting the operator.

**NO_COMMIT** Oracle Forms validates the changes and exits the current form without performing a commit or prompting the operator.

**NO_VALIDATE** Oracle Forms exits the current form without validating the changes, committing the changes, or prompting the operator.

**rollback_mode**

**TO_SAVEPOINT** Oracle Forms rolls back all uncommitted changes (including posted changes) to the current form’s savepoint.

**FULL_ROLLBACK** Oracle Forms rolls back all uncommitted changes (including posted changes) that were made during the current Runform session. You cannot specify a FULL_ROLLBACK from a form that is running in post–only mode. (Post–only mode can occur when your form issues a call to another form while unposted records exist in the calling form. To prevent losing the locks issued by the calling form, Oracle Forms prevents any commit processing in the called form.)

**NO_ROLLBACK** Oracle Forms exits the current form without rolling back to a savepoint. You can leave the top level form without performing a rollback, which means that you retain the locks across a NEW_FORM operation. These locks can also occur when running Oracle Forms from an external 3GL program. The locks remain in effect when Oracle Forms returns control to the program.

---

**Example:**

```c
/*
** Built-in:  EXIT_FORM
** Example:   Leave the called form, without rolling back the
**            posted changes so they may be posted and
**            committed by the calling form as part of the
**            same transaction.
*/
BEGIN
    Post;
```
/** Form_Status should be 'QUERY' if all records were */
IF :System.Form_Status <> 'QUERY' THEN
    Message('An error prevented the system from posting changes');
    RAISE Form_Trigger_Failure;
END IF;

</*
** By default, Exit_Form asks to commit and performs a
** rollback to savepoint. We've already posted, so we do
** not need to commit, and we don't want the posted changes
** to be rolled back.
*/
Exit_Form(NO_COMMIT, NO_ROLLBACK);
END;

---

**FETCH_RECORDS**

**Syntax:**

FETCH_RECORDS;

**Built-in Type:**

unrestricted procedure

**Enter Query Mode:**

no

**Description:**

When called from an On-Fetch trigger, initiates the default Oracle Forms processing for fetching records that have been identified by SELECT processing.

This built-in is included primarily for applications that will run against a non-ORACLE data source.

**Parameters:**

none

**Example:**

```/*
** Built-in: FETCH_RECORDS
** Example: Perform Oracle Forms record fetch processing during
** query time. Decide whether to use this built-in
** or a user exit based on a global flag setup at
** startup by the form, perhaps based on a
** parameter. The block property RECORDS_TO_FETCH
** allows you to know how many records Oracle Forms
** is expecting.
** Trigger: On-Fetch
*/
DECLARE
    numrecs NUMBER;
```
BEGIN
  /*
  ** Check the global flag we set during form startup
  */
  IF :Global.Using_Transactional_Triggers = 'TRUE' THEN
    /*
    ** How many records is the form expecting us to
    ** fetch?
    */
    numrecs := Get_Block_Property('EMP',RECORDS_TO_FETCH);
    /*
    ** Call user exit to determine if there are any
    ** more records to fetch from its cursor. User Exit
    ** will return failure if there are no more
    ** records to fetch.
    */
    User_Exit('my_fetch block=EMP remaining_records');
    /*
    ** If there ARE more records, then loop thru
    ** and create/populate the proper number of queried
    ** records. If there are no more records, we drop through
    ** and do nothing. Oracle Forms takes this as a signal that
    ** we are done.
    */
    IF Form_Success THEN
      /* Create and Populate 'numrecs' records */
      FOR j IN 1..numrecs LOOP
        Create_Queried_Record;
        /*
        ** User exit returns false if there are no more
        ** records left to fetch. We break out of the
        ** if we've hit the last record.
        */
        User_Exit('my_fetch block=EMP get_next_record');
        IF NOT Form_Success THEN
          EXIT;
        END IF;
      END LOOP;
      END IF;
    END IF;
    /*
    ** Otherwise, do the right thing.
    */
    ELSE
      Fetch_Records;
    END IF;
  END;
**FIND_ALERT**

**Syntax:**
```
FIND_ALERT(alert_name);
```

**Built-in Type:** unrestricted function  

**Returns:** Alert  

**Enter Query Mode:** yes  

**Description:** Searches the list of valid alerts in Oracle Forms. When the given alert is located, the subprogram returns an alert ID. You must return the ID to an appropriately typed variable. Define the variable with a type of Alert.

**Parameters:**
- `alert_name` Specifies the CHAR alert name.

**Example:**
```sql
DECLARE
    al_id     Alert;
    al_button NUMBER;
BEGIN
    al_id := Find_Alert('User_Warning');
    IF Id_Null(al_id) THEN
        Message('User_Warning alert does not exist');
        RAISE Form_Trigger_Failure;
    ELSE
        /*
        ** Show the warning alert
        */
        al_button := Show_Alert(al_id);
        /*
        ** If user pressed OK (button 1) then bring up another
        ** alert to confirm -- button mappings are specified
        ** in the alert design
        */
        IF al_button = ALERT_BUTTON1 THEN
            al_id := Find_Alert('Are_You_Sure');
            IF Id_Null(al_id) THEN
                Message('The alert named: Are you sure? does not exist');
                RAISE Form_Trigger_Failure;
            ELSE
                al_button := Show_Alert(al_id);
            END IF;
        END IF;
    END IF;
END;
```

**Built-in Subprograms**
IF al_button = ALERT_BUTTON2 THEN
   Erase_All_Employee_Records;
END IF;
END IF;
END IF;
END IF;
END;

---

FIND_BLOCK

Syntax: FIND_BLOCK(block_name);

Built-in Type: unrestricted function

Returns: Block

Enter Query Mode: yes

Description: Searches the list of valid blocks and returns a unique block ID. You must define an appropriately typed variable to accept the return value. Define the variable with a type of Block.

Parameters: block_name Specifies the CHAR block name.

Example:

```
/*
** Built-in: FIND_BLOCK
** Example: Return true if a certain blockname exists
*/
FUNCTION Does_Block_Exist( bk_name VARCHAR2 )
RETURN BOOLEAN IS
   bk_id Block;
BEGIN
   /*
   ** Try to locate the block by name
   */
   bk_id := Find_Block( bk_name );
   /*
   ** Return the boolean result of whether we found it.
   ** Finding the block means that its bk_id will NOT be NULL
   */
   RETURN (NOT Id_Null(bk_id));
END;
```
FIND_CANVAS

Syntax: FIND_CANVAS(canvas_name);

Built-in Type: unrestricted function

Returns: Canvas

Enter Query Mode: yes

Description: Searches the list of canvases and returns a canvas ID when it finds a valid canvas with the given name. You must define an appropriately typed variable to accept the return value. Define the variable with a type of Canvas.

Parameters: canvas_name Specifies the CHAR canvas name you gave the canvas when defining it.

Example:
```plaintext
DECLARE
  my_canvas Canvas;
BEGIN
  my_canvas := Find_Canvas('my_canvas');
END;
```

FIND_COLUMN

Syntax: FIND_COLUMN(recordgroup.groupcolumn_name);

Built-in Type: unrestricted function

Returns: GroupColumn

Enter Query Mode: yes

Description: Searches the list of record group columns and returns a groupcolumn ID when it finds a valid column with the given name. You must define an appropriately typed variable to accept the return value. Define the variable with a type of GroupColumn.

Parameters: recordgroup. groupcolumn_name Specifies the fully qualified CHAR record group column name.

Example:
```plaintext
/*
 ** Built-in: FIND_COLUMN
 ** Example: Get column IDs for three columns in a record
```
```
** group before performing multiple Get’s or Set’s 
** of the record group’s column values
*/
PROCEDURE Record_Machine_Stats( mach_number NUMBER, 
   pph NUMBER, 
   temperature NUMBER) IS
   rg_id  RecordGroup;
   col1   GroupColumn;
   col2   GroupColumn;
   col3   GroupColumn;
   row_no NUMBER;
BEGIN
   rg_id := Find_Group('machine');
   col1  := Find_Column('machine.machine_no');
   col2  := Find_Column('machine.parts_per_hour');
   col3  := Find_Column('machine.current_temp');
   /*
   ** Add a new row at the bottom of the 'machine' record 
   ** group, and make a note of what row number we just 
   ** added.
   */
   Add_Group_Row( rg_id, END_OF_GROUP);
   row_no := Get_Group_Row_Count(rg_id);
   Set_Group_Number_Cell(col1, row_no, mach_number);
   Set_Group_Number_Cell(col2, row_no, pph);
   Set_Group_Number_Cell(col3, row_no, temperature);
END;
```

---

** FIND_EDITOR **

** Syntax:**  
FIND_EDITOR(editor_name);

** Built-in Type:**  
unrestricted function

** Returns:**  
Editor

** Enter Query Mode:**  
yes

** Description:**  
Searches the list of editors and returns an editor ID when it finds a valid editor with the given name. You must define an appropriately typed variable to accept the return value. Define the variable with a type of Editor.

** Parameters:**  
editor_name  
Specifies a valid CHAR editor name.

** Example:**  
/*
** Built-in: FIND_EDITOR
*/
** Example: Find and show a user-defined editor  */
DECLARE
    ed_id  Editor;
    status BOOLEAN;
BEGIN
    ed_id := Find_Editor('Happy_Edit_Window');
    IF NOT Id_Null(ed_id) THEN
        Show_Editor(ed_id, NULL, :emp.comments, status);
    ELSE
        Message('Editor "Happy_Edit_Window" not found');
        RAISE Form_Trigger_Failure;
    END IF;
END;

---

** Example: Find a form's Id before inquiring about several of its properties  */
DECLARE
    fm_id  FormModule;
    tmpstr VARCHAR2(80);
BEGIN
    fm_id := Find_Form(:System.Current_Form);
    tmpstr := Get_Form_Property(fm_id,CURSOR_MODE);
    tmpstr := tmpstr||','||Get_Form_Property(fm_id,SAVEPOINT_MODE);
    Message('Form is configured as: '||tmpstr);
END;

---

** FIND_FORM **

Syntax: FIND_FORM(formmodule_name);

Built-in Type: unrestricted function

Returns: FormModule

Enter Query Mode: yes

Description: Searches the list of forms and returns a form module ID when it finds a valid form with the given name. You must define an appropriately typed variable to accept the return value. Define the variable with a type of Formmodule.

Parameters: formmodule_name Specifies a valid CHAR form name.

Example: /* Built-in: FIND_FORM */
** Example: Find a form's Id before inquiring about several of its properties */
DECLARE
    fm_id  FormModule;
    tmpstr VARCHAR2(80);
BEGIN
    fm_id := Find_Form(:System.Current_Form);
    tmpstr := Get_Form_Property(fm_id,CURSOR_MODE);
    tmpstr := tmpstr||','||Get_Form_Property(fm_id,SAVEPOINT_MODE);
    Message('Form is configured as: '||tmpstr);
END;
**FIND_GROUP**

Syntax: `FIND_GROUP(recordgroup_name);`

Built–in Type: unrestricted function

Returns: RecordGroup

Enter Query Mode: yes

Description: Searches the list of record groups and returns a record group ID when it finds a valid group with the given name. You must define an appropriately typed variable to accept the return value. Define the variable with a type of RecordGroup.

Parameters: `recordgroup_name` Specifies the valid CHAR record group name.

Restrictions: Performance of this function depends upon the number of record groups.

Example:

```c
/*
** Built-in: FIND_GROUP
** Example: See CREATE_GROUP and DELETE_GROUP_ROW
*/
```

**FIND_ITEM**

Syntax: `FIND_ITEM(block.item_name);`

Built–in Type: unrestricted function

Returns: Item

Enter Query Mode: yes

Description: Searches the list of items in a given block and returns an item ID when it finds a valid item with the given name. You must define an appropriately typed variable to accept the return value. Define the variable with a type of Item.

Parameters: `block_name` Specifies the fully qualified item name. The data type of the name is CHAR.

Example:

```c
/*
** Built-in: FIND_ITEM
** Example: Find an item's Id before setting several of its properties.
*/
```
FIND_LOV

Syntax: FIND_LOV(LOV_name);

Built-in Type: unrestricted function

Returns: LOV

Enter Query Mode: yes

Description: Searches the list of LOVs and returns an LOV ID when it finds a valid LOV with the given name. You must define an appropriately typed variable to accept the return value. Define the variable with a type of LOV.

Parameters: LOV_name Specifies the valid CHAR LOV name.

Example:
/*
** Built-in: FIND_LOV
** Example: Determine if an LOV exists before showing it.
*/
DECLARE
lv_id LOV;
status BOOLEAN;
BEGIN
lv_id := Find_LOV('My_Shared_LOV');
/*
** If the 'My_Shared_LOV' is not part of the current form,
** then use the 'My_Private_LOV' instead.
*/
IF Id_Null(lv_id) THEN
  lv_id := Find_LOV('My_Private_LOV');
END IF;
status := Show_LOV(lv_id,10,20);
END;

---

**FIND_MENU_ITEM**

**Syntax:**

```
FIND_MENU_ITEM(menu_name.menu_item_name);
```

**Built-in Type:**

unrestricted function

**Returns:**

MenuItem

**Enter Query Mode:**

yes

**Description:**

Searches the list of menu items and returns a menu item ID when it finds a valid menu item with the given name. You must define an appropriately typed variable to accept the return value. Define the variable with a type of MenuItem.

**Parameters:**

- `menu_name.menu_item_name` Specifies a valid fully-qualified CHAR menu item name.

**Example:**

```sql
PROCEDURE Toggle_AutoCommit_Mode IS
  mi_id MenuItem;
  val VARCHAR2(10);
  BEGIN
    mi_id := Find_Menu_Item('Preferences.AutoCommit');
    /*
    ** Determine the current checked state of the AutoCommit menu checkbox item
    */
    val := Get_Menu_Item_Property(mi_id,CHECKED);
    /*
    ** Toggle the checked state
    */
    IF val = 'TRUE' THEN
      Set_Menu_Item_Property(mi_id,CHECKED,PROPERTY_FALSE);
    ELSE
```
FIND_RELATION

**Syntax:**
FIND_RELATION(relation_name);

**Built-in Type:**
unrestricted function

**Returns:**
Relation

**Enter Query Mode:**
yes

**Description:**
Searches the list of relations and returns a relation ID when it finds a valid relation with the given name. You must define an appropriately typed variable to accept the return value. Define the variable with a type of Relation.

**Parameters:**
relation_name
Specifies a valid CHAR relation name.

**Example:**

```sql
FUNCTION Detail_of( Relation_Name VARCHAR2 ) RETURN VARCHAR2 IS
  rl_id Relation;
  BEGIN
    rl_id := Find_Relation( Relation_Name );
    /*
    ** Signal error if relation does not exist
    */
    IF Id_Null(rl_id) THEN
      Message('Relation ''|Relation_Name|'' does not exist.'); RAISE Form_Trigger_Failure;
    ELSE
      RETURN Get_Relation_Property(rl_id,DETAIL_NAME);
    END IF;
  END;
```

```sql
ru = Find_Relation('example_relation');
result = Get_Relation_Property(ru, 'example_property');
```
FIND_TIMER

Syntax: FIND_TIMER(timer_name);

Built-in Type: unrestricted function

Returns: Timer

Enter Query Mode: yes

Description: Searches the list of timers and returns a timer ID when it finds a valid timer with the given name. You must define an appropriately typed variable to accept the return value. Define the variable with a type of Timer.

Parameters: timer_name Specifies a valid CHAR timer name.

Example:

```/*
** Built-in: FIND_TIMER
** Example: If the timer exists, reset it. Otherwise create it.
*/
PROCEDURE Reset_Timer_Interval( Timer_Name VARCHAR2,
                                    Timer_Intv NUMBER ) IS
    tm_id       Timer;
    tm_interval NUMBER;
BEGIN
    /*
    ** User gives the interval in seconds, the timer subprograms
    ** expect milliseconds
    */
    tm_interval := 1000 * Timer_Intv;
    /* Lookup the timer by name */
    tm_id := Find_Timer(Timer_Name);
    /* If timer does not exist, create it */
    IF Id_Null(tm_id) THEN
        tm_id := Create_Timer(Timer_Name,tm_interval,NO_REPEAT);
    /*
    ** Otherwise, just restart the timer with the new interval
    */
    ELSE
        Set_Timer(tm_id,tm_interval,NO_REPEAT);
    END IF;
END;```
FIND_VIEW

Syntax: FIND_VIEW(viewcanvas_name);

Built-in Type: unrestricted function

Returns: ViewPort

Enter Query Mode: yes

Description: Searches the list of canvas-views and returns a view ID when it finds a valid canvas-view with the given name. You must define an appropriately typed variable to accept the return value. Define the variable with a type of ViewPort.

Parameters: viewcanvas_name Specifies the CHAR name of the canvas-view.

Example:

```/*
 ** Built-in:  FIND_VIEW
 ** Example:  Change the visual attribute and display position
 **          of a stacked view before making it visible to
 **          the user.
 */
DECLARE
vw_id ViewPort;
BEGIN
vw_id := Find_View('Sales_Summary');
Set_Canvas_Property('Sales_Summary', VISUAL_ATTRIBUTE, 'Salmon_On_Yellow');
Set_View_Property(vw_id, DISPLAY_X_POS, 30);
Set_View_Property(vw_id, DISPLAY_Y_POS, 5);
Set_View_Property(vw_id, VISIBLE, PROPERTY_TRUE);
END;```
**FIND_WINDOW**

Syntax:  
FIND_WINDOW(window_name);

Built-in Type:  
unrestricted function

Returns:  
Window

Enter Query Mode:  
yes

Description:  
Searches the list of windows and returns a window ID when it finds a valid window with the given name. You must define an appropriately typed variable to accept the return value. Define the variable with a type of Window.

Parameters:  
*window_name*  
Specifies the valid CHAR window name.

Example:

```sql
/*
** Built-in:  FIND_WINDOW
** Example:   Anchor the upper left corner of window2 at the bottom right corner of window1.
*/
PROCEDURE Anchor_Bottom_Right( Window2 VARCHAR2, Window1 VARCHAR2) IS
  wn_id1 Window;
  wn_id2 Window;
  x      NUMBER;
  y      NUMBER;
  w      NUMBER;
  h      NUMBER;
BEGIN
  /*   ** Find Window1 and get its (x,y) position, width, and height.   */
  wn_id1 := Find_Window(Window1);
  x      := Get_Window_Property(wn_id1,X_POS);
  y      := Get_Window_Property(wn_id1,Y_POS);
  w      := Get_Window_Property(wn_id1,WIDTH);
  h      := Get_Window_Property(wn_id1,HEIGHT);
  /*   ** Anchor Window2 at (x+w,y+h)   */
  wn_id2 := Find_Window(Window2);
  Set_Window_Property(wn_id2,X_POS, x+w );
  Set_Window_Property(wn_id2,Y_POS, y+h );
END;
```
FIRST_RECORD

Syntax:  FIRST_RECORD;

Built-in Type:  restricted procedure

Enter Query Mode:  no

Description:  Navigates to the first record in the block’s list of records.

Parameters:  none

Example:  /* Built-in: FIRST_RECORD
** Example:  Have a button toggle between the first and last
**           records in a block.
** Trigger:  When-Button-Pressed
*/
BEGIN
  /*
   ** If we’re not at the bottom, then go to the last record
   */
  IF :System.Last_Record <> 'TRUE' THEN
    Last_Record;
  ELSE
    First_Record;
  END IF;
END;

FORM_FAILURE

Syntax:  FORM_FAILURE;

Built-in Type:  unrestricted function

Returns:  BOOLEAN

Enter Query Mode:  yes

Description:  Returns a value that indicates the outcome of the action most recently
              performed during the current Runform session.
If no action has executed in the current Runform session, FORM_FAILURE returns FALSE.

Use FORM_FAILURE to test the outcome of a built-in to determine further processing within any trigger. To get the correct results, you must perform the test immediately after the action executes. That is, another action should not occur prior to the test.

Note: "Another action" includes both built-ins and PL/SQL assignment statements. If another action occurs, FORM_FAILURE may not reflect the status of the built-in you are testing, but of the other, more recently executed action. A more accurate technique is, for example, when performing a COMMIT_FORM, to check that the SYSTEM.FORM_STATUS variable is set to 'QUERY' after the operation is done.

Parameters: none

Example:
/*
** Built-in: FORM_FAILURE
** Example: Determine if the most recently executed built-in
** failed.
*/
BEGIN
  GO_BLOCK('Success_Factor');
/*
** If some validation failed and prevented us from leaving
** the current block, then stop executing this trigger.
**
** Generally it is recommended to test
** IF NOT Form_Success THEN ...
** Rather than explicitly testing for FORM_FAILURE
*/
  IF Form_Failure THEN
    RAISE Form_Trigger_Failure;
  END IF;
END;
FORM_FATAL

Syntax:  FORM_FATAL;

Built-in Type:  unrestricted function

Return Type:  BOOLEAN

Enter Query Mode:  yes

Description:  Returns the outcome of the action most recently performed during the current Runform session.

<table>
<thead>
<tr>
<th>Outcome</th>
<th>Returned Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>FALSE</td>
</tr>
<tr>
<td>failure</td>
<td>FALSE</td>
</tr>
<tr>
<td>fatal error</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

Use FORM_FATAL to test the outcome of a built-in to determine further processing within any trigger. To get the correct results, you must perform the test immediately after the action executes. That is, another action should not occur prior to the test.

Note: "Another action" includes both built-ins and PL/SQL assignment statements. If another action occurs, FORM_FATAL may not reflect the status of the built-in you are testing, but of the other, more recently executed action. A more accurate technique is, for example, when performing a COMMIT_FORM, to check that the SYSTEM.FORM_STATUS variable is set to 'QUERY' after the operation is done.

Parameters:  none

Example:  
/*
 ** Built-in:  FORM_FATAL
 ** Example:  Check whether the most-recently executed built-in
 ** had a fatal error.
 */
BEGIN
  User_Exit('Calculate_Line_Integral control.start control.stop');
  /*
   ** If the user exit code returned a fatal error, print a
   ** message and stop executing this trigger.
   **
   ** Generally it is recommended to test   **
   ** IF NOT FORM_SUCCESS THEN ...   **
   ** Rather than explicitly testing for FORM_FATAL
   */
IF Form_Fatal THEN
    Message('Cannot calculate the Line Integral due to internal error. ');
    RAISE Form_Trigger_Failure;
END IF;
END;

FORM_SUCCESS

Syntax:       FORM_SUCCESS;
Built-in Type: unrestricted function
Return Type:  BOOLEAN
Enter Query Mode: yes
Description:  Returns the outcome of the action most recently performed during the current Runform session.

<table>
<thead>
<tr>
<th>Outcome</th>
<th>Returned Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>TRUE</td>
</tr>
<tr>
<td>failure</td>
<td>FALSE</td>
</tr>
<tr>
<td>fatal error</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Use FORM_SUCCESS to test the outcome of a built-in to determine further processing within any trigger. To get the correct results, you must perform the test immediately after the action executes. That is, another action should not occur prior to the test. “Another action” includes both built-ins and PL/SQL assignment statements. If another action occurs, FORM_SUCCESS may not reflect the status of the built-in you are testing, but of the other, more recently executed action.

Note: FORM_SUCCESS should not be used to test whether a COMMIT_FORM or POST built-in has succeeded. Because COMMIT_FORM may cause many other triggers to fire, when you evaluate FORM_SUCCESS it may not reflect the status of COMMIT_FORM but of some other, more recently executed built-in. A more accurate technique is to check that the SYSTEM.FORM_STATUS variable is set to ‘QUERY’ after the operation is done.

Parameters:  none
Example:
```java
/*
** Built-in: FORM_SUCCESS
** Example: Check whether the most-recently executed built-in
** succeeded.
*/
BEGIN
  /*
  ** Force validation to occur
  */
  Enter;
  /*
  ** If the validation succeeded, then Commit the data.
  */
  IF Form_Success THEN
    Commit;
    IF :System.Form_Status <> 'QUERY' THEN
      Message('Error prevented Commit');
      RAISE Form_Trigger_Failure;
    END IF;
  END IF;
END;
```

**FORMS_DDL**

**Syntax:**

FORMS_DDL(`statement`);

**Built-in Type:**

unrestricted function

**Enter Query Mode:**

yes

**Description:**

Issues dynamic SQL statements at runtime, including server-side PL/SQL and DDL.

**Note:**

All DDL operations issue an implicit COMMIT and will end the current transaction without allowing Oracle Forms to process any pending changes.

**Parameters:**

- `statement`:
  - Any string expression up to 32K:
    - a literal
    - an expression or a variable representing the text of a block of dynamically created PL/SQL code
    - a DML statement or
```
• a DDL statement

Restrictions:  The statement you pass to FORMS_DDL may not contain bind variable references in the string, but the values of bind variables can be concatenated into the string before passing the result to FORMS_DDL. For example, this statement is not valid:

Forms_DDL ('Begin Update_Employee (:emp.empno); End;');

However, this statement is valid, and would have the desired effect:

Forms_DDL ('Begin Update_Employee ('||TO_CHAR(:emp.empno) ||');End;');

However, you could also call a stored procedure directly, using Oracle7’s shared SQL area over multiple executions with different values for emp.empno:

Update_Employee (:emp.empno);

SQL statements and PL/SQL blocks executed using FORMS_DDL cannot return results to Oracle Forms directly. (See Example 4.)

In addition, some DDL operations cannot be performed using FORMS_DDL, such as dropping a table or database link, if Oracle Forms is holding a cursor open against the object being operated upon.

Usage Notes:  Commit (or roll back) all pending changes before you issue the FORMS_DDL command. All DDL operations issue an implicit COMMIT and will end the current transaction without allowing Oracle Forms to process any pending changes, as well as losing any locks Forms may have acquired.

Some supplied stored procedures issue COMMIT or ROLLBACK commands as part of their logic. Make sure all pending changes in the form are committed or rolled back before you call those built-ins. Use the SYSTEM.FORM_STATUS variable to check whether there are pending changes in the current form before you issue the FORMS_DDL command. (See Example 4.)

If you use FORMS_DDL to execute a valid PL/SQL block:

• Use semicolons where appropriate.
• Enclose the PL/SQL block in a valid BEGIN/END block structure.
• Do not end the PL/SQL block with a slash.
• Line breaks, while permitted, are not required.

If you use FORMS_DDL to execute a single DML or DDL statement:
• Omit the trailing semicolon to avoid an invalid character error.

To check whether the statement issued using FORMS_DDL executed correctly, use the FORM_SUCCESS or FORM_FAILURE Boolean functions. If the statement did not execute correctly, check the error code and error text using DBMS_ERROR_CODE and DBMS_ERROR_TEXT. Note that the values of DBMS_ERROR_CODE and DBMS_ERROR_TEXT are not automatically reset following successful execution, so their values should only be examined after an error has been detected by a call to FORM_SUCCESS or FORM_FAILURE.

Example 1:
/*
 ** Built-in:  FORMS_DDL
 ** Example:   The expression can be a string literal.
 */
BEGIN
  Forms_DDL('create table temp(n NUMBER)');
  IF NOT Form_Success THEN
    Message ('Table Creation Failed');
  ELSE
    Message ('Table Created');
  END IF;
END;

Example 2:
/*
 ** Built-in:  FORMS_DDL
 ** Example:   The string can be an expression or variable.
 **            Create a table with n Number columns.
 **            TEMP(COL1, COL2, ..., COLn).
 */
PROCEDURE Create_N_Column_Number_Table (n NUMBER) IS
  my_stmt VARCHAR2(2000);
BEGIN
  my_stmt := 'create table temp(n NUMBER)';
  FOR I in 2..N LOOP
    my_stmt := my_stmt||',COL'||TO_CHAR(i)||' NUMBER';
  END LOOP;
  my_stmt := my_stmt||')';
  /*
  ** Now, create the table...
  */
  Forms_DDL(my_stmt);
  IF NOT Form_Success THEN
    Message ('Table Creation Failed');
  ELSE
    Message ('Table Created');
  END IF;
END;
Example 3:

```sql
/*
** Built-in: FORMS_DDL
** Example: The statement parameter can be a block
** of dynamically created PL/SQL code.
*/
DECLARE
    procname VARCHAR2(30);
BEGIN
    IF :global.flag = 'TRUE' THEN
        procname := 'Assign_New_Employer';
    ELSE
        procname := 'Update_New_Employer';
    END IF;
    Forms_DDL('Begin '|| procname ||'; End;');
    IF NOT Form_Success THEN
        Message ('Employee Maintenance Failed');
    ELSE
        Message ('Employee Maintenance Successful');
    END IF;
END;
```

Example 4:

```sql
/*
** Built-in: FORMS_DDL
** Example: Issue the SQL statement passed in as an argument,
** and return a number representing the outcome of
** executing the SQL statement.
** A result of zero represents success.
*/
FUNCTION Do_Sql (stmt VARCHAR2, check_for_locks BOOLEAN := TRUE) RETURN NUMBER IS
    SQL_SUCCESS CONSTANT NUMBER := 0;
BEGIN
    IF stmt IS NULL THEN
        Message ('DO_SQL: Passed a null statement.');
        RETURN SQL_SUCCESS;
    END IF;
    IF Check_For_Locks AND :System.Form_Status = 'CHANGED' THEN
        Message ('DO_SQL: Form has outstanding locks pending.');
        RETURN SQL_SUCCESS;
    END IF;
    Forms_DDL(stmt);
    IF Form_Success THEN
        RETURN SQL_SUCCESS;
    ELSE
        RETURN Dbms_Error_Code;
    END IF;
END;
```
FORMS_OLE.ACTIVATE_SERVER

Syntax:  
FORMS_OLE.ACTIVATE_SERVER(item_id);
FORMS_OLE.ACTIVATE_SERVER(item_name);

Built-in Type:  
unrestricted procedure

Enter Query Mode:  
no

Description:  
Activates an OLE server associated with an OLE container and prepares the OLE server to receive OLE automation events from the OLE container.

Parameters:  

- **item_id**  
  Specifies the unique ID that Oracle Forms assigns to the item when created. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is Item.

- **item_name**  
  Specifies the name of the object created at design time. The data type of the name is CHAR string.

Usage Notes:  
- The OLE container must contain an OLE object and the OLE Server must be available for activation.
- If the OLE In-place Activation property is True and the OLE object is embedded, Forms_OLE.Activate_Server starts up OLE in-place activation. When the OLE In-place Activation property is False, Forms_OLE.Activate_Server starts up OLE external activation.

Restrictions:  
Valid only on Microsoft Windows and Macintosh.

Example:  
/*
 ** Built-in: FORMS_OLE.ACTIVATE_SERVER
 ** Example: Activates the OLE server associated with the object
 ** in the OLE container.
 ** Trigger: When-Button-Pressed
 */
DECLARE
  item_id ITEM;
  item_name VARCHAR(25) := 'OLEITM';
BEGIN
  item_id := Find_Item(item_name);
  IF Id_Null(item_id) THEN
    message('No such item: '||item_name);
  ELSE
    Forms_OLE.Activate_Server(item_id);
  END IF;
END;
FORMS_OLE.CLOSE_SERVER

Syntax: 
FORMS_OLE.CLOSE_SERVER(item_id);
FORMS_OLE.CLOSE_SERVER(item_name);

Built-in Type: unrestricted procedure

Enter Query Mode: no

Description: Deactivates the OLE server associated with an OLE container. 
Terminates the connection between an OLE server and the OLE container.

Parameters: 
item_id 
Specifies the unique ID that Oracle Forms assigns to the item when created. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is Item.

item_name 
Specifies the name of the object created at design time. The data type of the name is CHAR string.

Restrictions: Valid only on Microsoft Windows and Macintosh.

Example: 
/*
** Built-in: FORMS_OLE.CLOSE_SERVER
** Example: Deactivates the OLE server associated with the object
** in the OLE container.
** Trigger: When-Button-Pressed
*/
DECLARE
    item_id ITEM;
    item_name VARCHAR(25) := 'OLEITM';
BEGIN
    item_id := Find_Item(item_name);
    IF Id_Null(item_id) THEN
        message('No such item: '||item_name);
    ELSE
        Forms_OLE.Close_Server(item_id);
    END IF;
END;
FORMS_OLE.EXEC_VERB

Syntax: 

FORMS_OLE.EXEC_VERB(item_id, verb_index);
FORMS_OLE.EXEC_VERB(item_id, verb_name);
FORMS_OLE.EXEC_VERB(item_name, verb_index);
FORMS_OLE.EXEC_VERB(item_name, verb_name);

Built-in Type: unrestricted procedure

Enter Query Mode: no

Description: Causes the OLE server to execute the verb identified by the verb name or the verb index. An OLE verb specifies the action that you can perform on an OLE object.

Parameters:

- item_id: Specifies the unique ID that Oracle Forms assigns to the item when created. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is Item.
- item_name: Specifies the name of the object created at design time. The data type of the name is CHAR string.
- verb_index: Specifies the numeric index of a verb. Use the Forms_OLE.Find_OLE_Verb built-in to obtain this value. The data type of index is CHAR string.
- verb_name: Specifies the name of a verb. Use the Forms_OLE.Get_Verb_Name built-in to obtain this value. The data type of the name is CHAR char.

Restrictions: Valid only on Microsoft Windows and Macintosh.

Example:

```sql
DECLARE
    item_id ITEM;
    item_name VARCHAR(25) := 'OLEITM';
    verb_cnt_str VARCHAR(20);
    verb_cnt NUMBER;
    verb_name VARCHAR(20);
    loop_cnt NUMBER;
BEGIN
    item_id := Find_Item(item_name);
    IF Id_Null(item_id) THEN
        message('No such item: ' || item_name);
    END IF;
END;
```

ELSE
    verb_cnt_str := Forms_OLE.Get_Verb_Count(item_id);
    verb_cnt := TO_NUMBER(verb_cnt_str);
    FOR loop_cntr in 1..verb_cnt LOOP
        verb_name := Forms_OLE.Get_Verb_Name(item_id, loop_cntr);
        IF verb_name = 'Edit' THEN
            Forms_OLE.Exec_Verb(item_id, verb_name);
        END IF;
    END LOOP;
END IF;
END;

---

**FORMS_OLE.FIND_OLE_VERB**

**Syntax:**

```
FORMS_OLE.FIND_OLE_VERB(item_id, verb_name);
FORMS_OLE.FIND_OLE_VERB(item_name, verb_name);
```

**Returns:** CHAR

**Built-in Type:** unrestricted function

**Enter Query Mode:** no

**Description:** Returns an OLE verb index. An OLE verb specifies the action that you can perform on an OLE object, and each OLE verb has a corresponding OLE verb index. The OLE verb index is returned as a CHAR string and must be converted to NUMBER when used in FORMS_OLE.EXE_VERB. You must define an appropriately typed variable to accept the return value.

**Parameters:**

- **item_id**
  Specifies the unique ID that Oracle Forms assigns to the item when created. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is Item.

- **item_name**
  Specifies the name of the object created at design time. The data type of the name is CHAR string.

- **verb_name**
  Specifies the name of an OLE verb. An OLE verb specifies the action that you can perform on an OLE object. Use the Forms_OLE.Get_Verb_Name built-in to obtain this value. The data type of the name is CHAR string.

**Restrictions:** Valid only on Microsoft Windows and Macintosh.
Example:

```java
/*
** Built-in: FORMS_OLE.EXEC_VERB
** Example: Finds an OLE verb index for use with the
**           Forms_OLE.Exec_Verb built-in.
** ** Trigger: When-Button-Pressed
*/
DECLARE
    item_id ITEM;
    item_name VARCHAR(25) := 'OLEITM';
    verb_index_str VARCHAR(20);
    verb_index NUMBER;
BEGIN
    item_id := Find_Item(item_name);
    IF Id_Null(item_id) THEN
        message('No such item: '||item_name);
    ELSE
        verb_index_str := Forms_OLE.Find_OLE_Verb(item_id,'Edit');
        verb_index := TO_NUMBER(verb_index_str);
        Forms_OLE.Exec_Verb(item_id,verb_index);
    END IF;
END;
```

**FORMS_OLE.GET_INTERFACE_POINTER**

| Syntax           | FORMS_OLE.GET_INTERFACE_POINTER(item_id);
|------------------| FORMS_OLE.GET_INTERFACE_POINTER(item_name); |
| Returns          | PLS_INTEGER |
| Built-in Type    | unrestricted function |
| Enter Query Mode | no |
| Description      | Returns a handle to an OLE2 automation object. |
| Parameters       | item_id |
|                  | item_name |

Specifies the unique ID that Oracle Forms assigns to the item when created. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is Item.

Specifies the name of the object created at design time. The data type of the name is CHAR string.

Restrictions: Valid only on Microsoft Windows and Macintosh.

Example:

```java
/*
** Built-in: FORMS_OLE.GET_INTERFACE_POINTER
*/
```
** Example: Finds a handle to an OLE object */
FUNCTION HandleMap(MapName VARCHAR2) RETURN OLE2.obj_type is
BEGIN
   RETURN(Forms_OLE.Get_interface_Pointer(MapName));
END;

FORMS_OLE.GET_VERB_COUNT

Syntax:  
FORMS_OLE.GET_VERB_COUNT(item_id);
FORMS_OLE.GET_VERB_COUNT(item_name);

Returns:  CHAR

Built-in Type:  unrestricted function

Enter Query Mode:  no

Description:  Returns the number of verbs that an OLE server recognizes. An OLE verb specifies the action that you can perform on an OLE object, and the number of verbs available depends on the OLE server. The number of verbs is returned as a CHAR string and must be converted to NUMBER for use in determining the verb index and verb name for each verb. You must define an appropriately typed variable to accept the return value.

Parameters:  

item_id  Specifies the unique ID that Oracle Forms assigns to the item when created. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is Item.

item_name  Specifies the name of the object created at design time. The data type of the name is CHAR string.

Restrictions:  Valid only on Microsoft Windows and Macintosh.

Example:  
/*
** Built-in: FORMS_OLE.GET_VERB_COUNT
** Example: Obtains the number of verbs that the OLE server issues and recognizes when executed from the OLE container.
** Trigger: When-Button-Pressed
*/
DECLARE
   item_id ITEM;
   item_name VARCHAR(25) := 'OLEITM';
   verb_cnt_str VARCHAR(20);
BEGIN
    item_id := Find_Item(item_name);
    IF Id_Null(item_id) THEN
        message('No such item: ' || item_name);
    ELSE
        verb_cnt_str := Forms_OLE.Get_Verb_Count(item_id);
        verb_cnt := TO_NUMBER(verb_cnt_str);
        FOR loop_cntr in 1..verb_cnt LOOP
            verb_name := Forms_OLE.Get_Verb_Name(item_id, loop_cntr);
            IF verb_name = 'Edit' THEN
                Forms_OLE.Exec_Verb(item_id, verb_name);
            END IF;
        END LOOP;
    END IF;
END;

---

**FORMS_OLE.GET_VERB_NAME**

**Syntax:**

```
FORMS_OLE.GET_VERB_NAME(item_id, verb_index);
FORMS_OLE.GET_VERB_NAME(item_name, verb_index);
```

**Returns:** CHAR

**Built-in Type:** unrestricted function

**Enter Query Mode:** no

**Description:** Returns the name of the verb that is associated with the given verb index. An OLE verb specifies the action that you can perform on an OLE object, and each OLE verb has a corresponding OLE verb index. You must define an appropriately typed variable to accept the return value.

**Parameters:**

- **item_id**
  Specifies the unique ID that Oracle Forms assigns to the item when created. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is Item.

- **item_name**
  Specifies the name of the object created at design time. The data type of the name is CHAR.
verb_index Specifies the numeric index of a verb. Use the FORMS_OLE.FIND_OLE_VERB built-in to obtain this value. The data type of index is CHAR.

Restrictions: Valid only on Microsoft Windows and Macintosh.

Example: /*
** Built-in: FORMS_OLE.GET_VERB_COUNT
** Example: See FORMS_OLE.EXEC_VERB and FORMS_OLE.GET_VERB_COUNT
*/

FORMS_OLE.INITIALIZE_CONTAINER

Syntax: FORMS_OLE.INITIALIZE_CONTAINER(item_id, file_name);
FORMS_OLE.INITIALIZE_CONTAINER(item_name, file_name);

Built-in Type: unrestricted procedure

Enter Query Mode: no

Description: Inserts an OLE object from a server-compatible file into an OLE container.

Parameters: item_id Specifies the unique ID that Oracle Forms assigns to the item when created. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is Item.

item_name Specifies the name of the object created at design time. The data type of the name is CHAR string.

file_name Specifies the name of the file containing the object for insertion into an OLE container. Include the path of the file location.

Restrictions: Valid only on Microsoft Windows and Macintosh.

Example: /*
** Built-in: FORMS_OLE.INITIALIZE_CONTAINER
** Example: Initializes an OLE container by inserting an object
** from a specified file into an OLE container.
** Trigger: When-Button-Pressed
*/
DECLARE
  item_id ITEM;
  item_name VARCHAR(25) := 'OLEITM';
BEGIN
FORMS_OLE.SERVER_ACTIVE

Syntax:
FORMS_OLE.SERVER_ACTIVE(item_id);
FORMS_OLE.SERVER_ACTIVE(item_name);

Returns: BOOLEAN

Built-in Type: unrestricted function

Enter Query Mode: no

Description: Indicates whether or not the server associated with a given container is running:
- Returns TRUE if the OLE server is running.
- Returns FALSE if the OLE server is not running.
You must define an appropriately typed variable to accept the return value.

Parameters: 

item_id: Specifies the unique ID that Oracle Forms assigns to the item when created. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is Item.

item_name: Specifies the name of the object created at design time. The data type of the name is CHAR string.

Restrictions: Valid only on Microsoft Windows and Macintosh.

Example:
/*
** Built-in: FORMS_OLE.SERVER_ACTIVE
** Example: Checks to see if the OLE server is active.
** Trigger: When-Button-Pressed
*/
DECLARE
    item_id ITEM;
    item_name VARCHAR(25) := 'OLEITM';
    active_serv BOOLEAN;

item_id := Find_Item(item_name);
IF Id_Null(item_id) THEN
    message('No such item: ' || item_name);
ELSE
    Forms_OLE.Initialize_Container(item_id,'c:\OLE\oleobj.xls');
END IF;
END;
BEGIN
    item_id := Find_Item(item_name);
    IF Id_Null(item_id) THEN
        message('No such item: ' || item_name);
    ELSE
        active_serv := Forms_OLE.Server_Active(item_id);
        IF active_serv = FALSE THEN
            Forms_OLE.Activate_Server(item_id);
        END IF;
    END IF;
END;

GENERATE_SEQUENCE_NUMBER

Syntax: GENERATE_SEQUENCE_NUMBER;

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Initiates the default Oracle Forms processing for generating a unique sequence number when a record is created. When a sequence object is defined in the database, you can reference it as a default value for an item by setting the Default property to SEQUENCE.my_seq.NEXTVAL. By default, Oracle Forms gets the next value from the sequence whenever a record is created. When you are connecting to a non–ORACLE data source, you can include a call to this built–in in the On–Sequence–Number trigger. Refer to Chapter 2, ”Triggers” for more information.

Parameters: none

Restrictions: Valid only in an On–Sequence–Number trigger.

Example:
/*
** Built-in: GENERATE_SEQUENCE_NUMBER
** Example: Perform Oracle Forms standard sequence number processing based on a global flag setup at startup by the form, perhaps based on a parameter.
** Trigger: On-Sequence-Number
*/
BEGIN
    /*
    ** Check the global flag we setup at form startup
    */
IF :Global.Using_Transactional_Triggers = 'TRUE' THEN
    User_Exit('my_seqnum seq=EMPNO_SEQ');
    /*
    ** Otherwise, do the right thing.
    */
    ELSE
    Generate_Sequence_Number;
    END IF;
END;

GET_APPLICATION_PROPERTY

Syntax: GET_APPLICATION_PROPERTY(property);

Built-in Type: unrestricted function

Returns: CHAR

Enter Query Mode: yes

Description: The GET_APPLICATION_PROPERTY built-in returns information about the current Oracle Forms application. You must call this built-in once for each value you want to retrieve.

Parameters: property

Specify one of the following constants to return information about your application:

APPLICATION_INSTANCE Returns the name of an instance handle. Only applies to the Microsoft Windows platform. For all other platforms, Oracle Forms returns NULL.

CALLING_FORM Returns the name of the calling form if the current form was invoked by the CALL_FORM built-in. If the current form is not a called form, Oracle Forms returns NULL.

CONNECT_STRING Returns the database connect string of the current operator. If the current operator does not have a connect string, Oracle Forms returns NULL.

CURRENT_FORM Returns the .FMX file name of the form currently being executed.
CURRENT_FORM_NAME Returns the name of the current form as indicated by the form module Name property.

CURSOR_STYLE Returns the name of the current cursor style property. Valid CHAR return values are BUSY, CROSSHAIR, DEFAULT, HELP, and INSERTION.

DATASOURCE Returns the name of the database that is currently in use. Valid return values are NULL, ORACLE, DB2, NONSTOP, TERADATA, NCR/3600/NCR/3700, and SQLSERVER. This call returns the database name only for connections established by Oracle Forms, not for connections established by On–Logon triggers.

DISPLAY_HEIGHT Returns the height of the display. The size of each unit depends on how you defined the Coordinate System property for the form module.

DISPLAY_WIDTH Returns the width of the display. The size of each unit depends on how you defined the Coordinate System property for the form module.

OPERATING_SYSTEM Returns the name of the operating system that is currently in use. Valid return values are MSWINDOWS, UNIX, SunOS, MACINTOSH, VMS, and HP–UX.

PASSWORD Returns the password of the current operator.

SAVEPOINT_NAME Returns the name of the last savepoint Oracle Forms has issued. This call is valid only from an On–Savepoint or On–Rollback trigger. It is included primarily for developers who are using transactional triggers to access a non–ORACLE data source.

TIMER_NAME Returns the name of the most recently expired timer. Oracle Forms returns NULL in response to this constant if there is no timer.

USER_INTERFACE Returns the name of the user interface that is currently in use. Valid return
values are MOTIF, MACINTOSH, MSWINDOWS, PM, CHARMODE, BLOCKMODE, X, UNKNOWN.

**USER_NLS_CHARACTER_SET** Returns the current value of the character set portion only of the NLS_LANG environment variable defined for the form operator.

**USER_NLS_LANG** Returns the complete current value of the NLS_LANG environment variable defined for the form operator, for national language support. USER_NLS_LANG is the equivalent of concatenating USER_NLS_LANGUAGE, USER_NLS_TERRITORY, and USER_NLS_CHARACTER_SET.

**USER_NLS_LANGUAGE** Returns the current value of the language portion only of the NLS_LANG environment variable defined for the form operator.

**USER_NLS_TERRITORY** Returns the current value of the territory portion only of the NLS_LANG environment variable defined for the form operator.

**USERNAME** Returns the name of the current operator.

**Restrictions:** To retrieve the timer name of the most recently executed timer, you must initiate a call to GET_APPLICATIONPROPERTY from within a When–Timer–Expired trigger. Otherwise, the results of the built-in are undefined.

**Usage Notes:** To request a complete logon, including an appended connect string, use the Username, Password, and Connect_String properties. For instance, assume that the user has initiated an Microsoft Windows Runform session specifying the following connect string:

```plaintext
f40run my_form scott/tiger@t:corp:DB1
```

Oracle Forms returns the following string as the result of a call to GET_APPLICATION_PROPERTY(USERNAME):

```plaintext
scott
```

Oracle Forms returns the following string as the result of a call to GET_APPLICATION_PROPERTY(PASSWORD):

```plaintext

```
Example 1:

**Built-in:** GET_APPLICATIONPROPERTY

**Example:** Determine the name of the timer that just expired, and based on the username perform a task.

**Trigger:** When-Timer-Expired

```sql
DECLARE
    tm_name  VARCHAR2(40);
BEGIN
    tm_name  := Get_Application_Property(TIMER_NAME);
    IF tm_name = 'MY_ONCE_EVERY_FIVE_MINUTES_TIMER' THEN
        :control.onscreen_clock := SYSDATE;
    ELSIF tm_name = 'MY_ONCE_PER_HOUR_TIMER' THEN
        Go_Block('connected_users');
        Execute_Query;
    END IF;
END;
```

Example 2:

**Built-in:** GET_APPLICATIONPROPERTY

**Example:** Capture the username and password of the currently logged-on user, for use in calling another Tool.

```sql
PROCEDURE Get_Connect_Info( the_username IN OUT VARCHAR2,
    the_password IN OUT VARCHAR2,
    the_connect  IN OUT VARCHAR2) IS
BEGIN
    the_username := Get_Application_Property(USERNAME);
    the_password := Get_Application_Property(PASSWORD);
    the_connect  := Get_Application_Property(CONNECT_STRING);
END;
```
GET_BLOCK_PROPERTY

Syntax: GET_BLOCK_PROPERTY(block_id, property);
       GET_BLOCK_PROPERTY(block_name, property);

Built-in Type: unrestricted function

Returns: CHAR

Enter Query Mode: yes

Description: Returns information about a specified block. You must issue a call to the built-in once for each property value you want to retrieve.

Parameters: block_id Specifies the unique ID Oracle Forms assigns when it creates the block. The data type of the ID is Block.

block_name Specifies the CHAR name that you gave to the block when defining it. The data type of the name is CHAR.

property Specify one of the following constants to return information about the given block:

BASE_TABLE Returns the name of the base table for the indicated block or NULL if the block does not have a base table. Corresponds to the Base Table block property.

COLUMN_SECURITY Returns the CHAR value of TRUE if column security is set to True, and the character string FALSE if it is set to False.

COORDINATION_STATUS For a block that is a detail block in a master-detail block relation, this property specifies the coordination status of the block with respect to its master block(s). Returns the CHAR value COORDINATED if the block is coordinated with all of its master blocks. If it is not coordinated with all of its master blocks, the built-in returns the CHAR value NON_COORDINATED. Immediately after records are fetched to the detail block, the status of the detail block is COORDINATED. When a different record becomes the current record in the master block, the status of the detail block again becomes NON_COORDINATED.
CURRENT_RECORD  Returns the number of the current record.

CURRENT_RECORD_ATTRIBUTE  Returns the CHAR name of the named visual attribute of the given block.

DEFAULT_WHERE  Returns the default WHERE clause in effect for the block, as indicated by the current setting of the WHERE block property.

DELETE_ALLOWED  Returns the CHAR value TRUE if the Delete Allowed block property is True, FALSE if it is False. This property determines whether the operator or the application is allowed to delete records in the block.

ENTERABLE  Returns the CHAR value TRUE if the block is enterable, that is, if any item in the block has its Enabled and Navigable properties set to True. Returns the character string FALSE if the block is not enterable.

FIRST_DETAIL_RELATION  Returns the CHAR name of the first relation in which the given block is a detail. Returns NULL if one does not exist.

FIRST_ITEM  Returns the CHAR name of the first item in the given block.

FIRST_MASTER_RELATION  Returns the CHAR name of the first relation in which the given block is a master. Returns NULL if one does not exist.

INSERT_ALLOWED  Returns the CHAR value TRUE if the Insert Allowed block property is True, FALSE if it is False. This property determines whether the operator or the application is allowed to insert records in the block.

KEY_MODE  Returns the CHAR value that indicates the current setting of the Key Mode block property. Return values for this property are UNIQUE_KEY, UPDATEABLE_PRIMARY_KEY, or NON_UPDATEABLE_PRIMARY_KEY.

LAST_ITEM  Returns the name of the last item in the given block.

LOCKING_MODE  Returns the CHAR value IMMEDIATE if rows are to be locked immediately
on a change to a base table item; otherwise, it
returns the CHAR value DELAYED if row locks are
to be attempted just prior to a commit.

**NAVIGATION_STYLE**  Returns the CHAR value
that indicates the current setting of the block’s
NAVIGATION_STYLE property, either
SAME_RECORD, CHANGE_RECORD, or
CHANGE_BLOCK.

**NEXTBLOCK**  Returns the name of the next block.
Returns NULL if the indicated block is the last
block in the form.  Note that the setting of the
block’s NEXT_NAVIGATION_BLOCK property
has no effect on the value of NEXTBLOCK.

**NEXT_NAVIGATION_BLOCK**  Returns the
CHAR name of the block’s next navigation block.
By default, the next navigation block is the next
block as defined by the order of blocks in the
Object Navigator; however, the
NEXT_NAVIGATION_BLOCK block property can
be set to override the default block navigation
sequence.

**OPTIMIZER_HINT**  Returns a hint in the form of
a character string that Oracle Forms passes on to
the RDBMS optimizer when constructing queries.

**ORDER_BY**  Returns the default ORDER BY
clause in effect for the block, as indicated by the
current setting of the ORDER BY block property.

**PREVIOUSBLOCK**  Returns the name of the block
that has the next lower sequence in the form, as
defined by the order of blocks in the Object
Navigator. Returns NULL if the indicated block is
the first block in the form.  Note that the setting of
the block’s PREVIOUS_NAVIGATION_BLOCK
property has no effect on the value of
PREVIOUSBLOCK.

**PREVIOUS_NAVIGATION_BLOCK**  Returns the
CHAR name of the block’s previous navigation
block.  By default, the previous navigation block is
the block with the next lower sequence, as defined
by the order of blocks in the Object Navigator;
however, the NEXT_NAVIGATION_BLOCK block
property can be set to override the default block navigation sequence.

**PRIMARY_KEY** Returns the CHAR value TRUE if the Primary Key property is set to True for the block. Otherwise, if the Primary Key property is set to False, this parameter returns the CHAR value FALSE.

**QUERY_ALLOWED** Returns the CHAR value TRUE if the Query Allowed block property is True, FALSE if it is False. This property determines whether the operator or the application is allowed to query records in the block.

**QUERY_HITS** Returns the CHAR value that indicates the number of records identified by the COUNT_QUERY operation. If this value is examined while records are being retrieved from a query, QUERY_HITS specifies the number of records that have been retrieved.

**QUERY_OPTIONS** Returns the CHAR values VIEW, FOR_UPDATE, COUNT_QUERY, or a null value if there are no options. You can call GET_BLOCK_PROPERTY with this parameter from within a transactional trigger when your user exit needs to know what type of query operation Oracle Forms would be doing by default if you had not circumvented default processing.

**RECORDS_DISPLAYED** Returns the number of records that the given block can display. Corresponds to the Records Displayed block property.

**RECORDS_TO_FETCH** Returns the number of records Oracle Forms expects an On–Fetch trigger to fetch and create as queried records.

**STATUS** Returns the CHAR value NEW if the block contains only new records, CHANGED if the block contains at least one changed record, and QUERY if the block contains only valid records that have been retrieved from the database.

**TOP_RECORD** Returns the record number of the topmost visible record in the given block.
UPDATE_ALLOWED

Returns the CHAR value TRUE if the Update Allowed block property is True, FALSE if it is False. This property determines whether the operator or the application is allowed to update records in the block.

UPDATE_CHANGED_COLUMNS

Specifies that only those columns updated by an operator will be sent to the database. When Update Changed Columns is set to FALSE, all columns are sent, regardless of whether they have been updated. This can result in considerable network traffic, particularly if the block contains a LONG data type.

Example:

```sql
/*
** Built-in: GET_BLOCK_PROPERTY
** Example: Return the screen line of the current record in a multi-record block. Could be used to dynamically position LOV to a place on the screen above or below the current line so as to not obscure the current record in question.
*/
FUNCTION Current_Screen_Line
RETURNS NUMBER IS
  cur_blk VARCHAR2(40) := :System.Cursor_Block;
  cur_rec NUMBER;
  top_rec NUMBER;
  itm_lin NUMBER;
  cur_lin NUMBER;
  bk_id  Block;
BEGIN
  /*
  ** Get the block id since we’ll be doing multiple Get_Block_Property operations for the same block
  */
  bk_id := Find_Block( cur_blk );
  /*
  ** Determine the (1) Current Record the cursor is in,
  ** (2) Current Record which is visible at the first (top) line of the multirecord block.
  */
  cur_rec := Get_Block_Property( bk_id, CURRENT_RECORD);
  top_rec := Get_Block_Property( bk_id, TOP_RECORD);
  /*
  ** Determine the position on the screen the first field in the multirecord block
  */
```
itm_lin := Get_Item_Property( Get_Block_Property
                       (bk_id,FIRST_ITEM),Y_POS);

/*
** Add the difference between the current record and the
** top record visible in the block to the screen position
** of the first item in the block to get the screen
** position of the current record:
*/
cur_lin := itm_lin + (cur_rec - top_rec);
RETURN cur_lin;
END;

GET_CANVAS_PROPERTY

Syntax: GET_CANVASPROPERTY(canvas_id, property);
       GET_CANVASPROPERTY(canvas_name, property);

Built-in Type: unrestricted function

Returns: CHAR

Enter Query Mode: yes

Description: Returns the given canvas property for the given canvas.

Parameters:

canvas_id Specifies the unique ID that Oracle Forms assigns
            the canvas when it creates the object. Use the
            FIND_CANVAS built-in to return the ID to an
            appropriately typed variable. The data type of the
            ID is Canvas.

canvas_name Specifies the name that you gave the object when
              defining it.

property Specifies the property whose state you want to get
          for the given canvas. You can enter the following
          constants for return values:

          HEIGHT Returns the height of the canvas,
                      specified in the form coordinate units indicated by
                      the Coordinate System form property.

          WIDTH Returns the width of the canvas, specified
                   in the form coordinate units indicated by the
                   Coordinate System form property.
**VISUAL_ATTRIBUTE**  Returns the name of the visual attribute currently in force. If no named visual attribute is assigned to the canvas, returns CUSTOM for a custom visual attribute or DEFAULT for a default visual attribute.

**Example:**

```java
/*
** Built-in: GET_CANVAS_PROPERTY
** Example: Can get the width/height of the canvas.
*/
DECLARE
    the_width  NUMBER;
    the_height NUMBER;
    cn_id      Canvas;
BEGIN
    cn_id     := Find_Canvas('My_Canvas_1');
    the_width := Get_Canvas_Property(cn_id, WIDTH);
    the_height := Get_Canvas_Property(cn_id, HEIGHT);
END;
```

---

**GET_FORM_PROPERTY**

**Syntax:**

```java
GET_FORM_PROPERTY(formmodule_id, property);
GET_FORM_PROPERTY(formmodule_name, property);
```

**Built-in Type:** unrestricted function

**Returns:** CHAR

**Enter Query Mode:** yes

**Description:** Returns information about the given form. If your application is a multi-form application, then you can call this built-in to return information about the calling form, as well as about the current, or called form.
Parameters:

formmodule_id
  Specifies the unique ID Oracle Forms assigns when it creates the form module. Use the FIND_FORM built-in to return the ID to an appropriately typed variable. The data type of the ID is FormModule.

formmodule_name
  Specifies the CHAR name that you gave to the form module when you defined it.

property
  Returns information about specific elements of the form based on which of the following constants are supplied to the built-in:

  CHARACTER_CELL_HEIGHT
    Returns the dimensions of the character cell in the form units specified by the Coordinate System property. When Coordinate System is Character Cells, the value is returned in pixels.

  CHARACTER_CELL_WIDTH
    Returns the dimensions of the character cell in the form units specified by the Coordinate System property. When Coordinate System is Character Cells, the value is returned in pixels.

  COORDINATE_SYSTEM
    Returns a character string indicating the coordinate system used in the form module.
    - CHARACTER_CELL if the current coordinate system for the form is character cell based.
    - POINTS if the current coordinate system for the form is points.
    - CENTIMETERS if the current coordinate system for the form is centimeters.
    - INCHES if the current coordinate system for the form is inches.
    - PIXELS if the current coordinate system for the form is pixels.

  CURRENT_RECORD_ATTRIBUTE
    Returns the CHAR name of the named visual attribute that should be used for the current row.

  CURSOR_MODE
    Returns the setting that indicates the intended effect of a commit action on existing cursors.
**DEFER_REQUIRED_ENFORCEMENT**  Returns the setting that indicates whether enforcement of required fields has been deferred from item validation to record validation.

**DIRECTION**  Returns the layout direction for bidirectional objects. Valid return values are RIGHT_TO_LEFT, LEFT_TO_RIGHT.

**FILE_NAME**  Returns the name of the file where the named form is stored.

**FIRST_BLOCK**  Returns the name of the block with the lowest sequence number in the indicated form.

**FORM_NAME**  Returns the name of the form.

**FIRST_NAVIGATION_BLOCK**  Returns the name of the block into which Oracle Forms attempts to navigate at form startup. By default, the first navigation block is the first block defined in the Object Navigator; however, the FIRST_NAVIGATION_BLOCK block property can be set to specify a different block as the first block at form startup.

**LAST_BLOCK**  Returns the name of the block with the highest sequence number in the indicated form.

**MODULE_NLS_CHARACTER_SET**  Returns the current value of the character set portion only of the NLS_LANG environment variable defined for the form.

**MODULE_NLS_LANG**  Returns the complete current value of the NLS_LANG environment variable defined for the form, for national language support. MODULE_NLS_LANG is the equivalent of concatenating MODULE_NLS_LANGUAGE, MODULE_NLS_TERRITORY, and MODULE_NLS_CHARACTER_SET.

**MODULE_NLS_LANGUAGE**  Returns the current value of the language portion only of the NLS_LANG environment variable defined for the form.
MODULE_NLS_TERRITORY  Returns the current value of the territory portion only of the
NLS_LANG environment variable defined for the form.

SAVEPOINT_MODE  Returns TRUE or FALSE to indicate whether savepoints are supported in the
data source.

VALIDATION  Returns TRUE or FALSE to indicate whether default Oracle Forms validation is enabled.

VALIDATION_UNIT  Returns a character string indicating the current validation unit for the form:

- FORM_SCOPE if the current validation unit is the form.
- BLOCK_SCOPE if the current validation unit is the block.
- RECORD_SCOPE if the current validation unit is the record.
- ITEM_SCOPE if the current validation unit is the item or if the current validation unit is set to DEFAULT.

Example 1:
/*
** Built-in: GET_FORM_PROPERTY
** Example: Determine the name of the first block in the form.
*/
DECLARE
  curform VARCHAR2(40);
  blkname VARCHAR2(40);
BEGIN
  curform := :System.Current_Form;
  blkname := Get_Form_Property(curform,FIRST_BLOCK);
END;

Example 2:
/*
** Built-in: GET_FORM_PROPERTY
** Example: Evaluate the current setting of the Validate property.
*/
BEGIN
  IF Get_Form_Property('curform', VALIDATION) = 'FALSE'
  THEN
    Message ('Form currently has Validation turned OFF');
  END IF;
END;
GET_GROUP_CHAR_CELL

Syntax: 
GET_GROUP_CHAR_CELL(groupcolumn_id, row_number);
GET_GROUP_CHAR_CELL(groupcolumn_name, row_number);

Built–in Type: 
unrestricted function

Returns: 
CHAR

Enter Query Mode: 
yes

Description: 
Returns the CHAR or LONG value for a record group cell identified by the given row and column. A cell is an intersection of a row and column.

Parameters: 

groupcolumn_id 
Specifies the unique ID that Oracle Forms assigns when it creates the record group column. Use the FIND_COLUMN built–in to return the ID to an appropriately typed variable. The data type of the ID is GroupColumn.

groupcolumn_name 
Specifies the fully qualified CHAR record group column name you gave the column when you defined it. If the column was defined as a result of a query, its name is the same as its corresponding database column.

row_number 
Specifies the row from which to retrieve the value of the cell.

Restrictions: 
The row_number specified must be within the bounds implied by the number of rows in the record group. A non–existent row_number results in an index out of bounds error.

Example: 
/
** Built-in: GET_GROUP_CHAR_CELL
** Example: Search thru names in a static record group to
determine if the value passed into this subprogram
exists in the list. Returns the row number
where the record was first located, or zero (0)
** if no match was found. */
FUNCTION Is_Value_In_List( the_value VARCHAR2,
 the_rg_name VARCHAR2,
 the_rg_column VARCHAR2)
RETURN NUMBER IS
 the_Rowcount NUMBER;
 rg_id RecordGroup;
BEGIN
/*
** Determine if record group exists, and if so get its ID.
*/
rg_id := Find_Group( the_rg_name );

IF Id_Null(rg_id) THEN
  Message('Record Group '||the_rg_name||' does not exist.');
  RAISE Exit_Function;
END IF;

/*
** Make sure the column name specified exists in the
** record Group.
*/
gc_id := Find_Column( the_rg_name||'.'||the_rg_column );

IF Id_Null(gc_id) THEN
  Message('Column '||the_rg_column||' does not exist.');
  RAISE Exit_Function;
END IF;

/*
** Get a count of the number of records in the record
** group
*/
the_Rowcount := Get_Group_Row_Count( rg_id );

/*
** Loop through the records, getting the specified column's
** value at each iteration and comparing it to 'the_value'
** passed in. Compare the values in a case insensitive
** manner.
*/
FOR j IN 1..the_Rowcount LOOP
  col_val := Get_Group_Char_Cell( gc_id, j );
  /*
** If we find a match, stop and return the
** current row number.
*/
  IF UPPER(col_val) = UPPER(the_value) THEN
    RETURN j;
  END IF;
END LOOP;

/*
** If we get here, we didn't find any matches.
```sql
/*
  RAISE Exit_Function;
EXCEPTION
    WHEN Exit_Function THEN
      RETURN 0;
END;

GET_GROUP_DATE_CELL

Syntax:  
GET_GROUP_DATE_CELL(groupcolumn_id, row_number);
GET_GROUP_DATE_CELL(groupcolumn_name, row_number);

Built–in Type:  unrestricted function

Returns:  DATE

Enter Query Mode:  yes

Description:  Returns the DATE value for a record group cell identified by the given row and column.  A cell is an intersection of a row and column.

Parameters:  

  groupcolumn_id  
  Specifies the unique ID that Oracle Forms assigns when it creates the record group column.  Use the FIND_COLUMN built-in to return the ID to an appropriately typed variable.  The data type of the ID is GroupColumn.

  groupcolumn_name  
  Specifies the fully qualified CHAR record group column name you gave the column when you defined it.  If the column was defined as a result of a query, its name is the same as its corresponding database column.

  row_number  
  Specifies the row from which to retrieve the value of the cell.

Restrictions:  The row_number specified must be within the bounds implied by the number of rows in the record group.  A non-existent row_number results in an index out of bounds error.

Example:  
  /*
    ** Built-in:  GET_GROUP_DATE_CELL
    ** Example:  Lookup a row in a record group, and return the minimum order date associated with that row in the record group.  Uses the 'is_value_in_list' function from the GET_GROUP_CHAR_CELL example.
  */
```
FUNCTION Max_Order_Date_Of( part_no VARCHAR2 )
RETURN DATE IS
  fnd_row NUMBER;
BEGIN
  /*
   * Try to lookup the part number among the temporary part
   * list record group named 'TMPPART' in its 'PARTNO'
   * column.
   */
  fnd_row := Is_Value_In_List( part_no, 'TMPPART', 'PARTNO');
  IF fnd_row = 0 THEN
    Message('Part Number '''||part_no||''' not found.');
    RETURN NULL;
  ELSE
    /*
     * Get the corresponding Date cell value from the
     * matching row.
     */
    RETURN Get_Group_Date_Cell( 'TMPPART.MAXORDDATE', fnd_row );
  END IF;
END;

GET_GROUP_NUMBER_CELL

Syntax: GET_GROUP_NUMBER_CELL(groupcolumn_id, row_number);
GET_GROUP_NUMBER_CELL(groupcolumn_name, row_number);

Built-in Type: unrestricted function

Returns: NUMBER

Enter Query Mode: yes

Description: Returns the NUMBER value for a record group cell identified by the
given row and column. A cell is an intersection of a row and column.

Parameters:  
  groupcolumn_id  Specifies the unique ID that Oracle Forms assigns
                 when it creates the record group column. Use the
                 FIND_COLUMN built-in to return the ID to an
                 appropriately typed variable. The data type of the
                 ID is GroupColumn.

  groupcolumn_name  Specifies the fully qualified CHAR record group
                 column name you gave the column when you
                 defined it. If the column was defined as a result of
a query, its name is the same as its corresponding database column.

**row_number** Specifies the row from which to retrieve the value of the cell.

**Restrictions:** The row_number specified must be within the bounds implied by the number of rows in the record group. A non-existent row_number results in an index out of bounds error.

**Example:**

```sql
/*
** Built-in: GET_GROUP_NUMBER_CELL
** Example: Lookup a row in a record group, and return the
** minimum order quantity associated with that row
** in the record group. Uses the
** 'is_value_in_list' function from the
** GET_GROUP_CHAR_CELL example.
*/
FUNCTION Min_Order_Qty_Of( part_no VARCHAR2 )
RETURN NUMBER IS
    fnd_row NUMBER;
BEGIN
    /*
    ** Try to lookup the part number among the temporary part
    ** list record group named 'TMPPART' in its 'PARTNO'
    ** column.
    */
    fnd_row := Is_Value_In_List( part_no, 'TMPPART', 'PARTNO');
    IF fnd_row = 0 THEN
        Message('Part Number ' || part_no || ' not found.');
        RETURN NULL;
    ELSE
        /*
        ** Get the corresponding Number cell value from the
        ** matching row.
        */
        RETURN Get_Group_Number_Cell( 'TMPPART.MINQTY', fnd_row );
    END IF;
END;
```
GET_GROUP_RECORD_NUMBER

Syntax:
GET_GROUP_RECORD_NUMBER(groupcolumn_id, cell_value);
GET_GROUP_RECORD_NUMBER(groupcolumn_name, cell_value);

Built-in Type: unrestricted function

Returns: NUMBER

Enter Query Mode: yes

Description: Returns the record number of the first record in the record group with a
column value equal to the cell_value parameter. If there is no match, 0
(zero) is returned.

Parameters:

- groupcolumn_id Specifies the unique ID that Oracle Forms assigns
to the record group column when it creates it. Use
the FIND_COLUMN built-in to return the ID to a
variable. The data type of the ID is GroupColumn.

- groupcolumn_name Specifies the name of the record group column that
you gave to the group when creating it. The data
type of the name is CHAR.

- cell_value Specifies the value to find in the specified record
group column. The data type of the name is
CHAR, NUMBER, or DATE.

Restrictions: The datatype of the cell_value parameter must match the datatype of
the record group column. The comparison is case-sensitive for CHAR
comparisons.

Example:

```/*
** Built-in:
** Example: Find the first record in the record group with a
**           cell in a column that is identical to the value
**           specified in the cell_value parameter.
***/
DECLARE
  rg_id        RecordGroup;
  match        NUMBER := 2212;
  status       NUMBER;
  the_recordnum NUMBER;
BEGIN
  rg_id := Create_Group_From_Query('QGROUP',
    'SELECT ENAME,EMPNO,SAL FROM EMP ORDER BY SAL DESC');
  status := Populate_Group( rg_id );
  */ *** Zero status is success*** /
  IF status = 0 THEN
```
the_recordnum := Get_Group_Record_Number('QGROUP.ENAME', match);
Message('The first match is record number
'||to_char(the_recordnum));
ELSE
Message('Error creating query record group.');
RAISE Form_Trigger_Failure;
END IF;
END;

---

**GET_GROUP_ROW_COUNT**

**Syntax:**
GET_GROUP_ROW_COUNT(recordgroup_id);
GET_GROUP_ROW_COUNT(recordgroup_name);

**Built-in Type:** unrestricted function

**Returns:** NUMBER

**Enter Query Mode:** yes

**Description:**
Returns the number of rows in the record group.

**Parameters:**
- `recordgroup_id`:
  Specifies the unique ID that Oracle Forms assigns to the record group when it creates it. Use the FIND_GROUP built-in to return the ID to a variable. The data type of the ID is RecordGroup.

- `recordgroup_name`:
  Specifies the name of the record group that you gave to the group when creating it. The data type of the name is CHAR.

**Example:**
/*
** Built-in: GET_GROUP_ROW_COUNT
** Example: Determine how many rows were retrieved by a Populate_Group for a query record group.
*/
DECLARE
  rg_id RecordGroup;
  status NUMBER;
  the_rowcount NUMBER;
BEGIN
  rg_id := Create_Group_From_Query('MY_QRY_GROUP',
  'SELECT ENAME,EMPNO,SAL FROM EMP ORDER BY SAL DESC');
  status := Populate_Group( rg_id );
  /* *** Zero status is success*** */
  IF status = 0 THEN
    the_rowcount := Get_Group_Row_Count( rg_id );
  */
$\text{Message('The query retrieved } \| \text{to_char(the_rowcount)} \| \text{' record(s)'\');}$

ELSE
    $\text{Message('Error creating query record group.'\');}$
    $\text{RAISE Form_Trigger_Failure;}$
END IF;
END;

---

**GET_GROUP_SELECTION**

**Syntax:**

GET_GROUP_SELECTION(recordgroup_id, selection_number);
GET_GROUP_SELECTION(recordgroup_name, selection_number);

**Built-in Type:**

unrestricted function

**Returns:**

NUMBER

**Enter Query Mode:**

yes

**Description:**

Retrieves the sequence number of the selected row for the given group.

**Parameters:**

*recordgroup_id* Specifies the unique ID that Oracle Forms assigns to the record group when it creates it. Use the FIND_GROUP built-in to return the ID to a variable. The data type of the ID is RecordGroup.

*recordgroup_name* Specifies the name of the record group that you gave to the group when creating it.

*selection_number* Identifies the selected rows in order of their selection. For example, given that rows 3, 7, and 21 are selected, their respective selection values are 1, 2, and 3. The selection_number argument takes a value of the NUMBER data type.

**Example:**

```/*
** Built-in: GET_GROUP_SELECTION
** Example: Return a comma-separated list (string) of the
** selected part numbers from the presumed
** existent PARTNUMS record group.
*/
FUNCTION Comma_Separated_PartNumbers
RETURN VARCHAR2 IS
    tmp_str VARCHAR2(2000);
    rg_id RecordGroup;
    gc_id GroupColumn;
    the_Rowcount NUMBER;
```
SEL_ROW NUMBER;
THE_VAL VARCHAR2(20);
BEGIN
  RG_ID := Find_Group('PARTNUMS');
  GC_ID := Find_Column('PARTNUMS.PARTNO');
  /
  ** Get a count of how many rows in the record group have
  ** been marked as "selected"
  */
  THE_ROWCOUNT := Get_Group_Selection_Count( RG_ID );
  FOR J IN 1..THE_ROWCOUNT LOOP
    /
    ** Get the Row number of the J-th selected row. 
    */
    SEL_ROW := Get_Group_Selection( RG_ID, J );
    /
    ** Get the (char) value of the J-th row. 
    */
    THE_VAL := Get_Group_Char_Cell( GC_ID, SEL_ROW );
    IF J = 1 THEN
      TMP_STR := THE_VAL;
    ELSE
      TMP_STR := TMP_STR || ',' || THE_VAL;
    END IF;
  END LOOP;
  RETURN TMP_STR;
END;
**GET_GROUP_SELECTION_COUNT**

**Syntax:**
```
GET_GROUP_SELECTION_COUNT(recordgroup_id);
GET_GROUP_SELECTION_COUNT(recordgroup_name);
```

**Built-in Type:** unrestricted function

**Returns:** NUMBER

**Enter Query Mode:** yes

**Description:** Returns the number of rows in the indicated record group that have been programmatically marked as selected by a call to SET_GROUP_SELECTION.

**Parameters:**
- `recordgroup_id` Specifies the unique ID that Oracle Forms assigns to the record group when it creates it. Use the FIND_GROUP built-in to return the ID to a variable. The data type of the ID is RecordGroup.
- `recordgroup_name` Specifies the name of the record group that you gave to the group when creating it.

**See Also:** SET_GROUP_SELECTION, GET_GROUP_SELECTION, UNSET_GROUP_SELECTION, RESET_GROUP_SELECTION.

**Example:**
```
/*
 ** Built-in: GET_GROUP_SELECTION_COUNT
 ** Example: See GET_GROUP_SELECTION
 */
```
GET_ITEM_PROPERTY

Syntax:  
GET_ITEM_PROPERTY(item_id, property);
GET_ITEM_PROPERTY(item_name, property);

Built-in Type:  unrestricted function

Returns:  CHAR

Enter Query Mode:  yes

Description:  Returns information about a specified item. You may be able to get but
not set certain object properties.

Parameters:  

item_id  Specifies the unique ID that Oracle Forms assigns
the object at the time it creates it. Use the
FIND_ITEM built-in to return the ID to an
appropriately typed variable. The data type of the
ID is Item.

item_name  Specifies the name that you gave the object when
creating it.

property  Specifies the property you want to set for the given
item. The possible properties are as follows:

ALIGNMENT  Returns the text alignment for text
items and display items only. Valid return values
are START, END, LEFT, CENTER, RIGHT.

AUTO_HINT  Returns the character string TRUE
if the auto hint property is set to True, and the
character string FALSE if it is set to False.

AUTO_SKIP  Returns the character string TRUE if
AutoSkip is set to True for the item, and the string
FALSE if it is set to False for the item.

BASE_TABLE  Returns the character string TRUE if the item is a base table item (that is, it
corresponds to a column in a base table), and
FALSE if the item is not a base table item.
Corresponds to the Base Table item property.

BORDER_BEVEL  Returns RAISED if the border
for the item is raised, LOWERED if the border is
lowered, or NONE if the item has no border.

CASE_INSENSITIVE_QUERY  Returns the
character string TRUE if this property is set to True
for the item, and the string FALSE if the property is set to False.

**CASE_RESTRICTION** Returns UPPERCASE if text for the item is to display in upper case, LOWERCASE if the text is to display in lower case, or NONE if no case restriction is in force.

**CURRENT_RECORD_ATTRIBUTE** Returns the CHAR name of the named visual attribute of the given item.

**DATABASE_VALUE** For a base table item, returns the value that was originally fetched from the database.

**DATATYPE** Returns the data type of the item: ALPHA, CHAR, DATE, JDATE, EDATE, DATETIME, INT, RINT, MONEY, RMONEY, NUMBER, RNUMBER, TIME, LONG, GRAPHICS, or IMAGE. Note that some item types, such as buttons and charts, do not have data types. To avoid an error message in these situations, screen for item type before getting data type.

**DIRECTION** Returns the layout direction for bidirectional objects. Valid return values are RIGHT_TO_LEFT, LEFT_TO_RIGHT.

**DISPLAYED** Returns the character string TRUE if the property is set to True for the item, and the character string FALSE if the property is set to False for the item.

**ECHO** Returns the character string TRUE if the Secure property is set to False for the item, and the character string FALSE if the Secure property is set to True for the item.

**EDITOR_NAME** Returns the name of the editor attached to the text item.

**EDITOR_X_POS** Returns the x coordinate of the editor attached to the text item. (Corresponds to the Editor Position property.)

**EDITOR_Y_POS** Returns the y coordinate of the editor attached to the edit item. (Corresponds to the Editor Position property.)
ENFORCE_KEY Returns the name of the item whose value is copied to this item as a foreign key when a new record is created as part of a master-detail relation. (Corresponds to the Copy property.)

ENABLED Returns TRUE if enabled property is set to True, FALSE if set to False.

FIXED_LENGTH Returns the character string TRUE if the property is set to True for the item, and the character string FALSE if the property is set to False for the item.

FORMAT_MASK Returns the format mask used for the text item.

HEIGHT Returns the height of the item. The size of the units depends on the Coordinate System and default font scaling you specified for the form.

HINT_TEXT Returns the item-specific help text displayed on the message line at runtime.

ICON_NAME Returns the file name of the icon resource associated with a button item having the iconic property set to TRUE.

ICONIC_BUTTON Returns the CHAR value TRUE if the button is defined as iconic, and the CHAR value FALSE if it is not an iconic button.

INSERT_ALLOWED Returns the CHAR value TRUE if operators are allowed to enter or change the item’s value in a new record. Returns FALSE if the item does not accept modification.

ITEM_CANVAS Returns the name of the canvas to which the item is assigned.

ITEM_IS_VALID Returns the character string TRUE if the current item is valid, and the character string FALSE if the current item is not valid.

ITEM_NAME Returns the name of the item.

ITEM_TYPE Returns the type of the item. Returns BUTTON if the item is a button, CHART ITEM if the item is a chart item, CHECKBOX if the item is a check box, DISPLAY ITEM if the item is a display item, IMAGE if the item is an image item, LIST if
the item is a list item, OLE OBJECT if the item is an OLE container, RADIO GROUP if the item is a radio group, TEXT ITEM if the item is a text item, USER AREA if the item is a user area, and VBX CONTROL if the item is a custom item that is a VBX control.

**KEEP_POSITION** Returns the character string TRUE if the cursor is to re-enter at the identical location it was in when it left the item, and the character string FALSE if the cursor is to re-enter the item at its default position.

**LABEL** Returns the CHAR value defined for the item’s Label property. This property is valid only for items that have labels, such as buttons and check boxes.

**LIST** Returns the character string TRUE if the item is a text item to which a list of values (LOV) is attached; otherwise returns the character string FALSE.

**LOCK_RECORD_ON_CHANGE** Returns the character string TRUE if Oracle Forms should attempt to lock a row based on a potential change to this item, and returns the character string FALSE if no lock should be attempted.

**LOV_X_POS** Returns the x coordinate of the LOV associated with the text item. (Corresponds to the LOV Position property.)

**LOV_VALIDATION** Returns the character string TRUE if Oracle Forms should validate the value of the text item against the values in the attached LOV; otherwise returns the character string FALSE.

**LOV_Y_POS** Returns the y coordinate of the LOV associated with the text item. (Corresponds to the LOV Position property.)

**MAX_LENGTH** Returns the maximum length setting for the item. The value is returned as a whole NUMBER.

**MOUSE_NAVIGATE** Returns the character string TRUE if Mouse Navigate is set to TRUE for the item,
and the character string FALSE if it is set to False for the item.

**MULTI_LINE** Returns the CHAR value TRUE if the item is a multi-line text item, and the character string FALSE if it is a single-line text item.

**NAVIGABLE** Returns the character string TRUE if the item can be navigated to, and the character string FALSE if it cannot be navigated to.

**NEXTITEM** Returns the name of the next item in the default navigation sequence, as defined by the order of items in the Object Navigator.

**NEXT_NAVIGATION_ITEM** Returns the name of the item that is defined as the “next navigation item” with respect to this current item.

**POPUPMENU_CONTENT_ITEM** Returns the setting for any of the OLE popup menu item properties:

- POPUPMENU_COPY_ITEM
- POPUPMENU_CUT_ITEM
- POPUPMENU_DELOBJ_ITEM
- POPUPMENU_INSOBJ_ITEM
- POPUPMENU_LINKS_ITEM
- POPUPMENU_OBJECT_ITEM
- POPUPMENU_PASTE_ITEM
- POPUPMENU_PASTESPEC_ITEM

Returns the character string HIDDEN if the OLE popup menu item is not displayed. Returns the character string ENABLED if the OLE popup menu item is displayed and enabled. Returns the character string DISABLED if the OLE popup menu item is displayed and not enabled. Returns the character string UNSUPPORTED if the platform is not Microsoft Windows.

**PREVIOUSITEM** Returns the name of the previous item.
PREVIOUS_NAVIGATION_ITEM  Returns the name of the item that is defined as the “previous navigation item” with respect to this current item.

PRIMARY_KEY  Returns the CHAR value TRUE if the item is a primary key, and the character string FALSE if it is not.

QUERYABLE  Returns the character string TRUE if the item can be included in a query, and the character string FALSE if it cannot.

QUERY_LENGTH  Returns the number of characters an operator is allowed to enter in the text item when the form is in Enter Query mode.

QUERY_ONLY  Returns the character string TRUE if property is set toTrue for the item, and the character string FALSE if the property is set toFalse for the item.

RANGE_HIGH  Returns the high value of the range limit.  (Corresponds to the Range property.)

RANGE_LOW  Returns the low value of the range limit.  (Corresponds to the Range property.)

REQUIRED  For multi–line text items, returns the character string TRUE if the item is required, and the character string FALSE if the item is not required.

SCROLLBAR  Returns the character string TRUE if the Scroll Bar property is True, and the character string FALSE if the Scroll Bar property is False.

SECURE  Returns the character string TRUE if the text an operator types into the text item is to be hidden, and the character string FALSE if the text an operator types into the text item is to be displayed.

UPDATEABLE  Returns the character string TRUE if the item is updateable, and FALSE if it is not.  (Corresponds to the Update Allowed property.)

UPDATE_COLUMN  Returns the character string TRUE if Oracle Forms should treat the item as updated, and FALSE if it should not.
**UPDATE_NULL** Returns the character string TRUE if the item should be updated only if it is NULL, and the character string FALSE if it can always be updated. (Corresponds to the Update if NULL property.)

**UPDATE_PERMISSION** Returns the character string TRUE if the UPDATE_PERMISSION property is set to ON, turning on the item’s UPDATEABLE and UPDATE_NULL properties. The character string FALSE indicates that UPDATEABLE and UPDATE_NULL are turned off.

**VISUAL_ATTRIBUTE** Returns the name of the visual attribute currently in force. If no named visual attribute is assigned to the item, returns CUSTOM for a custom visual attribute or DEFAULT for a default visual attribute.

**WIDTH** Returns the width of the item.

**WINDOW_HANDLE** Returns the a unique internal character constant that is used to refer to objects. Returns the number 0 if the platform is not Microsoft Windows.

**WRAP_STYLE** Returns CHARACTER if the item has wrap style set to character, WORD if word wrap is set, NONE if no wrap style is specified for the item.

**X_POS** Returns the x coordinate that reflects the current placement of the item’s upper left corner relative to the upper left corner of the canvas.

**Y_POS** Returns the y coordinate that reflects the current placement of the item’s upper left corner relative to the upper left corner of the canvas.

**Usage Notes:** If you attempt to use GET_ITEMPROPERTY to get a property for an item that is not valid for that item, an error will occur. For example, an error will occur when you attempt to get LIST from a radio group because LIST is valid only for text items.
Example:

```
/*
** Built-in: GET_ITEM_PROPERTY
** Example: Navigate to the next required item in the current block. */
PROCEDURE Go_Next_Required_Item IS
  cur_blk VARCHAR2(40);
  cur_itm VARCHAR2(80);
  orig_itm VARCHAR2(80);
  first_itm VARCHAR2(80);
  wrapped BOOLEAN := FALSE;
  found BOOLEAN := FALSE;
  Exit_Procedure EXCEPTION;
/
** Local function returning the name of the item after the one passed in. Using NVL we make the item after the last one in the block equal the first item again.
*/
FUNCTION The_Item_After(itm VARCHAR2) RETURN VARCHAR2 IS
  BEGIN
    RETURN cur_blk||'.'||NVL(Get_Item_Property(itm,NEXTITEM),
                             first_itm);
  END;
BEGIN
  cur_blk := :System.Cursor_Block;
  first_itm := Get_Block_Property( cur_blk, FIRST_ITEM );
  orig_itm := :System.Cursor_Item;
  cur_itm := The_Item_After(orig_itm);
  /
  ** Loop until we come back to the item name where we started */
  WHILE (orig_itm <> cur_itm) LOOP
    /
    ** If required item, set the found flag and exit procedure */
    IF Get_Item_Property(cur_itm,REQUIRED) = 'TRUE' THEN
      found := TRUE;
      RAISE Exit_Procedure;
    END IF;
    /
    ** Setup for next iteration */
    cur_itm := The_Item_After(cur_itm);
  END LOOP;
```
GET_LIST_ELEMENT_COUNT

Syntax: GET_LIST_ELEMENT_COUNT(list_id); GET_LIST_ELEMENT_COUNT(list_name);

Built-in Type: unrestricted function

Returns: CHAR

Enter Query Mode: yes

Description: Returns the total number of list item elements in a list, including elements with NULL values.

Parameters:

list_id Specifies the unique ID that Oracle Forms assigns when it creates the list item. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.

list_name The name you gave to the list item when you created it. The data type of the name is CHAR.

Example:

/*
** Built-in: GET_LIST_ELEMENT_COUNT
** Example: Add an element to the list item. Before adding
** the element, verify that the element is not in
** the current list.
*/
DECLARE
  total_list_count  NUMBER(2);
/*
loop_index_var       NUMBER(2) := 1;
list_element         VARCHAR(50);
list_element_value   VARCHAR(50);
list_element_to_add  VARCHAR(50);
list_value_to_add    VARCHAR(50);
list_value_to_add    VARCHAR(50);
list_value_to_add    VARCHAR(50);
element_match        VARCHAR(5) := 'TRUE';
value_match          VARCHAR(5) := 'TRUE';
BEGIN
/*
** Determine the total number of list elements.
*/
total_list_count := Get_List_Element_Count(list_id);
/*
** Compare the current list item elements with the element that
** will be added.
*/
LOOP
    list_element := Get_List_Element_Value(list_id,
        loop_index_var);
    loop_index_var := loop_index_var + 1;
    IF list_element_to_add = list_element THEN
        element_match := 'FALSE';
    END IF;
    EXIT WHEN list_element = list_element_to_add OR
        loop_index_var = total_list_count;
END LOOP;
/*
** Compare the current list item values with the value that
** will be added.
*/
loop_index_var := 1;
LOOP
    list_element_value := Get_List_Element_Value(list_id,
        loop_index_var);
    loop_index_var := loop_index_var + 1;
    IF list_value_to_add = list_element_value THEN
        value_match := 'FALSE';
    END IF;
    EXIT WHEN list_element_value = list_value_to_add OR
        loop_index_var = total_list_count;
END LOOP;
/*
** Add the element and value if it is not in the current list
*/
IF element_match AND value_match = 'TRUE' THEN
    Add_List_Element(list_id, list_name, list_element_to_add,
        list_value_to_add);
END IF
END;
GET_LIST_ELEMENT_LABEL

Syntax:  
GET_LIST_ELEMENT_LABEL(list_id, list_name, list_index);  
GET_LIST_ELEMENT_LABEL(list_name, list_index);

Built-in Type:  
unrestricted function

Returns:  
CHAR

Enter Query Mode:  
yes

Description:  
Returns information about the requested list element label.

Parameters:  

list_id  
Specifies the unique ID that Oracle Forms assigns when it creates the list item. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.

list_name  
The name you gave to the list item when you created it. The data type of the name is CHAR.

list_index  
Specifies the list index value. The list index is 1 based. If the index is greater than the count of elements in the list, GET_LIST_ELEMENT_LABEL will fail.

Usage Notes:  
The value associated with a list item element is not necessarily the list item's current value. That is, the value of :block.list_item.

Example:  
/*
** Built-in: GET_LIST_ELEMENT_LABEL
** Example: See GET_LIST_ELEMENT_COUNT
*/
GET_LIST_ELEMENT_VALUE

**Syntax:**

GET_LIST_ELEMENT_VALUE(list_id, list_index);
GET_LIST_ELEMENT_VALUE(list_name, list_index);

**Built-in Type:** unrestricted function

**Returns:** CHAR

**Enter Query Mode:** yes

**Description:** Returns the value associated with the specified list item element.

**Parameters:**

- `list_id` Specifies the unique ID that Oracle Forms assigns when it creates the list item. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.
- `list_name` The name you gave to the list item when you created it. The data type of the name is CHAR.
- `list_index` Specifies the list index value. The list index is 1 based. It will return a string containing the value of the requested element. If the index is greater than the count of elements in the list, GET_LIST_ELEMENT_VALUE will fail.

**Example:**

```/*
 ** Built-in: GET_LIST_ELEMENT_VALUE
 ** Example: See GET_LIST_ELEMENT_COUNT
 */```
GET_LOV_PROPERTY

Syntax:   
```
GET_LOV_PROPERTY(lov_id, property);
GET_LOV_PROPERTY(lov_name, property);
```

Built-in Type: unrestricted function

Returns: CHAR

Enter Query Mode: yes

Description: Returns information about a specified list of values (LOV).

You must issue a call to the built-in once for each property value you want to retrieve.

Parameters:  

-lov_id-
Specifies the unique ID that Oracle Forms assigns the object at the time it creates it. Use the FIND_LOV built-in to return the ID to an appropriately typed variable. The data type of the ID is LOV.

-lov_name-
Specifies the name that you gave the object when creating it.

-property-
Specifies the property you want to set for the given LOV. The possible properties are as follows:

-AUTO_REFRESH- Returns the character string TRUE if the property is set to True; that is, if Oracle Forms re-executes the query each time the LOV is invoked. Returns the character string FALSE if the property is set to False.

-GROUP_NAME- Returns the name of the record group currently associated with this LOV. The data type of the name is CHAR.

-HEIGHT- Returns the height of the LOV. The size of the units depends on the Coordinate System and default font scaling you specified for the form.

-WIDTH- Returns the width of the LOV. The size of the units depends on the Coordinate System and default font scaling you specified for the form.

-X_POS- Returns the x coordinate that reflects the current placement of the LOV’s upper left corner relative to the upper left corner of the screen.
Y_POS  Returns the y coordinate that reflects the current placement of the LOV’s upper left corner relative to the upper left corner of the screen.

Example:
/*
** Built-in: GET_LOVPROPERTY
** Example: Can get the width/height of the LOV.
*/
DECLARE
    the_width  NUMBER;
    the_height NUMBER;
    lov_id      LOV;
BEGIN
    lov_id     := Find_LOV('My_LOV_1');
    the_width := Get_LOV_Property(lov_id, WIDTH);
    the_height := Get_LOV_Property(lov_id,HEIGHT);
END;

GET_MENU_ITEM_PROPERTY

Syntax:  GET_MENU_ITEM_PROPERTY(menuitem_id, property);
         GET_MENU_ITEM_PROPERTY(menu_name.menuitem_name, property);

Built–in Type:  unrestricted function

Returns:  CHAR

Enter Query Mode:  yes

Description:  Returns the state of the menu item given the specific property. You can use this built–in function to get the state and then you can change the state of the property with the SET_MENU_ITEM_PROPERTY built–in.

Parameters:

menuitem_id  Specifies the unique ID Oracle Forms assigns when it creates the menu item. Use the FIND_MENU_ITEM built–in to return the ID to an appropriately typed variable. The data type of the ID is MenuItem.

menu_name.menuitem_name  Specify the CHAR name you gave to the menu item when you defined it. If you specify the menu item by name, include the qualifying menu name, for example, menu_name.menuitem_name.
Specify one of the following constants to retrieve information about the menu item:

**CHECKED** Returns the character string TRUE if a check box menu item is checked, FALSE if it is unchecked. Returns the character string TRUE if a radio menu item is the selected item in the radio group, FALSE if some other radio item in the group is selected. Returns TRUE for other menu item types.

**DISPLAYED** Returns the character string TRUE if a menu item is displayed, FALSE if it is hidden from view.

**ENABLED** Returns the character string TRUE if a menu item is enabled, FALSE if it is disabled (thus grayed out and unavailable).

**LABEL** Returns the character string for the menu item label.

Example:

```sql
/*
 ** Built-in: GET_MENU_ITEM_PROPERTY
 ** Example: Toggle the enabled/disable status of the menu item whose name is passed in. Pass in a string of the form 'MENUNAME.MENUITEM'.
 */
PROCEDURE Toggle_Enabled( menuitem_name VARCHAR2) IS
    mi_id MenuItem;
BEGIN
    mi_id := Find_Menu_Item( menuitem_name );
    IF Get_Menu_Item_Property(mi_id,ENABLED) = 'TRUE' THEN
        Set_Menu_Item_Property(mi_id,ENABLED,PROPERTY_FALSE);
    ELSE
        Set_Menu_Item_Property(mi_id,ENABLED,PROPERTY_TRUE);
    END IF;
END;
```
GET_MESSAGE

Syntax: GET_MESSAGE;

Built–in Type: unrestricted function

Returns: CHAR

Enter Query Mode: yes

Description: Returns the current message, regardless of type.

Parameters: none

Restrictions: GET_MESSAGE is only instantiated when a message is directed to the display device, either by Oracle Forms or by a call to the MESSAGE built–in. If you redirect messages using the On–Message trigger, a call to GET_MESSAGE does not return a value. Refer to the On–Message trigger description in Chapter 2, “Triggers” for more information.

Example:
/*
** Built-in: GET_MESSAGE
** Example: Capture the contents of the Message Line in a local variable
*/
DECLARE
    string_var VARCHAR2(80);
BEGIN
    string_var := Get_Message;
END;
**GET_PARAMETER_ATTR**

**Syntax:**

GET_PARAMETER_ATTR(list, key, paramtype, value);
GET_PARAMETER_ATTR(name, key, paramtype, value);

**Built-in Type:**
unrestricted procedure that returns two OUT parameters

**Enter Query Mode:**
yes

**Description:**
Returns the current value and type of an indicated parameter in an indicated parameter list.

**Parameters:**

- **list or name**
  Specifies the parameter list to which the parameter is assigned. The actual parameter can be either a parameter list ID of type PARAMLIST, or the CHAR name of the parameter list.

- **key**
  The CHAR name of the parameter.

- **paramtype**
  An OUT parameter of type NUMBER. The actual parameter you supply must be a variable of type NUMBER, and cannot be an expression. Executing the parameter sets the value of the variable to one of the following numeric constants:

  - **DATA_PARAMETER** Indicates that the parameter’s value is the name of a record group.
  - **TEXT_PARAMETER** Indicates that the parameter’s value is an actual data value.

- **value**
  An OUT parameter of type CHAR. If the parameter is a data type parameter, the value is the name of a record group. If the parameter is a text parameter, the value is an actual text parameter.

For an overview of using OUT parameters with PL/SQL procedures, refer to the [PL/SQL 2.0 User's Guide and Reference](#).
GET_PARAMETER_LIST

Syntax: 

```
GET_PARAMETER_LIST(name);
```

Built-in Type: unrestricted function

Returns: ParamList

Enter Query Mode: yes

Description: Searches the list of parameter lists and returns a parameter list ID when it finds a valid parameter list with the given name. You must define an variable of type PARAMLIST to accept the return value. This function is similar to the FIND_ functions available for other objects.

Parameters: 

- `name` Specifies a valid CHAR parameter list name.

Example: See CREATE_PARAMETER_LIST

---

GET_RADIO_BUTTON_PROPERTY

Syntax: 

```
GET_RADIO_BUTTON_PROPERTY(item_id, button_name, property);
GET_RADIO_BUTTON_PROPERTY(item_name, button_name, property);
```

Built-in Type: unrestricted function

Returns: CHAR

Enter Query Mode: yes

Description: Returns information about a specified radio button.

Parameters:

- `item_id` Specifies the radio group item ID. Oracle Forms assigns the unique ID at the time it creates the object. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.

- `item_name` Specifies the name of the radio group. The radio group is the owner or parent of its subordinate radio buttons. The data type of the name is CHAR.

- `button_name` Specifies the name of the radio button whose property you want. The data type of the name is CHAR.
Specifies the property for which you want the current state. The possible property constants you can indicate are as follows:

**DISPLAYED** Returns the character string TRUE if property is set to True, returns and the character string FALSE if property is set to False.

**ENABLED** Returns the character string TRUE if property is set to True, and the character string FALSE if property is set to False.

**HEIGHT** Returns the height of the radio button. The value is returned as a CHAR and is expressed in the units as set for the form by the form module Coordinate System property.

**LABEL** Returns the actual string label for that radio button.

**VISUAL_ATTRIBUTE** Returns the name of the visual attribute currently in force. If no named visual attribute is assigned to the radio button, returns CUSTOM for a custom visual attribute or DEFAULT for a default visual attribute.

**WIDTH** Returns the width of the radio button, including the label part. The value is returned as a CHAR and is expressed in the units as set for the form by the form module Coordinate System property.

**WINDOW_HANDLE** Returns the a unique internal character constant that is used to refer to objects. Returns the number 0 if the platform is not Microsoft Windows.

**X_POS** Returns the x coordinate that reflects the current placement of the button's upper left corner relative to the upper left corner of the canvas. The value is returned as a CHAR and is expressed in the units defined by the form module Coordinate System property.

**Y_POS** Returns the y coordinate that reflects the current placement of the button's upper left corner relative to the upper left corner of the canvas. The value is returned as a CHAR and is expressed in
the units defined by the form module Coordinate System property.

Example:

```sql
/*
 ** Built-in: GET_RADIO_BUTTONPROPERTY
 ** Example: Determine whether a given radio button is
 ** displayed and has a particular visual
 ** attribute.
 */
DECLARE
   it_id   Item;
   disp    VARCHAR2(5);
   va_name VARCHAR2(40);
BEGIN
   it_id := Find_Item('My_Favorite_Radio_Grp');
   disp := Get_Radio_Button_Property( it_id, 'REJECTED', DISPLAYED);
   va_name := Get_Radio_Button_Property( it_id, 'REJECTED', VISUAL_ATTRIBUTE);

   IF disp = 'TRUE' AND va_name = 'BLACK_ON_PEACH' THEN
      Message('You win a prize!');
   ELSE
      Message('Sorry, no luck today.');
   END IF;
END;
```
GET_RECORD_PROPERTY

Syntax: &GET_RECORD_PROPERTY; (record_number, block_name, property);

Built–in Type: unrestricted function

Returns: CHAR

Enter Query Mode: yes

Description: Returns the value for the given property for the given record number in the given block. The three parameters are required. If you do not pass the proper constants, Oracle Forms issues an error. For example, you must pass a valid record number as the argument to the record_number parameter.

Parameters:
- record_number: Specifies the record in a block for which you want property information. The number must correspond to a record number.
- block_name: Specifies the block containing the target record.
- property: Specifies the property for which you want the current state. One property constant is supported: Status.

STATUS returns NEW if the record is marked as new and there is no changed record in the block. Returns CHANGED if the record is marked as changed. Returns QUERY if the record is marked as query. Returns INSERT if the record is marked as insert.

Usage Notes: The following table illustrates the situations which return a NEW status.

<table>
<thead>
<tr>
<th>Created record with no modified fields</th>
<th>Record Status</th>
<th>Block Status</th>
<th>Form Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW</td>
<td>&lt;N</td>
<td>Q</td>
<td>C&gt;</td>
</tr>
<tr>
<td>...and all records in current block are NEW</td>
<td>NEW</td>
<td>NEW</td>
<td>&lt;N</td>
</tr>
<tr>
<td>...and all blocks in current form are NEW</td>
<td>NEW</td>
<td>NEW</td>
<td>NEW</td>
</tr>
</tbody>
</table>

The following table illustrates the effect on record, block, and form status of changes to base table items and control item in base table and control blocks.
<table>
<thead>
<tr>
<th>Type of Block/Type of Item Changed</th>
<th>Record Status Before Change</th>
<th>Record Status After Change</th>
<th>Block Status</th>
<th>Form Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>In a Base Table Block: Change a Base Table Item</td>
<td>NEW</td>
<td>INSERT</td>
<td>CHANGED</td>
<td>CHANGED</td>
</tr>
<tr>
<td>In a Base Table Block: Change a Base Table Item</td>
<td>QUERY</td>
<td>CHANGED</td>
<td>CHANGED</td>
<td>CHANGED</td>
</tr>
<tr>
<td>In a Base Table Block: Change a Control Item</td>
<td>QUERY</td>
<td>QUERY</td>
<td>&lt;Q</td>
<td>C&gt;</td>
</tr>
<tr>
<td>...and no record in current block is changed</td>
<td>QUERY</td>
<td>QUERY</td>
<td>QUERY</td>
<td>&lt;Q</td>
</tr>
<tr>
<td>...and no block in current form is changed</td>
<td>QUERY</td>
<td>QUERY</td>
<td>QUERY</td>
<td>QUERY</td>
</tr>
<tr>
<td>In a Base Table Block: Change a Control Item</td>
<td>NEW</td>
<td>INSERT</td>
<td>&lt;Q</td>
<td>C&gt;</td>
</tr>
<tr>
<td>In a Control Block: Change a Control Item</td>
<td>NEW</td>
<td>INSERT</td>
<td>&lt;Q&gt;</td>
<td>&lt;Q</td>
</tr>
<tr>
<td>...and no record in current block is changed</td>
<td>INSERT</td>
<td>QUERY</td>
<td>&lt;Q</td>
<td>C&gt;</td>
</tr>
<tr>
<td>...and no block in current form is changed</td>
<td>INSERT</td>
<td>QUERY</td>
<td>QUERY</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** Both GET_RECORD_PROPERTY and the system variable SYSTEM.RECORD_STATUS return the status of a record in a given block, and in most cases, they return the same status. However, there are specific cases in which the results may differ.

GET_RECORD_PROPERTY always has a value of NEW, CHANGED, QUERY, or INSERT, because GET_RECORD_PROPERTY returns the status of a specific record without regard to the processing sequence or whether the record is the current record.

SYSTEM.RECORD_STATUS, on the other hand, can in certain cases return a value of NULL, because SYSTEM.RECORD_STATUS is undefined when there is no current record in the system. For example, in a When–Clear–Block trigger, Oracle Forms is at the block level in its...
processing sequence, so there is no current record to report on, and the value of SYSTEM.RECORD_STATUS is NULL.

Example:

```sql
/*
** built-in:  GET_RECORD_PROPERTY
** Example: Obtain the status of a record in given block
*/
BEGIN
    IF Get_Record_Property(1,'orders',STATUS) = 'NEW' AND
       Get_Record_Property(1,'customers',STATUS) = 'NEW' THEN
        Message('You must enter a customer and order first!');
        RAISE Form_Trigger_Failure;
    END IF;
END;
```

### GET_RELATIONPROPERTY

**Syntax:**

```
GET_RELATIONPROPERTY(relation_id, property);
GET_RELATIONPROPERTY(relation_name, property);
```

**Built-in Type:** unrestricted function

**Returns:** CHAR

**Enter Query Mode:** yes

**Description:** Returns the state of the given relation property.

**Parameters:**

- `relation_id`: Specifies the unique ID Oracle Forms assigns when it creates the relation. Use the FIND_RELATION built-in to return the ID to an appropriately typed variable. The data type of the ID is Relation.

- `relation_name`: Specifies the CHAR name you gave to the relation when you defined it, or the name that Oracle Forms assigned to the relation when created.

- `property`: Specifies the property for which you want the current state. The property constants you can use are as follows:

  - `AUTOQUERY`: Returns the CHAR value TRUE if the Auto-query relation property is True, FALSE if it is False. When the Deferred relation property is set to True, this property determines whether Oracle Forms automaticallypopulates the detail
block when a different record becomes the current record in the master block.

**DEFERRED_COORDINATION** Returns the CHAR value TRUE if the Deferred relation property is True, FALSE if it is False. This property determines whether the detail block is to be immediately coordinated with the current master record, or left clear until the operator navigates to the detail block.

**DETAIL_NAME** Returns the CHAR name of the detail block in the given master–detail relationship.

**MASTER_DELETES** Returns one of the following CHAR values to indicate the current setting of the block’s Master Deletes property: NON_ISOLATED, ISOLATED, or CASCADING.

**MASTER_NAME** Returns the CHAR name of the master block in the given master–detail relationship.

**NEXT_DETAIL_RELATION** Returns the CHAR name of the next detail relation, if one exists. To get the name of the first detail for a given block, issue a call to GET_BLOCK_PROPERTY. Returns NULL if none exists.

**NEXT_MASTER_RELATION** Returns the CHAR name of the next relation, if one exists. To get the name of the first relation for a given block, issue a call to GET_BLOCK_PROPERTY. Returns NULL if one does not exist.

**PREVENT_MASTERLESS_OPERATION** Returns the CHAR value TRUE if this relation property is True, FALSE if it is False. When set to True, Oracle Forms does not allow records to be inserted in the detail block when there is no master record in the master block, and does not allow querying in the detail block when there is no master record from the database.

**Example:**

```c
/*
** Built-in: GET_RELATION_PROPERTY
** Example: If the relation is not deferred, then go
** coordinate the detail block. Otherwise, mark
** the detail block as being in need of
** coordination for an eventual deferred query.
*/
```
PROCEDURE Query_The_Details(rel_id Relation,  
detail VARCHAR2) IS  
BEGIN  
  IF Get_Role_Property(rel_id, DEFERRED_COORDINATION) = 'FALSE' THEN  
    Go_Block(detail);  
    IF NOT Form_Success THEN  
      RAISE Form_Trigger_Failure;  
    END IF;  
    Execute_Query;  
  ELSE  
    Set_Block_Property(detail, coordination_status,  
      NON_COORDINATED);  
  END IF;  
End;

GET_VIEW_PROPERTY

Syntax:  
GET_VIEW_PROPERTY(view_id, property);  
GET_VIEW_PROPERTY(view_name, property);

Built-in Type: unrestricted function

Returns: CHAR

Enter Query Mode: yes

Description: Returns the indicated property setting for the indicated canvas-view.

Parameters:  
view_id Specifies the unique ID that Oracle Forms assigns the canvas-view when it creates the object. Use the FIND_VIEW built-in to return the ID to an appropriately typed variable. The data type of the ID is ViewPort.

view_name Specifies the name that you gave the object when defining it.

property Specifies the property whose state you want to get for the given canvas-view. You must make a separate call to GET_VIEW_PROPERTY for each property you need, as shown in the example. You can enter one of the following constants to obtain return values:
DIRECTION  Returns the layout direction for bidirectional objects. Valid return values are RIGHT_TO_LEFT, LEFT_TO_RIGHT.

DISPLAY_X_POS For a stacked canvas–view, returns the x coordinate that reflects the current placement of the view’s upper left corner relative to the upper left corner of the window’s current content canvas. For a content view, returns 0. The value is returned as a CHAR and is expressed in the units defined by the form module Coordinate System property.

DISPLAY_Y_POS For a stacked canvas–view, returns the y coordinate that reflects the current placement of the view’s upper left corner relative to the upper left corner of the window’s current content canvas. For a content view, returns 0. The value is returned as a CHAR and is expressed in the units defined by the form module Coordinate System property.

HEIGHT Returns the height of the view. For a content view, the height of the view is actually the height of the window in which the view is currently displayed. The size of each unit depends on how you defined the Coordinate System property for the form module.

VISIBLE Returns the CHAR value TRUE if the view is visible, FALSE if it is not. A view is reported visible when it is a) in front of all other views in the window or b) only partially obscured by another view. A view is reported not visible when it is a) a stacked view that is behind the content view or b) completely obscured by a single stacked view. Note that this property is independent of the current window display state. Thus a view can be reported visible even when its window is currently hidden or iconified.

WIDTH Returns the width of the view. For a content view, the width of the view is actually the width of the window in which the view is currently displayed. The size of each unit depends on how you defined the Coordinate System property for the form module.
**WINDOW_NAME** Returns the name of the window where this canvas-view is displayed.

**X_POS_ON_CANVAS** Returns the x coordinate that reflects the current placement of the view's upper left corner relative to the upper left corner of its canvas. The value is returned as a CHAR and is expressed in the units defined by the form module Coordinate System property.

**Y_POS_ON_CANVAS** Returns the y coordinate that reflects the current placement of the view's upper left corner relative to the upper left corner of its canvas. The value is returned as a CHAR and is expressed in the units defined by the form module Coordinate System property.

Example:

```/*
** Built-in: GET_VIEW_PROPERTY
** Example: Use the Width, and display position of one
** stacked view (View1) to determine where to
** position another one (View2) immediately to its
** right.
*/
PROCEDURE Anchor_To_Right( View2 VARCHAR2, View1 VARCHAR2) IS
  vw_id1 ViewPort;
  vw_id2 ViewPort;
  x      NUMBER;
  y      NUMBER;
  w      NUMBER;
BEGIN
  /*
  ** Find View1 and get its (x,y) position, width
  */
  vw_id1 := Find_View(View1);
  x := Get_View_Property(vw_id1,DISPLAY_X_POS);
  y := Get_View_Property(vw_id1,DISPLAY_Y_POS);
  w := Get_View_Property(vw_id1,WIDTH);
  /*
  ** Anchor View2 at (x+w,y+h)
  */
  vw_id2 := Find_View(View2);
  Set_View_Property(vw_id2,DISPLAY_X_POS, x+w );
  Set_View_Property(vw_id2,DISPLAY_Y_POS, y );
END;```
GET_WINDOWPROPERTY

Syntax: GET_WINDOWPROPERTY(window_id, property);
       GET_WINDOWPROPERTY(window_name, property);

Built-in Type: unrestricted function

Returns: CHAR

Enter Query Mode: yes

Description: Returns the current setting for the indicated window property for the given window.

Usage Notes: On Microsoft Windows, you can reference the MDI application window with the constant FORMS_MDI_WINDOW.

Parameters:

window_id Specifies the unique ID that Oracle Forms assigns the window at the time it creates it. Use the FIND_WINDOW built-in to return the ID to an appropriately typed variable. The data type of the ID is Window.

window_name Specifies the name that you gave the window when creating it.

property You must make a separate call to GET_WINDOWPROPERTY for each property you need, as shown in the FIND_WINDOW example. Specify one of the following constants to get the current value or state of the property:

  DIRECTION Returns the layout direction for bidirectional objects. Valid return values are RIGHT_TO_LEFT, LEFT_TO_RIGHT.

  HEIGHT Returns the height of the window.

  REMOVE_ON_EXIT Returns the CHAR value TRUE if the window has the Remove On Exit property set to True, otherwise, it is FALSE.

  TITLE Returns the title of the window.

  VISIBLE Returns the CHAR value TRUE if the window is visible, FALSE if it is not. A window is reported visible if it is currently mapped to the screen, even if it is entirely hidden behind another window or iconified (minimized).
WIDTH Returns the width of the window.

WINDOW_HANDLE Returns the a unique internal character constant that is used to refer to objects. Returns the number 0 if the platform is not Microsoft Windows.

WINDOW_SIZE Returns the width and height of the window as a string, separated by commas.

WINDOW_STATE Returns the current display state of the window. The display state of a window is the character string NORMAL, MAXIMIZE, or MINIMIZE.

X_POS Returns the x coordinate that reflects the window’s current display position on the screen.

Y_POS Returns the y coordinate that reflects the window’s current display position on the screen.

---

**GO_BLOCK**

**Syntax:**
```
GO_BLOCK(block_name);
```

**Built-in Type:** restricted procedure

**Enter Query Mode:** no

**Description:** GO_BLOCK navigates to an indicated block. If the target block is non-enterable, an error occurs.

**Parameters:**
- `block_name` Specifies the name you gave the block when defining it. The data type of the name is CHAR.

**Example:**
```
/*
 ** Built-in: GO_BLOCK
 ** Example: Navigate to a block by name. Make sure to check that the Go_Block succeeds by checking FORM_SUCCESS.
 */
BEGIN
 IF :Global.Flag_Indicator = 'NIGHT' THEN
  Go_Block('Night_Schedule');
  /*
   ** One method of checking for block navigation success.
   */
  IF NOT FORM_SUCCESS THEN
   RAISE Form_Trigger_Failure;
 END;
```
GO_FORM

Syntax:  
GO_FORM(form_name);
GO_FORM(form_id);

Built-in Type: restricted procedure

Enter Query Mode: no

Description: In a multiple-form application, navigates from the current form to the indicated target form. When navigating with GO_FORM, no validation occurs and no triggers fire except WHEN-WINDOW-DEACTIVATED, which fires for the form that initiates navigation, and WHEN-WINDOW-ACTIVATED, which fires for the target window in the target form.

Attempting to navigate to a form that has not yet been opened raises an error.

Parameters:  

form_name  
The name of the target form. The data type of name is CHAR.

form_id  
The unique ID that is assigned to the form dynamically when it is instantiated at runtime. Use the FIND_FORM built-in to return the ID to an appropriately typed variable. The data type of the ID is FORMMODULE.
Restrictions: The target form cannot be a form that is currently disabled as a result of having invoked another form with CALL_FORM.

GO_ITEM

Syntax:  

```sql
GO_ITEM(item_id);
GO_ITEM(item_name);
```

Built-in Type: restricted procedure

Enter Query Mode: yes

Description: GO_ITEM navigates to an indicated item. GO_ITEM succeeds even if the target item has the Navigable property set to False.

Parameters:

- `item_id`  
  Specifies the unique ID that Oracle Forms assigns to the item when created. The data type of the ID is Item.

- `item_name`  
  Specifies the string you defined as the name of the item at design time. The data type of the name is CHAR.

Restrictions:

- If you create items with duplicate names in different blocks and use this name in a call to GO_ITEM, Oracle Forms issues a runtime error. If you have a need for duplicate names, fully qualify the item name in the argument to the built-in, as follows:

  ```sql
  GO_ITEM('emp.ename');
  ```

- In Enter Query mode, GO_ITEM cannot be used to navigate to an item in a different block.

- You cannot use GO_ITEM to navigate to a non-navigable item, such as a chart item or display item.

Example:

```sql
/*
** Built-in: GO_ITEM
** Example: Invoke a dialog window by navigating to an item which is on the canvas which the window displays.
*/
PROCEDURE Open_Preference_Dialog IS
  BEGIN
    Go_Item('pref_dialog.printer_name');
  END;
```
GO_RECORD

Syntax:  

GO_RECORD(record_number);

Built-in Type:  restricted procedure

Enter Query Mode:  no

Description:  Navigates to the record with the specified record number.

Parameters:  

record_number  Specifies any integer value that PL/SQL can evaluate to a number. This includes values derived from calls to system variables, such as

TO_NUMBER(:SYSTEM.TRIGGER_RECORD) + 8.

You can use the system variables SYSTEM.CURSOR_RECORD or SYSTEM.TRIGGER_RECORD to determine a record’s sequence number.

Restrictions:  

• The specified record number must evaluate to a positive integer.

• If the query is open and the specified record number is greater than the number of records already fetched, Oracle Forms fetches additional records to satisfy the call to this built-in.

Example:  

/*
** Built-in:  GO_RECORD
** Example:  Navigate to a record in the current block
** by record number. Also see FIRST_RECORD and
** LAST_RECORD built-ins.
*/
BEGIN
  Go_Record( :control.last_record_number );
END;
HELP

Syntax: HELP;

Built–in Type: restricted procedure

Enter Query Mode: yes

Description: Displays the current item’s hint message on the message line. If the hint message is already displayed, HELP displays the detailed help screen for the item.

Parameters: none

Example:
/*
** Built-in: HELP
** Example: Gives item-level hint/help.
*/
BEGIN
  Help;
END;

HIDE_MENU

Syntax: HIDE_MENU;

Built–in Type: unrestricted procedure

Enter Query Mode: yes

Description: On character mode platforms, makes the current menu disappear if it is currently displayed, uncovering any part of the form display that the menu had covered. The menu will redisplay if the SHOW_MENU built–in is invoked or the operator presses [Menu].

Parameters: none

Example:
/*
** Built-in: HIDE_MENU
** Example: Hides the menu from view on character-mode or block-mode devices
*/
BEGIN
  Hide_Menu;
END;
HIDE_VIEW

Syntax:   

\[
\text{HIDE\_VIEW}(\text{view\_id}); \\
\text{HIDE\_VIEW}(\text{view\_name});
\]

Built-in Type:  unrestricted procedure

Enter Query Mode:  yes

Description:  Hides the indicated canvas-view.

Parameters:

- \textit{view\_id}  
  Specifies the unique ID that Oracle Forms assigns the view at the time it creates it. Use the FIND\_VIEW built-in to return the ID to an appropriately typed variable. The data type of the ID is ViewPort.

- \textit{view\_name}  
  Specifies the name that you gave the view when creating it.

Example:

\[
\text{PROCEDURE \textit{Hide\_Button\_Bar} IS} \\
\text{BEGIN} \\
\quad \text{Hide\_View(\textquoteleft \textit{Button\_Bar}\textquoteleft);} \\
\text{END};
\]

HIDE_WINDOW

Syntax:   

\[
\text{HIDE\_WINDOW}(\text{window\_id}); \\
\text{HIDE\_WINDOW}(\text{window\_name});
\]

Built-in Type:  unrestricted procedure

Enter Query Mode:  yes

Description:  Hides the given window. HIDE\_WINDOW is equivalent to setting VISIBLE to False by calling SET\_WINDOW\_PROPERTY.

Parameters:

- \textit{window\_id}  
  Specifies the unique ID that Oracle Forms assigns the window at the time it creates it. Use the FIND\_WINDOW built-in to return the ID to an
appropriately typed variable. The data type of the ID is Window.

*window_name* Specifies the name that you gave the window when creating it.

---

**Example:**

```(/
** Built-in: HIDE_WINDOW
** Example: When a main window is closed, hide other
** “subordinate” windows automatically. To
** establish this window hierarchy we might define
** a static record group in the form called
** 'WINDOW_HIERARCHY' with a structure of:
**
<table>
<thead>
<tr>
<th>Parent_Window</th>
<th>Child_Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>DETAIL1</td>
</tr>
<tr>
<td>MAIN</td>
<td>DETAIL2</td>
</tr>
<tr>
<td>DETAIL1</td>
<td>DETAIL3</td>
</tr>
<tr>
<td>DETAIL1</td>
<td>DETAIL4</td>
</tr>
<tr>
<td>DETAIL2</td>
<td>DETAIL5</td>
</tr>
<tr>
<td>DETAIL3</td>
<td>DETAIL6</td>
</tr>
</tbody>
</table>
**
** We also have to make sure we navigate to some
** item not on any of the canvases shown in the
** windows we are closing, or else that window
** will automatically be re-displayed by forms
** since it has input focus.
*/
```

```vbnet
PROCEDURE Close_Window( wn_name VARCHAR2,
dest_item VARCHAR2 ) IS
rg_id        RecordGroup;
gc_parent    GroupColumn;
gc_child     GroupColumn;
the_Rowcount NUMBER;

PROCEDURE Close_Win_With_Children( parent_win VARCHAR2 ) IS
the_child  VARCHAR2(40);
the_parent VARCHAR2(40);
BEGIN
FOR j IN 1..the_Rowcount LOOP
  the_parent := Get_Group_Char_Cell(gc_parent,j);
  /* If we find a matching parent in the table */
  IF UPPER(the_parent) = UPPER(parent_win) THEN
    the_child := Get_Group_Char_Cell(gc_child,j);
    /*
    ** Close this child and any of its children
    */
```

---

Built-in Subprograms 3 – 177
Close_Win_With_Children( the_child );
END IF;
END LOOP;
/*
** Close the Parent
*/
    Hide_Window( parent_win );
END;
BEGIN
  /*
  ** Setup
  */
    rg_id := Find_Group('WINDOW_HIERARCHY');
    gc_parent := Find_Column('WINDOW_HIERARCHY.PARENT_WINDOW');
    gc_child := Find_Column('WINDOW_HIERARCHY.CHILD_WINDOW');
    the_Rowcount := Get_Group_Row_Count(rg_id);
  /* Close all the child windows of 'wn_name' */
    Close_Win_With_Children( wn_name );
  /* Navigate to the Destination Item supplied by the caller */
    Go_Item( dest_item );
END;

HOST

** Syntax:**
HOST(system_command_string);
HOST(system_command_string, screen_action);

** Built-in Type:** unrestricted procedure

** Enter Query Mode:** yes

** Description:** Executes an indicated operating system command.

** Parameters:**
- `system_command_string` Specifies the system command you want to pass to your particular operating system.
- `screen_action` Specifies one of the following constants:
  - `no parameter` Specifies that Oracle Forms will:
    - clear the screen
    - prompt the operator to return from the command
  - `NO_PROMPT` Specifies that Oracle Forms will:
• clear the screen (does not prompt the operator to return from the command)

**NO_SCREEN** Specifies that Oracle Forms will:

• not clear the screen
• not prompt the operator to return from the system command

(The HOST command should not send output to the screen when using the NO_SCREEN parameter.)

**Note:** The screen_action parameter is only relevant to applications running in character mode, where the output of the Host command is displayed in the same window as the form. In GUI applications, the output of the Host command is displayed in a separate window.

Example:

```/*
** built-in: HOST
** Example: Execute an operating system command in a
** subprocess or subshell. Uses the
** 'Get_Connect_Info' procedure from the
** GET_APPLICATION_PROPERTY example.
*/
PROCEDURE Mail_Warning( send_to VARCHAR2) IS
  the_username VARCHAR2(40);
  the_password VARCHAR2(40);
  the_connect  VARCHAR2(40);
  the_command  VARCHAR2(2000);
BEGIN
  /*
  ** Get Username, Password, Connect information
  */
  Get_Connect_Info(the_username,the_password,the_connect);
  /*
  ** Concatenate together the static text and values of
  ** local variables to prepare the operating system command
  ** string.
  */
  the_command := 'orasend ' ||
    ' to=' ||send_to||
    ' std_warn.txt ' ||
    ' subject="## LATE PAYMENT ##"' ||
    ' user=' ||the_username||
    ' password=' ||the_password||
    ' connect=' ||the_connect;
*/```
Message('Sending Message...', NO_ACKNOWLEDGE);
Synchronize;
/**
** Execute the command string as an O/S command The
** NO_SCREEN option tells forms not to clear the screen
** while we do our work at the O/S level "silently".
*/
Host( the_command, NO_SCREEN );
/**
** Check whether the command succeeded or not
*/
IF NOT Form_Success THEN
  Message('Error -- Message not sent.');
ELSE
  Message('Message Sent.');
END IF;
END;

ID_NULL

Syntax:    ID_NULL(object_id);
Built-in Type: unrestricted function
Returns:   BOOLEAN
Enter Query Mode: yes
Description: Returns a BOOLEAN value that indicates whether the object ID is available.
Parameters: object_id    You can call this function to test results of the following object ID types:
                   • Alert
                   • Block
                   • Canvas
                   • Editor
                   • FormModule
                   • GroupColumn
                   • Item
                   • LOV
                   • MenuItem
                   • ParamList
Usage Notes: Use ID_NULL when you want to check for the existence of an object created dynamically at runtime. For example, if a specific record group already exists, you will receive an error message if you try to create that record group. To perform this check, follow this general process:

- Use the appropriate FIND_ built-in to obtain the object ID.
- Use ID_NULL to check whether an object with that ID already exists.
- If the object does not exist, proceed to create it.

Example: See CREATE_GROUP

**IMAGE_ZOOM**

**Syntax:**

```plaintext
IMAGE_ZOOM(image_id, zoom_type);
IMAGE_ZOOM(image_name, zoom_type);
IMAGE_ZOOM(image_id, zoom_type, zoom_factor);
IMAGE_ZOOM(image_name, zoom_type, zoom_factor);
```

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Zooms the image in or out using the effect specified in `zoom_type` and the amount specified in `zoom_factor`.

**Parameters:**

- `image_id` Specifies the unique ID Oracle Forms assigns when it creates the image item. The data type of the ID is ITEM.
- `image_name` Specifies the name you gave the image when defining it.
- `zoom_type` Specify one of the following constants to describe the effect you want to have on the image displayed:
  - **ADJUST_TO_FIT** Scales the image to fit within the display rectangle: the entire image is visible
and the image fills as much of the image item as possible without distorting the image.

**SELECTION_RECTANGLE** Scales the image so the selected region fully fills the image item.

**ZOOM_IN_FACTOR** Enlarges the image by the `zoom_factor`.

**ZOOM_OUT_FACTOR** Reduces the image by the `zoom_factor`.

**ZOOM_PERCENT** Scales the image to the percentage indicated in `zoom_factor`.

`zoom_factor` Specifies either the factor or the percentage to which you want the image zoomed. Supply a whole number for this argument.

**Usage Notes:**
- Check `zoom_factor` for reasonableness. For example, specifying a ZOOM_IN_FACTOR of 100 would increase the size of your image 100 times, and could cause your application to run out of memory.
- When specifying ZOOM_IN_FACTOR or ZOOM_OUT_FACTOR, you can use any positive integer value for `zoom_factor`, but performance is optimal if you use 2, 4, or 8.
- When specifying ZOOM_PERCENT, you can use any positive integer value for `zoom_factor`. To enlarge the image, specify a percentage greater than 100.
- The operator must use the mouse to select a region before specifying SELECTION_RECTANGLE, or Oracle Forms will return an error message.
- Your design should include scroll bars on images that use SELECTION_RECTANGLE.
- Valid for both color and black–and–white images.

**Example:** The following example shows a When–Button–Pressed trigger that doubles the size of the image every time the button is pressed.

```
Image_Zoom('my_image', zoom_in_factor, 2);
```
### INSERT_RECORD

**Syntax:**

```plaintext
INSERT_RECORD;
```

**Built-in Type:**

restricted procedure

**Enter Query Mode:**

no

**Description:**

When called from an On–Insert trigger, inserts the current record into the database during Post and Commit Transactions processing. This built-in is included primarily for applications that will run against a non–ORACLE datasource.

**Parameters:**

none

**Restrictions:**

Valid only in an On–Insert trigger.

**Example:**

```plaintext
/*
** Built-in: INSERT_RECORD
** Example : Perform Oracle Forms standard insert processing
** based on a global flag setup at startup by the form, perhaps based on a parameter.
** Trigger: On-Insert
*/
BEGIN
  /*
   ** Check the global flag we setup at form startup
   */
  IF :Global.Using_Transactional_Triggers = 'TRUE' THEN
    User_Exit('my_insrec block=EMP');
  /*
   ** Otherwise, do the right thing.
   */
  ELSE
    Insert_Record;
  END IF;
END;
```
ISSUE_ROLLBACK

Syntax: ISSUE_ROLLBACK(savepoint_name);

Built-in Type: unrestricted procedure

Enter Query Mode: no

Description: When called from an On-Rollback trigger, initiates the default Oracle Forms processing for rolling back to the indicated savepoint.

This built-in is included primarily for applications that will run against a non-ORACLE data source.

Parameters: savepoint name Name of the savepoint to which you want to rollback. A null savepoint_name causes a full rollback.

Restrictions: Results are unpredictable when ISSUE_ROLLBACK is used outside an On-Rollback trigger or when used with a savepoint other than that provided by a call to GET_APPLICATION_PROPERTY(SAVEPOINT_NAME).

Example: /*
** Built-in: ISSUE_ROLLBACK
** Example: Perform Oracle Forms standard Rollback processing.
** Decide whether to use this built-in based on a global flag setup at startup by the form.
** perhaps based on a parameter.
** Trigger: On-Rollback
*/
DECLARE
    sp_name VARCHAR2(80);
BEGIN
    /*
    ** Get the name of the savepoint to which Forms needs to rollback. (NULL = Full Rollback)
    */
    sp_name := Get_Application_Property(SAVEPOINT_NAME);
    /*
    ** Check the global flag we setup at form startup
    */
    IF :Global.Using_Transactional_Triggers = 'TRUE' THEN
        User_Exit('my_rollbk name=' || sp_name);
    ELSE
        Issue_Rollback(sp_name);
    END IF;
    END;
ISSUE_SAVEPOINT

Syntax: ISSUE_SAVEPOINT(savepoint_name);

Built-in Type: unrestricted procedure

Enter Query Mode: no

Description: When called from an On-Savepoint trigger, ISSUE_SAVEPOINT initiates the default processing for issuing a savepoint. You can use GET_APPLICATION_PROPERTY (SAVEPOINT_NAME) to determine the name of the savepoint that Oracle Forms would be issuing by default, if no On-Savepoint trigger were present.

This built-in is included primarily for applications that will run against a non-ORACLE datasource.

Parameters: savepoint_name Name of the savepoint you want to be issued

Restrictions: Never issue a savepoint with the name FM_<number>, unless the savepoint name was provided by a call to GET_APPLICATION_PROPERTY. Doing so may cause a conflict with savepoints issued by Oracle Forms.

Example: /* Built-in: ISSUE_SAVEPOINT
** Example: Perform Oracle Forms standard savepoint processing.
** Decide whether to use this built-in based on a global flag setup at startup by the form,
** perhaps based on a parameter.
** Trigger: On-Savepoint
*/
DECLARE
   sp_name VARCHAR2(80);
BEGIN
   /* Get the name of the savepoint Forms needs to issue */
   sp_name := Get_Application_Property(SAVEPOINT_NAME);
   /* Check the global flag we setup at form startup */
   IF :Global.Using_Transactional_Triggers = 'TRUE' THEN
      User_Exit('my_savept name='||sp_name);
   /* Otherwise, do the right thing. */
   ELSE
      Issue_Savepoint(sp_name);
   END IF;
END;
ITEM_ENABLED

Syntax:  ITEM_ENABLED (mnunam, itmnam);

Built–in Type:  unrestricted function

Returns:  BOOLEAN

Enter Query Mode:  yes

Description:  Returns the Boolean value TRUE when the menu item is enabled. Returns the Boolean value FALSE when the menu item is disabled.

Note:  ITEM_ENABLED is equivalent to GET_MENU_ITEM_PROPERTY (MENU_ITEM, ENABLED).

Parameters:  

- mnunam  Specifies the CHAR name of the menu.
- itmnam  Specifies the CHAR name of the menu item.

LAST_RECORD

Syntax:  LAST_RECORD;

Built–in Type:  restricted procedure

Enter Query Mode:  no

Description:  Navigates to the last record in the block’s list of records. If a query is open in the block, Oracle Forms fetches the remaining selected records into the block’s list of records, and closes the query.

Parameters:  none

Example:  See FIRST_RECORD
LIST_VALUES

Syntax:  \texttt{LIST\_VALUES (kwd)};

Built-in Type:  restricted procedure

Enter Query Mode:  no

Description:  \texttt{LIST\_VALUES} displays the list of values for the current item, as long as the input focus is in a text item that has an attached LOV. The list of values remains displayed until the operator dismisses the LOV or selects a value.

By default, \texttt{LIST\_VALUES} uses the NO\_RESTRICT parameter. This parameter causes Oracle Forms not to use the automatic search and complete feature. If you use the RESTRICT parameter, Oracle Forms uses the automatic search and complete feature.

\textbf{Automatic Search and Complete Feature}  With the automatic search and complete feature, an LOV evaluates a text item’s current value as a search value. That is, if an operator presses [List] in a text item that has an LOV, Oracle Forms checks to see if the item contains a value.

If the text item contains a value, Oracle Forms automatically uses that value as if the operator had entered the value into the LOV’s search field and pressed [List] to narrow the list.

If the item value would narrow the list to only one value, Oracle Forms does not display the LOV, but automatically reads the correct value into the field.

Parameters:  \texttt{kwd}  

Specifies one of the following constants:

\begin{itemize}
  \item NO\_RESTRICT  Specifies that Oracle Forms will not use the automatic search and complete feature.
  \item RESTRICT  Specifies that Oracle Forms will use the automatic search and complete feature.
\end{itemize}
LOCK_RECORD

Syntax: LOCK_RECORD;

Built-in Type: unrestricted procedure

Enter Query Mode: no

Description: Attempts to lock the row in the database that corresponds to the current record. LOCK_RECORD locks the record immediately, regardless of whether the Locking Mode block property is set to Immediate (the default) or Delayed.

When executed from within an On-Lock trigger, LOCK_RECORD initiates default database locking. The following example illustrates this technique.

Parameters: none

Example:
```/*
** Built-in:  LOCK_RECORD
** Example:   Perform Oracle Forms standard record locking on the
**            queried record which has just been deleted or
**            updated. Decide whether to use default
**            processing or a user exit by consulting a
**            global flag setup at startup by the form,
**            perhaps based on a parameter.
**            Trigger: On-Lock
*/
BEGIN
  /*
  ** Check the global flag we set up at form startup
  */
  IF :Global.Non_Oracle_Datasource = 'TRUE' THEN
    User_Exit('my_lockrec block=EMP');
  /*
  ** Otherwise, do the right thing.
  */
  ELSE
    Lock_Record;
  END IF;
END;```
**LOGON**

**Syntax:**

- `LOGON(username, password);`
- `LOGON(username, password, logon_screen_on_error);`

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Performs the default Oracle Forms logon processing with an indicated username and password. Call this procedure from an On-Logon trigger when you want to augment default logon processing.

**Parameters:**

This built-in takes the following arguments:

- `username` Any valid username of up to 80 characters.
- `password` Any valid password of up to 80 characters, including a database connect string.
- `logon_screen_on_error` An optional BOOLEAN parameter that, when set to TRUE (default), causes Oracle Forms to automatically display the logon screen if the logon specified fails (usually because of a incorrect username/password). When `logon_screen_on_error` is set to FALSE and the logon fails, the logon screen will not display and FORMS_FAILURE is set to TRUE so the designer can handle the condition in an appropriate manner.

**Restrictions:**

- Must not already be connected to a database. (Use the LOGOUT built-in first, if necessary.)
- If you identify a remote database, a SQL*Net connection to that database must exist at runtime.
- Oracle Forms can connect to only one database at a time. However, database links may be used to access multiple databases with a single connection.

**Example:**

```c
/*
** Built-in: LOGON
** Example: Perform Oracle Forms standard logon to the ORACLE
database. Decide whether to use Forms built-in
processing or a user exit by consulting a
** global flag setup at startup by the form,
** perhaps based on a parameter. This example
** uses the 'Get_Connect_Info' procedure from the
** GET_APPLICATION_PROPERTY example.
** Trigger: On-Logon
*/
```
DECLARE
    un  VARCHAR2(80);
    pw  VARCHAR2(80);
    cn  VARCHAR2(80);
BEGIN
    /*
    ** Get the connection info
    */
    Get_Connect_Info(un,pw,cn);
    /*
    ** If at startup we set the flag to tell our form that we
    ** are not running against ORACLE, then call our
    ** appropriate MY_LOGON userexit to logon.
    */
    IF :Global.Non_Oracle_Datasource = 'TRUE' THEN
        User_Exit('my_logon username='||un||' password='||pw);
    /*
    ** Otherwise, call the LOGON built-in
    */
    ELSE
        IF cn IS NOT NULL THEN
            LOGON(un,pw||'@'||cn);
        ELSE
            LOGON(un,pw);
        END IF;
    END IF;
END;

LOGON_SCREEN

Syntax:   LOGON_SCREEN;

Built-in Type:  unrestricted procedure

Enter Query Mode:  yes

Description:  Displays the default Oracle Forms logon screen and requests a valid username and password. Most commonly, you will include this built-in subprogram in an On-Logon trigger to connect to a non-ORACLE data source.

Parameters:  none

Restrictions:  • When first entering a form, if the On-Logon trigger fails with an unhandled exception, no other triggers are executed, and the current
Runform session is aborted. Otherwise, to change connections in the middle of a session, you can log off, then log on again.

- You must issue a call to the LOGON built-in to create the connection to your data source.

**Example:**

```sql
DECLARE
  un  VARCHAR2(80);
  pw  VARCHAR2(80);
  cn  VARCHAR2(80);
BEGIN
  /*
   ** Bring up the logon screen
   */
  Logon_Screen;
  /*
   ** Get the username, password and
   ** connect string.
   */
  Get_Connect_Info( un, pw, cn );
  /*
   ** Log the user onto the database
   */
  IF cn IS NOT NULL THEN
    LOGON(un,pw||'@'||cn);
  ELSE
    LOGON(un,pw);
  END IF;
END;
```
LOGOUT

Syntax: LOGOUT;

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Disconnects the application from the ORACLE RDBMS. All open cursors are automatically closed when you issue a call to the LOGOUT built-in. You can programmatically log back on with LOGON. If you LOGOUT of a multiple-form application with multiple connections, Oracle Forms tries to re-establish all of those connections when you subsequently execute LOGON.

Parameters: none

Example:

```
/*
** Built-in: LOGOUT
** Example: Perform Oracle Forms standard logout. Decide whether to use Forms built-in processing or a user exit by consulting a global flag setup at startup by the form, perhaps based on a parameter.
** Trigger: On-Logout
*/
BEGIN
  /*
  ** Check the flag we setup at form startup
  */
  IF :Global.Non_Oracle_Datasource = 'TRUE' THEN
    User_Exit('my_logout');
  /*
  ** Otherwise, do the right thing.
  */
  ELSE
    Logout;
  END IF;
```
### MAIN_MENU

**Syntax:**
```
MAIN_MENU;
```

**Built–in Type:** restricted procedure

**Enter Query Mode:** yes

**Description:** MAIN_MENU navigates to the main menu of the current application.

**Parameters:** none

**Restrictions:** This built–in is not available in a pull–down menu.

### MENU_CLEAR_FIELD

**Syntax:**
```
MENU_CLEAR_FIELD;
```

**Built–in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** MENU_CLEAR_FIELD clears the current field’s value from the current cursor position to the end of the field. If the current cursor position is to the right of the last nonblank character, MENU_CLEAR_FIELD clears the entire field, making its value NULL.

**Parameters:** none

**Restrictions:** The Enter Parameter Values dialog must be displayed.
MENU_NEXT_FIELD

Syntax:  MENU_NEXT_FIELD;
Built–in Type:  restricted procedure
Enter Query Mode:  yes
Description:  MENU_NEXT_FIELD navigates to the next field in an Enter Parameter Values dialog.
Parameters:  none
Restrictions:  You must be in an Enter Parameter Values dialog.

MENU_PARAMETER

Syntax:  MENU_PARAMETER;
Built–in Type:  unrestricted procedure
Enter Query Mode:  yes
Description:  MENU_PARAMETER displays all the parameters associated with the current menu, and their current values, in the Enter Parameter Values dialog.
Parameters:  none
Restrictions:  Valid only for menus running in full–screen display style.
**MENU_PREVIOUS_FIELD**

Syntax: `MENU_PREVIOUS_FIELD;`

Built–in Type: unrestricted procedure

Enter Query Mode: yes

Description: `MENU_PREVIOUS_FIELD` returns to the previous field in an Enter Parameter Values dialog.

Parameters: none

Restrictions: You must be in an Enter Parameter Values dialog box.

---

**MENU_REDISPLAY**

Syntax: `MENU_REDISPLAY;`

Built–in Type: unrestricted procedure

Enter Query Mode: yes

Description: This procedure redraws the screen in a menu.

Parameters: none

Restrictions: You must be on a character mode or block mode platform.

---

**MENU_SHOW_KEYS**

Syntax: `MENU_SHOW_KEYS;`

Built–in Type: unrestricted procedure

Enter Query Mode: yes

Description: `MENU_SHOW_KEYS` displays the Show Keys screen for the menu module at runtime.

Parameters: none

Restrictions: `MENU_SHOW_KEYS` is available in any context.
MESSAGE

Syntax: MESSAGE(message_string, user_response);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Displays specified text on the message line.

Parameters:

- message_string Specify a character string enclosed in single quotes or a variable of CHAR data type.
- user_response Specifies one of the following constants:
  - ACKNOWLEDGE Specifies that Oracle Forms is to display a modal alert that the operator must dismiss explicitly, whenever two consecutive messages are issued. ACKNOWLEDGE forces the first message to be acknowledged before the second message can be displayed. This is the default.
  - NO_ACKNOWLEDGE Specifies that, when two consecutive messages are issued, the operator is not expected to respond to the first message displayed before Oracle Forms displays a second message. Using NO_ACKNOWLEDGE creates a risk that the operator may not see the first message, because the second message immediately overwrites it without prompting the operator for acknowledgement.

Restrictions: message_string can be up to 200 characters long. Note, however, that several factors affect the maximum number of characters that can be displayed, including the current font and the limitations of the runtime window manager.

Example:

```/*
** Built-in: MESSAGE
** Example: Display several messages to the command line throughout the progress of a particular subprogram. By using the NO_ACKNOWLEDGE parameter, we can avoid the operator’s having to acknowledge each message explicitly.
*/
PROCEDURE Do_Large_Series_Of_Updates IS
BEGIN
  Message('Working... (0%)', NO_ACKNOWLEDGE);
```

MESSAGE_CODE

Syntax: MESSAGE_CODE;

Built-in Type: unrestricted function

Returns: NUMBER

Enter Query Mode: yes

Description: Returns a message number for the message that Oracle Forms most recently generated during the current Runform session. MESSAGE_CODE returns zero at the beginning of a session, before Oracle Forms generates any messages.

Use MESSAGE_CODE to test the outcome of a user action (e.g., pressing a key) to determine processing within an On-Message trigger.

Refer to the Messages appendix for a list of messages and message numbers.

Parameters: none

Example:

```sql
DECLARE
    msgnum NUMBER := MESSAGE_CODE;
    msgtxt VARCHAR2(80) := MESSAGE_TEXT;
    msgtyp VARCHAR2(3) := MESSAGE_TYPE;
END;
```
BEGIN
  IF msgnum = 40400 THEN
    Message('Your changes have been made permanent.');
  ELSIF msgnum = 40401 THEN
    Message('You have no unsaved changes outstanding.');
  ELSE
    /*
    ** Print the Normal Message that would have appeared
    **
    ** FRM–12345: Message Text Goes Here
    */
    Message(msgtyp||'–'||TO_CHAR(msgnum)||': '||msgtxt);
  END IF;
END;

MESSAGE_TEXT

Syntax: MESSAGE_TEXT;

Built-in Type: unrestricted function

Returns: CHAR

Enter Query Mode: yes

Description: Returns message text for the message that Oracle Forms most recently generated during the current Runform session. MESSAGE_TEXT returns NULL at the beginning of a session, before Oracle Forms generates any messages.

Use MESSAGE_TEXT to test the outcome of a user action (e.g., pressing a key) to determine processing within an On–Message trigger.

Note: If your applications must be supported in more than one language, use the MESSAGE_CODE built-in instead of the MESSAGE_TEXT built-in. Referencing message codes rather than message text is particularly useful in applications that provide national language support.

Parameters: none

Example:
  /*
   ** Built-in: MESSAGE_CODE,MESSAGE_TEXT,MESSAGE_TYPE
   ** Example: Reword certain FRM message messages by checking
   **          the Message_Code in an ON–MESSAGE trigger
   ** Trigger: On–Message
   */
DECLARE
    msgnum NUMBER       := MESSAGE_CODE;
    msgtxt VARCHAR2(80) := MESSAGE_TEXT;
    msgtyp VARCHAR2(3)  := MESSAGE_TYPE;
BEGIN
    IF msgnum = 40400 THEN
        Message('Your changes have been made permanent.');
    ELSIF msgnum = 40401 THEN
        Message('You have no unsaved changes outstanding.');
    ELSE
        /*
        ** Print the Normal Message that would have appeared
        **
        ** FRM-12345: Message Text Goes Here
        */
        Message(msgtyp||'-'||TO_CHAR(msgnum)||': '||msgtxt);
    END IF;
END;

MESSAGE_TYPE

Syntax: MESSAGE_TYPE;
Built-in Type: unrestricted function
Returns: CHAR
MESSAGE_TYPE returns one of three values for the message type:
FRM Indicates that an Oracle Forms message was generated.
ORA Indicates that an ORACLE message was generated.
NULL Indicates that Oracle Forms has not yet issued any messages during the session.

Enter Query Mode: yes
Description: Returns a message type for the message that Oracle Forms most recently generated during the current Runform session.
Use MESSAGE_TYPE to test the outcome of a user action (e.g., pressing a key) to determine processing within an On–Message trigger.
Parameters: none
Example: /*
** Built-in: MESSAGE_CODE,MESSAGE_TEXT,MESSAGE_TYPE
** Example: Reword certain FRM message messages by checking
** the Message_Code in an ON-MESSAGE trigger
** Trigger: On-MESSAGE
*/
DECLARE
  msgnum NUMBER := MESSAGE_CODE;
  msgtxt VARCHAR2(80) := MESSAGE_TEXT;
  msgtyp VARCHAR2(3) := MESSAGE_TYPE;
BEGIN
  IF msgnum = 40400 THEN
    Message('Your changes have been made permanent.');
  ELSIF msgnum = 40401 THEN
    Message('You have no unsaved changes outstanding.');
  ELSE
    /*
    ** Print the Normal Message that would have appeared
    **
    ** FRM–12345: Message Text Goes Here
    */
    Message(msgtyp||'–'||TO_CHAR(msgnum)||': '||msgtxt);
  END IF;
END;

MOVE_WINDOW

Syntax:
 MOVE_WINDOW(window_id, x, y);
 MOVE_WINDOW(window_name, x, y);

Built–in Type: unrestricted function

Enter Query Mode: yes

Description: Moves the given window to the location specified by the given coordinates.

If you have specified the form property Coordinate System as Character, then your x, y coordinates are specified in characters. If the Coordinate System is specified as Real, then your x, y coordinates are specified in the real units you have selected—pixels, inches, centimeters, or points.

Parameters: window_id

Specifies the unique ID that Oracle Forms assigns the window when created. Use the FIND_WINDOW built–in to return the ID to an appropriately typed variable. The data type of the ID is Window.
Example:

```
/*
** Built-in:  MOVE_WINDOW
** Example:   Move window2 to be anchored at the bottom right
**            corner of window1.
*/
PROCEDURE Anchor_Bottom_Right2( Window2 VARCHAR2, Window1 VARCHAR2) IS
  wn_id1 Window;
  wn_id2 Window;
  x      NUMBER;
  y      NUMBER;
  w      NUMBER;
  h      NUMBER;
BEGIN
  /*
  ** Find Window1 and get its (x,y) position, width, and
  ** height.
  */
  wn_id1 := Find_Window(Window1);
  x := Get_Window_Property(wn_id1,X_POS);
  y := Get_Window_Property(wn_id1,Y_POS);
  w := Get_Window_Property(wn_id1,WIDTH);
  h := Get_Window_Property(wn_id1,HEIGHT);
  /*
  ** Anchor Window2 at (x+w,y+h)
  */
  wn_id2 := Find_Window(Window2);
  Move_Window( wn_id2, x+w, y+h );
END;
```
NAME_IN

Syntax:  NAME_IN(variable_name);

Built–in Type:  unrestricted function

Returns:  CHAR

Description:  Returns the value of the indicated variable.

The returned value is in the form of a character string. However, you can use NAME_IN to return numbers and dates as character strings and then convert those strings to the appropriate data types. You can use the returned value as you would use any value within an executable statement.

If you nest the NAME_IN function, Oracle Forms evaluates the individual NAME_IN functions from the innermost one to the outermost one.

Parameters:  variable_name  Specifies a valid variable or text item. The data type of the name is CHAR.

Example:
/*
** Built–in:  NAME_IN
** Example:  Simple implementation of a Last-In–First–Out stack mechanism using Global variables.
** For each named stack, a global variable GLOBAL.<stackname>_PTR points to the largest element on the stack. PUSH increments this value as new elements are added. Values PUSH’ed on or POP’ed off the named stack are actually stored in GLOBAL variables of a conveniently formed name: GLOBAL.<stackname>nnn where ’nnn’ is the number of the element on the stack.
** Usage:
** Push(’MYSTACKNAME’, ’1’);
** Push(’MYSTACKNAME’, ’2’);
**
** str_var := Pop(’MYSTACKNAME’); -- Gets ’2’
** str_var := Pop(’MYSTACKNAME’); -- Gets ’1’
** str_var := Pop(’MYSTACKNAME’); -- Gets ’EOS’
**
*/
PROCEDURE Push ( the_stackname VARCHAR2,
the_value      VARCHAR2 ) IS
ptr_name VARCHAR2(40); -- This stack's pointer name
prefix VARCHAR2(40); -- Common prefix for storage vars
elt_name VARCHAR2(40); -- Name of storage element
new_idx VARCHAR2(4); -- New stack pointer value

BEGIN

/*
** For any named stack that we reference, the global
** variables used for storing the stack's values and the
** stack's pointer all begin with a common prefix:
** GLOBAL.<stackname>
*/
prefix := 'GLOBAL.' || the_stackname;

/*
** This named stack's pointer resides in
** GLOBAL.<stackname>_PTR Remember that this is the "name"
** of the pointer.
*/
ptr_name := prefix || '_PTR';

/*
** Initialize the stack pointer with a default value of
** zero if the stack pointer did not exist previously, ie
** the GLOBAL.<stackname>_PTR had yet to be created.
*/
Default_Value( '0', ptr_name );

/*
** Since we're PUSH'ing a new element on the stack,
** increment the stack pointer to reflect this new
** element's position. Remember that GLOBAL variables are
** always of type CHAR, so we must convert them TO_NUMBER
** before any calculations.
*/
new_idx := TO_CHAR( TO_NUMBER( Name_In( ptr_name ) ) + 1 ) ;
Copy( new_idx   , ptr_name );

/*
** Determine the name of the global variable which will
** store the value passed in, GLOBAL.<stackname><new_idx>.
** This is simply the prefix concatenated to the new index
** number we just calculated above.
*/
elt_name := prefix||new_idx;
Copy( the_value , elt_name );
END;

FUNCTION Pop ( the_stackname VARCHAR2 )
RETURN VARCHAR2 IS

ptr_name VARCHAR2(40); -- This stack's pointer name
prefix VARCHAR2(40); -- Common prefix for storage vars
elt_name VARCHAR2(40); -- Name of storage element
new_idx VARCHAR2(4); -- New stack pointer value
cur_idx VARCHAR2(4); -- Current stack pointer value
the_val VARCHAR2(255);

EMPTY_STACK CONSTANT VARCHAR2(3) := 'EOS';
NO_SUCH_STACK CONSTANT VARCHAR2(3) := 'NSS';

BEGIN
  /*
  ** For any named stack that we reference, the global
  ** variables used for storing the stack’s values and the
  ** stack’s pointer all begin with a common prefix:
  ** GLOBAL.<stackname>
  */
  prefix := 'GLOBAL.' || the_stackname;
  /*
  ** This named stack’s pointer resides in
  ** GLOBAL.<stackname>._PTR Remember that this is the *name*
  ** of the pointer.
  */
  ptr_name := prefix || '_PTR';
  /*
  ** Force a default value of NULL so we can test if the
  ** pointer exists (as a global variable). If it does not
  ** exist, we can test in a moment for the NULL, and avoid
  ** the typical error due to referencing non-existent
  ** global variables.
  */
  Default_Value( NULL, ptr_name );
  /*
  ** If the *value* contained in the pointer is NULL, then
  ** the pointer must not have existed prior to the
  ** Default_Value statement above. Return the constant
  ** NO_SUCH_STACK in this case and erase the global
  ** variable that the Default_Value implicitly created.
  */
  IF Name_In( ptr_name ) IS NULL THEN
    the_val := NO_SUCH_STACK;
    Erase( ptr_name );
    /*
    ** Otherwise, the named stack already exists. Get the
    ** index of the largest stack element from this stack’s
    ** pointer.
    */
  ELSE
    cur_idx := Name_In( ptr_name ) ;
    /*
    ** If the index is zero, then the named stack is already
    ** empty, so return the constant EMPTY_STACK, and leave
    ** the stack’s pointer around for later use, ie don’t
** ERASE it.
**
** Note that a stack can only be empty if some values
** have been PUSHʼed and then all values subsequently
** POPʼed. If no values were ever PUSHʼed on this named
** stack, then no associated stack pointer would have
** been created, and we would flag that error with the
** NO_SUCH_STACK case above.
*/
IF cur_idx = '0' THEN
  the_val := EMPTY_STACK;
/*
** If the index is non-zero, then:
** (1) Determine the name of the global variable in
** which the value to be POPʼed is stored,
**     GLOBAL.<stackname><cur_idx>
** (2) Get the value of the (cur_idx)-th element to
**     return
** (3) Decrement the stack pointer
** (4) Erase the global variable which was used for
**     value storage
*/
ELSE
  elt_name:= prefix || cur_idx;
  the_val := Name_In( elt_name );
  new_idx := TO_CHAR( TO_NUMBER( Name_In(ptr_name) ) – 1 ) ;
  Copy( new_idx , ptr_name );
  Erase( elt_name );
END IF;
END IF;
RETURN the_val;
END;

** NEW_FORM 

Syntax: NEW_FORM(formmodule_name);
        NEW_FORM(formmodule_name, rollback_mode);
        NEW_FORM(formmodule_name, rollback_mode, query_mode);
        NEW_FORM(formmodule_name, rollback_mode, query_mode, paramlist_id);
        NEW_FORM(formmodule_name, rollback_mode, query_mode, paramlist_name);

Built-in Type: restricted procedure 

Enter Query Mode: no
Description: Exits the current form and enters the indicated form. The calling form is terminated as the parent form. If the calling form had been called by a higher form, Oracle Forms keeps the higher call active and treats it as a call to the new form. Oracle Forms releases memory (such as database cursors) that the terminated form was using.

Oracle Forms runs the new form with the same Runform options as the parent form. If the parent form was a called form, Oracle Forms runs the new form with the same options as the parent form.

Parameters: formmodule_name Specifies the formmodule name of the called form. The name must be enclosed in single quotes. The data type of the name is CHAR.

to_savepoint Oracle Forms rolls back all uncommitted changes (including posted changes) to the current form’s savepoint.

no_rollback Oracle Forms exits the current form without rolling back to a savepoint. You can leave the top level form without performing a rollback, which means that you retain any locks across a NEW_FORM operation. These locks can also occur when invoking Oracle Forms from an external 3GL program. The locks are still in effect when you regain control from Oracle Forms.

full_rollback Oracle Forms rolls back all uncommitted changes (including posted changes) that were made during the current Runform session. You cannot specify a FULL_ROLLBACK from a form that is running in post-only mode. (Post-only mode can occur when your form issues a call to another form while unposted records exist in the calling form. To avoid losing the locks issued by the calling form, Oracle Forms prevents any commit processing in the called form.)

query_mode Takes one of the following constants as an argument:

no_query_only Runs the indicated form normally, allowing the operator to perform inserts, updates, and deletes in the form.

query_only Runs the indicated form as a query-only form.
Specifies the unique ID Oracle Forms assigns when it creates the parameter list. Specify a parameter list when you want to pass parameters from the calling form to the new form. The data type of the ID is PARAMLIST. A parameter list passed to a form via NEW_FORM cannot contain parameters of type DATA_PARAMETER (a pointer to record group).

The name you gave the parameter list object when you defined it. The data type of the name is CHAR. A parameter list passed to a form via NEW_FORM cannot contain parameters of type DATA_PARAMETER (a pointer to record group).

Example:

```sql
/*
** Built-in:  NEW_FORM
** Example:   A generic procedure which will invoke the
**            formname passed–in using the method indicated
**            by the 'newform' and 'queryonly' parameters.
*/
PROCEDURE Generic_Call( formname  VARCHAR2,
                        newform   VARCHAR2,
                        queryonly VARCHAR2 ) IS

  msglvl         VARCHAR2(2);
  Error_Occurred EXCEPTION;
BEGIN
/*
** Remember the current message level and temporarily set
** it to 10 to suppress errors if an incorrect formname is
** called
*/
  msglvl := :System.Message_Level;
  :System.Message_Level := '10';

  IF newform = 'Y' THEN
    IF queryonly = 'Y' THEN
      New_Form( formname, TO_SAVEPOINT, QUERY_ONLY);
    ELSIF queryonly = 'N' THEN
      New_Form( formname );
    END IF;
  ELSIF newform = 'N' THEN
    IF queryonly = 'Y' THEN
      Call_Form( formname, HIDE, NO_REPLACE, QUERY_ONLY );
    ELSIF queryonly = 'N' THEN
      Call_Form( formname );
    END IF;
  END IF;
END IF;
```
IF NOT Form_Success THEN
    Message('Cannot call form '||upper(formname)||'. Please contact your SysAdmin for help.');
    RAISE Error_Occurred;
END IF;
:System.Message_Level := msglvl;
EXCEPTION
    WHEN Error_Occurred THEN
        :System.Message_Level := msglvl;
        RAISE Form_Trigger_Failure;
END;

NEXT_BLOCK

  Syntax:  NEXT_BLOCK;

  Built-in Type:  restricted procedure

  Enter Query Mode:  no

  Description:  Navigates to the first navigable item in the next enterable block in the navigation sequence. By default, the next block in the navigation sequence is the block with the next higher sequence number, as defined by the order of blocks in the Object Navigator. However, the Next Navigation Block block property can be set to specify a different block as the next block for navigation purposes.

  If there is no enterable block with a higher sequence, NEXT_BLOCK navigates to the enterable block with the lowest sequence number.

  Parameters:  none

  Example:  /*
** Built-in:  NEXT_BLOCK
** Example:  If the current item is the last item in the
** block, then skip to the next block instead of
** the default of going back to the first item in
** the same block
** Trigger:  Key-Next-Item
*/
DECLARE
    cur_itm VARCHAR2(80) := :System.Cursor_Item;
    cur_blk VARCHAR2(80) := :System.Cursor_Block;
    lst_itm VARCHAR2(80);
BEGIN
    lst_itm := cur_blk||'.'||Get_Block_Property(cur_blk,LAST_ITEM);
    IF cur_itm = lst_itm THEN
NEXT_FORM

Syntax: NEXT_FORM;

Built–in Type: restricted procedure

Enter Query Mode: no

Description: In a multiple–form application, navigates to the independent form with the next highest sequence number. (Forms are sequenced in the order they were invoked at runtime.) If there is no form with a higher sequence number, NEXT_FORM navigates to the form with the lowest sequence number. If there is no such form, the current form remains current.

When navigating with NEXT_FORM, no validation occurs and no triggers fire except WHEN–WINDOW–DEACTIVATED, which fires for the form that initiates navigation, and WHEN–WINDOW–ACTIVATED, which fires for the target form.

Restrictions: The target form cannot be a form that is currently disabled as a result of having invoked another form with CALL_FORM.
**NEXT_ITEM**

**Syntax:**
```plaintext
NEXT_ITEM;
```

**Built-in Type:** restricted procedure

**Enter Query Mode:** yes

**Description:**
Navigates to the navigable item with the next higher sequence number than the current item. If there is no such item, NEXT_ITEM navigates to the item with the lowest sequence number. If there is no such item, NEXT_ITEM navigates to the current item.

If the validation unit is the item, NEXT_ITEM validates any fields with sequence numbers greater than the current item or less than the target item.

The function of NEXT_ITEM from the last navigable item in the block depends on the setting of the Navigation Style block property. The valid settings for Navigation Style include:

- **Same Record** (Default): A Next Item operation from a block’s last item moves the input focus to the first navigable item in the block, in that same record.

- **Change Record**: A Next Item operation from a block’s last item moves the input focus to the first navigable item in the block, in the next record. If the current record is the last record in the block and there is no open query, Oracle Forms creates a new record. If there is an open query in the block (the block contains queried records), Oracle forms retrieves additional records as needed.

- **Change Block**: A Next Item operation from a block’s last item moves the input focus to the first navigable item in the first record of the next block.

**Parameters:** none

**Example:**
```plaintext
/*
   ** Built-in: NEXT_ITEM
   ** Example: See NEXT_BLOCK
*/
```
NEXT_KEY

Syntax: NEXT_KEY;

Built-in Type: restricted procedure

Enter Query Mode: yes

Description: Navigates to the enabled and navigable primary key item with the next higher sequence number than the current item. If there is no such item, NEXT_KEY navigates to the enabled and navigable primary key item with the lowest sequence number. If there is no primary key item in the current block, an error occurs.

If the validation unit is the item, NEXT_KEY validates any fields with sequence numbers greater than the current item or less than the target item.

Parameters: none

Example:
/**
 ** Built-in: NEXT_KEY
 ** Example: Jump the cursor to the next primary key item in
 **          in the current block.
 */
BEGIN
   Next_Key;
END;

NEXT_MENU_ITEM

Syntax: NEXT_MENU_ITEM;

Built-in Type: restricted procedure

Description: Navigates to the next menu item in the current menu.

Parameters: none

Restrictions: NEXT_MENU_ITEM is available only in a custom menu running in the full-screen menu display style.
NEXT_RECORD

Syntax: NEXT_RECORD;

Built-in Type: restricted procedure

Enter Query Mode: no

Description: Navigates to the first enabled and navigable item in the record with the next higher sequence number than the current record. If there is no such record, Oracle Forms will fetch or create a record. If the current record is a new record, NEXT_RECORD fails.

Parameters: none

Restrictions: Not allowed in Enter Query mode.

Example:

```sql
/*
** Built-in: NEXT_RECORD
** Example: If the current item is the last item in the
**          block, then skip to the next record instead of
**          the default of going back to the first item in
**          the same block
** Trigger: Key-Next-Item
*/
DECLARE
  cur_itm VARCHAR2(80) := :System.Cursor_Item;
  cur_blk VARCHAR2(80) := :System.Cursor_Block;
  lst_itm VARCHAR2(80);
BEGIN
  lst_itm := cur_blk||'.'||Get_Block_Property(cur_blk,LAST_ITEM);
  IF cur_itm = lst_itm THEN
    Next_Record;
  ELSE
    Next_Item;
  END IF;
END;
```
**NEXT_SET**

Syntax: NEXT_SET;

Built-in Type: restricted procedure

Enter Query Mode: no

Description: Fetches another set of records from the database and navigates to the first record that the fetch retrieves. NEXT_SET succeeds only if a query is open in the current block.

Parameters: none

Example:

```/*
** Built-in: NEXT_SET
** Example: Fetch the next set of records from the database
**          when a button is pressed.
** Trigger: When-Button-Pressed
*/
BEGIN
  Next_Set;
END;```

**OPEN_FORM**

Syntax: OPEN_FORM(form_name);
OPEN_FORM(form_name, activate_mode);
OPEN_FORM(form_name, activate_mode, session_mode);
OPEN_FORM(form_name, activate_mode, session_mode, paramlist_name);
OPEN_FORM(form_name, activate_mode, session_mode, paramlist_id);

Built-in Type: restricted procedure (cannot be called in Enter Query mode)

Enter Query Mode: no

Description: Opens the indicated form. Call OPEN_FORM to create multiple-form applications, that is, applications that open more than one form at the same time.

Parameters: no parameters

`form_name` Specifies the CHAR name of the form to open.

Opens the form and sets focus to it. Equivalent to Open_Form(ACTIVATE, NO_SESSION);
activate_mode

**ACTIVATE** Sets focus to the form to make it the active form in the application.

**NO_ACTIVATE** Opens the form but does not set focus to the form. The current form remains current.

**session_mode**

**NO_SESSION** Specifies that the opened form should share the same database session as the current form. A COMMIT operation in any form will cause validation and commit processing to occur for all forms running in the same session.

**SESSION** Specifies that a new, separate database session should be created for the opened form.

**paramlist_name** Specifies the CHAR name of a parameter list to be passed to the opened form.

**paramlist_id** Specifies the unique ID that Oracle Forms assigns to the parameter list at the time it is created. Use the GET_PARAMETER_LIST function to return the ID to a variable of type PARAMLIST.

**Restrictions:**

- Oracle Forms Runform must be running with the Session option turned On when you execute OPEN_FORM with the `session_mode` parameter set to SESSION. If the Session option is Off, Oracle Forms issues an error and does not open the indicated form.

You can set session On for all Runform invocations by setting the FORMS45_SESSION environment variable to TRUE. When you set the FORMS45_SESSION variable, all Runform invocations inherit its setting, unless you override the environment variable by setting the Session option from the Runform command line.

**Usage Notes:**

- Whether you open a form with ACTIVATE or NO_ACTIVATE specified, any startup triggers that would normally fire will execute in the opened form.

- When you open a form with ACTIVATE specified (the default), the opened form receives focus immediately; trigger statements that follow the call to OPEN_FORM never execute.

- When you open a form with NO_ACTIVATE specified, trigger statements that follow the call to OPEN_FORM will execute after the opened form has been loaded into memory and its initial start-up triggers have fired.

- On Microsoft Windows, if any window in the form that opens the independent form is maximized, the first window displayed by the
opened form will also be maximized, regardless of its original
design–time setting. (The GUI display state of a window is
controlled by the Window_State property.)

- For most applications, you should avoid using OPEN_FORM with
forms that contain root windows. Because there can be only one root
window displayed at a time, canvas–views that are assigned to the
root window in the current form and in the opened form will be
displayed in the same window. This causes the opened form to “take
over” the root window from the original form, thus hiding the
canvas–views in the original form partially or completely.

### PASTE_REGION

**Syntax:**

PASTE_REGION;

**Built–in Type:**

restricted procedure

**Enter Query Mode:**

yes

**Description:**

Pastes the contents of the clipboard (i.e., the selected region that was
cut or copied most recently), positioning the upper left corner of the
pasted area at the cursor position.

**Parameters:**

none

**Usage Notes:**

Use PASTE_REGION, as well as the other editing functions, on text
only. The cut and copy functions transfer the selected region of text
into the system clipboard until you indicate the paste target. At that
time, the cut or copied text is pasted onto the target location.

### PAUSE

**Syntax:**

PAUSE;

**Built–in Type:**

unrestricted procedure

**Enter Query Mode:**

yes

**Description:**

Suspend processing until the operator presses a function key. PAUSE
might display an alert.

**Parameters:**

none
POPULATE_GROUP

Syntax:  

POPULATE_GROUP (recordgroup_id);
POPULATE_GROUP (recordgroup_name);

Built-in Type:  
unrestricted function

Returns:  
NUMBER

Enter Query Mode:  
yes

Description:  
Executes the query associated with the given record group and returns a number indicating success or failure of the query. Upon a successful query, POPULATE_GROUP returns a 0 (zero). An unsuccessful query generates an ORACLE error number that corresponds to the particular SELECT statement failure. The rows that are retrieved as a result of a successful query replace any rows that exist in the group.

Note:  
Be aware that the POPULATE_GROUP array fetches 100 records at a time. To improve network performance, you may want to restrict queries, thus limiting network traffic.

Parameters:  

recordgroup_id  
The unique ID that Oracle Forms assigns when it creates the group. The data type of the ID is RecordGroup.

recordgroup_name  
The name you gave to the record group when creating it. The data type of the name is CHAR.

Restrictions:  
Valid only for record groups

- that were created at design time with a query
- that were created by a call to the CREATE_GROUP_FROM_QUERY built-in
- that have been previously populated with the POPULATE_GROUP_WITH_QUERY built-in (which associates a query with the record group)

Example:  
/*
** Built-in: POPULATE_GROUP
** Example: See GET_GROUP_ROW_COUNT and CREATE_GROUP_FROM_QUERY
*/
POPULATE_GROUP_WITH_QUERY

Syntax: 
POPULATE_GROUP_WITH_QUERY(recordgroup_id, query);
POPULATE_GROUP_WITH_QUERY(recordgroup_name, query);

Built-in Type: unrestricted function

Returns: NUMBER

Enter Query Mode: yes

Description: Populates a record group with the given query. The record group is cleared and rows that are fetched replace any existing rows in the record group.

If the SELECT statement fails, Oracle Forms returns an ORACLE error number. If the query is successful, this built-in returns 0 (zero).

You can use this built-in to populate record groups that were created by a call to either:

- the CREATE_GROUP built-in or
- the CREATE_GROUP_FROM_QUERY built-in

When you use this built-in, the indicated query becomes the default query for the group, and will be executed whenever the POPULATE_GROUP built-in is subsequently called.

Note: Be aware that the POPULATE_GROUP_WITH_QUERY array fetches 100 records at a time. To improve network performance, you may want to restrict queries, thus limiting network traffic.

Parameters:

- recordgroup_id: The unique ID that Oracle Forms assigns when it creates the group. The data type of the ID is RecordGroup.

- recordgroup_name: The name you gave to the record group when creating it. The data type of the name is CHAR.

- query: A valid SELECT statement, enclosed in single quotes. Any columns retrieved as a result of the query take the data types of the columns in the table. If you restrict the query to a subset of the columns in the table, then Oracle Forms creates only those columns in the record group. The data type of the query is CHAR.

Restrictions: You can populate a record group created in the Oracle Forms Designer if the record group is not a static record group.
The columns specified in the SELECT statement must match the record group columns in number and type.

Example:
```c
/*
** Built-in:  POPULATE_GROUP_WITH_QUERY
** Example: See CREATE_GROUP
*/
```

**POPULATE_LIST**

**Syntax:**
- `POPULATE_LIST(list_id, recgrp_id);
- `POPULATE_LIST(list_id, recgrp_name);
- `POPULATE_LIST(list_name, recgrp_id);
- `POPULATE_LIST(list_name, recgrp_name);

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:**
Removes the contents of the current list and populates the list with the values from a record group. The record group must be created at runtime and it must have the following two column (CHAR) structure:

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>the list label</td>
<td>the list value</td>
</tr>
</tbody>
</table>

**Parameters:**
- `list_id` Specifies the unique ID that Oracle Forms assigns when it creates the list item. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.
- `list_name` The name you gave to the list item when you created it. The data type of the name is CHAR.
- `recgrp_id` Specifies the unique ID that Oracle Forms assigns when it creates the record group. The data type of the ID is RecordGroup.
- `recgrp_name` The CHAR name you gave to the record group when you created it.

**Usage Notes:**
- Do not use the POPULATE_LIST built-in if the Other Values property is defined and there are queried records in the block. Doing so may cause Oracle Forms to be unable to display records that have already been fetched.
For example, assume that a list item contains the values A, B, and C and the Other Values property is defined. Assume also that these values have been fetched from the database (a query is open). At this point, if you populate the list using POPULATE_LIST, an error will occur because Oracle Forms will attempt to display the previously fetched values (A, B, and C), but will be unable to because these values were removed from the list and replaced with new values.

- Before populating a list, close any open queries. Use the ABORT_QUERY built-in to close an open query.

**Restrictions:** POPULATE_LIST returns error FRM-41337: Cannot populate the list from the record group if:

- the record group does not contain either the default value element or the other values element and the list does not meet the criteria specified for deleting these elements with DELETE_LIST_ELEMENT. Refer to the restrictions on DELETE_LIST_ELEMENT for more information.

- the record group contains an other value element but the list does not meet the criteria specified for adding an other value element with ADD_LIST_ELEMENT. Refer to the restrictions on ADD_LIST_ELEMENT for more information.

**Example:**

```/*
 ** Built-in:  POPULATE_LIST
 ** Example:   Retrieves the values from the current list item
 **             into record group one, clears the list, and
 **             populates the list with values from record group
 **             two when a button is pressed.
 ** Trigger:   When-Button-Pressed
 */
BEGIN
    Retrieve_List(list_id, 'RECGRP_ONE');
    Clear_List(list_id);
    Populate_List(list_id, 'RECGRP_TWO');
END;```
POST

Syntax: POST;
Built-in Type: restricted procedure
Enter Query Mode: no

Description: Writes data in the form to the database, but does not perform a database commit. Oracle Forms first validates the form. If there are changes to post to the database, for each block in the form Oracle Forms writes deletes, inserts, and updates to the database.

Any data that you post to the database is committed to the database by the next COMMIT_FORM that executes during the current Runform session. Alternatively, this data can be rolled back by the next CLEAR_FORM.

Parameters: none
Example: /* ** Built-in: POST ** Example: See EXIT_FORM */

PREVIOUS_BLOCK

Syntax: PREVIOUS_BLOCK;
Built-in Type: restricted procedure
Enter Query Mode: no

Description: Navigates to the first navigable item in the previous enterable block in the navigation sequence. By default, the previous block in the navigation sequence is the block with the next lower sequence number, as defined by the block order in the Object Navigator. However, the Previous Navigation Block block property can be set to specify a different block as the previous block for navigation purposes.

If there is no enterable block with a lower sequence, PREVIOUS_BLOCK navigates to the enterable block with the highest sequence number.

Parameters: none
Example:

```/*
** Built-in:  PREVIOUS_BLOCK
** Example:   If the current item is the first item in the
**            block, then skip back the previous block
**            instead of the default of going to the last
**            item in the same block
** Trigger:   Key-Previous-Item
*/
DECLARE
  frs_itm VARCHAR2(80);
BEGIN
  frs_itm := cur_blk||'.'||Get_Block_Property(cur_blk,FIRST_ITEM);
  IF cur_itm = frs_itm THEN
    Previous_Block;
  ELSE
    Previous_Item;
  END IF;
END;```

**PREVIOUS_FORM**

**Syntax:**
PREVIOUS_FORM;

**Built-in Type:**
restricted procedure

**Enter Query Mode:**
no

**Description:**
In a multiple-form application, navigates to the form with the next lowest sequence number. (Forms are sequenced in the order they were invoked at runtime.) If there is no form with a lower sequence number, PREVIOUS_FORM navigates to the form with the highest sequence number. If there is no such form, the current form remains current.

When navigating with PREVIOUS_FORM, no validation occurs and no triggers fire except WHEN-WINDOW-DEACTIVATED, which fires for the form that initiates navigation, and WHEN-WINDOW-ACTIVATED, which fires for the target form.

**Restrictions:**
The target form cannot be a form that is currently disabled as a result of having invoked another form with CALL_FORM.
PREVIOUS_ITEM

Syntax:  

Built-in Type: restricted procedure

Enter Query Mode: yes

Description: Navigates to the navigable item with the next lower sequence number than the current item. If there is no such item, PREVIOUS_ITEM navigates to the navigable item with the highest sequence number. If there is no such item, PREVIOUS_ITEM navigates to the current item.

The function of PREVIOUS_ITEM from the first navigable item in the block depends on the setting of the Navigation Style block property. The valid settings for Navigation Style include:

**Same Record** (Default): A Previous Item operation from a block’s first item moves the input focus to the last navigable item in the block, in that same record.

**Change Record**: A Previous Item operation from a block’s first item moves the input focus to the last navigable item in the block, in the previous record.

**Change Block**: A Previous Item operation from a block’s first item moves the input focus to the last navigable item in the current record of the previous block.

Parameters: none

Example:

```*/
  ** Built-in:  PREVIOUS_ITEM
  ** Example: See PREVIOUS_BLOCK
*/```
### PREVIOUS_MENU

**Syntax:** PREVIOUS_MENU;

**Built-in Type:** restricted procedure

**Description:** PREVIOUS_MENU navigates to the previously active item in the previous menu.

**Parameters:** none

**Restrictions:** PREVIOUS_MENU applies only in full-screen and bar menus.

### PREVIOUS_MENU_ITEM

**Syntax:** PREVIOUS_MENU_ITEM;

**Built-in Type:** restricted procedure

**Description:** PREVIOUS_MENU_ITEM navigates to the previous menu item in the current menu.

**Parameters:** none

**Restrictions:** PREVIOUS_MENU_ITEM applies only in full-screen menus.
PREVIOUS_RECORD

Syntax:  PREVIOUS_RECORD;

Built–in Type:  restricted procedure

Enter Query Mode:  no

Description:  Navigates to the first enabled and navigable item in the record with the next lower sequence number than the current record.

Parameters:  none

Example:  
/*
** Built–in:  PREVIOUS_RECORD
** Example:  If the current item is the first item in the block, then skip back to the previous record instead of the default of going to the last item in the same block
** Trigger:  Key-Previous-Item
*/
DECLARE
  cur_itm VARCHAR2(80) := :System.Cursor_Item;
  cur_blk VARCHAR2(80) := :System.Cursor_Block;
  frs_itm VARCHAR2(80);
BEGIN
  frs_itm := cur_blk||'.'||Get_Block_Property(cur_blk,FIRST_ITEM);
  IF cur_itm = frs_itm THEN
    Previous_Record;
  ELSE
    Previous_Item;
  END IF;
END;

PRINT

Syntax:  PRINT;

Built–in Type:  unrestricted procedure

Enter Query Mode:  yes

Description:  Prints the current window to a file or to the printer.

Parameters:  none

Example:  
/*
** Built–in:  PRINT
*/
** Example: Print the current window.

```
BEGIN
  Print;
END;
```

### QUERY_PARAMETER

** Syntax:**

```
QUERY_PARAMETER(parameter_string);
```

** Built-in Type:** unrestricted procedure

** Description:**

Displays the Query Parameter dialog showing the current values of the specified substitution parameters. Operators can set the value of any parameter you include in the list.

The Query Parameter dialog is modal, and control does not return to the calling trigger or procedure until the operator either accepts or cancels the dialog. This means that any PL/SQL statements that follow the call to QUERY_PARAMETER are not executed until the Query Parameter dialog is dismissed.

** Parameters:**

`parameter_string` Specifies a string of substitution parameters for a menu item. The syntax for specifying the `parameter_string` parameter requires the ampersand `&parm_name`. Substitution parameters are referenced in PL/SQL code with the colon syntax `:param_name` used for all bind variables).

** Example:**

```
/*
** Built-in: QUERY_PARAMETER
** Example: Prompt for several menu parameters programmatically, validating their contents.
*/
PROCEDURE Update_Warehouse IS
  validation_Err BOOLEAN;
BEGIN
  WHILE TRUE LOOP
    Query_Parameter(‘&p1 &q2 &z6’);
    /*
    ** If the user did not Cancel the box the Menu_Success function will return boolean TRUE.
    */
    IF Menu_Success THEN
      IF TO_NUMBER( :q2 ) NOT BETWEEN 100 AND 5000 THEN
        Message(‘Qty must be in the range 100..5000’);
        Bell;
      END IF;
    END IF;
  END LOOP;
END;
```
Validation_Err := TRUE;
END IF;
/
** Start a sub-block so we can catch a Value_Error
** exception in a local handler
*/
BEGIN
IF TO_DATE( :z6 ) < SYSDATE THEN
   Message('Target Date must name a day in the future.');
   Bell;
   Validation_Err := TRUE;
END IF;
EXCEPTION
WHEN VALUE_ERROR THEN
   Message('Target Date must be of the form DD-MON-YY');
   Bell;
   Validation_Err := TRUE;
END;
/
** If we get here, all parameters were valid so do the
** Update Statement.
*/
IF NOT Validation_Err THEN
   UPDATE WAREHOUSE
   SET QTY_TO_ORDER = QTY_TO_ORDER*0.18
   WHERE TARGET_DATE = TO_DATE(:z6)
   AND QTY_ON_HAND > TO_NUMBER(:q2)
   AND COST_CODE LIKE :p1 || '%';
END IF;
ELSE
/
** If Menu_Success is boolean false, then return back
** from the procedure since user cancelled the dialog
*/
RETURN;
END IF;
END LOOP;
END;
READ_IMAGE_FILE

Syntax: READ_IMAGE_FILE(file_name, file_type, item_id);
         READ_IMAGE_FILE(file_name, file_type, item_name);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Reads an image of the given type from the given file and displays it in the Oracle Forms image item.

Parameters:

- **file_name** Specifies a valid file name. The file name designation can include a full path statement appropriate to your particular operating system.

- **file_type** Specifies a valid image file type that conforms to the TIFF, JFIF, BMP, PCX, PICT, GIF, CALS, or RAS formats.

- **item_id** Specifies the unique ID that Oracle Forms assigns to the image item when it creates the item. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.

- **item_name** Specifies the name that you gave the image item when defining it. The data type of the name is CHAR.

Usage Notes: Oracle Forms searches for the image file along the same default path as it searches for an .FMX file. For more information on the specific search path for your platform, refer to the Oracle Forms documentation for your operating system.

Example:

```/*
** Built-in:  READ_IMAGE_FILE
** Example:   Read an image from the filesystem into an image
**            item on the form. In this example, the scanned
**            picture identification for each employee is NOT
**            saved to the database, but is stored on the
**            filesystem. An employee’s photo is a TIFF image
**            stored in a file named <Userid>.TIF Each employee’s
**            Userid is unique.
** Trigger:   Post-Query
*/
DECLARE
tiff_image_dir VARCHAR2(80) := '/usr/staff/photos/';
photo_filename VARCHAR2(80);```
BEGIN
   /*
   ** Set the message level high so we can gracefully handle
   ** an error reading the file if it occurs
   */
   :System.Message_Level := '25';
   /*
   ** After fetching an employee record, take the employee’s
   ** Userid and concatenate the ’.TIF’ extension to derive
   ** the filename from which to load the TIFF image. The EMP
   ** record has a non-database image item named ’EMP_PHOTO’
   ** into which we read the image.
   */
   photo_filename := tiff_image_dir||LOWER(:emp.userid)||'.tif';
   /*
   ** For example ’photo_filename’ might look like:
   **
   **   /usr/staff/photos/jgetty.tif
   **
   ** Now, read in the appropriate image.
   */
   Read_Image_File( photo_filename, 'TIFF', 'emp.emp_photo');
   IF NOT Form_Success THEN
      Message('This employee does not have a photo on file.');
   END IF;
   :System.Message_Level := '0';
END;

**Syntax:** REDISPLAY;

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Redraws the screen. This clears any existing system messages displayed on the screen.

**Parameters:** none
**REPLACE_CONTENT_VIEW**

**Syntax:**

REPLACE_CONTENT_VIEW(window_id, view_id);
REPLACE_CONTENT_VIEW(window_name, view_id);
REPLACE_CONTENT_VIEW(window_id, view_name);
REPLACE_CONTENT_VIEW(window_name, view_name);

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Replaces the content canvas–view currently displayed in the indicated window with a different content canvas–view.

**Parameters:**

- **window_id**
  Specifies the unique ID that Oracle Forms assigns the window when created. Use the FIND_WINDOW built-in to return the ID to an appropriately typed variable. The data type of the ID is Window.

- **window_name**
  Specifies the name that you gave the window when creating it. The data type of the name is CHAR.

- **view_id**
  Specifies the unique ID that Oracle Forms assigns the view when it creates the object. Use the FIND_VIEW built-in to return the ID to an appropriately typed variable. The data type of the ID is ViewPort.

- **view_name**
  Specifies the name that you gave the object when defining it. The data type of the name is CHAR.

**Restrictions:**

- The indicated canvas–view must be a content view, not a stacked or toolbar canvas–view.

- The canvas–view that replaces the window’s current content canvas–view must have been assigned to that window at design time. That is, you cannot replace a window’s content view with a content view from a different window.

- If you replace a content canvas–view that contains the item that currently has focus, Oracle Forms will immediately undo the replacement to keep the focus item visible to the operator.

**Example:**

```
/*
** Built-in: REPLACE_CONTENT_VIEW
** Example: Replace the ‘salary’ view with the ‘history’
** view in the ‘employee_status’ window.
*/
```
REPLACE_MENU

Syntax:

REPLACE_MENU;
REPLACE_MENU(menu_module_name);
REPLACE_MENU(menu_module_name, menu_type);
REPLACE_MENU(menu_module_name, menu_type, starting_menu_name);
REPLACE_MENU(menu_module_name, menu_type, starting_menu, group_name);
REPLACE_MENU(menu_module_name, menu_type, starting_menu, group_name, use_file);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Replaces the current menu with the specified menu, but does not make the new menu active. REPLACE_MENU also allows you to change the way the menu displays and the role.

Because REPLACE_MENU does not make the new menu active, Oracle Forms does not allow the menu to obscure any part of the active canvas. Therefore, all or part of the menu does not appear on the screen if the active canvas would cover it.

Usage Notes: REPLACE_MENU replaces the menu for all windows in the application. If you are using CALL_FORM, REPLACE_MENU will replace the menu for both the calling form and the called form with the specified menu.

Parameters:

- **menu_module_name**
  Specifies the menu module that should replace the current menu module. The data type of the name is CHAR. This parameter is optional; if it is omitted, Oracle Forms runs the form without a menu.

- **menu_type**
  Specifies the display style of the menu. The following constants can be passed as arguments for this parameter:
  - **PULL_DOWN** Specifies that you want Oracle Forms to display the menus in a pull-down style
that is characteristic of most GUI platforms and some character mode platforms.

**BAR** Specifies that you want Oracle Forms to display the menu in a bar style horizontally across the top of the root window.

**FULL_SCREEN** Specifies that you want Oracle Forms to display the menu in a full-screen style.

**starting_menu**
Specifies the menu within the menu module that Oracle Forms should use as the starting menu. The data type of the name is CHAR.

**group_name**
Specifies the security role that Oracle Forms is to use. If you do not specify a role name, Oracle Forms uses the current username to determine the role.

**use_file**
Indicates how Oracle Forms should locate the menu .MMX file to be run. Corresponds to the Use File form module property. The data type of use_file is BOOLEAN.

**NULL** Specifies that Oracle Forms should read the current form’s Use File property and execute REPLACE_MENU accordingly. For example, if the form module Use File property is set to True for the current form, Oracle Forms executes REPLACE_MENU as if the use_file actual parameter was TRUE.

**FALSE** Specifies that Oracle Forms should treat the menu_module value as a reference to a .MMB (binary) menu module in the database, and should query this module to get the actual name of the .MMX (executable).

**TRUE** Specifies that Oracle Forms should treat the menu_module value as a direct reference to a .MMX menu runfile in the file system.

**Example:**
```/*
** Built-in: REPLACE_MENU
** Example: Use a standard procedure to change which root menu in the current menu application appears in the menu bar. A single menu application may have multiple "root-menus" which an application can dynamically set at runtime.
*/```
PROCEDURE Change_Root_To(root_menu_name VARCHAR2) IS
    BEGIN
        Replace_Menu(‘MYAPPLSTD’, PULL_DOWN, root_menu_name);
    END;

RESET_GROUP_SELECTION

    Syntax:    RESET_GROUP_SELECTION(recordgroup_id);
                RESET_GROUP_SELECTION(recordgroup_name);

    Built-in Type:  unrestricted procedure

    Enter Query Mode:  yes

    Description:  Deselects any selected rows in the given group. Use this built-in to
deselect all record group rows that have been programatically
marked as selected by executing SET_GROUP_SELECTION on
individual rows.

    Parameters:  
        recordgroup_id    The unique ID that Oracle Forms assigns when it creates the group. The data type of the ID is
                        RecordGroup.
        recordgroup_name  The name you gave to the record group when creating it. The data type of the name is CHAR.

    Example:    /*
                ** Built-in:  RESET_GROUP_SELECTION
                ** Example:  If the user presses the (Cancel) button, forget
                **            all of the records in the ‘USERSEL’ record
                **            group that we may have previously marked as
                **            selected records.
                ** Trigger:  When-Button-Pressed
                */
                BEGIN
                    Reset_Group_Selection( ‘usersel’ );
                END;
RESIZE_WINDOW

Syntax:
RESIZE_WINDOW(window_id, width, height);
RESIZE_WINDOW(window_name, width, height);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Changes the size of the given window to the given width and height. A call to RESIZE_WINDOW sets the width and height of the window, even if the window is not currently displayed. RESIZE_WINDOW does not change the position of the window, as specified by the x and y coordinates of the window’s upper left corner on the screen.

On Microsoft Windows, you can resize the MDI application window by specifying the constant FORMS_MDI_WINDOW as the window name. You can also resize a window with SET_WINDOW_PROPERTY.

Parameters:
- **window_id**: Specifies the unique ID that Oracle Forms assigns the window when created. Use the FIND_WINDOW built-in to return the ID to an appropriately typed variable. The data type of the ID is Window.
- **window_name**: Specifies the name that you gave the window when creating it. The data type of the name is CHAR.
- **width**: Specifies the new width of the window, in form coordinate units.
- **height**: Specifies the new height of the window, in form coordinate units.

Example:
```sql
/*
** Built-in: RESIZE_WINDOW
** Example: Set Window2 to be the same size as Window1
*/
PROCEDURE Make_Same_Size_Win( Window1 VARCHAR2, Window2 VARCHAR2)
IS
    wn_id1 Window;
    w NUMBER;
    h NUMBER;
BEGIN
    /*
    ** Find Window1 and get it’s width and height.
    */
    wn_id1 := Find_Window(Window1);
```

Built-in Subprograms 3 – 233
Syntax:

RETRIEVE_LIST(list_id, recgrp_name);
RETRIEVE_LIST(list_id, recgrp_id);
RETRIEVE_LIST(list_name, recgrp_id);
RETRIEVE_LIST(list_name, recgrp_name);

Built-in Type: unrestricted procedure

Returns: CHAR

Enter Query Mode: yes

Description: Retrieves and stores the contents of the current list into the specified record group. The target record group must have the following two-column (CHAR) structure:

<table>
<thead>
<tr>
<th>Column 1:</th>
<th>Column 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>the list label</td>
<td>the list value</td>
</tr>
</tbody>
</table>

Storing the contents of a list item allows you to restore the list with its former contents.

Parameters:

- **list_id**: Specifies the unique ID that Oracle Forms assigns when it creates the list item. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.

- **list_name**: The name you gave to the list item when you created it. The data type of the name is CHAR.

- **recgrp_id**: Specifies the unique ID that Oracle Forms assigns when it creates the record group. The data type of the ID is RecordGroup.

- **recgrp_name**: The CHAR name you gave to the record group when you created it.

w := Get_Window_Property(wn_id1,WIDTH);
h := Get_Window_Property(wn_id1,HEIGHT);
/*
** Resize Window2 to the same size
*/
Resize_Window(Window2, w, h);
END;

RETRIEVE_LIST

RETRIEVE_LIST(list_id, recgrp_name);
RETRIEVE_LIST(list_id, recgrp_id);
RETRIEVE_LIST(list_name, recgrp_id);
RETRIEVE_LIST(list_name, recgrp_name);
/*
  ** Built-in: RETRIEVE_LIST
  ** Example: See POPULATE_LIST
*/

RUNPRODUCT

Syntax: RUNPRODUCT(product, document, commmode, execmode, location, list, display);
        RUNPRODUCT(product, document, commmode, execmode, location, name, display);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Invokes one of the supported Oracle tools products and specifies the name of the document or module to be run. If the called product is unavailable at the time of the call, Oracle Forms returns a message to the operator.

If you create a parameter list and then reference it in the call to RUNPRODUCT, the form can pass text and data parameters to the called product that represent values for command line parameters, bind or lexical references, and named queries. Parameters of type DATA_PARAMETER are pointers to record groups in Oracle Forms. You can pass DATA_PARAMETERs to Oracle Reports and Oracle Graphics, but not to Oracle Forms or Oracle Book.

Parameters: product Specifies a numeric constant for the Oracle product you want to invoke: FORMS specifies a Runform session. GRAPHICS specifies Oracle Graphics. REPORTS specifies Oracle Reports. BOOK specifies Oracle Book.

document Specifies the CHAR name of the document or module to be executed by the called product. Valid values are the name of a form module, report, Oracle Graphics display, or Oracle Book document. The application looks for the module or document in the default paths defined for the called product.

commmode Specifies the communication mode to be used when running the called product. Valid numeric constants for this parameter are SYNCHRONOUS and ASYNCHRONOUS.
SYNCHRONOUS specifies that control returns to Oracle Forms only after the called product has been exited. The operator cannot work in the form while the called product is running.

ASYNCHRONOUS specifies that control returns to the calling application immediately, even if the called application has not completed its display.

**execmode**

Specifies the execution mode to be used when running the called product. Valid numeric constants for this parameter are BATCH and RUNTIME. When you run Oracle Reports and Oracle Graphics, execmode can be either BATCH or RUNTIME. When you run Oracle Forms, always set execmode to RUNTIME.

**location**

Specifies the location of the document or module you want the called product to execute, either the file system or the database. Valid constants for this property are FILESYSTEM and DB.

**list or name**

Specifies the parameter list to be passed to the called product. Valid values for this parameter are the CHAR name of the parameter list, the ID of the parameter list, or NULL. To specify a parameter list ID, use a variable of type PARAMLIST.

**Note:** You can pass text parameters to called products in both SYNCHRONOUS and ASYNCHRONOUS mode. However, parameter lists that contain parameters of type DATA_PARAMETER (pointers to record groups) can only be passed to Oracle Reports and Oracle Graphics in SYNCHRONOUS mode. (SYNCHRONOUS mode is required when invoking Oracle Graphics to return an Oracle Graphics display that will be displayed in a form chart item.)

**Note:** You can prevent Oracle Graphics from logging on by passing a parameter list that includes a parameter with key set to LOGON and value set to NO.

**Note:** You cannot pass a DATA_PARAMETER to a child query in Oracle Reports. Data passing is supported only for master queries.
Specifies the CHAR name of the Oracle Forms chart item that will contain the display (such as a pie chart, bar chart, or graph) generated by Oracle Graphics. The name of the chart item must be specified in the format \texttt{block\_name.item\_name}. (This parameter is only required when you are using an Oracle Graphics chart item in a form.)

Example:

```plaintext
/*
** Built-in:  RUN_PRODUCT
** Example:   Call an Oracle Reports 2.5 report, passing the
data in record group 'EMP_RECS' to substitute
for the report's query named 'EMP_QUERY'.
** Presumes the Emp_ReCS record group already
exists and has the same column/data type
** structure as the report's Emp_ReCS query.
*/
PROCEDURE Run_Emp_Report IS
   pl_id ParamList;
BEGIN
   /*
   ** Check to see if the 'tmpdata' parameter list exists.
   */
   pl_id := Get_Parameter_List('tmpdata');
   /*
   ** If it does, then delete it before we create it again in
   ** case it contains parameters that are not useful for our
   ** purposes here.
   */
   IF NOT Id_Null(pl_id) THEN
      Destroy_Parameter_List(pl_id);
   END IF;
   /*
   ** Create the 'tmpdata' parameter list afresh.
   */
   pl_id := Create_Parameter_List('tmpdata');
   /*
   ** Add a data parameter to this parameter list that will
   ** establish the relationship between the named query
   ** 'EMP_QUERY' in the report, and the record group named
   ** 'EMP_RECS' in the form.
   */
   Add_Parameter(pl_id,’EMP_QUERY’,DATA_PARAMETER,’EMP_RECS’);
   /*
   ** Run the report synchronously, passing the parameter list
   */
   Run_Product(REPORTS, 'empreport', SYNCHRONOUS, RUNTIME, FILESYSTEM, pl_id, NULL);
END;
```
**SCROLL_DOWN**

Syntax: `SCROLL_DOWN;`

Built–in Type: restricted procedure

Enter Query Mode: no

Description: Scrolls the current block’s list of records so that previously hidden records with higher sequence numbers are displayed. If there are available records and a query is open in the block, Oracle Forms fetches records during SCROLL_DOWN processing. In a single–line block, SCROLL_DOWN displays the next record in the block’s list of records. SCROLL_DOWN puts the input focus in the instance of the current item in the displayed record with the lowest sequence number.

Parameters: none

Example:
```
/*
 ** Built-in:   SCROLL_DOWN
 ** Example:    Scroll records down some.
 */
BEGIN
 Scroll_Down;
 END;
```

**SCROLL_UP**

Syntax: `SCROLL_UP;`

Built–in Type: restricted procedure

Enter Query Mode: no

Description: Scrolls the current block’s list of records so that previously hidden records with lower sequence numbers are displayed. This action displays records that were “above” the block’s display.

SCROLL_UP puts the input focus in the instance of the current item in the displayed record that has the highest sequence number.

Parameters: none

Example:
```
/*
 ** Built-in:   SCROLL_UP
 ** Example:    Scroll records up some.
 */
```
BEGIN
    Scroll_Up;
END;

SCROLL_VIEW

Syntax:  

SCROLL_VIEW(view_id, x, y);
SCROLL_VIEW(view_name, x, y);

Built-in Type:  unrestricted procedure

Enter Query Mode:  yes

Description:  Moves the view to a different position on its canvas by changing the X Position on Canvas and Y Position on Canvas properties. Moving the view makes a different area of the canvas visible to the operator, but does not change the position of the view within the window.

Note: For a content or toolbar canvas-view, the window in which the canvas-view is displayed represents the view for that canvas. For a stacked canvas-view, the view size is controlled by setting the View Width and View Height properties.

Parameters:

view_id  Specifies the unique ID that Oracle Forms assigns the view when it creates the object. Use the FIND_VIEW built-in to return the ID to an appropriately typed variable. The data type of the ID is ViewPort.

view_name  Specifies the name that you gave the object when defining it. The data type of the name is CHAR.

x  Specifies the x coordinate of the view’s upper left corner relative to the upper left corner of the canvas.

y  Specifies the y coordinate of the view’s upper left corner relative to the upper left corner of the canvas.

Example:

/*
 ** Built-in:  SCROLL_VIEW
 ** Example:  Scroll the view whose name is passed in 10% to
 **            the right or left depending on the 'direction'
 **            parameter.
 */
PROCEDURE Scroll_Ten_Percent( viewname VARCHAR2,
direction VARCHAR2 ) IS
vw_id        ViewPort;
vw_wid       NUMBER;
vw_x         NUMBER;
cn_id        Canvas;
cn_wid       NUMBER;
ten_percent  NUMBER;
new_x        NUMBER;
old_y        NUMBER;
BEGIN
¦/*
¦** Get the id’s for the View and its corresponding canvas
¦*/
vw_id := Find_View( viewname );
cn_id := Find_Canvas( viewname );
¦/*
¦** Determine the view width and corresponding canvas
¦** width.
¦*/
vw_wid := Get_View_Property(vw_id,WIDTH);
cn_wid := Get_Canvas_Property(cn_id,WIDTH);
¦/*
¦** Calculate how many units of canvas width are outside of
¦** view, and determine 10% of that.
¦*/
ten_percent := 0.10 * (cn_wid – vw_wid);
¦/*
¦** Determine at what horizontal position the view
¦** currently is on the corresponding canvas
¦*/
vw_x := Get_View_Property(vw_id,X_POS_ON_CANVAS);
¦/*
¦** Calculate the new x position of the view on its canvas
¦** to effect the 10% scroll in the proper direction.
¦** Closer than ten percent of the distance to the edge
¦** towards which we are moving, then position the view
¦** against that edge.
¦*/
IF direction = 'LEFT' THEN
IF vw_x > ten_percent THEN
   new_x := vw_x – ten_percent;
ELSE
   new_x := 0;
END IF;
ELSIF direction = 'RIGHT' THEN
IF vw_x < cn_wid – vw_wid – ten_percent THEN
   new_x := vw_x + ten_percent;
ELSE
new_x := cn_wid - vw_wid;
END IF;
END IF;
/
** Scroll the view that much horizontally */
old_y := Get_View_Property(vw_id,Y_POS_ON_CANVAS);
Scroll_View(vw_id, new_x, old_y);
END;

SELECT_ALL

Syntax: SELECT_ALL;

Built-in Type: restricted procedure

Enter Query Mode: yes

Description: Selects the text in the current item. Call this procedure prior to issuing a call to CUT_REGION or COPY_REGION, when you want to cut or copy the entire contents of a text item.

Parameters: none
SELECT_RECORDS

Syntax: SELECT_RECORDS;

Built-in Type: restricted procedure

Enter Query Mode: no

Description: When called from an On-Select trigger, initiates default Oracle Forms SELECT processing. This built-in is included primarily for applications that run against a non-ORACLE data source, and use transactional triggers to replace default Oracle Forms transaction processing.

Parameters: none

Restrictions: Valid only within an On-Select trigger.

Example:

```
/*
** Built-in: SELECT_RECORDS
** Example: Perform Oracle Forms standard SELECT processing based on a global flag setup at startup by the form, perhaps based on a parameter.
** Trigger: On-Select
*/
BEGIN
    /*
    ** Check the flag variable we setup at form startup
    */
    IF :Global.Using_Transactional_Triggers = 'TRUE' THEN
        User_Exit('my_select block=EMP');
    /*
    ** Otherwise, do the right thing.
    */
    ELSE
        Select_Records;
    END IF;
END;
```
**SET_ALERT_BUTTON_PROPERTY**

**Syntax:**

```plaintext
SET_ALERT_BUTTON_PROPERTY(alert_id, button, property, value);
SET_ALERT_BUTTON_PROPERTY(alert_name, button, property, value);
```

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Changes the label on one of the buttons in an alert.

**Parameters:**

- `alert_id` Specifies the unique ID (data type ALERT) that Oracle Forms assigns to the alert when it is created. Use FIND_ALERT to return the ID to an appropriately typed variable.
- `alert_name` Specifies the CHAR name of the alert.
- `button` A constant that specifies the alert button you want to change, either ALERT_BUTTON1, ALERT_BUTTON2, or ALERT_BUTTON3.
- `property` `LABEL` Specifies the label text for the alert button.
- `value` Specifies the CHAR value to be applied to the property you specified.

**Usage Notes:** If the label specified is NULL, the button’s label reverts to the label specified at design time.
SET_ALERTPROPERTY

Syntax:

SET_ALERTPROPERTY(alert_id, property, message);
SET_ALERTPROPERTY(alert_name, property, message);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Changes the message text for an existing alert.

Parameters:

alert_id  Specifies the unique ID (data type ALERT) that Oracle Forms assigns to the alert when it is created. Return the ID to an appropriately typed variable.

alert_name  Specifies the CHAR name of the alert.

property  Specifies the specific alert property you are setting:

  ALERT_MESSAGE_TEXT  Specifies that you are setting the text of the alert message.
  TITLE  Specifies the title of the alert. Overrides the value specified in the Oracle Forms Designer unless the property value is NULL.

message  Specifies the message that is to replace the current alert message. Pass the message as a string enclosed in single quotes, as a variable, or in a string/variable construction.

Restrictions: If the message text exceeds 80 characters, it will be truncated.

Example:

/*
** Built-in: SET_ALERTPROPERTY
** Example: Places the error message into a user-defined alert named ‘My_Error_Alert’ and displays the alert.
** Trigger: On-Error
*/
DECLARE
  err_txt VARCHAR2(80) := Error_Text;
  al_id Alert;
  al_button Number;
BEGIN
  al_id := Find_Alert('My_Error_Alert');
  Set_Alert_Property(al_id, alert_message_text, err_txt);
  al_button := Show_Alert(al_id);
END;
SET_APPLICATION_PROPERTY

Syntax:  

SET_APPLICATION_PROPERTY(property, value)

Built-in Type:  

unrestricted procedure

Enter Query Mode:  

yes

Description:  

Sets the application property for the current application.

Parameters:  

property  

Specifies the property you want to set for the given application. The possible properties are as follows:

CURSOR_STYLE  Specifies the cursor style for the given application.

value  

The following GUI-specific constants can be passed as CHAR arguments to the property values described earlier:

BUSY  Specifies a busy symbol.

CROSSHAIR  Specifies a crosshair symbol.

DEFAULT  Specifies an arrow symbol.

HELP  Specifies a help symbol.

INSERTION  Specifies an insertion symbol.
SET_BLOCK_PROPERTY

Syntax:  
SET_BLOCK_PROPERTY(block_id, property, value);
SET_BLOCK_PROPERTY(block_name, property, value);

Built-in Type:  unrestricted procedure

Enter Query Mode:  yes

Description:  Sets the given block characteristic of the given block.

Parameters:  
- **block_id**: Specifies the unique ID Oracle Forms assigns when it creates the block. The data type of the ID is Block.
- **block_name**: Specifies the CHAR name that you gave to the block when defining it. The data type of the name is CHAR.
- **property**: 
  
  **COORDINATION_STATUS** Specifies a status that indicates whether a block that is a detail block in a master-detail relation is currently coordinated with all of its master blocks; that is, whether the detail records in the block correspond correctly to the current master record in the master block. Valid values are COORDINATED and NON_COORDINATED.

  **CURRENT_RECORD_ATTRIBUTE** Specify the CHAR name of a named visual attribute to be associated with the given block. If the named visual attribute does not exist, you will get an error message.

  **DEFAULT_WHERE** Specifies a default WHERE clause for the block, overriding any previous WHERE clause. Enclose in single quotes. The WHERE reserved word is optional. The default WHERE clause can include references to global variables, form parameters, and item values, specified with standard bind variable syntax.

  **DELETE_ALLOWED** Specifies whether the operator or the application is allowed to delete records in the given block. Valid values are PROPERTY_TRUE or PROPERTY_FALSE.

  **INSERT_ALLOWED** Specifies whether the operator or the application is allowed to insert
records in the given block. Valid values are PROPERTY_TRUE or PROPERTY_FALSE.

**KEY_MODE** Specifies the key mode for the block. This is particularly useful when running Oracle Forms against non-ORACLE data sources. Valid values are UPDATEABLE_PRIMARY_KEY and NONUPDATEABLE_PRIMARY_KEY.

**LOCKING_MODE** Specifies the block’s LOCKING_MODE property. Valid values are DELAYED or IMMEDIATE.

**NAVIGATION_STYLE** Specifies the block’s NAVIGATION_STYLE property. Valid values are SAME_RECORD, CHANGE_RECORD, or CHANGE_BLOCK.

**NEXT_NAVIGATION_BLOCK** Specifies the name of the block’s next navigation block. By default, the next navigation block is the block with the next higher sequence number; however, the NEXT_NAVIGATION_BLOCK block property can be set to override the default block navigation sequence.

**OPTIMIZER_HINT** Specifies a hint that Oracle Forms passes on to the RDBMS optimizer when constructing queries. This allows the form designer to achieve the highest possible performance when querying blocks.

**ORDER_BY** Specifies a default ORDER BY clause for the block, overriding any prior ORDER BY clause. Enclose in single quotes but do not include the actual words ‘ORDER BY’. Oracle Forms automatically prefixes the statement you supply with “ORDER BY.”

**PREVIOUS_NAVIGATION_BLOCK** Specifies the name of the block’s previous navigation block. By default, the previous navigation block is the block with the next lower sequence number; however, the NEXT_NAVIGATION_BLOCK block property can be set to override the default block navigation sequence.

**PRIMARY_KEY** Specifies that any record inserted or updated in the block must have a unique
characteristic in order to be committed to the
database. Valid values are PROPERTY_TRUE or
PROPERTY_FALSE.

**QUERY_ALLOWED** Specifies whether a query
can be issued from the block, either by an operator
or programmatically. Valid values are
PROPERTY_TRUE or PROPERTY_FALSE.

**QUERY_HITS** Specifies the NUMBER value that
indicates the number of records identified by the
COUNT_QUERY operation.

**UPDATE_ALLOWED** Specifies whether the
operator or the application is allowed to update
records in the given block. Valid values are
PROPERTY_TRUE or PROPERTY_FALSE.

The following constants can be passed as
arguments to the property values described earlier:

**COORDINATED** Specifies that the
COORDINATION_STATUS property should be set
to COORDINATED for a block that is a detail
block in a master–detail relation.

**DELAYED** Specifies that you want Oracle Forms
to lock detail records only at the execution of a
commit action.

**IMMEDIATE** Specifies that you want Oracle
Forms to lock detail records immediately whenever
a database record has been modified.

**NON_COORDINATED** Specifies that the
COORDINATION_STATUS property should be set
to NON_COORDINATED for a block that is a
detail block in a master–detail relation.

**NON_UPDATEABLE_PRIMARY_KEY** Specifies
that you want Oracle Forms to process records in
the block on the basis that the underlying data
source does not allow primary keys to be updated.

**PROPERTY_TRUE** Specifies that the property is
to be set to the TRUE state. Specifically, supply as
the value for DELETE_ALLOWED,
INSERT_ALLOWED, QUERY_HITS, and
UPDATE_ALLOWED.
**PROPERTY_FALSE** Specifies that the property is to be set to the FALSE state.

**UNIQUE_KEY** Specifies that you want Oracle Forms to process records in the block on the basis that the underlying data source uses some form of unique key, or ROWID.

**UPDATE_CHANGED_COLUMNS** Specifies that only those columns updated by an operator will be sent to the database. When Update Changed Columns is set to False, all columns are sent, regardless of whether they have been updated. This can result in considerable network traffic, particularly if the block contains a LONG data type.

**UPDATEABLE_PRIMARY_KEY** Specifies that you want Oracle Forms to process records in the block on the basis that the underlying data source allows for primary keys to be updated.

---

**Example:**

```sql
/*
** Built-in:  SET_BLOCK_PROPERTY
** Example:   Prevent future inserts, updates, and deletes to
**            queried records in the block whose name is
**            passed as an argument to this procedure.
**/
PROCEDURE Make_Block_Query_Only( blk_name IN VARCHAR2 ) IS
  blk_id Block;
BEGIN
  /* Lookup the block’s internal ID */
  blk_id := Find_Block(blk_name);
  /*
  ** If the block exists (ie the ID is Not NULL) then set
  ** the three properties for this block. Otherwise signal
  ** an error.
  */
  IF NOT Id_Null(blk_id) THEN
    Set_Block_Property(blk_id,INSERT_ALLOWED,PROPERTY_FALSE);
    Set_Block_Property(blk_id,UPDATE_ALLOWED,PROPERTY_FALSE);
    Set_Block_Property(blk_id,DELETE_ALLOWED,PROPERTY_FALSE);
  ELSE
    Message('Block ' || blk_name || ' does not exist.');
    RAISE Form_Trigger_Failure;
  END IF;
END;
```
SET_CANVAS_PROPERTY

Syntax:

SET_CANVAS_PROPERTY(canvas_id, property, value);
SET_CANVAS_PROPERTY(canvas_id, property, x);
SET_CANVAS_PROPERTY(canvas_id, property, x, y);
SET_CANVAS_PROPERTY(canvas_name, property, value);
SET_CANVAS_PROPERTY(canvas_name, property, x);
SET_CANVAS_PROPERTY(canvas_name, property, x, y);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Sets the given canvas property for the given canvas.

Parameters:

`canvas_id` Specifies the unique ID that Oracle Forms assigns the canvas when it creates the object. Use the FIND_CANVAS built-in to return the ID to an appropriately typed variable. The data type of the ID is Canvas.

`canvas_name` Specifies the name that you gave the object when defining it. The data type of the name is CHAR.

`property` Specifies the property you want to set for the given canvas. The possible properties are as follows:

- **CANVAS_SIZE** Specifies a width, height pair indicating the size of the canvas.
- **HEIGHT** Specifies the height in characters of the canvas.
- **VISUAL_ATTRIBUTE** Specifies either a valid named visual attribute that exists in the current form, or the name of a logical attribute definition in a runtime resource file that you want Oracle Forms to apply to the canvas.
- **WIDTH** Specifies the width in characters of the canvas.

`value` Specifies the CHAR value to be applied to the property you specified.

`x` Specifies the NUMBER value of the x coordinate or the width, depending on the property you specified. Specify the argument in form coordinate system units.
y

Specifies the NUMBER value of the y coordinate or the height, depending on the property you specified. Specify the argument in form coordinate system units.

Restrictions:

• You cannot enter negative numbers for the x, y, width, or height parameters.
• You cannot enter a non-existent named visual attribute.
• If Oracle Forms cannot find a named visual attribute by the name you supply, it looks for the display attribute in your Oracle*Terminal resource file.

Example:

```/*
** Built-in:  SET_CANVAS_PROPERTY
** Example:   Change the “background color” by setting the
**            canvas color dynamically at runtime to the name
**            of a visual attribute you created.
**            BEGIN
**            Set_Canvas_Property('my_main_cnv',VISUAL_ATTRIBUTE,'blue_text');
**            END;
*/```
visual attribute does not exist, you will get an error message.

**CURSOR_MODE** Specifies the cursor state Oracle Forms should attempt to define. Primarily used when connecting to non–ORACLE data sources. Valid values are OPEN_AT_COMMIT and CLOSE_AT_COMMIT.

**DEFER_REQUIRED_ENFORCEMENT** Specifies whether enforcement of required fields has been deferred from item validation to record validation. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

**DIRECTION** Specifies the layout direction for bidirectional objects. Valid values are DIRECTION_DEFAULT, RIGHT_TO_LEFT, LEFT_TO_RIGHT.

**FIRST_NAVIGATION_BLOCK** Returns the name of the block into which Oracle Forms attempts to navigate at form startup. By default, the first navigation block is the first block defined in the Object Navigator; however, the FIRST_NAVIGATION_BLOCK block property can be set to specify a different block as the first block at form startup.

**SAVEPOINT_MODE** Specifies whether Oracle Forms is to issue savepoints. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

**VALIDATION** Specifies whether Oracle Forms is to perform default validation. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

**VALIDATION_UNIT** Specifies the scope of validation for the form. Valid values are DEFAULT_SCOPE, BLOCK_SCOPE, RECORD_SCOPE, and ITEM_SCOPE.

The following constants can be passed as arguments to the property values described earlier:

**BLOCK_SCOPE** Specify when you want Oracle Forms to validate data only at the block level. This means, for instance, that Oracle Forms validates all
the records in a block when a navigation event forces validation by leaving the block.

**CLOSE_AT_COMMIT** Specify when you do not want cursors to remain open across database commits; for example, when a form is running against a non–ORACLE database.

**DEFAULT_SCOPE** Sets the Validation Unit form module property to the default setting. On GUI window managers, the default validation unit is ITEM.

**FORM_SCOPE** Specify when you want validation to occur at the form level only.

**ITEM_SCOPE** Specify when you want Oracle Forms to validate at the item level. This means, for instance, that Oracle Forms validates each changed item upon navigating out of an item as a result of a navigation event.

**OPEN_AT_COMMIT** Specify when you want cursors to remain open across database commits. This is the normal setting when running against ORACLE.

**PROPERTY_TRUE** Specifies that the property is to be set to the TRUE state.

**PROPERTY_FALSE** Specifies that the property is to be set to the FALSE state.

**RECORD_SCOPE** Specify when you want Oracle Forms to validate at the record level. This means that Oracle Forms validates each changed record when, for instance, it leaves the record.

Example 1:

```/*
** Built-in:  SET_FORM_PROPERTY
** Example:   Set the Cursor Mode property in the current form
**            to CLOSE_AT_COMMIT and changes the form
**            Validation unit to the Block level.
**/
DECLARE
fm_id FormModule;
BEGIN
  fm_id := Find_Form(:System.Current_Form);
  Set_Form_Property(fm_id,CURSOR_MODE,CLOSE_AT_COMMIT);
  Set_Form_Property(fm_id,VALIDATION_UNIT,BLOCK_SCOPE);
END;```
Example 2:

```
/*
** Built-in:  SET_FORM_PROPERTY
** Example:   Setup form and block properties required to run
**            against a particular non-Oracle datasource.
**            Procedure accepts the appropriate numerical
**            constants like DELAYED as arguments.
**
** Usage:     Setup_Non_Oracle(PROPERTY_FALSE,
**                             CLOSE_AT_COMMIT,
**                             UPDATEABLE_PRIMARY_KEY,
**                             DELAYED);
*/
PROCEDURE Setup_Non_Oracle( the_savepoint_mode NUMBER,
                             the_cursor_mode    NUMBER,
                             the_key_mode       NUMBER,
                             the_locking_mode   NUMBER ) IS

  fm_id   FormModule;
  bk_id   Block;
  bk_name VARCHAR2(40);
BEGIN
  /* ** Validate the settings of the parameters ** */
  IF the_savepoint_mode NOT IN (PROPERTY_TRUE,PROPERTY_FALSE) THEN
    Message('Invalid setting for Savepoint Mode.');
    RAISE Form_Trigger_Failure;
  END IF;
  IF the_cursor_mode NOT IN (CLOSE_AT_COMMIT,OPEN_AT_COMMIT) THEN
    Message('Invalid setting for Cursor Mode.');
    RAISE Form_Trigger_Failure;
  END IF;
  IF the_key_mode NOT IN (UNIQUE_KEY,UPDATEABLE_PRIMARY_KEY,
                          NON_UPDATEABLE_PRIMARY_KEY) THEN
    Message('Invalid setting for Key Mode.');
    RAISE Form_Trigger_Failure;
  END IF;
  IF the_locking_mode NOT IN (IMMEDIATE,DELAYED) THEN
    Message('Invalid setting for Locking Mode.');
    RAISE Form_Trigger_Failure;
  END IF;
  /*
  ** Get the id of the current form
  */
  fm_id := Find_Form(:System.Current_Form);
  /*
  ** Set the two form-level properties
  */
  Set_Form_Property(fm_id, SAVEPOINT_MODE, the_savepoint_mode);
  Set_Form_Property(fm_id, CURSOR_MODE, the_cursor_mode);
  /*
  ** Set the block properties for each block in the form
```
/*
   bk_name := Get_Form_Property(fm_id,FIRST_BLOCK);
   WHILE bk_name IS NOT NULL LOOP
       bk_id := Find_Block(bk_name);

       Set_Block_Property(bk_id,LOCKING_MODE,the_locking_mode);

       Set_Block_Property(bk_id,KEY_MODE,the_key_mode);
       IF the_key_mode IN (UPDATEABLE_PRIMARY_KEY,
               NON_UPDATEABLE_PRIMARY_KEY) THEN
           Set_Block_Property(bk_id,PRIMARY_KEY,PROPERTY_TRUE);
       END IF;

       bk_name := Get_Block_Property(bk_id, NEXTBLOCK);
   END LOOP;
END;

SET_GROUP_CHAR_CELL

Syntax:  SET_GROUP_CHAR_CELL(groupcolumn_id, row_number, cell_value);
         SET_GROUP_CHAR_CELL(groupcolumn_name, row_number, cell_value);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Sets the value for the record group cell identified by the given row and column.

Parameters: 

   groupcolumn_id     The unique ID that Oracle Forms assigns when it creates the column for the record group. The data type of the ID is GroupColumn.

   groupcolumn_name   The name you gave to the column when you created it. The data type of the name is CHAR.

   row_number         Specifies the row number that contains the cell whose value you intend to set. Specify as a whole NUMBER.

   cell_value         For a CHAR column, specifies the CHAR value you intend to enter into a cell; for a LONG column, specifies the LONG value you intend to enter into a cell.
Restrictions:  
- The data type of the value you set must be the same as or convertible to the data type of its corresponding column.
- You must create the specified row before setting the value of a cell in that row. Oracle Forms does not automatically create a new row when you indicate one in this built-in. Explicitly add the row with the ADD_GROUP_ROW built-in or populate the group with either POPULATE_GROUP or POPULATE_GROUP_WITH_QUERY.
- Not valid for a static record group. A static record group is a record group that was created at design time and that has the Record Group Type property set to Static.

Example:  
```/*  
** Built-in:  SET_GROUP_DATE_CELL  
** Example:  See ADD_GROUP_ROW  
*/  
```

---

**SET_GROUP_DATE_CELL**

**Syntax:**  
```python  
SET_GROUP_DATE_CELL(groupcolumn_id, row_number, cell_value);  
SET_GROUP_DATE_CELL(groupcolumn_name, row_number, cell_value);  
```

**Built-in Type:**  
unrestricted procedure

**Enter Query Mode:**  
yes

**Description:**  
Sets the value for the record group cell identified by the given row and column.

**Parameters:**  
- `groupcolumn_id`  
  The unique ID that Oracle Forms assigns when it creates the column for the record group. The data type of the ID is GroupColumn.
- `groupcolumn_name`  
  The name you gave to the column when you created it. The data type of the name is CHAR.
- `row_number`  
  Specifies the row number that contains the cell whose value you intend to set. Specify as a whole NUMBER.
- `cell_value`  
  Specifies the DATE value you intend to enter into a cell.

**Restrictions:**  
- The data type of the value you set must be the same as or convertible to the data type of its corresponding column.
• You must create the specified row before setting the value of a cell in that row. Oracle Forms does not automatically create a new row when you indicate one in this built-in. Explicitly add the row with the ADD_GROUP_ROW built-in or populate the group with either POPULATE_GROUP or POPULATE_GROUP_WITH_QUERY.

• Not valid for a static record group. A static record group is a record group that was created at design time and that has the Record Group Type property set to Static.

Example:

```
/*
** Built-in: SET_GROUP_DATE_CELL
** Example: Lookup a row in a record group, and set the
**           minimum order date associated with that row in
**           the record group. Uses the 'is_value_in_list'
**           function from the GET_GROUP_CHAR_CELL example.
*/
PROCEDURE Set_Max_Order_Date_Of( part_no  VARCHAR2,
                                 new_date DATE ) IS
                    fnd_row NUMBER;
BEGIN
  /*
   ** Try to lookup the part number among the temporary part list
   ** record group named 'TMPPART' in its 'PARTNO' column.
   */
  fnd_row := Is_Value_In_List( part_no, 'TMPPART', 'PARTNO');
  IF fnd_row = 0 THEN
    Message('Part Number '||part_no|| ' not found.');
    RETURN;
  ELSE
    /*
     ** Set the corresponding Date cell value from the
     ** matching row.
     */
    Set_Group_Date_Cell('TMPPART.MAXORDDATE',fnd_row,new_date );
    END IF;
END;
```
SET_GROUP_NUMBER_CELL

Syntax: 
SET_GROUP_NUMBER_CELL(groupcolumn_id, row_number, cell_value);
SET_GROUP_NUMBER_CELL(groupcolumn_name, row_number, cell_value);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Sets the value for the record group cell identified by the given row and column.

Parameters:

- **groupcolumn_id**: The unique ID that Oracle Forms assigns when it creates the column for the record group. The data type of the ID is GroupColumn.
- **groupcolumn_name**: The name you gave to the column when you created it. The data type of the name is CHAR.
- **row_number**: Specifies the row number that contains the cell whose value you intend to set. Specify as a whole NUMBER.
- **cell_value**: Specifies the NUMBER value you intend to enter into a cell.

Restrictions:

- The data type of the value you set must be the same as the data type of its corresponding column.
- You must create the specified row before setting the value of a cell in that row. Explicitly add a row with the ADD_GROUP_ROW built-in or populate the group with either POPULATE_GROUP or POPULATE_GROUP_WITH_QUERY.
- Not valid for a static record group. A static record group is a record group that was created at design time and that has the Record Group Type property set to Static.

Example:

```/*
** Built-in: SET_GROUP_NUMBER_CELL
** Example: See ADD_GROUP_ROW
*/```
SET_GROUP_SELECTION

Syntax: SET_GROUP_SELECTION(recordgroup_id, row_number);
        SET_GROUP_SELECTION(recordgroup_name, row_number);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Marks the specified row in the given record group for subsequent programmatic row operations. Rows are numbered sequentially starting at 1. If you select rows 3, 8, and 12, for example, those rows are considered by Oracle Forms to be selections 1, 2, and 3. You can undo any row selections for the entire group by calling the RESET_GROUP_SELECTION built-in.

Parameters:

recordgroup_id Specifies the unique ID that Oracle Forms assigns to the record group when created. Use the FIND_GROUP built-in to return the ID to a variable. The data type of the ID is RecordGroup.

recordgroup_name Specifies the name of the record group that you gave to the group when creating it. The data type of the name is CHAR.

row_number Specifies the number of the record group row that you want to select. The value you specify is a NUMBER.

Example:

/*
** Built-in: SET_GROUP_SELECTION
** Example: Set all of the even rows as selected in the record group whose id is passed-in as a parameter.
*/
PROCEDURE Select_Even_Rows ( rg_id RecordGroup ) IS
BEGIN
    FOR j IN 1..Get_Group_Row_Count(rg_id) LOOP
        IF MOD(j,2)=0 THEN
            Set_Group_Selection( rg_id, j );
        END IF;
    END LOOP;
END;
SET_INPUT_FOCUS

Syntax: 

```
SET_INPUT_FOCUS(MENU);
```

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Sets the input focus on the menu of the current form. Once trigger processing is completed, Oracle Forms activates the menu.

Parameters: 

- **MENU**

Restrictions: Only for use in character mode and block mode environments.

Example: 

```
/*
** Built-in:  SET_INPUT_FOCUS
** Example:   Directs the users input focus to the Menu when
**             used with the only support parameter, MENU.
**             Only has an effect on character-mode or
**             block-mode devices.
*/
BEGIN
  Set_Input_Focus(MENU);
END;
```

SET_ITEM_PROPERTY

Syntax: 

```
SET_ITEM_PROPERTY(item_id, property, value);
SET_ITEM_PROPERTY(item_id, property, value, x, y);
SET_ITEM_PROPERTY(item_name, property, value);
SET_ITEM_PROPERTY(item_name, property, value, x, y);
SET_ITEM_PROPERTY(item_id, property, x);
SET_ITEM_PROPERTY(item_name, property, x, y);
SET_ITEM_PROPERTY(item_id, property, x);
SET_ITEM_PROPERTY(item_name, property, x);
```

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Modifies all instances of an item in a block by changing a specified item property. You may be able to get but not set certain object properties.

Parameters: 

- **item_id** 
  Specifies the unique ID that Oracle Forms assigns the object when created. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.
Specifies the name you gave the item when creating it. The data type of the name is CHAR.

Specifies the property you want to set for the given item. The possible properties are as follows:

ALIGNMENT  Specifies the text alignment for text items and display items only. Valid values are ALIGNMENT_START, ALIGNMENT_END, ALIGNMENT_LEFT, ALIGNMENT_CENTER, ALIGNMENT_RIGHT.

AUTO_HINT  Specifies whether help hints should automatically be displayed on the status line when the input focus is in the indicated item. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

AUTO_SKIP  Specifies whether the cursor should skip to the next item automatically when the operator enters the last character in a text item. Valid only for a text item. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

CASE_INSENSITIVE_QUERY  Specifies whether query conditions entered in the item should be case-sensitive. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

CASE_RESTRICTION  Specifies the case restriction applied to any text entered in the indicated text item. Valid values are UPPERCASE, LOWERCASE, or NONE.

CURRENT_RECORD_ATTRIBUTE  Specifies the CHAR name of a named visual attribute to be associated with the given item. If the named visual attribute does not exist, you will get an error message.

DIRECTION  Specifies the layout direction for bidirectional objects. Valid values are DIRECTION_DEFAULT, RIGHT_TO_LEFT, LEFT_TO_RIGHT.

DISPLAYED  Specifies whether the indicated item should be displayed or hidden. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.
Note: Setting Displayed to False will cause other item property settings to change. Consult the "Propagation of Property Changes" section for details.

ECHO Specifies whether characters an operator types into a text item should be visible. When Echo is False, the characters typed are hidden. Used for password protection. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

ENABLED Specifies whether operators should be able to manipulate an item. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

Note: Setting Enabled to False will cause other item property settings to change. Consult the "Propagation of Property Changes" section for details.

FIXED_LENGTH Specifies whether the item’s value should be validated against the setting of the item’s Max Length property. When FIXED_LENGTH is True, the item is valid only if the number of characters in its value equals the Max Length setting. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

FORMAT_MASK Specifies the display format and input accepted for data in text items.

HEIGHT Specifies the height of the item.

ICON_NAME Specifies the file name of the icon resource associated with a button item having the iconic property set to ON.

INSERT_ALLOWED In a new record, allows operator to insert items normally when set to PROPERTY_TRUE. Specify PROPERTY_FALSE to specify that the item does not accept modification, but is displayed normally (not grayed out). (Insert_Allow does not propagate changes to the Enabled property.)

ITEM_IS_VALID Specifies whether the current item should be considered valid. Set to PROPERTY_TRUE or PROPERTY_FALSE.
ITEM_SIZE  Specifies a width and height for the item as two numbers separated by a comma. Use the syntax that includes \( x, y \).

KEEP_POSITION  Specifies whether the Keep Position property should be True or False. When Keep Position is True, the cursor returns to the same position it was in when it left the text item. When Keep Position is False, the cursor returns to the default position in the text item. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

LABEL  Specifies the CHAR string that you want displayed as the label of the item. This property is only valid for items that have labels, such as buttons.

LOCK_RECORD_ON_CHANGE  Specify the constant PROPERTY_TRUE if you want the record to be locked when this item is changed. Specify the constant PROPERTY_FALSE if you do not want the record locked when this item is changed. Use primarily when connecting to a non–ORACLE data source that does not have row–level locking.

LOV_NAME  Specify the CHAR name of an LOV to be associated with the given item. If the LOV name does not exist, you will get an error message.

LOV_VALIDATION  Specifies that Oracle Forms should validate the value of the text item against the values in the attached LOV when set to PROPERTY_TRUE. Specify PROPERTY_FALSE to specify that Oracle Forms should not use the LOV for validation.

MOUSE_NAVIGATE  Specifies whether Oracle Forms should navigate and set focus to the item when the operator activates the item with the mouse. Specify the constant PROPERTY_TRUE if you want the operator to be able to navigate to the item using the mouse. Specify the constant PROPERTY_FALSE if you want a mouse click to keep the input focus in the current item.

NAVIGABLE  Specify the constant PROPERTY_TRUE if you want the operator to be able to navigate to the item using default keyboard
navigation. Specify the constant
PROPERTY_FALSE if you want to disable default
keyboard navigation to the item. (Navigable does
not propagate changes to the Enabled property.)

**NEXT_NAVIGATION_ITEM** Returns the name of
the item that is defined as the “next navigation
item” with respect to this current item.

**POPUPMENU_CONTENT_ITEM** Specifies the
setting for any of the OLE popup menu item
properties:

- POPUPMENU_COPY_ITEM
- POPUPMENU_CUT_ITEM
- POPUPMENU_DELOBJ_ITEM
- POPUPMENU_INSOBJ_ITEM
- POPUPMENU_LINKS_ITEM
- POPUPMENU_OBJECT_ITEM
- POPUPMENU_PASTE_ITEM
- POPUPMENU_PASTESPEC_ITEM

Specify the character string HIDDEN for the OLE
popup menu item not to be displayed on the OLE
popup menu. Specify the character string
ENABLED for the OLE popup menu item to be
displayed and enabled. Specify the character
string DISABLED for the OLE popup menu item to
be displayed and not enabled.

**POSITION** Specify the x, y coordinates for the
item as NUMBERS separated by a comma. Use the
syntax that includes x, y.

**PREVIOUS_NAVIGATION_ITEM** Returns the
name of the item that is defined as the “previous
navigation item” with respect to this current item.

**PRIMARY_KEY** Specify the constant
PROPERTY_TRUE to indicate that any record
inserted or updated in the block must have a
unique characteristic in order to be committed to
the database. Otherwise, specify the constant
PROPERTY_FALSE.
QUERYABLE Specify the constant PROPERTY_TRUE if you want the operator to be able to initiate a query against the item. Specify the constant PROPERTY_FALSE if you want to disallow the use of the item in a query.

QUERY_ONLY Specify an item to be queried, preventing that item from becoming part of insert or update statements. QUERY_ONLY is applicable to text items, radio groups, and check boxes. Enclose the fully-qualified item name in single quotes.

REQUIRED Specify the constant PROPERTY_TRUE if you want to force the operator to enter a value for the item. Specify the constant PROPERTY_FALSE if the item is not to be required.

SECURE Specify the constant PROPERTY_TRUE if you want the item to remain blank or otherwise obscured when the operator enters a value. Specify the constant PROPERTY_FALSE if you want any value that is typed into the text item to be visible.

UPDATEABLE Specify the constant PROPERTY_TRUE if you want the operator to be able to update the item. Specify the constant PROPERTY_FALSE if you want the item protected from update. (Corresponds to the Update_Allowed property.)

UPDATE_COLUMN Specify the constant PROPERTY_TRUE if this column should be treated as updated, and included in the columns to be written to the database. Specify the constant PROPERTY_FALSE if this column should be treated as not updated, and not be included in the columns to be written to the database.

UPDATE_NULL Specify the constant PROPERTY_TRUE if you want the operator to be able to update the item only if its value is NULL. Specify the constant PROPERTY_FALSE if you want the operator to be able to update the value of the item regardless of whether the value is NULL.
**UPDATE_PERMISSION** Use UPDATE_ALLOWED when you run against non-ORACLE data sources. Specify the constant PROPERTY_TRUE to turn on the item’s UPDATEABLE and UPDATE_NULL properties. Specify the constant PROPERTY_FALSE to turn off the item’s UPDATEABLE and UPDATE_NULL properties.

**VISUAL_ATTRIBUTE** Specify a valid named visual attribute that exists in the current form.

**Note:** You cannot set the visual attribute for an image item.

**WIDTH** Specify the width of the item as a NUMBER. The size of the units depends on how you set the Coordinate System property and default font scaling for the form.

**X_POS** Specify the x coordinate as a NUMBER.

**Y_POS** Specify the y coordinate as a NUMBER.

Specify the value to be applied to the given property. The data type of the property determines the data type of the value you enter. For instance, if you want to set the DISPLAYED property to True, you specify the constant PROPERTY_TRUE for the value. If you want to change the LABEL for the item, you specify the value, in other words, the label, as a CHAR string.

**PROPERTY_TRUE** Specifies that the property is to be set to the TRUE state.

**PROPERTY_FALSE** Specifies that the property is to be set to the FALSE state.

**x** Specifies the NUMBER value of the x coordinate or the width, depending on the property you specified. Specify the argument in form coordinate system units.

**y** Specifies the NUMBER value of the y coordinate or the height, depending on the property you specified. Specify the argument in form coordinate system units.
Usage Notes: The following issues can affect your decisions on how to apply certain property values to an item:

- validation of property changes
- propagation of property changes

Validation of Property Changes When you specify a change through the SET_ITEM_PROPERTY built-in, Oracle Forms validates the change before it adjusts the property. If the change is validated, Oracle Forms makes the change and leaves it in effect until another SET_ITEMPROPERTY changes the same property or the current form is exited.

Illegal Settings If the change is not validated, Oracle Forms issues an error message. You cannot use SET_ITEMPROPERTY to set the following item properties True or False, given the following target item conditions.

<table>
<thead>
<tr>
<th>You can't set this property parameter...</th>
<th>To this restricted setting</th>
<th>If this target item condition is true:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(All) True/False</td>
<td></td>
<td>• NULL-canvas item (item's canvas property is null)</td>
</tr>
<tr>
<td>ENABLED True/False</td>
<td>True/False</td>
<td>• current item</td>
</tr>
<tr>
<td>NAVIGABLE True/False</td>
<td>True/False</td>
<td>• current item</td>
</tr>
<tr>
<td>DISPLAYED True/False</td>
<td>True/False</td>
<td>• Displayed item property is False</td>
</tr>
<tr>
<td>UPDATEABLE True/False (Update Allowed)</td>
<td>True True</td>
<td>• Enabled item property is False</td>
</tr>
<tr>
<td>UPDATE_NULL True/False (Update if NULL)</td>
<td>True True</td>
<td>• Secure item property is True</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Enabled item property is False</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Secure item property is True</td>
</tr>
</tbody>
</table>
You can’t set this property parameter... To this restricted setting If this target item condition is true:

<table>
<thead>
<tr>
<th>QUERYABLE (Query Allowed) True</th>
<th>• Displayed item property is False</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUIRED True True</td>
<td>• Enabled item property is False</td>
</tr>
<tr>
<td>• Update Allowed and Update if NULL item properties are set to False</td>
<td></td>
</tr>
</tbody>
</table>

Oracle Forms does not consider the current contents of an item before allowing a property change. If SET_ITEM_PROPERTY changes an item property that would affect how Oracle Forms validates the data in an item (for example, FIXED_LENGTH or REQUIRED), the validation consequences are not retroactive. The new validation rules do not apply to the item until Oracle Forms next validates it under normal circumstances.

For example, suppose the application has a required text item, such as Employee ID. In the application, the operator needs to be able to leave this item (behavior not allowed for a REQUIRED item), so you temporarily set the REQUIRED property to False. At this point, Oracle Forms marks an existing NULL value as VALID. Later in the application, when you set the REQUIRED property to True again, Oracle Forms does not automatically change the VALID/INVALID marking. In order to have a NULL value marked as INVALID (expected for a REQUIRED item), you must make a change in the item that will cause Oracle Forms to validate it, such as:

```
IF :block.item IS NULL
THEN :block.item := NULL;
```

**Propagation of Property Changes** You can only specify a change to one item property at a time through the SET_ITEM_PROPERTY built-in. However, one SET_ITEM_PROPERTY statement can cause changes to more than one item property if the additional changes are necessary to complete, or propagate, the intended change. This is included primarily for compatibility with prior versions. You should upgrade your existing properties to the new properties to take advantage of advanced features.

The following table shows the SET_ITEM_PROPERTY settings that cause Oracle Forms to propagate changes across item properties:
### Example:

```sql
/*
** Built-in:  SET_ITEM_PROPERTY
** Example:  Change the icon of an iconic button dynamically
** at runtime by changing its icon_name. The user
** clicks on this button to go into enter query
** mode, then clicks on it again (after the icon
** changed) to execute the query. After the query
** is executed the user sees the original icon
** again.
** Trigger:   When-Button-Pressed
*/
DECLARE
   it_id Item;
BEGIN
   it_id := Find_Item('CONTROL.QUERY_BUTTON');
   IF :System.Mode = 'ENTER-QUERY' THEN
      /*
      ** Change the icon back to the enter query icon, and
      ** execute the query.
      */
      Set_Item_Property(it_id,ICON_NAME,'entquery');
      Execute_Query;
   ELSE
      /*
```
** Change the icon to the execute query icon and get into enter query mode. */
Set_Item_Property(it_id,ICON_NAME,'exequery');
Enter_Query;
END IF;
END;

---

**SET_LOV_COLUMN_PROPERTY**

**Syntax:**

```
SET_LOV_COLUMN_PROPERTY(lov_id, colnum, property, value);
SET_LOV_COLUMN_PROPERTY(lov_name, colnum, property, value);
```

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Sets the given LOV property for the given LOV.

**Parameters:**

- **lov_id**
  Specifies the unique ID that Oracle Forms assigns the LOV when created. Use the FIND_LOV built-in to return the ID to an appropriately typed variable. The data type of the ID is LOV.

- **lov_name**
  Specifies the LOV name (as a CHAR).

- **colnum**
  Specifies the column to be modified (as an INTEGER). Column numbers are 1-based. That is, the first column is column 1.

- **property**
  Specifies the property you want to set for the given LOV. The possible properties are as follows:

  - **TITLE**
    Sets the Column Title property that controls the title that displays above an LOV column.

  - **VALUE**
    The CHAR or NUMBER value that represents the desired property setting.

  - **WIDTH**
    Specifies the width to be reserved in the LOV for displaying column values.

**Note:** Setting the column title to NULL resets the column title to the title specified at design time.

**Note:** Setting the column width to NULL results in a hidden, or non-displayed, column.
**SET_LOVPROPERTY**

**Syntax:**

```plaintext
SET_LOVPROPERTY(lovid, property, value);
SET_LOVPROPERTY(lov_name, property, value);
SET_LOVPROPERTY(lovid, property, x, y);
SET_LOVPROPERTY(lov_name, property, x, y);
```

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Sets the given LOV property for the given LOV.

**Parameters:**

- `lovid` Specifies the unique ID that Oracle Forms assigns the LOV when created. Use the FIND_LOV built-in to return the ID to an appropriately typed variable. The data type of the ID is LOV.
- `lov_name` Specifies the LOV name (as a CHAR).
- `property` Specifies the property you want to set for the given LOV. The possible properties are as follows:
  - `AUTO_REFRESH` Specifies whether Oracle Forms re-executes the query each time the LOV is invoked.
  - `GROUP_NAME` Specifies the record group with which the LOV is associated.
  - `LOV_SIZE` Specifies a width, height pair indicating the size of the LOV.
  - `POSITION` Specifies an x, y pair indicating the position of the LOV.
  - `TITLE` Specifies the title of the LOV. Overrides the value specified in the Oracle Forms Designer unless the property value is NULL.
- `value` Specify one of the following constants:
  - `PROPERTY_TRUE` Specifies that the property is to be set to the TRUE state.
  - `PROPERTY_FALSE` Specifies that the property is to be set to the FALSE state.
  - `Recordgroup Name` Specify the CHAR name of the record group you are setting. You can create this record group in the Designer or programmatically, as long as the
record group exists when the
SET_LOV_PROPERTY is called.

\( x \)  Specify either the x coordinate or the width,
depending on the property you specified.

\( y \)  Specify either the y coordinate or the height,
depending on the property you specified.

Restrictions:
• You cannot enter negative numbers for the x, y, width, or height
  parameters.
• You can set only one property per call to the built-in.

Example:
/*
** Built-in:  SET_LOV_PROPERTY
** Example:   if LOV is currently base on GROUP1,
**            make LOV use GROUP2
 */
DECLARE
  lov_id      LOV;
BEGIN
  lov_id     := Find_LOV('My_LOV_1');
  IF Get_LOV_Property(lov_id,GROUP_NAME) = 'GROUP1' THEN
    Set_LOV_Property(lov_id,GROUP_NAME,'GROUP2');
  ENDIF;
END;

SET_MENU_ITEM_PROPERTY

Syntax:
SET_MENU_ITEM_PROPERTY(menuitem_id, property, value);  
SET_MENU_ITEM_PROPERTY(menu_name.menuitem_name, property, value);

Built-in Type:  unrestricted procedure

Enter Query Mode:  yes

Description:  Modifies the given properties of a menu item.

Parameters:  

\( menuitem_id \)  Specifies the unique ID Oracle Forms assigns when
it creates the menu item. Use the
FIND_MENU_ITEM built-in to return the ID to an
appropriately typed variable. The data type of the
ID is MenuItem.

\( menu_name \)  Specifies the CHAR name you gave to the menu
item when you defined it. If you specify the menu
item by name, include the qualifying menu name, as shown in the syntax.

**property**

Specify one of the following constants to set information about the menu item:

**CHECKED** Specifies the Checked property, which indicates if a check box menu item or a radio menu item is in the checked state or unchecked state.

**DISPLAYED** Specifies whether the menu item is visibly displayed.

**ENABLED** Specifies whether the menu item is enabled (thus active) or disabled (thus greyed out and unavailable to the operator).

**LABEL** Specifies the character string for the menu item label.

**value**

Specify one of the following constants:

**PROPERTY_TRUE** Specifies that the property is to be set to the TRUE state.

**PROPERTY_FALSE** Specifies that the property is to be set to the FALSE state.

**Label**

Specify the CHAR label name.

**Restrictions:** These restrictions apply only if the menu module’s Use Security property is set to True:

- If the menu module Use Security property is True, whether you can set the property of a menu item using SET_MENU_ITEM_PROPERTY depends on whether the form operator has access privileges for that item.

- If the menu item is hidden and the operator does not have security access to a menu item, Runform does not display that item. You cannot set the property of a menu item using SET_MENU_ITEM_PROPERTY if the item is currently hidden.

- If the menu item is displayed, but disabled and the Display w/o Priv property for this menu item was set in the Designer, Runform displays the item in a disabled state. In this case, you can set the menu item properties programmatically.

**Example:**

```c *
** Built-in:  SET_MENU_ITEM_PROPERTY
** Example:  See GET_MENU_ITEM_PROPERTY
*/```
SET_PARAMETER_ATTR

Syntax:

SET_PARAMETER_ATTR(list, key, paramtype, value);
SET_PARAMETER_ATTR(name, key, paramtype, value);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Sets the type and value of an indicated parameter in an indicated parameter list.

Parameters:

list or name Specifies the parameter list. The actual parameter can be either a parameter list ID of type PARAMLIST, or the CHAR name of the parameter list.

key The CHAR name of the parameter.

paramtype Specifies the type of parameter you intend to pass:

DATA_PARAMETER Indicates that the parameter’s value is the name of a record group.

TEXT_PARAMETER Indicates that the parameter’s value is an actual data value.

value The value of the parameter specified as a CHAR string.

SET_RADIO_BUTTON_PROPERTY

Syntax:

SET_RADIO_BUTTON_PROPERTY(item_id, button_name, property, value);
SET_RADIO_BUTTON_PROPERTY(item_id, button_name, property, x, y);
SET_RADIO_BUTTON_PROPERTY(item_name, button_name, property, x, y);
SET_RADIO_BUTTON_PROPERTY(item_name, button_name, property, value);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Sets the given property for a radio button that is part of the given radio group specified by the item_name or item_id.

Parameters:

item_id Specifies the radio group item ID. Oracle Forms assigns the unique ID at the time it creates the
object. Use the FIND_ITEM built–in to return the ID to an appropriately typed variable.

item_name

Specifies the name of the radio group. The radio group is the owner or parent of its subordinate radio buttons. The data type of the name is CHAR.

button_name

Specifies the name of the radio button whose property you want to set. The data type of the name is CHAR.

property

Specifies the property you want to set. The possible property constants you can set are as follows:

DISPLAYED Specify PROPERTY_TRUE constant if you want the radio button to be displayed. Specify PROPERTY_FALSE constant if you want the radio button to be hidden.

ENABLED Specify PROPERTY_TRUE constant if you want to enable the radio button. Specify PROPERTY_FALSE if you want to disable the radio button from operator control.

HEIGHT Specify the height of the given radio button. Specify the value as a number.

ITEM_SIZE Sets the width and height of the given radio button. Use the syntax that shows an x,y coordinate pair and specify the values as numbers.

LABEL Specify the actual string label for that radio button.

POSITION Sets the position of the given radio button. Use the syntax that shows an x,y coordinate pair and specify the values as numbers.

VISUAL_ATTRIBUTE Specifies either a valid named visual attribute that exists in the current form, or the name of a logical attribute definition in a runtime resource file that you want Oracle Forms to apply to the radio button.

WIDTH Specify the width of the given radio button. Specify the value as a number.

X_POS Specify the x–coordinate for the radio button. Specify the value as a number.
**Y_POS** Specify the y-coordinate for the radio button. Specify the value as a number.

value Specifies a NUMBER or a CHAR value. The data type of the value you enter is determined by the data type of the property you specified. If you enter a CHAR value, you must enclose it in quotes, unless you reference a text item or variable.

**PROPERTY_TRUE** Specifies that the property is to be set to the TRUE state.

**PROPERTY_FALSE** Specifies that the property is to be set to the FALSE state.

x Specifies the first numeric value for the ITEM_SIZE and POSITION properties.

y Specifies the second numeric value for the ITEM_SIZE and POSITION properties.

Example: /*
** Built-in: SET_RADIO_BUTTON_PROPERTY
** Example: Set a particular radio button to disabled.
*/
BEGIN
    Set_Radio_Button_Property('MYBLOCK.FLIGHT_STATUS',
        'GROUNDED',ENABLED,PROPERTY_FALSE);
END;

---

**SET_RECORD_PROPERTY**

**Syntax:** SET_RECORD_PROPERTY(record_number, block_name, property, value);

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Sets the specified record property to the specified value.

**Parameters:**

- **record_number** Specifies the number of the record whose status you want to set. The record number is the record’s position in the block. Specify as a whole number.

- **block_name** Specifies the name of the block in which the target record exists. The data type of the name is CHAR.

- **property** Use the following property:
STATUS Specifies that you intend to change the record status. STATUS is a constant.

Use one of the following values:

CHANGED_STATUS Specifies that the record should be marked for update and should be treated as an update when the next commit action occurs.

INSERT_STATUS Specifies that the record is to be marked as an INSERT and should be inserted into the appropriate table when the next commit action occurs.

NEW_STATUS Specifies that the record is to be treated as a NEW record, that is, a record that has not been marked for insert, update, or query.

QUERY_STATUS Specifies that the record is to be treated as a QUERY record, whether it actually is. See also the CREATE_QUERIED_RECORD built-in.

Restrictions: The following table illustrates the valid transition states of a record.

<table>
<thead>
<tr>
<th>Current Status</th>
<th>Target Status</th>
<th>NEW</th>
<th>QUERY</th>
<th>INSERT</th>
<th>CHANGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW</td>
<td></td>
<td>yes</td>
<td>yes^1</td>
<td>yes^2</td>
<td>no</td>
</tr>
<tr>
<td>QUERY</td>
<td></td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>INSERT</td>
<td></td>
<td>yes</td>
<td>yes^3</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>CHANGED</td>
<td></td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

1. Adheres to the rules described in footnotes 2 and 3.
2. This transition is not allowed in query mode, because QUERY and INSERT are not valid in query mode.
3. If this transition is performed while Runform is running in Unique Key mode and not all of the transactional triggers exist, then you must enter a valid value in the ROWID field. Put another way, if you are connected to a non-ORACLE data source that does not support ROWID, but you are using a unique key, you must supply the key for a record that goes from Insert to Query, in one of the transactional triggers, either On-Lock, On-Update, or On-Delete. Otherwise Oracle Forms returns an error.
Example: /*
** Built-in: SET_RECORD_PROPERTY
** Example: Mark the third record in the EMP block as if it
**           were a queried record.
*/
BEGIN
  Set_Record_Property( 3, 'EMP', STATUS, QUERY_STATUS);
END;

SET_RELATION_PROPERTY

Syntax: SET_RELATION_PROPERTY(relation_id, property, value);
         SET_RELATION_PROPERTY(relation_name, property, value);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Sets the given relation property in a master-detail relationship.

Parameters:

relation_id Specifies the unique ID that Oracle Forms assigns the relation when it creates the relation object. This can occur automatically when you define a master-detail relationship in the Oracle Forms Designer, or you can explicitly create the relation in the Designer. The data type of the ID is Relation.

relation_name Specifies the name you or Oracle Forms gave the relation object when defining it. The data type of the name is CHAR.

property Use one of the following relation properties, which can be passed to the built-in as a constant:

AUTOQUERY Specifies that the detail block of this relation is to be automatically coordinated upon instantiation of the block. This allows potentially expensive processing to be deferred until blocks that are involved in relations are actually visited. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

DEFERRED_COORDINATION Specifies that a block requiring coordination is to be marked but not coordinated until the detail blocks are instantiated. Deferred coordination refers only to the population phase of coordination. Even
deferred detail blocks are cleared during the clear phase of coordination to present the form in a visually consistent state. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

**MASTER_DELETES** Specifies the default relation behavior for deletion of a detail record in the detail block when there is a corresponding master record in the master block. Valid values are NON-ISOLATED, ISOLATED, or CASCADING. The ability to set this property programmatically is included only for designers who are coding their own master-detail coordination. It does not alter a default relation that was created at design time.

**PREVENT_MASTERLESS_OPERATION**
Specifies that operations in a detail block are not allowed when no corresponding master record exists. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

The following constants can be supplied for the properties described earlier:

**CASCADING** Specifies that the MASTER_DELETES property is to be set so that when an operator deletes a master record, its corresponding detail records are locked at the same time as the master records are locked.

**ISOLATED** Specifies that the MASTER_DELETES property is to be set so that an operator can delete a master record for which detail records exist. This does not cause subsequent locking and deletion of detail records, however, Oracle Forms still initiates detail block coordination in this case.

**NON_ISOLATED** Specifies that the MASTER_DELETES property is to be set so that if the operator attempts to delete a master record for which detail records exist, Oracle Forms issues an error message and disallows the deletion.

**PROPERTY_TRUE** Specifies that the property is to be set to the TRUE state.

**PROPERTY_FALSE** Specifies that the property is to be set to the FALSE state.
Restrictions: You can only set one property per call to this built-in.

Example:
/*
** Built-in:  SET_RELATION_PROPERTY
** Example:   Set the coordination behavior of a relation to
**            be deferred, and auto-query.
*/
PROCEDURE Make_Relation_Deferred( rl_name VARCHAR2 ) IS
  rl_id Relation;
BEGIN
  /*
  ** Look for the relation’s ID
  */
  rl_id := Find_Relation( rl_name );
  /*
  ** Set the two required properties
  */
  Set_Relation_Property(rl_id,AUTOQUERY,PROPERTY_TRUE);
END;

SET_TIMER

Syntax:      SET_TIMER(timer_id, milliseconds, iterate);
             SET_TIMER(timer_name, milliseconds, iterate);

Built-in Type:  unrestricted procedure

Enter Query Mode:  yes

Description: Changes the settings for an existing timer. You can modify the interval,
              the repeat parameter, or both.

Parameters:  

  timer_id  Specifies the unique ID that Oracle Forms assigns
             when it creates the timer, specifically as a response
             to a successful call to the CREATE_TIMER built-in.
             Use the FIND_TIMER built-in to return the ID to
             an appropriately typed variable. The data type of
             the ID is Timer.

  timer_name  Specifies the name you gave the timer when you
                defined it. The data type of the name is CHAR.

  milliseconds  Specifies the duration of the timer in milliseconds.
                The range of values allowed for this parameter is 1
                to 2147483648 milliseconds. Values > 2147483648
                will be rounded down to 2147483648. Note that
                only positive numbers are allowed. The data type
of the parameter is NUMBER. See Restrictions below for more information.

**NO_CHANGE** Specifies that the milliseconds property is to remain at its current setting.

\textit{iterate} Specifies the iteration of the timer.

**REPEAT** Indicates that the timer should repeat upon expiration. Default.

**NO_REPEAT** Indicates that the timer should not repeat upon expiration, but is to be used once only, until explicitly called again.

**NO_CHANGE** Specifies that the iterate property is to remain at its current setting.

### Restrictions:

- If you intend to change only the REPEAT/NO_REPEAT parameter, you must indicate NO_CHANGE for the interval, unless you directly reference the REPEAT/NO_REPEAT parameter.
- Values > 2147483648 will be rounded down to 2147483648.
- A value less than 1 results in a runtime error.
- A value greater than the stated upper bound results in an integer overflow.
- Milliseconds cannot be expressed as a negative number.
- No two timers can share the same name in the same form instance, regardless of case.
- If there is no When–Timer–Expired trigger defined at the execution of a timer, Oracle Forms returns an error.
- If there is no When–Timer–Expired trigger defined at the execution of a timer, and the timer is a repeating timer, subsequent repetitions are canceled, but the timer is retained.

### Example:

```c
/*
** Built-in: SET_TIMER
** Example: See FIND_TIMER
*/```
**SET_VIEW_PROPERTY**

**Syntax:**

```
SET_VIEW_PROPERTY(view_id, property, value);
SET_VIEW_PROPERTY(view_id, property, x, y);
SET_VIEW_PROPERTY(view_name, property, value);
SET_VIEW_PROPERTY(view_name, property, x, y);
```

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Sets a property for the indicated canvas-view. You can set only one property per call to the built-in. In other words, you cannot split the argument in such a way that the x coordinate applies to X_POS and the y coordinate applies to the HEIGHT.

**Parameters:**

- `view_id` Specifies the unique ID that Oracle Forms assigns the view when it creates the object. Use the FIND_VIEW built-in to return the ID to an appropriately typed variable. The data type of the ID is ViewPort.

- `view_name` Specifies the name that you gave the object when defining it. The data type of the name is CHAR

- `property` Specifies one of the following properties:
  - **DIRECTION** Specifies the layout direction for bidirectional objects. Valid values are DIRECTION_DEFAULT, RIGHT_TO_LEFT, LEFT_TO_RIGHT.
  - **DISPLAY_POSITION** For a stacked view, specifies the position of the view’s upper–left corner relative to the window’s content view, as an x, y pair. Determines where the view is displayed in the window.
  - **DISPLAY_X_POS** For a stacked view, specifies the x coordinate for the view’s upper–left corner relative to the window’s content view.
  - **DISPLAY_Y_POS** For a stacked view, specifies the y coordinate for the view’s upper–left corner relative to the window’s content view.
  - **HEIGHT** For a stacked canvas-view, specifies the height of the view. To change the size of the canvas itself, use SET_CANVAS_PROPERTY.
**POSITION_ON_CANVAS** Specifies an x, y pair indicating the location of the view’s upper–left corner relative to its canvas.

**VIEW_SIZE** For a stacked canvas–view, specifies the size of the view, as a width, height pair. To change the size of the canvas itself, use `SET_CANVAS_PROPERTY`.

**VISIBLE** Specifies whether the view is to be displayed. Valid values are `PROPERTY_TRUE` and `PROPERTY_FALSE`.

**WIDTH** For a stacked canvas–view, specifies the width of the view. To change the size of the canvas itself, use `SET_CANVAS_PROPERTY`.

**X_POS_ON_CANVAS** Specifies the x coordinate for the view’s upper–left corner relative to its canvas.

**Y_POS_ON_CANVAS** Specifies the y coordinate for the view’s upper–left corner relative to its canvas.

Specify the value appropriate to the property you are setting:

**PROPERTY_TRUE** Specifies that the property is to be set to the TRUE state.

**PROPERTY_FALSE** Specifies that the property is to be set to the FALSE state.

**x** Specifies the NUMBER value of the x coordinate or the width, depending on the property you specified. Specify the argument in form coordinate system units.

**y** Specifies the NUMBER value of the y coordinate or the height, depending on the property you specified. Specify the argument in form coordinate system units.

**Example:**

```
/*
** Built-in:  SET_TIMER
** Example:  See GET_VIEW_PROPERTY and FIND_VIEW
*/
```
**SET_WINDOW_PROPERTY**

**Syntax:**

```
SET_WINDOW_PROPERTY(window_id, property, value);
SET_WINDOW_PROPERTY(window_id, property, x);
SET_WINDOW_PROPERTY(window_id, property, x, y);
SET_WINDOW_PROPERTY(window_name, property, value);
SET_WINDOW_PROPERTY(window_name, property, x);
SET_WINDOW_PROPERTY(window_name, property, x, y);
```

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Sets a property for the indicated window.

**Parameters:**

- `window_id` Specifies the unique ID that Oracle Forms assigns the window when created. Use the FIND_WINDOW built-in to return the ID to an appropriately typed variable. The data type of the ID is Window.

- `window_name` Specifies the name that you gave the window when creating it. The data type of the name is CHAR.

- `property` Specify one of the following window properties:
  - **DIRECTION** Specifies the layout direction for bidirectional objects. Valid values are DIRECTION_DEFAULT, RIGHT_TO_LEFT, LEFT_TO_RIGHT.
  - **HEIGHT** Specifies the height of the window.
  - **POSITION** Specifies an x, y pair indicating the location for the window on the screen.
  - **REMOVE_ON_EXIT** Specifies whether Oracle Forms hides the current window automatically when the operator navigates to an item in another window. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.
  - **TITLE** Sets the title of the window.
  - **VISIBLE** Specifies whether the window is to be displayed. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.
  - **WINDOW_SIZE** Specifies a width, height pair indicating the size of the window on the screen.
**WINDOW_STATE** Specifies the current display state of the window. Valid values are NORMAL, MAXIMIZE, or MINIMIZE.

**WIDTH** Specifies the width of the window.

**X_POS** Sets the x coordinate for the window’s upper left corner on the screen.

**Y_POS** Sets the y coordinate for the window’s upper left corner on the screen.

The following constants can be passed as arguments to the property values described earlier:

**PROPERTY_TRUE** Specifies that the property is to be set to the TRUE state. This applies specifically to the VISIBLE property.

**PROPERTY_FALSE** Specifies that the property is to be set to the FALSE state. This applies specifically to the VISIBLE property.

The following constants can be passed as arguments for use with the WINDOW_STATE property:

**NORMAL** Specifies that the window is displayed normally according to the current Width, Height, X Position, and Y Position property settings.

**MAXIMIZE** Specifies that the window is enlarged to fill the screen according to the display style of the window manager.

**MINIMIZE** Specifies that the window is minimized, or iconified.

**x** Specifies the NUMBER value of the x coordinate or the width, depending on the property you specified. Specify the argument in form coordinate system units.

**y** Specifies the NUMBER value of the y coordinate or the height, depending on the property you specified. Specify the argument in form coordinate system units.

**Restrictions:**
- You can set only one property per call to the built-in.
- If you change the size or position of a window, the change remains in effect for as long as the form is running, or until you explicitly
change the window’s size or position again. Closing the window and reopening it does not reset the window to its design–time defaults. You must assign the design–time defaults to variables if you intend to set the window back to those defaults.

**Usage Notes:** On Microsoft Windows, forms run inside the MDI *application window*. You can use `SET_WINDOW_PROPERTY` to set the following properties of the MDI application window:

- TITLE
- POSITION
- WIDTH, HEIGHT
- WINDOW_SIZE
- WINDOW_STATE
- X_POS, Y_POS

To reference the MDI application window in a call to `SET_WINDOW_PROPERTY`, use the constant `FORMS_MDI_WINDOW`:

```plaintext
Set_Window_Property(FORMS_MDI_WINDOW, POSITION, 5,10)
Set_Window_Property(FORMS_MDI_WINDOW, WINDOW_STATE, MINIMIZE)
```

**Example:**

```plaintext
/*
  ** Built-in: SET_WINDOW_PROPERTY
  ** Example: See FIND_WINDOW
  */
```

---

**SHOW_ALERT**

**Syntax:**

```plaintext
SHOW_ALERT(alert_id);
SHOW_ALERT(alert_name);
```

**Built-in Type:** unrestricted function

**Returns:** A numeric constant corresponding to the button the operator selected from the alert. Button mappings are specified in the alert design.

<table>
<thead>
<tr>
<th>If the operator selects...</th>
<th>Oracle Forms returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button 1</td>
<td>ALERT_BUTTON1</td>
</tr>
<tr>
<td>Button 2</td>
<td>ALERT_BUTTON2</td>
</tr>
<tr>
<td>Button 3</td>
<td>ALERT_BUTTON3</td>
</tr>
</tbody>
</table>
Enter Query Mode: yes

Description: Displays the given alert, and returns a numeric value when the operator selects one of three alert buttons.

Parameters: 

- `alert_id` The unique ID that Oracle Forms assigns the alert when the alert is created. Use the `FIND_ALERT` built-in to return the ID to an appropriately typed variable. The data type of the ID is `Alert`.

- `alert_name` The name you gave the alert when you defined it. The data type of the name is `CHAR`.

Example:

```markdown
/*
** Built-in:  SHOW_ALERT
** Example:   See FIND_ALERT and SET_ALERT_PROPERTY
*/
```

SHOW_BACKGROUND_MENU

Syntax: `SHOW_BACKGROUND_MENU;`

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Displays the background menu.

Parameters: none

Restrictions: If you issue a call to this built-in when no background menu is defined for the current application, Oracle Forms issues error message `FRM-10207: No background menu present`.

Example:

```markdown
/*
** Built-in:  SHOW_BACKGROUND_MENU
** Example:   Display the background menu on a button press
** Trigger:   When-Button-Pressed
*/
BEGIN
  Show_Background_Menu;
END;
```
**SHOW_EDITOR**

Syntax:

```
SHOW_EDITOR(editor_id, message_in, message_out, result);
SHOW_EDITOR(editor_id, message_in, x, y, message_out, result);
SHOW_EDITOR(editor_name, message_in, message_out, result);
SHOW_EDITOR(editor_name, message_in, x, y, message_out, result);
```

Built-in Type: unrestricted procedure that returns two OUT parameters (*result* and *message_out*)

Enter Query Mode: yes

Description: Displays the given editor at the given coordinates and passes a string to the editor, or retrieves an existing string from the editor. If no coordinates are supplied, the editor is displayed in the default position specified for the editor at design time.

Parameters:

- **editor_id**: Specifies the unique ID that Oracle Forms assigns when it creates the editor. Use the FIND_EDITOR built-in to return the ID to a variable of the appropriate data type. The data type of the ID is Editor.

- **editor_name**: Specifies the name you gave to the editor when you defined it. The data type of the name is CHAR.

- **message_in**: Specifies a required IN parameter of CHAR data type. The value passed to this parameter can be NULL. You can also reference a text item or variable.

- **x**: Specifies the x coordinate of the editor. Supply a whole number for this argument.

- **y**: Specifies the y coordinate of the editor. Supply a whole number for this argument.

- **message_out**: Specifies a required OUT parameter of CHAR data type. You can also reference a text item or variable. If the operator cancels the editor, *result* is FALSE and *message_out* is NULL.

- **result**: Specifies a required OUT parameter of BOOLEAN data type. If the operator accepts the editor, *result* is TRUE. If the operator cancels the editor, *result* is FALSE and *message_out* is NULL.

Restrictions:

- Each *message_in* and *message_out* string cannot exceed 32Kb.
• *Message_out* should be at least as long as *message_in*, because the length of the variable or text item specified for *message_out* determines the maximum number of characters the editor can accept.

• The *message_in* parameter values are always converted to CHAR by Oracle Forms when passed to the editor. However, if you are passing *message_out* to something other than a CHAR type object, you must first perform the conversion by passing the value to a variable and then perform type conversion on that variable with PL/SQL functions TO_DATE or TO_NUMBER.

• The Width must be at least wide enough to display the buttons at the bottom of the editor window.

Example:

```sql
/*
** Built-in:  SHOW_EDITOR
** Example:   Accept input from the operator in a user-defined
**            editor. Use the system editor if the user has
**            checked the “System_Editor” menu item under the
**            “Preferences” menu in our custom menu module.
*/
DECLARE
   ed_id   Editor;
   mi_id   MenuItem;
   ed_name VARCHAR2(40);
   val     VARCHAR2(32000);
   ed_ok   BOOLEAN;
BEGIN
   mi_id := Find_Menu_Item('PREFERENCES.SYSTEM_EDITOR');
   IF Get_Menu_Item_Property(mi_id,CHECKED) = 'TRUE' THEN
      ed_name := 'system_editor';
   ELSE
      ed_name := 'my_editor1';
   END IF;
   ed_id := Find_Editor(ed_name);
   /*
   ** Show the appropriate editor at position (10,14) on the
   ** screen. Pass the contents of the :emp.comments item
   ** into the editor and reassign the edited contents if
   ** ‘ed_ok’ returns boolean TRUE.
   */
   val := :emp.comments;
   Show_Editor(ed_id, val, 10,14, val, ed_ok);
   IF ed_ok THEN
      :emp.comments := val;
   END IF;
END;
```
SHOW_KEYS

Syntax: SHOW_KEY;

Built–in Type: unrestricted procedure

Enter Query Mode: yes

Description: Displays the Show Keys screen. When the operator presses a function key, Oracle Forms redisplays the form as it was before invoking the SHOW_KEYS built–in.

Parameters: none

Example:
```sql
/* ** Built-in: SHOW_KEYS
 ** Example: Display valid function key bindings */
BEGIN
    Show_Keys;
END;
```

SHOW_LOV

Syntax: SHOW_LOV(lov_id);
SHOW_LOV(lov_id, x, y);
SHOW_LOV(lov_name);
SHOW_LOV(lov_name, x, y);

Built–in Type: unrestricted function

Returns: BOOLEAN

Enter Query Mode: yes

Description: Displays a list of values (LOV) window at the given coordinates, and returns TRUE if the operator selects a value from the list, and FALSE if the operator Cancels and dismisses the list.

Parameters:

- `lov_id`: Specifies the unique ID that Oracle Forms assigns the LOV when created. Use the FIND_LOV built–in to return the ID to an appropriately typed variable. The data type of the ID is LOV.

- `lov_name`: The name you gave to the LOV when you defined it. The data type of the name is CHAR.
x Specifies the x coordinate of the LOV.
y Specifies the y coordinate of the LOV.

Usage Notes: Because SHOW_LOV is a restricted built-in, when you use it to display an LOV, Oracle Forms ignores the LOV’s Auto-Skip property.

If you want to move the cursor to the next navigable item, use the LIST_VALUES built-in. LIST_VALUES is an unrestricted built-in.

Restrictions: If the lov_name argument is not supplied and there is no LOV associated with the current item, Oracle Forms issues an error.

If the record group underlying the LOV contains 0 records, the BOOLEAN return value for SHOW_LOV will be FALSE.

Example:
*/
** Built-in: SHOW_LOV
** Example: Display a named List of Values (LOV)
*/
DECLARE
  a_value_chosen BOOLEAN;
BEGIN
  a_value_chosen := Show_Lov('my_employee_status_lov');
  IF NOT a_value_chosen THEN
    Message('You have not selected a value.');
    Bell;
    RAISE Form_Trigger_Failure;
  END IF;
END;

SHOW_MENU

Syntax: SHOW_MENU;

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Displays the current menu if it is not currently displayed. It does not make the menu active.

Because SHOW_MENU does not make the menu active, Oracle Forms does not allow the menu to obscure any part of the current canvas. Therefore, all or part of the menu does not appear on the screen if the current canvas would cover it.

Parameters: none
**Restrictions:** Only for use in character mode environments.

**Example:**
```
/*
** Built-in: SHOW_MENU
** Example: Display the menu if no canvas overlays it.
*/
BEGIN
    Show_Menu;
END;
```

### SHOW_VIEW

**Syntax:**
```
SHOW_VIEW(view_id);
SHOW_VIEW(view_name);
```

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Displays the indicated canvas–view at the coordinates specified by the canvas–view’s X Position and Y Position property settings. If the view is already displayed, SHOW_VIEW raises it in front of any other views in the same window.

**Parameters:**
- **view_id** Specifies the unique ID that Oracle Forms assigns the view when it creates the object. Use the FIND_VIEW built-in to return the ID to an appropriately typed variable. The data type of the ID is ViewPort.
- **view_name** Specifies the name that you gave the view when defining it. The data type of the name is CHAR.

**Example:**
```
/*
** Built-in: SHOW_VIEW
** Example: Programmatically display a view in the window to which it was assigned at design time.
*/
BEGIN
    Show_View('My_Stacked_Overlay');
END;
```
SHOW_WINDOW

Syntax: SHOW_WINDOW(window_id);
SHOW_WINDOW(window_id, x, y);
SHOW_WINDOW(window_name);
SHOW_WINDOW(window_name, x, y);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Displays the indicated window at either the optionally included X,Y coordinates, or at the window's current X,Y coordinates. If the indicated window is a modal window, SHOW_WINDOW is executed as a GO_ITEM call to the first navigable item in the modal window.

Parameters:

- **window_id** Specifies the unique ID that Oracle Forms assigns the window when created. Use the FIND_WINDOW built-in to return the ID to an appropriately typed variable. The data type of the ID is Window.

- **window_name** Specifies the name that you gave the window when defining it. The data type of the name is CHAR.

- **x** Specifies the x coordinate of the window. Supply a whole number for this argument.

- **y** Specifies the y coordinate of the window. Specify this value as a whole NUMBER.

Example:

```
/*
** Built-in: SHOW_WINDOW
** Example: Override the default (x,y) coordinates for a
** windows location while showing it.
*/
BEGIN
  Show_Window('online_help',20,5);
END;
```
**SYNCHRONIZE**

<table>
<thead>
<tr>
<th>Syntax:</th>
<th>SYNCHRONIZE;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Built-in Type:</td>
<td>unrestricted procedure</td>
</tr>
<tr>
<td>Enter Query Mode:</td>
<td>yes</td>
</tr>
<tr>
<td>Description:</td>
<td>Synchronizes the terminal screen with the internal state of the form. That is, SYNCHRONIZE updates the screen display to reflect the information that Oracle Forms has in its internal representation of the screen.</td>
</tr>
<tr>
<td>Parameters:</td>
<td>none</td>
</tr>
<tr>
<td>Restrictions:</td>
<td>SYNCHRONIZE only updates the screen display if both of the following conditions are true:</td>
</tr>
<tr>
<td></td>
<td>• Oracle Forms is at the item level in the forms hierarchy (i.e., SYSTEM.CURRENT_ITEM is not NULL).</td>
</tr>
<tr>
<td></td>
<td>• Navigation to a different canvas-view has taken place (i.e., the current item and the previous item were on different canvas-views).</td>
</tr>
</tbody>
</table>
| Example:     | /*
** Built-in:  SYNCHRONIZE
** Example:   Achieve an odometer effect by updating the screen as items value changes quickly.
** Without synchronize, the screen is typically only updated when Forms completes all trigger execution and comes back for user input.
*/
BEGIN
  FOR j IN 1..1000 LOOP
    :control.units_processed := j;
    SYNCHRONIZE;
    Process_Element(j);
  END LOOP;
END; |
### TERMINATE

**Syntax:**

`TERMINATE;`

**Built-in Type:** restricted function

**Description:** TERMINATE terminates input in a form or dialog box. This function is equivalent to the operator pressing [ACCEPT].

**Parameters:** none

**Restrictions:** Terminate applies only in the Enter Parameter Values dialog.

### UNSET_GROUP_SELECTION

**Syntax:**

`UNSET_GROUP_SELECTION(recordgroup_id, row_number);`

`UNSET_GROUP_SELECTION(recordgroup_name, row_number);`

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Unmarks the specified row in the indicated record group. Use the procedure to unmark rows that have been programmatically selected by a previous call to SET_GROUP_SELECTION.

Rows are numbered sequentially starting at 1. If you select rows 3, 8, and 12, for example, those rows are considered by Oracle Forms to be selections 1, 2, and 3. You can undo any row selections for the entire group by calling the RESET_GROUP_SELECTION built-in.

**Parameters:**

- `recordgroup_id` Specifies the unique ID that Oracle Forms assigns to the record group when created. Use the FIND_GROUP built-in to return the ID to a variable. The data type of the ID is RecordGroup.

- `recordgroup_name` Specifies the name of the record group that you gave to the group when creating it. The data type of the name is CHAR.

- `row_number` Specifies the number of the record group row that you want to select. The value you specify is a NUMBER.

**Example:**

```
/*
** Built-in: UNSET_GROUP_SELECTION
** Example: Clear all of the even rows as selected in the
```
** record group whose id is passed-in as a parameter. */
PROCEDURE Clear_Even_Rows ( rg_id RecordGroup ) IS
BEGIN
  FOR j IN 1..Get_Group_Row_Count(rg_id) LOOP
    IF MOD(j,2)=0 THEN
      Unset_Group_Selection( rg_id, j );
    END IF;
  END LOOP;
END;

UP

Syntax: UP;
Built-in Type: restricted procedure
Enter Query Mode: no
Description: Navigates to the instance of the current item in the record with the next lowest sequence number.
Parameters: none

UPDATE_RECORD

Syntax: UPDATE_RECORD;
Built-in Type: restricted procedure
Enter Query Mode: no
Description: When called from an On-Update trigger, initiates the default Oracle Forms processing for updating a record in the database during the Post and Commit Transaction process.
This built-in is included primarily for applications that run against a non-ORACLE data source.
Parameters: none
Restrictions: Valid only in an On-Update trigger.
Example:

```/*
 * Built-in: UPDATE_RECORD
 * Example: Perform Oracle Forms standard update processing
 * based on a global flag setup at startup by the
 * form, perhaps based on a parameter.
 * Trigger: On-Update
 */
 BEGIN
     BEGIN
     /*
     * Check the global flag we setup at form startup
     */
     IF :Global.Using_Transactional_Triggers = 'TRUE' THEN
         User_Exit('my_updrec block=EMP');
     /*
     * Otherwise, do the right thing.
     */
     ELSE
         Update_Record;
     END IF;
 END;
```

**USER_EXIT**

**Syntax:**

```plaintext
USER_EXIT(user_exit_string);
USER_EXIT(user_exit_string, error_string);
```

**Built-in Type:** unrestricted procedure

**Enter Query Mode:** yes

**Description:** Calls the user exit named in the user_exit_string.

For more information, refer to *Oracle Forms Advanced Techniques*, Ch. 3, "Writing User Exits."

**Parameters:**

- `user_exit_string` Specifies the name of the user exit you want to call, including any parameters.
- `error_string` Specifies a user-defined error message that Oracle Forms should display if the user exit fails.

**Example:**

```/*
 ** Built-in: USER_EXIT
 ** Example: Invoke a 3GL program by name which has been
 ** properly linked into your current Oracle Forms
 ** executable. The user exit subprogram must parse
 ** the string argument it is passed to decide what
 ** functionality to perform.
 */
```
PROCEDURE Command_Robotic_Arm (cmd_string VARCHAR2) IS
BEGIN
   /*
   ** Call a C function 'RobotLnk' to initialize the
   ** communication card before sending the robot a message.
   */
   User_Exit('RobotLnk INITIALIZE Unit=6,CmdToFollow=1');
   IF NOT Form_Success THEN
      Message('Failed to initialize Robot 6');
      RAISE Form_Trigger_Failure;
   END IF;
   /*
   ** Pass the string argument as a command to the robot
   */
   User_Exit('RobotLnk SEND Unit=6,Msg=' || cmd_string);
   IF NOT Form_Success THEN
      Message('Command not understood by Robot 6');
      RAISE Form_Trigger_Failure;
   END IF;
   /*
   ** Close the robot’s communication channel
   */
   User_Exit('RobotLnk DEACTIVATE Unit=6');
   IF NOT Form_Success THEN
      Message('Failed to Deactivate Robot');
      RAISE Form_Trigger_Failure;
   END IF;
   /*
   ** The user exit will deposit a timing code into the item
   ** called 'CONTROL.ROBOT_STATUS'.
   */
   Message('Command Successfully Completed by Robot 6'||
            ' in '||TO_CHAR(:control.robot_timing)||
            ' seconds.');
END;
VALIDATE

Syntax: VALIDATE(validation_scope);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: VALIDATE forces Oracle Forms to immediately execute validation processing for the indicated validation scope.

Parameters: validation scope Specify one of the following scopes:

  DEFAULT_SCOPE Perform normal validation for the default scope, determined by the runtime platform.

Note: If you change the scope via SET_FORM_PROPERTY(VALIDATION UNIT) and then call VALIDATE(DEFAULT_SCOPE), you will override the default scope as defined in the form module. In this case, Oracle Forms will not validate at the default scope but at the scope defined by SET_FORM_PROPERTY.

  FORM_SCOPE Perform normal validation for the current form.

  BLOCK_SCOPE Perform normal validation for the current block.

  RECORD_SCOPE Perform normal validation for the current record.

  ITEM_SCOPE Perform normal validation for the current item.

Example: /*
  ** Built-in:  VALIDATE
  ** Example:  Deposits the primary key value, which the user has typed, into a global variable, and then validates the current block.
  ** Trigger:  When-New-Item-Instance
  */
BEGIN
  IF :Emp.Empno IS NOT NULL THEN
    :Global.Employee_Id := :Emp.Empno;
    Validate(block_scope);
    IF NOT Form_Success THEN
      RAISE Form_Trigger_Failure;
    END IF;
  END IF;
**VBX.FIRE_EVENT**

**Syntax:**  
VBX.FIRE_EVENT(item_id, event_name, paramlist_id);
VBX.FIRE_EVENT(item_id, event_name, paramlist_name);
VBX.FIRE_EVENT(item_name, event_name, paramlist_id);
VBX.FIRE_EVENT(item_name, event_name, paramlist_name);

**Built-in Type:**  
unrestricted procedure

**Enter Query Mode:**  
yes

**Description:**  
Raises an event for the VBX control.

**Parameters:**  
- *item_id*  
  Specifies the unique ID that Oracle Forms assigns to the item when created. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.

- *item_name*  
  Specifies the name of the object created at design time. The data type of the name is CHAR string.

- *event_name*  
  Specifies the name of a VBX event sent to the VBX control. The data type of the name is CHAR string.

- *paramlist_id*  
  Specifies the unique ID Oracle Forms assigns when a parameter list is created. The data type of the ID is PARAMLIST.

- *paramlist_name*  
  The name you give the parameter list object when it is defined. The data type of the name is CHAR string.

**Restrictions:**  
Valid only on Microsoft Windows.

**Example:**  
```/*
** Built-in: VBX.FIRE_EVENT
** Example: The VBX.FIRE_EVENT built-in triggers a SpinDown event for the SpinButton VBX control.
** Trigger: When-Button-Pressed
*/
DECLARE
  ItemName VARCHAR2(40) := 'SPINBUTTON';
  Pl_ID PARAMLIST;
  PL_NAME VARCHAR2(20) := 'EName';```
BEGIN
    PL_ID := Get_Parameter_List(PL_NAME);
    IF id_null(PL_ID) THEN
        PL_ID := Create_Parameter_List(PL_NAME);
        Add_Parameter(PL_ID,'sb_event',TEXT_PARAMETER,'SpinDown');
    END IF;
    VBX.FIRE_EVENT(ItemName,'SpinDown',PL_ID);
END;

VBX.GET_PROPERTY

Syntax:  
    VBX.GET_PROPERTY(item_id, property);
    VBX.GET_PROPERTY(item_name, property);

Built–in Type:  
    unrestricted function

Returns:  
    CHAR

Enter Query Mode:  
    yes

Description:  
    Obtains the value of a property from a VBX control.

Parameters:  
    item_id  
        Specifies the unique ID that Oracle Forms assigns to the item when created. Use the FIND_ITEM built–in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.

    item_name  
        Specifies the name of the object created at design time. The data type of the name is CHAR string.

    property  
        Specifies a property or an element of a property array for a VBX control. A set of VBX properties exists for any given VBX control. Examples of VBX properties are Width, Height, and FontSize. The data type of property is a CHAR string.

Restrictions:  
    Valid only on Microsoft Windows.

Example:  
    /*
    ** Built-in:  VBX.GET_PROPERTY
    ** Example:  Uses the VBX.GET_PROPERTY built-in to obtain the CURRTAB property of the VBX item named TABCONTROL.
    ** The property value of CURRTAB is returned to the TabNumber variable and is used as input in the user-defined Goto_Tab_Page subprogram.
    ** Trigger:  When-Custom-Item-Event
    */
DECLARE
    TabEvent    varchar2(80);
    TabNumber   char;
BEGIN
    TabEvent := :system.custom_item_event;
    IF (UPPER(TabEvent) = 'CLICK') then
        TabNumber := VBX.Get_Property('TABCONTROL','CurrTab');
        Goto_Tab_Page(TO_NUMBER(TabNumber));
    END IF;
END;

**VBX.GET_VALUE_PROPERTY**

**Syntax:**
- VBX.GET_VALUE_PROPERTY(item_id);
- VBX.GET_VALUE_PROPERTY(item_name);

**Built-in Type:** unrestricted function

**Returns:** property

**Enter Query Mode:** yes

**Description:**
Gets the VBX Control Value Property of a VBX control.

**Parameters:**
- **item_id**
  Specifies the unique ID that Oracle Forms assigns to the item when created. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.

- **item_name**
  Specifies the name of the object created at design time. The data type of the name is CHAR string.

**Restrictions:**
Valid only on Microsoft Windows.

**Example:**
/*
 ** Built-in:  VBX.GET_VALUE_PROPERTY
 ** Example:   Passes the VBX Control Value to the user-defined
 **            Verify_Item_Value subprogram. Verify_Item_Value
 **            ensures the display value is the expected value.
 */
DECLARE
    ItemName   VARCHAR2(40) := 'SPINBUTTON';
    VBX_VAL_PROP VARCHAR2(40);
BEGIN
    VBX_VAL_PROP := VBX.Get_Value_Property(ItemName);
    Verify_Item_Value(VBX_VAL_PROP);
END;
VBX.INVOKE_METHOD

Syntax:  
VBX.INVOKE_METHOD(item_id, method_name, w, x, y, z);
VBX.INVOKE_METHOD(item_name, method_name, w, x, y, z);

Built-in Type:  unrestricted procedure

Enter Query Mode:  yes

Description:  Invokes the specified method on the item. If the method takes arguments, they should be specified. The arguments should be provided in the order that the VBX control expects them. The methods that are valid for VBX controls and a listing of the arguments they expect can be found in the documentation that accompanies the VBX control.

Parameters:  

item_id  Specifies the unique ID that Oracle Forms assigns to the item when created. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.

item_name  Specifies the name of the object created at design time. The data type of the name is CHAR string.

method_name  Specifies the name of the method to invoke. The data type of the name is CHAR string.

w, x, y, z  Specifies optional arguments that might be required for VBX controls. The data type of the arguments is CHAR string.

Restrictions:  Valid only on Microsoft Windows.

Example:  
/*
** Built-in:  VBX.INVOKE_METHODPROPERTY
** Example:  Adds an entry to a combobox. The entry to add to the combobox is the first optional argument.
** The position where the entry appears is the second optional argument.
*/
DECLARE
    ItemName VARCHAR2(40) := ‘COMBOBOX’;
BEGIN
    VBX.Invoke_Method(ItemName,’ADDITEM’,’blue’,’2’);
END;
VBX.SET_PROPERTY

Syntax:  

```vb
VBX.SET_PROPERTY(item_id, property, value);
VBX.SET_PROPERTY(item_name, property, value);
```

Built-in Type:  

unrestricted procedure

Enter Query Mode:  

yes

Description:  

Sets the specified property for a VBX control.

Parameters:  

- `item_id` Specifies the unique ID that Oracle Forms assigns to the item when created. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.

- `item_name` Specifies the name of the object created at design time. The data type of the name is CHAR string.

- `property` Specifies a property or an element of a property array for a VBX control. A set of VBX properties exists for any given VBX control. Examples of VBX properties are Width, Height, and FontSize. The data type of property is a CHAR string.

- `value` Specifies the value to be applied to the VBX property. The data type of value is a CHAR string.

Restrictions:  

Valid only on Microsoft Windows.

Example:  

```vb
/*
** Built-in:  VBX.SET_PROPERTY
** Example:  Uses the VBX.SET_PROPERTY built-in to set the Index property of the SpinButton VBX control.
** Trigger:  When-Button-Pressed
*/
DECLARE
 (ItemName   VARCHAR(40) := 'SPINBUTTON';
  VBX_VAL_PROP VARCHAR(40);
  VBX_VAL    VARCHAR(40);
BEGIN
  IF :System.Custom_Item_Event = 'SpinDown' THEN
    VBX_VAL_PROP := 'Index';
    VBX_VAL := '5';
    VBX.Set_Property(ItemName, VBX_VAL_PROP, VBX_VAL);
  END IF;
END;
```
VBX.SET_VALUE_PROPERTY

Syntax:  
VBX.SET_VALUE_PROPERTY(item_id, property);
VBX.SET_VALUE_PROPERTY(item_name, property);

Built-in Type:  unrestricted procedure

Enter Query Mode:  yes

Description:  Sets the VBX Control Value Property of a VBX control.

Parameters:

- item_id  
  Specifies the unique ID that Oracle Forms assigns to the item when created. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM.

- item_name  
  Specifies the name of the object created at design time. The data type of the name is CHAR string.

- property  
  Specifies a property for the Oracle Forms VBX Control Value Property. A set of VBX properties exists for any given VBX control. Examples of VBX properties are Width, Height, and FontSize. The data type of property is a CHAR string.

Restrictions:  Valid only on Microsoft Windows.

Example:  /*
** Built-in:  VBX.SET_VALUE_PROPERTY
** Example:  Uses the VBX.SET_VALUE_PROPERTY built-in to set the
**            VBX Control Value property.
** */
DECLARE  
  ItemName  VARCHAR(40) := 'SPINBUTTON';  
  VBX_VAL_PROP  VARCHAR(40);
BEGIN  
  IF :System.Custom_Item_Event = 'SpinDown' THEN  
    VBX_VAL_PROP := 'Index';  
    VBX.Set_Value_Property(ItemName, VBX_VAL_PROP);  
  END IF;
END;
WHERE_DISPLAY

Syntax: WHERE_DISPLAY;

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Toggles the Where menu navigation option on and off. In a full-screen menu, the Where option displays information about the operator’s current location in the menu hierarchy.

Parameters: none

Restrictions: WHERE_DISPLAY is valid only in a full-screen menu.

WRITE_IMAGE_FILE

Syntax: WRITE_IMAGE_FILE(file_name, file_type, item_id);
WRITE_IMAGE_FILE(file_name, file_type, item_name);

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description: Writes the image stored in an Oracle Forms image item into the file.

Parameters:

- file_name
  Specifies the name of the file in which the image is stored. The file name must adhere to your operating system requirements.

- file_type
  Specifies the file type of the image: TIFF, JFIF, PCX, PICT, PCD, GIF, CALS, RAS, or BMP. The parameter takes a CHAR argument.

- item_id
  Specifies the unique ID that Oracle Forms assigns to the image item when it creates the item. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. The data type of the ID is ITEM..

- item_name
  Specifies the name that you gave the image item when defining it. The data type of the name is CHAR.
Restrictions:

- All images written to the database are stored in Oracle format, regardless of the original file type of the image.
- The indicated file type must be compatible with the actual file type of the image.
- Supported image formats are TIFF, JFIF, PCX, PICT, PCD, GIF, CALS, RAS, or BMP formats.
- As with any file, if you write the image to an existing file, you overwrite the contents of that file with the contents of the image item.

Example:

```/*
** built-in: WRITE_IMAGE_FILE
**
** Example: Save the contents of an image item out to a file on the filesystem in a supported image format. */
BEGIN
  Write_Image_File('output.tif','TIFF','emp.photo_image_data');
END;```
System Variables

This chapter includes information about Oracle Forms system variables.
About System Variables

A system variable is an Oracle Forms variable that keeps track of an internal Oracle Forms state. You can reference the value of a system variable to control the way an application behaves.

Oracle Forms maintains the values of system variables on a per form basis. That is, the values of all system variables correspond only to the current form. The following list presents the names of the available system variables:

- SYSTEM.BLOCK_STATUS
- SYSTEM.COORDINATION_OPERATION
- SYSTEM.CURRENT_BLOCK
- SYSTEM.CURRENT_DATETIME
- SYSTEM.CURRENT_FORM
- SYSTEM.CURRENT_ITEM
- SYSTEM.CURRENT_VALUE
- SYSTEM.CURSOR_BLOCK
- SYSTEM.CURSOR_ITEM
- SYSTEM.CURSOR_RECORD
- SYSTEM.CURSOR_VALUE
- SYSTEM.DATE_THRESHOLD*
- SYSTEM.EFFECTIVE_DATE*
- SYSTEM.EVENT_WINDOW
- SYSTEM.FORM_STATUS
- SYSTEM.LAST_QUERY
- SYSTEM.LAST_RECORD
- SYSTEM.MASTER_BLOCK
- SYSTEM.MESSAGE_LEVEL*
- SYSTEM.MODE
- SYSTEM.MOUSE_BUTTON_PRESSED
- SYSTEM.MOUSE_BUTTON SHIFT_STATE
- SYSTEM.MOUSE_ITEM
- SYSTEM.MOUSE_CANVAS
• SYSTEM.MOUSE_X_POS
• SYSTEM.MOUSE_Y_POS
• SYSTEM.MOUSE_RECORD
• SYSTEM.MOUSE_RECORD_OFFSET
• SYSTEM.RECORD_STATUS
• SYSTEM.SUPPRESS_WORKING*
• SYSTEM.TRIGGER_BLOCK
• SYSTEM.TRIGGER_ITEM
• SYSTEM.TRIGGER_RECORD

All system variables, except the four indicated with an asterisk (*), are read-only variables. These four variables are the only system variables to which you can explicitly assign values.

Date and Time System Default Values

Oracle Forms also supplies six special default values—$$DATE$$, $$DATETIME$$, $$TIME$$, $$DBDATE$$, $$DBDATETIME$$, and $$DBTIME$$—that supply date and time information and have special restrictions on their use:

• If you’re building client/server applications, consider the performance implications of going across the network to get date and time information.

• If you’re accessing a non-ORACLE datasource, avoid using $$DBDATE$$ and $$DBDATETIME$$. Instead, use a When–Create–Record trigger to select the current date in a datasource–specific manner.

• Use $$DATE$$, $$DATETIME$$, and $$TIME$$ to obtain the local system date/time; use $$DBDATE$$, $$DBDATETIME$$, and $$DBTIME$$ to obtain the database date/time, which may differ from the local system date/time if, for example, you’re connecting to a remote database in a different time zone.

• Use these variables only to set the value of the Default Value, Range Low Value or Range High Value property.
Local Variables

Because system variables are derived, if the value is not expected to change over the life of the trigger, you can save the system value in a local variable and use the local variable multiple times.

Example: Assume that you want to create a Key-NXTBLK trigger at the form level that navigates depending on what the current block is. The following trigger performs this function, using :SYSTEM.CURSOR_BLOCK stored in a local variable.

```sql
DECLARE
  curblk VARCHAR2(30);
BEGIN
  curblk := :System.Cursor_Block;
  IF curblk = 'Orders'
    THEN Go_Block('Items');
  ELSIF curblk = 'Items'
    THEN Go_Block('Customers');
  ELSIF curblk = 'Customers'
    THEN Go_Block('Orders');
  END IF;
END;
```

Uppercase Return Values

All system variables are case-sensitive, and most return their arguments as uppercase values. This will affect the way you compare results in IF statements.
### Syntax:

$$$DATE$$

### Description:

$$$DATE$$ retrieves the current operating system date. You can use $$$DATE$$ to designate a default value or range for a text item using the Default or Range property. The text item must be of the CHAR, DATE, or DATETIME data type.

You also can use $$$DATE$$ as a default value for form parameters. In this case, the parameter's value is computed once, at form startup.

### Usage Notes:

The difference between $$$DATE$$ and $$$DATETIME$$ is that the time component for $$$DATE$$ is always fixed to 00:00:00, compared to $$$DATETIME$$, which includes a meaningful time component, such as 09:17:59.

The display of system variables is governed by the format mask, either a default data type format mask or one you specify. For example, if you want to use the default DD–MON–YY format, you must specify a DATE data type. (Note that the default format mask depends on the value of NLS_LANG.)

Although $$$DATE$$ displays only the date, its underlying value includes a time component which is saved at commit time. If you specify a DATETIME data type and provide $$$DATE$$ as the default, the underlying value will be DD–MON–YYYY HH:MI:SS. For example, 01–DEC–1994 00:00:00 (although only 01–DEC–1994 will be displayed).

Use $$$DATE$$ when you want to compare the contents of this field with a field whose format mask does not have a time component, such as a SHIPDATE field of data type DATE. In this case, both $$$DATE$$ and SHIPDATE will have a time component of 00:00:00, so the comparison of two dates evaluating to the same day will be successful.

### Example 1:

Assume that you want the value of a DATE text item, called ORDERDATE, to default to the current date. When you define the ORDERDATE text item, specify $$$DATE$$ in the text item Default Value property.

### Example 2:

If you use $$$DATE$$ in a parameter, such as

:PARAMETER.STARTUP_DATE, then every time you reference that parameter, the date you started the application will be available:

```sql
IF :PARAMETER.Startup_Date + 1 < :System.Current_Datetime
   THEN Message ('You have been logged on for more than a day.');
ELSE Message ('You just logged on today.');
END IF;
```
$$DATETIME$$

Syntax:  $$DATETIME$$

Description:  $$DATETIME$$ retrieves the current operating system date and time. You can use $$DATETIME$$ to designate a default value or range for a text item using the Default or Range property. The text item must be of the CHAR or DATETIME data type.

You also can use $$DATETIME$$ as a default value for form parameters. In this case, the parameter’s value is computed once, at form startup.

Usage Notes:  The display of system variables is governed by the format mask, either a default data type format mask or one you specify. For example, if you want the default DD–MON–YY HH:MI:SS format, you must specify a DATETIME data type. (Note that the default format mask depends on the value of NLS_LANG.)

The difference between $$DATE$$ and $$DATETIME$$ is that the time component for $$DATE$$ is always fixed to 00:00:00, compared to $$DATETIME$$, which includes a meaningful time component, such as 09:17:59.

Note:  Do not use $$DATETIME$$ instead of $$DATE$$ unless you plan to specify the time component. If, for example, you use $$DATETIME$$ with the default DATE format mask of DD–MON–YY, you would be committing values to the database that the user would not see, because the format mask does not include a time component. Then, because you had committed specific time information, when you later queried on date, the values would not match and you would not return any rows.

Example:  Assume that you want the value of a DATETIME text item, called ORDERDATE, to default to the current operating system date and time. When you define the ORDERDATE text item, specify $$DATETIME$$ in the Default Value property.
**$DBDATE$$**

**Syntax:**

```
$DBDATE$
```

**Description:**

$DBDATE$ retrieves the current database date. You can use $DBDATE$ to designate a default value or range for a text item using the Default or Range property. The text item must be of the CHAR, DATE, or DATETIME data type.

**Restrictions:**

- Use this variable only to set the value of the Default Value, Range Low Value, or Range High Value property.
- If you are accessing a non-ORACLE datasource, avoid using $DBDATE$. Instead, use a When–Create–Record trigger to select the current date in a datasource–specific manner.

**Usage Notes:**

The difference between $DBDATE$ and $DBDATETIME$ is that the time component for $DBDATE$ is always fixed to 00:00:00, compared to $DBDATETIME$, which includes a meaningful time component, such as 09:17:59.

Use $DBDATE$ when you want to default a DATE item to the current date on the server machine, for example, when connecting to a remote database that may be in a different time zone from the client’s time zone.

The display of system variables is governed by the format mask, either a default data type format mask or one you specify. For example, if you want the default DD–MON–YY format, you must specify a DATE data type. (Note that the default format mask depends on the value of NLS_LANG.)

Although $DBDATE$ displays only the date, its underlying value includes a time component which is saved at commit time. If you specify a DATETIME data type and provide $DBDATE$ as the default, the underlying value will be DD–MON–YYYY HH:MI:SS: for example, 01–DEC–1994 00:00:00 (although only 01–DEC–1994 will be displayed).

**Example:**

To have the value of a DATE text item called ORDERDATE default to the current database date, for the ORDERDATE text item, specify $DBDATE$ in the Default Value property.
**Syntax:**

$$DBDATETIME$$ retrieves the current date and time from the local database. You can use $$DBDATETIME$$ to designate a default value or range for a text item using the Default or Range property. The text item must be of the CHAR or DATETIME data type.

**Description:**

- Use this variable only to set the value of the Default Value, Range Low Value, or Range High Value property.
- If you are accessing a non–ORACLE datasource, avoid using $$DBDATETIME$$. Instead, use a When–Create–Record trigger to select the current date and time in a datasource–specific manner.

**Usage Notes:**

Use $$DBDATETIME$$ when you want to default a DATE item to the current date on the server machine, for example, when connecting to a remote database that may be in a different time zone from the client’s time zone.

The display of system variables is governed by the format mask, either a default data type format mask or one you specify. For example, if you want a DD–MON–YY HH:MI:SS format, you must specify a DATETIME or CHAR data type. (Note that the default format mask depends on the value of NLS_LANG.)

If you are building a client–server application, using $$DBDATETIME$$ could have performance implications, depending on the complexity of your network configuration.

**Note:** Do not use $$DBDATETIME$$ instead of $$DBDATE$$ unless you plan to specify the time component. If, for example, you use $$DBDATETIME$$ with the default DATE format mask of DD–MON–YY, you would be committing values to the database that the user would not see, because the format mask does not include a time component. Then, because you had committed specific time information, when you later queried on date, the values would not match and you would not return any rows.

**Example:**

Assume that you want the value of a DATETIME text item, called ORDERDATE, to default to the current database date and time. When you define the ORDERDATE text item, specify $$DBDATETIME$$ in the Default Range property.
$$DBTIME$$

Syntax: $$DBTIME$$

Description: $$DBTIME$$ retrieves the current time from the local database. You can use $$DBTIME$$ to designate a default value or range for a text item using the Default or Range property. The text item must be of the CHAR or TIME data type.

Restrictions:

- Use this variable only to set the value of the Default Value, Range Low Value, or Range High Value property.
- If you are accessing a non–ORACLE datasource, avoid using $$DBTIME$$. Instead, use a When–Create–Record trigger to select the current time in a datasource–specific manner.

Usage Notes:

Use $$DBTIME$$ when connecting to a remote database that may be in a different time zone from the client’s time zone.

The display of system variables is governed by the format mask, either a default data type format mask or one you specify. For example, if you want the default HH:MI:SS format, you must specify a TIME data type. (Note that the default format mask depends on the value of NLS_LANG.)

If you are building a client–server application, using $$DBTIME$$ could have performance implications, depending on the complexity of your network configuration.

Example: Assume that you want the value of a TIME text item, called ORDERTIME, to default to the current database time. When you define the ORDERTIME text item, specify $$DBTIME$$ in the Default Value property.

$$TIME$$

Syntax: $$TIME$$

Description: $$TIME$$ retrieves the current operating system time. You can use $$TIME$$ to designate a default value or range for a text item using the Default or Range property. The text item must be of the CHAR or TIME data type.

You also can use $$TIME$$ as a default value for form parameters. In this case, the parameter’s value is computed once, at form startup.
The display of system variables is governed by the format mask, either a default data type format mask or one you specify. For example, if you want the default HH:MI:SS format, you must specify a TIME data type. (Note that the default format mask depends on the value of NLS_LANG.)

Assume that you want the value of a TIME text item, called ORDERTIME, to default to the current operating system time. When you define the ORDERTIME item, specify $$\$TIME\$$ in the Default Value property.

**SYSTEM.BLOCK_STATUS**

**Syntax:** `SYSTEM.BLOCK_STATUS`

**Description:** `SYSTEM.BLOCK_STATUS` represents the status of the block where the cursor is located, or the current block during trigger processing. The value can be one of three character strings:

- **CHANGED**: Indicates that the block contains at least one Changed record.
- **NEW**: Indicates that the block contains only New records.
- **QUERY**: Indicates that the block contains only Valid records that have been retrieved from the database.

Each time this value is referenced, it must be constructed by Oracle Forms. If your block contains a large numbers of records, using `SYSTEM.BLOCK_STATUS` could affect performance.

Assume that you want to create a trigger that performs a commit before clearing a block if there are changes to commit within that block. The following Key–CLRBLK trigger performs this function.

```sql
IF :System.Block_Status = 'CHANGED'
   THEN Commit_Form;
END IF;
Clear_Block;
```

**See Also:** `SYSTEM.RECORD_STATUS` and `SYSTEM.FORM_STATUS` system variables
SYSTEM.COORDINATION_OPERATION

Syntax: SYSTEM.COORDINATION_OPERATION

Description: This system variable works with its companion SYSTEM.MASTER_BLOCK to help an On-Clear-Details trigger determine what type of coordination-causing operation fired the trigger, and on which master block of a master-detail relation.

The values of the two system variables remain constant throughout the clearing phase of any block synchronization. SYSTEM.MASTER_BLOCK represents the name of the driving master block, and SYSTEM.COORDINATION_OPERATION represents the coordination-causing event that occurred on the master block.

The Clear_All_Master_Details procedure, which is automatically generated when a relation is created, checks the value of SYSTEM.COORDINATION_OPERATION to provide special handling for the CLEAR_RECORD and SYNCHRONIZE events, which may be different from the handling for other coordination-causing events. The Clear_All_Master_Details procedure also checks the value of SYSTEM.MASTER_BLOCK, to verify that while it is processing the master block of a relation coordination, it is searching only for blocks containing changes.

For example, given the relation hierarchy between blocks shown below, moving to the next record using the [Next Record] key or the Record, Next menu command while in Block C would cause blocks E, F, G, and H to be cleared (and perhaps subsequently queried, depending on the Deferred_Coordination property of the C->E and the C->F relations).
When the On-Clear-Details trigger fires for block C, the result is:

```plaintext
:System.Cooordination_Operation = 'NEXT_RECORD'
:System.Master_Block = 'C'
```

The Clear_All_Master_Details procedure will clear all of block C’s details, causing a “chain reaction” of Clear_Block operations. Consequently, block F is cleared.

Since F is a master for both G and H, and it is being cleared, an On-Clear-Details trigger will fire for block F as well. However, since the clearing of block F was driven (indirectly) by a coordination-causing event in block C, these remain the values in the On-Clear-Details trigger for block F:

```plaintext
:System.Cooordination_Operation = 'NEXT_RECORD'
:System.Master_Block = 'C'
```

**Note:** The values of these two system variables are well-defined only in the scope of an On-Clear-Details trigger, or any program unit called by that trigger. Outside this narrow context, the values of these two variables are undefined and should not be used.

The possible values of `SYSTEM.COORDINATION_OPERATION`, when it is appropriate to check that variable, are described in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Caused By</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOUSE</td>
<td>Mouse to non-current record</td>
<td>Mouse</td>
</tr>
<tr>
<td>UP</td>
<td>Move up a record</td>
<td>Menu, key, PL/SQL</td>
</tr>
<tr>
<td>DOWN</td>
<td>Move down a record</td>
<td>Menu, key, PL/SQL</td>
</tr>
<tr>
<td>SCROLL_UP</td>
<td>Scroll up records</td>
<td>Menu, key, PL/SQL</td>
</tr>
<tr>
<td>SCROLL_DOWN</td>
<td>Scroll down records</td>
<td>Mouse, key, PL/SQL</td>
</tr>
<tr>
<td>CLEAR_BLOCK</td>
<td>Clear current block</td>
<td>Menu, key, PL/SQL</td>
</tr>
<tr>
<td>CLEAR_RECORD</td>
<td>Clear current record</td>
<td>Menu, key, PL/SQL</td>
</tr>
<tr>
<td>CREATE_RECORD</td>
<td>Create new record</td>
<td>Mouse, menu, key, PL/SQL</td>
</tr>
<tr>
<td>DELETE_RECORD</td>
<td>Delete current record</td>
<td>Menu, key, PL/SQL</td>
</tr>
<tr>
<td>DUPLICATE_RECORD</td>
<td>Duplicate current record</td>
<td>Menu, key, PL/SQL</td>
</tr>
<tr>
<td>FIRST_RECORD</td>
<td>Move to first record</td>
<td>PL/SQL</td>
</tr>
<tr>
<td>LAST_RECORD</td>
<td>Move to last record</td>
<td>PL/SQL</td>
</tr>
<tr>
<td>NEXT_RECORD</td>
<td>Move to next record</td>
<td>Mouse, menu, key, PL/SQL</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
<td>Caused By</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>PREVIOUS_RECORD</td>
<td>Move to previous record</td>
<td>Mouse, menu, key, PL/SQL</td>
</tr>
<tr>
<td>GO_RECORD</td>
<td>Jump to record by number</td>
<td>PL/SQL</td>
</tr>
<tr>
<td>ENTER_QUERY</td>
<td>Enter Query mode</td>
<td>Menu, key, PL/SQL</td>
</tr>
<tr>
<td>EXECUTE_QUERY</td>
<td>Execute query</td>
<td>Menu, key, PL/SQL</td>
</tr>
<tr>
<td>COUNT_QUERY</td>
<td>Count queried records</td>
<td>Menu, key, PL/SQL</td>
</tr>
<tr>
<td>NEXT_SET</td>
<td>Fetch next set of records</td>
<td>Menu, key, PL/SQL</td>
</tr>
<tr>
<td>SYNCHRONIZE_BLOCS</td>
<td>Resume after commit error</td>
<td>Internal only</td>
</tr>
</tbody>
</table>

**SYSTEM.CURRENT_BLOCK**

**Syntax:**  
```
SYSTEM.CURRENT_BLOCK
```

**Description:**  
The value that the SYSTEM.CURRENT_BLOCK system variable represents depends on the current navigation unit:

- If the current navigation unit is the block, record, or item (as in the Pre– and Post– Item, Record, and Block triggers), the value of SYSTEM.CURRENT_BLOCK is the name of the block that Oracle Forms is processing or that the cursor is in.

- If the current navigation unit is the form (as in the Pre– and Post–Form triggers), the value of SYSTEM.CURRENT_BLOCK is NULL.

The value is always a character string.

**Note:**  
SYSTEM.CURRENT_BLOCK is included for compatibility with previous versions. Oracle Corporation recommends that you use SYSTEM.CURSOR_BLOCK and SYSTEM.TRIGGER_BLOCK instead.
SYSTEM.CURRENT_DATETIME

Syntax: SYSTEM.CURRENT_DATETIME

Description: SYSTEM.CURRENT_DATETIME is a variable representing the operating system date. The value is a CHAR string in the following format:

DD–MON–YYYY HH24:MI:SS

Default: current date

Usage Notes: SYSTEM.CURRENT_DATETIME is useful when you want to use the current operating system date and time in a PL/SQL trigger or procedure. By using SYSTEM.CURRENT_DATETIME instead of $$DBDATETIME$$, you can avoid the performance impact caused by querying the database.

Note: Local time and database time may differ.

Example:

```sql
/*
** Trigger: WHEN–TIMER–EXPIRED
** Example: Update on-screen time every 30 seconds
*/
DECLARE
    time VARCHAR2(20);
BEGIN
    time := :System.Current_Datetime;
    :control.onscreen := SUBSTR(time, instr(time, ' ') + 1);
END;
```
**SYSTEM.CURRENT_FORM**

**Syntax:**  
```sql
SYSTEM.CURRENT_FORM
```

**Description:** SYSTEM.CURRENT_FORM represents the name of the form that Oracle Forms is executing. The value is always a character string.

**Usage Notes:** You can use the GET_APPLICATION_PROPERTY built-in to obtain the name of the current form.

**Example:** Assume that you want any called form to be able to identify the name of the form that called it. You can invoke the following user-defined procedure before Oracle Forms issues a call. This procedure stores the name of the current form in a global variable named CALLING_FORM.

```sql
PROCEDURE STORE_FORMNAME IS
BEGIN
  :GLOBAL.Calling_Form := :System.Current_Form;
END;
```

**SYSTEM.CURRENT_ITEM**

**Syntax:**  
```sql
SYSTEM.CURRENT_ITEM
```

**Description:** The value that the SYSTEM.CURRENT_ITEM system variable represents depends on the current navigation unit:

- If the current navigation unit is the item (as in the Pre- and Post-Item triggers), the value of SYSTEM.CURRENT_ITEM is the name of the item that Oracle Forms is processing or that the cursor is in. The returned item name does not include a block name prefix.

- If the current navigation unit is the record, block, or form (as in the Pre- and Post- Record, Block, and Form triggers), the value of SYSTEM.CURRENT_ITEM is NULL.

The value is always a character string.

**Note:** SYSTEM.CURRENT_ITEM is included for compatibility with previous versions. Oracle Corporation recommends that you use SYSTEM.CURSOR_ITEM or SYSTEM.TRIGGER_ITEM instead.
### SYSTEM.CURRENT_VALUE

**Syntax:**

```
SYSTEM.CURRENT_VALUE
```

**Description:**

SYSTEM.CURRENT_VALUE represents the value of the item that is registered in SYSTEM.CURRENT_ITEM.

The value is always a character string.

**Note:** SYSTEM.CURRENT_VALUE is included for compatibility with previous versions. Oracle Corporation recommends that you use SYSTEM.CURSOR_ITEM and SYSTEM.CURSOR_VALUE instead.

### SYSTEM.CURSOR_BLOCK

**Syntax:**

```
SYSTEM.CURSOR_BLOCK
```

**Description:**

The value that the SYSTEM.CURSOR_BLOCK system variable represents depends on the current navigation unit:

- If the current navigation unit is the block, record, or item (as in the Pre– and Post– Item, Record, and Block triggers), the value of SYSTEM.CURSOR_BLOCK is the name of the block where the cursor is located. The value is always a character string.

- If the current navigation unit is the form (as in the Pre– and Post–Form triggers), the value of SYSTEM.CURSOR_BLOCK is NULL.

**Example:**

Assume that you want to create a Key–NXTBLK trigger at the form level that navigates depending on what the current block is. The following trigger performs this function, using :SYSTEM.CURSOR_BLOCK stored in a local variable.

```sql
DECLARE
  curblk VARCHAR2(30);
BEGIN
  curblk := :System.Cursor_Block;
  IF curblk = 'ORDERS'
    THEN Go_Block('ITEMS');
  ELSIF curblk = 'ITEMS'
    THEN Go_Block('CUSTOMERS');
  ELSIF curblk = 'CUSTOMERS'
    THEN Go_Block('ORDERS');
  END IF;
END;
```
SYSTEM.CURSOR_ITEM

Syntax: SYSTEM.CURSOR_ITEM

Description: SYSTEM.CURSOR_ITEM represents the name of the block and item, block.item, where the input focus (cursor) is located.

The value is always a character string.

Restrictions:

- Use SYSTEM.CURSOR_ITEM in triggers where the current navigation unit is the item, such as Pre–Item and Post–Item triggers. In these triggers, the value of SYSTEM.CURSOR_ITEM is block.item.

- Avoid using SYSTEM.CURSOR_ITEM in triggers where the current navigation unit is not the item, such as Pre– and Post–Record, Block, and Form triggers. In these triggers, the value of SYSTEM.CURSOR_ITEM is NULL.

Usage Note: Within a given trigger, the value of SYSTEM.CURSOR_ITEM changes when navigation takes place. This differs from SYSTEM.TRIGGER_ITEM, which remains the same from the beginning to the end of single trigger.

Example: Assume that you want to create a user–defined procedure that takes the value of the item where the cursor is located (represented by SYSTEM.CURSOR_VALUE), then multiplies the value by a constant, and then reads the modified value into the same item. The following user–defined procedure uses the COPY built–in to perform this function.

```sql
PROCEDURE CALC_VALUE IS
new_value NUMBER;
BEGIN
new_value := TO_NUMBER(:System.Cursor_Value) * .06;
Copy(TO_CHAR(new_value), :System.Cursor_Item);
END;
```
SYSTEM.CURSOR_RECORD

Syntax: SYSTEM.CURSOR_RECORD

Description: SYSTEM.CURSOR_RECORD represents the number of the record where the cursor is located. This number represents the record’s current physical order in the block’s list of records. The value is always a character string.

Example: Assume that you want to redefine [Previous Item] on the first text item of the ITEMS block so that it navigates to the last text item of the ORDERS block if the current record is the first record. The following Key–PRV–ITEM trigger on the ITEMS.ORDERID text item performs this function.

```
IF :System.Cursor_Record = '1'
   THEN Go_Item('orders.total');
   ELSE Previous_Item;
END IF;
```
SYSTEM.CURSOR_VALUE

Syntax:  

```plaintext
SYSTEM.CURSOR_VALUE
```

Description: SYSTEM.CURSOR_VALUE represents the value of the item where the cursor is located. The value is always a character string.

Restrictions:  

- Use SYSTEM.CURSOR_VALUE in triggers that fire after the user interacts with an item, such as When–Validate–Item or Post–Item, Post–Record, and Post–Block triggers. In these triggers, the value of SYSTEM.CURSOR_VALUE is value of the item where the cursor is located.

- Avoid using SYSTEM.CURSOR_VALUE in Pre–Form and Post–Form triggers, where the value of SYSTEM.CURSOR_VALUE is NULL.

Usage Note: Be aware that in triggers where the current navigation unit is not the item, such as Pre–Item, Pre–Record, and Pre–Block triggers, SYSTEM.CURSOR_VALUE will contain the value of the item navigated from, rather than the value of the item navigated to.

Example: Assume that you want to create a user–defined procedure that takes the value of the item where the cursor is located, multiplies the value by a constant, and then reads the modified value into the same item. The following user–defined procedure uses the COPY built–in to perform this function.

```plaintext
PROCEDURE CALC_VALUE IS
    new_value NUMBER;
BEGIN
    new_value := TO_NUMBER(:System.Cursor_Value) * .06;
    Copy(TO_CHAR(new_value), :System.Cursor_Item);
END;
```
SYSTEM.CUSTOM_ITEM_EVENT

Syntax:  SYSTEM.CUSTOM_ITEM_EVENT

Description:  SYSTEM.CUSTOM_ITEM_EVENT stores the name of the event fired by a VBX control.

Example:  Checks to see if the SpinDown event was fired by the SpinButton VBX control before navigating to the previous item.

IF  :System.CUSTOM_ITEM_EVENT = 'SpinDown' THEN
    :QTY := :QTY –1;
END IF;

SYSTEM.CUSTOM_ITEM_EVENT_PARAMETERS

Syntax:  SYSTEM.CUSTOM_ITEM_EVENT_PARAMETERS

Description:  SYSTEM.CUSTOM_ITEM_EVENT_PARAMETERS stores the supplementary arguments for an event fired by a VBX control.

Example:  Obtains the value of the 'Button' parameter that stores the value of a VBX control event, and passed the 'Button' value to the user-defined Move_Image subprogram.

DECLARE
    parmType  NUMBER;
    parmValue VARCHAR2(80);
BEGIN
    Get_Parameter_Attr(:System.CUSTOM_ITEM_EVENT_PARAMETERS,
        'Button',parmType,parmValue);
    /*
    ** The value of the 'Button' parameter represents the
    ** direction to move an image. The user-defined Move_Image
    ** subprogram moves an image two pixels in the direction
    ** specified by 'Button' parameter.
    */
    Move_Image(parmValue);
END;
Syntax:  

Description:  

Default:  

Usage Notes:
**SYSTEM.EFFECTIVE_DATE**

**Syntax:**  
SYSTEM.EFFECTIVE_DATE

**Description:**  
SYSTEM.EFFECTIVE_DATE represents the effective database date. The variable value must always be in the following format:

DD-MON-YYYY HH24:MI:SS

**Default:**  
RDBMS date

**Restrictions:**  
This variable is only valid when your database contains a definition of the DUAL table.

**Usage Notes:**  
This system variable is convenient for testing. Since you can use it to set a specific time, the time on the screen during an application would not cause subsequent test results to appear different than the known valid output.

**Example:**  
Assume you have set a DATE or TIME text item to one of the three system variables $$DBDATE$$, $$DBDATETIME$$, or $$DBTIME$$.

If you want to override that date or time, you can set the SYSTEM.EFFECTIVE_DATE system variable to a specific date and/or time.

:System.Effective_Date := ’31–DEC–1994 10:59:00’

Note that the effective date “rolls forward” with the database clock. For example, if you were to set the date as in the immediately preceding example, in an hour, the date would appear as follows:

:System.Effective_Date := ’31–DEC–1994 11:59:00’

The value is synchronized to the RDBMS date. If your database administrator changes the RDBMS date, the SYSTEM.EFFECTIVE_DATE is automatically changed by the same amount of change between old and new RDBMS dates. Oracle Forms polls the RDBMS whenever a reference to the effective date is required by the application.
The SYSTEM.EVENT_WINDOW system variable represents the name of the last window that was affected by an action that caused one of the window event triggers to fire. The following triggers cause this variable to be updated:

- WHEN–WINDOW–ACTIVATED
- WHEN–WINDOW–CLOSED
- WHEN–WINDOW–DEACTIVATED
- WHEN–WINDOW–RESIZED

From within these triggers, you can assign the value of the variable to any of the following:

- global variable
- parameter
- variable
- item, including a null canvas item

The following example sets the input focus into a particular item, depending on the window affected:

```plaintext
IF :System.Event_Window = 'ROOT_WINDOW' THEN
  Go_Item('EMPNO');
ELSIF :System.Event_Window = 'DEPT_WINDOW' THEN
  Go_Item('DEPTNO');
END IF;
```
SYSTEM.FORM_STATUS

Syntax: SYSTEM.FORM_STATUS

Description: SYSTEM.FORM_STATUS represents the status of the current form. The value can be one of three character strings:

CHANGED Indicates that the form contains at least one block with a Changed record. The value of SYSTEM.FORM_STATUS becomes CHANGED only after at least one record in the form has been changed and the associated navigation unit has also changed.

NEW Indicates that the form contains only New records.

QUERY Indicates that a query is open. The form contains at least one block with QUERY records and no blocks with CHANGED records.

Usage Notes: Each time this value is referenced, it must be constructed by Oracle Forms. If your form contains a large number of blocks and many records, using SYSTEM.FORM_STATUS could affect performance.

Example: Assume that you want to create a trigger that performs a commit before clearing a form if there are changes to commit within that form. The following Key–CLRFRM trigger performs this function.

IF :System.Form_Status = 'CHANGED'
   THEN Commit_Form;
END IF;
Clear_Form;

See Also: SYSTEM.BLOCK_STATUS and SYSTEM.RECORD_STATUS system variables

SYSTEM.LAST_FORM

Syntax: SYSTEM.LAST_FORM

Description: SYSTEM.LAST_FORM represents the form module ID of the previous form in a multi–form application, where multiple forms have been invoked using OPEN_FORM. The value can be one of two character strings: either the form module ID or NULL.

Usage Notes: SYSTEM.LAST_FORM is not valid with CALL_FORM.
SYSTEM LAST_QUERY

Syntax: SYSTEM.LAST_QUERY

Description: SYSTEM.LAST_QUERY represents the query SELECT statement that Oracle Forms most recently used to populate a block during the current Runform session. The value is always a character string.

Example 1: Assume that you want to generate a report in Oracle Reports that retrieves information identical to a query you perform in Oracle Forms. The following examples show how to use SYSTEM.LAST_QUERY to extract the WHERE/ORDER BY clause from the last query so you can pass the results to Oracle Reports using the RUN_PRODUCT built-in.

FUNCTION Last_Where_Clause
RETURN VARCHAR2
IS
    tmp_lstqry VARCHAR2(10000) := :System.Last_Query;
    tmp_curblk VARCHAR2(40);
    tmp_index NUMBER;
    tmp_where VARCHAR2(2000);
BEGIN
    /*
    ** See if we can find the word 'WHERE' in
    ** the text of the Last Query
    */
    tmp_index:= INSTR(tmp_lstqry,'WHERE');
    /*
    ** If we found it (at a positive index into
    ** the string), we extract the remainder of
    ** the text that follows the word 'WHERE' as
    ** the Where clause. This might include ORDER BY
    ** clause, too.
    */
    IF tmp_index > 0 THEN
        tmp_where := SUBSTR(tmp_lstqry, tmp_index + 6);
    END IF;
    RETURN tmp_where;
EXCEPTION
    WHEN OTHERS THEN
        RETURN NULL;
END;

PROCEDURE Run_Report_For_Last_Query
IS
    pl ParamList;
    wc VARCHAR2(2000); -- The Where Clause to Pass
BEGIN
    /*
    ** Create a parameter list for parameter passing
pl := Create_Parameter_List('tmp');

wc := Last_Where_Clause;

IF wc IS NOT NULL THEN
    Add_Parameter(pl, 'the_Where_Clause', TEXT_PARAMETER, wc);
END IF;

Run_Product(REPORTS, 'rep0058.rdf', SYNCHRONOUS, BATCH, FILESYSTEM, pl);

Destroy_Parameter_List(pl);
END;
**SYSTEM.LAST_RECORD**

**Syntax:**

```plaintext
SYSTEM.LAST_RECORD
```

**Description:**

SYSTEM.LAST_RECORD indicates whether the current record is the last record in a block’s list of records. The value is one of the following two CHAR values:

- **TRUE**
  - Indicates that the current record is the last record in the current block’s list of records.
- **FALSE**
  - Indicates that the current record is not the last record in the current block’s list of records.

**Example:**

Assume that you want to create a user-defined procedure that displays a custom message when an operator navigates to the last record in a block’s list of records. The following user-defined procedure performs the basic function.

```plaintext
PROCEDURE LAST_RECORD_MESSAGE IS
BEGIN
  IF :System.Last_Record = 'TRUE'
    THEN Message('You are on the last row');
  END IF;
END;
```

You can then redefine [Down], [Next Record], and [Scroll Down] to call this user-defined procedure in addition to their normal processing.

---

**SYSTEM.MASTER_BLOCK**

**Syntax:**

```plaintext
SYSTEM.MASTER_BLOCK
```

**Description:**

This system variable works with its companion SYSTEM.COORDINATION_OPERATION to help an On-Clear-Details trigger determine what type of coordination-causing operation fired the trigger, and on which master block of a master-detail relation.

The values of the two system variables remain constant throughout the clearing phase of any block synchronization.

SYSTEM.MASTER_BLOCK represents the name of the driving master block, and SYSTEM_COORDINATION_OPERATION represents the coordination-causing event that occurred on the master block.

More details are in the description for SYSTEM.COORDINATION_OPERATION.
SYSTEM.MESSAGE_LEVEL

Syntax: `SYSTEM.MESSAGE_LEVEL`

Description: `SYSTEM.MESSAGE_LEVEL` represents one of the following message severity levels: 0, 5, 10, 15, 20, or 25. The value is always a character string.

During a Runform session, Oracle Forms suppresses all messages with a severity level that is the same or lower (less severe) than the indicated severity level.

Assign a value to the `SYSTEM.MESSAGE_LEVEL` system variable with standard PL/SQL syntax:

```
:System.Message_Level := value;
```

The legal values for `SYSTEM.MESSAGE_LEVEL` are 0, 5, 10, 15, 20, and 25. Oracle Forms does not suppress prompts or vital error messages, no matter what severity level you select.

Default: 0

Example: Assume that you want Oracle Forms to display only the most severe messages (level 25). The following Pre–Form trigger suppresses all messages at levels 20 and below:

```
:System.Message_Level := ‘20’;
```
Usage Notes: If you are using SYSTEM.MODE to check whether the current block is in Enter Query mode, be aware that if you are testing from a When–Button–Pressed trigger in a control block, you will never be in Enter Query mode because the control block is not the current block.

Example: Assume that you want Oracle Forms to display an LOV when the operator enters query mode and the input focus is in a particular text item. The following trigger accomplishes that operation.

```/*
** When–New–Item–Instance Trigger
*/
BEGIN
  IF :System.Cursor_Item = 'EMP.EMPNO' and 
     :System.Mode = 'ENTER–QUERY'
  THEN
    IF NOT Show_Lov('my_lov') THEN
      RAISE Form_Trigger_Failure;
    END IF;
  END IF;
END;
```

**SYSTEM.MOUSE_BUTTON_PRESSED**

**Syntax:** `SYSTEM.MOUSE_BUTTON_PRESSED`

**Description:** `SYSTEM.MOUSE_BUTTON_PRESSED` indicates the number of the button that was clicked. Mouse button support is limited to buttons 1 and 2 (left or middle) on a three button mouse. The value is always a character string.

**Restrictions:** Mouse button 3, or the right mouse button, is reserved for future popup support.

**Example:**

```/*
** Trigger: When–Mouse–Click
** Example: When mouse button 1 is pressed,
**          a help window appears.
*/
DECLARE
  the_button_pressed  VARCHAR(1);
BEGIN
  the_button_pressed := :System.Mouse_Button_Pressed;
  IF the_button_pressed = '1' THEN
    Show_Window('online_help',20,5);
  END IF;
END;
```
SYstem.MOUSE_BUTTON_SHIFT_STATE

**Syntax:** SYSTEM.MOUSE_BUTTON_SHIFT_STATE

**Description:** SYSTEM.MOUSE_BUTTON_SHIFT_STATE indicates the key that was pressed during the click, such as SHIFT, ALT, or CONTROL. The value is always a character string.

<table>
<thead>
<tr>
<th>Key Pressed</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIFT</td>
<td>Shift+</td>
</tr>
<tr>
<td>CONTROL</td>
<td>Control+</td>
</tr>
<tr>
<td>ALT</td>
<td>Alt+</td>
</tr>
<tr>
<td>SHIFT+CONTROL</td>
<td>Shift+Control+</td>
</tr>
</tbody>
</table>

**Example:**

```/*
**  Trigger:  When–Mouse–Click
**  Example:  If the operator presses down on the Shift key and
**            then clicks on a boilerplate image, a window
**            appears.
*/
DECLARE
  key_pressed  VARCHAR(5) := 'FALSE';
  x_position_clicked NUMBER(1);
  y_position_clicked NUMBER(1);
BEGIN
  key_pressed := :System.Mouse_Button_Shift_State;
  x_position_clicked := To_Number(:System.Mouse_X_Pos);
  y_position_clicked := To_Number(:System.Mouse_Y_Pos);
  /*
  **  If the operator shift–clicked within the x and y
  **  coordinates of a boilerplate image, display a window.
  */
  IF  key_pressed = 'Shift+' AND x_position_clicked
      BETWEEN 10 AND 20 AND y_position_clicked BETWEEN 10
      AND 20 THEN
    Show_Window('boilerplate_image_window');
  END IF;
END;```
**SYSTEM.MOUSE_CANVAS**

**Syntax:**  
SYSTEM.MOUSE_CANVAS

**Description:** If the mouse is in a canvas, SYSTEM.MOUSE_CANVAS represents the name of that canvas as a CHAR value. If the mouse is in an item, this variable represents the name of the canvas containing the item.

SYSTEM.MOUSE_CANVAS is NULL if:
- the mouse is not in a canvas
- the platform is non-GUI

**Example:**
```sql
/*
** Trigger:  When-Mouse-Move
** Example:  When the mouse enters any one of several overlapping
**            canvases, Oracle Forms brings that canvas to the
**            front.
*/
DECLARE
    canvas_to_front  VARCHAR(50);
BEGIN
    canvas_to_front := :System.Mouse_Canvas;
    Show_View(canvas_to_front);
END;
```

**SYSTEM.MOUSE_FORM**

**Syntax:**  
SYSTEM.MOUSE_FORM

**Description:** If the mouse is in a form module, SYSTEM.MOUSE_FORM represents the name of that form module as a CHAR value. For example, if the mouse is in Form_Module1, the value for SYSTEM.MOUSE_ITEM is FORM_MODULE1.

**Note:** SYSTEM.MOUSE_FORM is NULL if the platform is not a GUI platform.
SYSTEM.MOUSE_ITEM

**Syntax:**

SYSTEM.MOUSE_ITEM

**Description:**

If the mouse is in an item, SYSTEM.MOUSE_ITEM represents the name of that item as a CHAR value. For example, if the mouse is in Item1 in Block2, the value for SYSTEM.MOUSE_ITEM is :BLOCK2.ITEM1.

SYSTEM.MOUSE_ITEM is NULL if:

- the mouse is not in an item
- the platform is not a GUI platform

**Example:**

```plaintext
/* Trigger:  When–Mouse–Click
**  Example:  Dynamically repositions an item if:
**             1) the operator clicks mouse button 2
**                on an item and
**             2) the operator subsequently clicks mouse button
**                2 on an area of the canvas that is
**                not directly on top of another item.
*/

DECLARE
    item_to_move       VARCHAR(50);
    the_button_pressed VARCHAR(50);
    target_x_position  NUMBER(3);
    target_y_position  NUMBER(3);
    the_button_pressed VARCHAR(1);
BEGIN
    /* Get the name of the item that was clicked.
    */
    item_to_move := :System.Mouse_Item;
    the_button_pressed := :System.Mouse_Button_Pressed;
    /*
    ** If the mouse was clicked on an area of a canvas that is
    ** not directly on top of another item, move the item to
    ** the new mouse location.
    */
    IF item_to_move IS NOT NULL AND the_button_pressed = '2' THEN
        target_x_position := To_Number(:System.Mouse_X_Pos);
        target_y_position := To_Number(:System.Mouse_Y_Pos);
        Set_Item_Property(item_to_move,position,
            target_x_position,target_y_position);
        target_x_position := NULL;
        target_y_position := NULL;
        item_to_move := NULL;
    END IF;
END;
```
**SYSTEM.MOUSE_RECORD**

**Syntax:**
SYSTEM.MOUSE_RECORD

**Description:**
If the mouse is in a record, SYSTEM.MOUSE_RECORD represents that record’s record number as a CHAR value.

**Note:** SYSTEM.MOUSE_RECORD is 0 if the mouse is not in an item (and thus, not in a record).

**Example:**
/*
**  Trigger:  When–Mouse–Move
**  Example:  If the mouse is within a record, display a window
**            that contains an editing toolbar.
***/
DECLARE
  mouse_in_record  NUMBER(7);
BEGIN
  mouse_in_record := To_Number(:System.Mouse_Record);
  IF mouse_in_record > 0 THEN
    Show_Window('editing_toolbar');
  END IF;
END;

**SYSTEM.MOUSE_RECORD_OFFSET**

**Syntax:**
SYSTEM.MOUSE_RECORD_OFFSET

**Description:**
If the mouse is in a record, SYSTEM.MOUSE_RECORD_OFFSET represents the offset from the first visible record as a CHAR value.

For example, if the mouse is in the second of five visible records in a multi–record block, SYSTEM.MOUSE_RECORD_OFFSET is 2. (SYSTEM.MOUSE_RECORD_OFFSET uses a 1–based index).

**Note:** SYSTEM.MOUSE_RECORD_OFFSET is 0 if the mouse is not in an item (and thus, not in a record).
SYSTEM.MOUSE_X_POS

Syntax: SYSTEM.MOUSE_X_POS

Description: SYSTEM.MOUSE_X_POS represents (as a CHAR value) the x coordinate of the mouse in the units of the current form coordinate system. If the mouse is in an item, the value is relative to the upper left corner of the item’s bounding box. If the mouse is on a canvas, the value is relative to the upper left corner of the canvas.

Note: SYSTEM.MOUSE_X_POS is always NULL on character mode platforms.

Example: /*
** Example: See SYSTEM.MOUSE_ITEM and
** SYSTEM.MOUSE_BUTTON_SHIFT_STATE.
*/

SYSTEM.MOUSE_Y_POS

Syntax: SYSTEM.MOUSE_Y_POS

Description: SYSTEM.MOUSE_Y_POS represents (as a CHAR value) the y coordinate of the mouse, using units of the current coordinate system. If the mouse is in an item, the value is relative to the upper left corner of the item’s bounding box. If the mouse is on a canvas, the value is relative to the upper left corner of the canvas.

Note: SYSTEM.MOUSE_Y_POS is always NULL on character mode platforms.

Example: /*
** Example: See SYSTEM.MOUSE_ITEM and
** SYSTEM.MOUSE_BUTTON_SHIFT_STATE.
*/
SYSTEM.RECORD_STATUS

Syntax: SYSTEM.RECORD_STATUS

Description: SYSTEM.RECORD_STATUS represents the status of the record where the cursor is located. The value can be one of four character strings:

- CHANGED: Indicates that a queried record’s validation status is Changed.
- INSERT: Indicates that the record’s validation status is Changed and that the record does not exist in the database.
- NEW: Indicates that the record’s validation status is New.
- QUERY: Indicates that the record’s validation status is Valid and that it was retrieved from the database.

Usage Notes: Both SYSTEM.RECORD_STATUS and the GET_RECORD_PROPERTY built-in return the status of a record in a given block, and in most cases, they return the same status. However, there are specific cases in which the results may differ.

SYSTEM.RECORD_STATUS can in certain cases return a value of NULL, because SYSTEM.RECORD_STATUS is undefined when there is no current record in the system. For example, in a When-Clear-Block trigger, Oracle Forms is at the block level in its processing sequence, so there is no current record to report on, and the value of SYSTEM.RECORD_STATUS is NULL.

GET_RECORD_PROPERTY, on the other hand, always has a value of NEW, CHANGED, QUERY, or INSERT, because it returns the status of a specific record without regard to the processing sequence or whether the record is the current record.

Example: Assume that you want to create a trigger that performs a commit before clearing a Changed record. The following Key-CLRREC trigger performs this function.

```sql
IF :System.Record_Status IN ('CHANGED', 'INSERT') THEN
    Commit_Form;
END IF;
Clear_Record;
```

See Also: GET_RECORD_PROPERTY and SET_RECORD_PROPERTY built-ins;
SYSTEM.BLOCK_STATUS and SYSTEM.RECORD_STATUS variables
SYSTEM.SUPPRESS_WORKING

Syntax: SYSTEM.SUPPRESS_WORKING

Description: SYSTEM.SUPPRESS_WORKING suppresses the “Working...” message in Runform, in order to prevent the screen update usually caused by the display of the "Working..." message. The value of the variable is one of the following two CHAR values:

- **TRUE** Prevents Oracle Forms from issuing the "Working..." message.
- **FALSE** Allows Oracle Forms to continue to issue the "Working..." message.

Example: Assume that you want to have the form filled with data when the operator enters the form. The following When–New–Form–Instance trigger will prevent the unwanted updates that would normally occur when you fill the blocks with data.

```
:System.Suppress Working := 'TRUE';
Go_Block ('DEPT');
Execute_Query;
Go_Block ('EMP');
Execute_Query;
Go_Block ('DEPT');
:System.Suppress Working := 'FALSE';
```

SYSTEM.TRIGGER_BLOCK

Syntax: SYSTEM.TRIGGER_BLOCK

Description: SYSTEM.TRIGGER_BLOCK represents the name of the block where the cursor was located when the current trigger initially fired. The value is NULL if the current trigger is a Pre– or Post–Form trigger. The value is always a character string.

Example: Assume that you want to write a form–level procedure that navigates to the block where the cursor was when the current trigger initially fired. The following statement performs this function.

```
Go_Block(Name_In('System.Trigger_Block'));
```
**SYSTEM.TRIGGER_ITEM**

- **Syntax:** `SYSTEM.TRIGGER_ITEM`

- **Description:** `SYSTEM.TRIGGER_ITEM` represents the item (`BLOCK.ITEM`) in the scope for which the trigger is currently firing. When referenced in a key trigger, it represents the item where the cursor was located when the trigger began. The value is always a character string.

- **Restrictions:**
  - Use `SYSTEM.TRIGGER_ITEM` in triggers where the current navigation unit is the item, such as Pre-Item and Post-Item triggers. In these triggers, the value of `SYSTEM.TRIGGER_ITEM` is `block.item`.
  - Avoid using `SYSTEM.TRIGGER_ITEM` in triggers where the current navigation unit is not the item, such as Pre- and Post-Record, Block, and Form triggers. In these triggers, the value of `SYSTEM.TRIGGER_ITEM` is NULL.

- **Usage Note:** `SYSTEM.TRIGGER_ITEM` remains the same from the beginning to the end of given trigger. This differs from `SYSTEM.CURSOR_ITEM`, which may change within a given trigger when navigation takes place.

- **Example:** Assume that you want to write a user-defined procedure that navigates to the item where the cursor was when the current trigger initially fired. The following statement performs this function.

  ```
  Go_Item(:System.Trigger_Item);
  ```
SYSTEM.TRIGGER_RECORD

Syntax: SYSTEM.TRIGGER_RECORD

Description: SYSTEM.TRIGGER_RECORD represents the number of the record that Oracle Forms is processing. This number represents the record’s current physical order in the block’s list of records. The value is always a character string.

Example: In the following anonymous block, the IF statement uses SYSTEM.TRIGGER_RECORD to identify the current record before processing continues.

```
IF :System.Trigger_Record = '1'
   THEN Message('First item in this order.');
END IF;
```
Index

A
ABORT_QUERY, 3 – 17, 8 – 2
Accelerator, 5 – 21
Access keys, definition, 5 – 21
ADD_GROUP_COLUMN , 3 – 18
ADD_GROUP_ROW, 3 – 20
ADD_LIST_ELEMENT, 3 – 22
ADD_PARAMETER, 3 – 23
Add_Triggers option, 1 – 23
Adjust, 5 – 184
Adjust–to–fit, 5 – 160
Alert built–ins
   FIND_ALERT, 3 – 89
   ID_NULL, 3 – 180
   SET_ALERT_BUTTON_PROPERTY, 3 – 243
   SET_ALERT_PROPERTY, 3 – 244
   SHOW_ALERT, 3 – 286
Alert Properties
   Alert Style, 5 – 22
   Button 1, Button 2, Button 3, 5 – 31
   Message, 5 – 130
   table of Alert Properties, 5 – 5
Alert Style, 5 – 22
Alignment, 5 – 23
ALPHA data type, 5 – 56
Application built–ins
   DO_KEY, 3 – 72
   GET_APPLICATION_PROPERTY, 3 – 119
   HOST, 3 – 178
   PAUSE, 3 – 215
   SET_APPLICATION_PROPERTY, 3 – 245
   USER_EXIT, 3 – 297
Application Instance, 5 – 24
Application menu (V.3 term), A – 24
Application Properties
   Calling_Form, 5 – 32
   Connect_String, 5 – 44
   Current_Form, 5 – 50
   Current_Form_Name, 5 – 51
   Datasource, 5 – 61
   Directory, 5 – 72
   Display_Height, 5 – 75
   Display_Width, 5 – 76
   Operating_System, 5 – 152
   Password, 5 – 155
   Savepoint_Name, 5 – 177
   table of Application Properties, 5 – 5
   Timer_Name, 5 – 188
   User_Interface, 5 – 202
   User_NLS_Lang, 5 – 203
   Username, 5 – 202
APPLICATION_PARAMETER, 3 – 24
Array Processing option, 1 – 11
Asynchronously, Run Modules, 1 – 37
Auto Hint, 5 – 24
Auto–Confirm, 5 – 25
Auto–Display , 5 – 25
Auto–Query, 5 – 25
Auto–Refresh , 5 – 26
Auto–Skip (item), 5 – 28
Auto–Skip (LOV), 5 – 28
B
Background color, 5 – 214
BACKGROUND_MENU, 3 – 25
Base Table (Block), 5 – 29
Base Table (Item), 5 – 29
Batch option, 1 – 23
BELL, 3 – 26
Bevel, 5 – 30
Beveling, A – 46
Bg color, 5 – 214
Binary files, converting to text files, A – 10
Block built–ins
  BLOCK_MENU, 3 – 26
  CLEAR_BLOCK, 3 – 34
  FIND_BLOCK, 3 – 90
  GET_BLOCK_PROPERTY, 3 – 123
  GO_BLOCK, 3 – 171
  ID_NULL, 3 – 180
  NEXT_BLOCK, 3 – 208
  PREVIOUS_BLOCK, 3 – 220
  SET_BLOCK_PROPERTY, 3 – 246
Block Properties
  Base Table (Block), 5 – 29
  Column Security property, 5 – 40
  Current Record, 5 – 52
  Delete Allowed, 5 – 66
  Enterable, 5 – 79
  First_Block, 5 – 83
  First_Detail_Relation, 5 – 84
  First_Master_Relation, 5 – 85
  In Menu/Block Description, 5 – 102
  Insert Allowed, 5 – 103
  Join Condition, 5 – 107
  Key Mode, 5 – 109
  Last_Item, 5 – 112
  Locking Mode, 5 – 114
  Navigation Style, 5 – 139
  Next Navigation Block, 5 – 140
  NextBlock, 5 – 141
  Optimizer_Hint, 5 – 152
  Order By Clause, 5 – 217
  Previous Navigation Block, 5 – 156
  PreviousBlock, 5 – 158
  Primary Key (Block), 5 – 159
  Query Allowed, 5 – 160
  Query_Hits, 5 – 163
  Query_Options, 5 – 164
  Record Orientation, 5 – 169
  Records Buffered, 5 – 169
  Records Displayed, 5 – 170
  Records Fetched, 5 – 171
  Records_to_Fetch, 5 – 171
  Scroll Bar, 5 – 178
  Status, 5 – 186
  table of Block Properties, 5 – 6
  Top_Record, 5 – 190
  Transactional Triggers, 5 – 190
  Update Allowed, 5 – 193
  Where Clause/Order By Clause, 5 – 217
  BLOCK_MENU, 3 – 26, 8 – 3
  Block_Menu (Runform option), 1 – 11
  Blocks, flushing programmatically, 3 – 36
  Bottom Title, 5 – 31
  BREAK, 3 – 27
  Buffer Records in File option, 1 – 12
  Built–in subprograms, 3 – 1
    ABORT_QUERY, 3 – 17
    ADD_GROUP_COLUMN, 3 – 18
    ADD_GROUP_ROW, 3 – 20
    ADD_LIST_ELEMENT, 3 – 22
    ADD_PARAMETER, 3 – 23
    APPLICATION_PARAMETER, 3 – 24
    BACKGROUND_MENU, 3 – 25
    BELL, 3 – 26
    BLOCK_MENU, 3 – 26
    BREAK, 3 – 27
    CALL_FORM, 3 – 28
    CALL_INPUT, 3 – 31
    CHECK_RECORD_UNIQUENESS, 3 – 33
    CHECKBOX_CHECKED, 3 – 31
    COMMIT_FORM, 3 – 41
    CLEAR_EOL, 3 – 35
    CLEAR_FORM, 3 – 36
    CLEAR_ITEM, 3 – 37
    CLEAR_LIST, 3 – 38
    CLEAR_MESSAGE, 3 – 39
    CLEAR_RECORD, 3 – 40
    CLOSE_FORM, 3 – 41
    COMMIT_FORM, 3 – 41
    CONVERT_OTHER_VALUE, 3 – 43
    COPY, 3 – 44
    COPY_REGION, 3 – 45
    COUNT_REGION, 3 – 46
HIDE_VIEW, 3 – 176
HIDE_WINDOW, 3 – 176
HOST, 3 – 178
ID_NULL, 3 – 180
IMAGE_ZOOM, 3 – 181
INSERT_RECORD, 3 – 183
ISSUE_ROLLBACK, 3 – 184
ISSUE_SAVEPPOINT, 3 – 185
ITEM_ENABLED, 3 – 186
LAST_RECORD, 3 – 186
LIST_VALUES, 3 – 187
LOCK_RECORD, 3 – 188
LOGON, 3 – 189
LOGON_SCREEN, 3 – 190
LOGOUT, 3 – 192
MAIN_MENU, 3 – 193
MENU_CLEAR_FIELD, 3 – 193
MENU_NEXT_FIELD, 3 – 194
MENU_PARAMETER, 3 – 194
MENU_PREVIOUS_FIELD, 3 – 195
MENU_REDISPLAY, 3 – 195
MENU_SHOW_KEYS, 3 – 195
MESSAGE, 3 – 196
MESSAGE_CODE, 3 – 197
MESSAGE_TEXT, 3 – 198
MESSAGE_TYPE, 3 – 199
MOVE_WINDOW, 3 – 200
NAME_IN, 3 – 202
new for Version 4.5, A – 29
NEW_FORM, 3 – 205
NEXT_BLOCK, 3 – 208
NEXT_FORM, 3 – 209
NEXT_ITEM, 3 – 210
NEXT_KEY, 3 – 211
NEXT_MENU_ITEM, 3 – 211
NEXT_RECORD, 3 – 212
NEXT_SET, 3 – 213
OPEN_FORM, 3 – 213
PASTE_REGION, 3 – 215
PAUSE, 3 – 215
POPULATE_GROUP, 3 – 216
POPULATE_GROUP_WITH_QUERY, 3 – 217
POPULATE_LIST, 3 – 218
POST, 3 – 220
PREVIOUS_BLOCK, 3 – 220
PREVIOUS_FORM, 3 – 221
PREVIOUS_ITEM, 3 – 222
PREVIOUS_MENU, 3 – 223
PREVIOUS_MENU_ITEM, 3 – 223
PREVIOUS_RECORD, 3 – 224
PRINT, 3 – 224
QUERY_PARAMETER, 3 – 225
READ_IMAGE_FILE, 3 – 227
REDISPLAY, 3 – 228
REPLACE_CONTENT_VIEW, 3 – 229
REPLACE_MENU, 3 – 230
RESET_GROUP_SELECTION, 3 – 232
RESIZE_WINDOW, 3 – 233
RETRIEVE_LIST, 3 – 234
RUN_PRODUCT, 3 – 235
SCROLL_DOWN, 3 – 238
SCROLL_UP, 3 – 238
SHOW_VIEW, 3 – 239
SELECT, 3 – 241
SELECT_ALL, 3 – 241
SELECT_RECORDS, 3 – 242
SET_ALERT_BUTTON_PROPERTY, 3 – 243
SET_ALERT_PROPERTY, 3 – 244
SET_APPLICATION_PROPERTY, 3 – 245
SET_BLOCK_PROPERTY, 3 – 246
SET_CANVAS_PROPERTY, 3 – 250
SET_FORM_PROPERTY, 3 – 251
SET_GROUP_CHAR_CELL, 3 – 255
SET_GROUP_DATE_CELL, 3 – 256
SET_GROUP_NUMBER_CELL, 3 – 258
SET_GROUP_SELECTION, 3 – 259
SET_INPUT_FOCUS, 3 – 260
SET_ITEM_PROPERTY, 3 – 260
SET_LOV_COLUMNPROPERTY, 3 – 270
SET_LOV_PROPERTY, 3 – 271
SET_MENU_ITEM_PROPERTY, 3 – 272
SET_PARAMETER_ATTR, 3 – 274
SET_RADIO_BUTTON_PROPERTY, 3 – 274
SET_RECORD_PROPERTY, 3 – 276
SET_RELATION_PROPERTY, 3 – 278
SET_TIMER, 3 – 280
SET_WINDOW_PROPERTY, 3 – 284
SHOW_ALERT, 3 – 286
SHOW_BACKGROUND_MENU, 3 – 287
SHOW_EDITOR, 3 – 288
SHOW_KEYS, 3 – 290
SHOW_LOV, 3 – 290
SHOW_MENU, 3 – 291
SHOW_VIEW, 3 – 292
SHOW_WINDOW, 3 – 293
Synchronize, 3 – 294
TERMINATE, 3 – 295
UNSET_GROUP_SELECTION, 3 – 295
UP, 3 – 296
UPDATE_RECORD, 3 – 296
USER_EXIT, 3 – 297
VALIDATE, 3 – 299
VBX.FIRE EVENT, 3 – 300
VBX.GET PROPERTY, 3 – 301
VBX.GET_VALUE_PROPERTY, 3 – 302
VBX.INVOKE METHOD, 3 – 303
VBX.SETroperty, 3 – 304
VBX.SET_VALUE_PROPERTY, 3 – 305
WHERE DISPLAY, 3 – 306
WRITE_IMAGE_FILE, 3 – 306
Built-in Subprograms Tables, 3 – 7
Button 1, Button 2, Button 3, 5 – 31

C
CALL_FORM, 3 – 28, 8 – 4
CALL_INPUT, 3 – 31, 8 – 5
Calling a form
issuing savepoints for a called form, 3 – 29
passing parameters with a parameter list, 3 – 51
Calling_Form, 5 – 32
Canvas
Canvas property, 5 – 32
SET_CANVAS_PROPERTY, 3 – 250
setting VISUAL_ATTRIBUTE property, 3 – 266
Canvas and view built-ins
FIND_CANVAS, 3 – 91
FIND_VIEW, 3 – 99
GET_CANVAS_PROPERTY, 3 – 128
GET_VIEW_PROPERTY, 3 – 167
HIDE_VIEW, 3 – 176
ID_NULL, 3 – 180
PRINT, 3 – 224
SCROLL_VIEW, 3 – 239
SET_CANVAS_PROPERTY, 3 – 250
SET_VIEW_PROPERTY, 3 – 282
SHOW_VIEW, 3 – 292
Canvas–View Properties
Display X Position, 5 – 73
Display Y Position, 5 – 73
Displayed, 5 – 74
Horizontal Scroll Bar, 5 – 97
table of Canvas–view Properties, 5 – 7
Canvas–view Properties
Raise on Entry, 5 – 164
Size, 5 – 182
View Height, 5 – 210
View Horizontal Scroll Bar, 5 – 210
View Vertical Scroll Bar, 5 – 211
View Width, 5 – 210
Window, 5 – 218
X Position on Canvas, 5 – 224
Y Position on Canvas, 5 – 224
Canvas–view Type property, 5 – 33
Case Insensitive Query, 5 – 34
Case Restriction, 5 – 35
CHANGE_ALERT_MESSAGE, (now
SET_ALERT_PROPERTY), 3 – 244
CHAR data type, 5 – 56
Character Cell WD/HT, 5 – 36
Character mode logical attributes, 5 – 214
Characteristic (V.3 term), A – 24
Check Block for Query, 8 – 6
Check Block for Update, 8 – 7
Check Box Other Values, 5 – 36
Check Item for Edit, 8 – 8
Check Record for Update, 8 – 11
Check Record Uniqueness, 8 – 10
CHECK_RECORD_UNIQUENESS, 3 – 33
CHECKBOX_CHECKED, 3 – 31
Checked, 5 – 37
Checked Value, 5 – 37
Checked value (Check Box), 5 – 36
Class, 5 – 38
CLEAR_BLOCK, 3 – 34
CLEAR_EOL, 3 – 35, 8 – 13
CLEAR_FORM, 3 – 36, 8 – 14
CLEAR_ITEM, 8 – 15
CLEAR_ITEM (V.3 “CLEAR_FIELD”), 3 – 37
CLEAR_LIST, 3 – 38
CLEAR_MESSAGE, 3 – 39
CLEAR_RECORD, 3 – 40, 8 – 16
Close the Query, 8 – 17
keeping open across commits, 5 – 52
Optimize SQL Processing option, 1 – 15
Optimize Transaction Mode Processing, 1 – 16
Statistics (Runform option), 1 – 20
CUT_REGION, 3 – 56

D
Data Type, 5 – 55
Data types
ALPHA, 5 – 56
CHAR, 5 – 56
DATE, 5 – 56
DATETIME, 5 – 56
INT, 5 – 57
LONG, 5 – 58
NUMBER, 5 – 58
Database
cursors, D – 3
login, 1 – 8
Database storage
libraries, B – 8
menus, B – 6
Database_Value, 5 – 60
Datasource, 5 – 61
Date and time system default values, 4 – 3
Date and Time System Variables
$$DATE$$, 4 – 5
$$DATETIME$$, 4 – 6
$$DBDATE$$, 4 – 7
$$DBDATETIME$$, 4 – 8
$$DBTIME$$, 4 – 9
DATE data type, 5 – 56
$$DATE$$, 4 – 5
Dates, formatting, 5 – 88
DATETIME data type, 5 – 56
$$DATETIME$$, 4 – 6
$$DBDATE$$, 4 – 7
$$DBDATETIME$$, 4 – 8
DBMS_ERROR_CODE, 3 – 57
DBMS_ERROR_TEXT, 3 – 58
$$DBTIME$$, 4 – 9
Debug Messages option, 1 – 12
Debug Mode (Generate option), 1 – 24
Debug Mode (Runtime option), 1 – 12
DEBUG_MODE, 3 – 59
Default Alert Button, 5 – 61
Default Block facility, Item Type, 5 – 105
Default Button, 5 – 62
Default Directory, 5 – 72
Default file extensions, B – 2
Default Font Scaling, 5 – 62
Default Order By Clause, 5 – 217
Default value, 5 – 63, 5 – 65
Default Value (Form Parameter), 5 – 63
Default Value (Item), 5 – 63
Default Value (Menu Substitution Parameter),
5 – 65
DEFAULT_VALUE, 3 – 60, 8 – 22
Defer_Required_Enforcement, 5 – 65
Deferred, 5 – 65
Deferred (Coordination property), 5 – 47
Delete Allowed, 5 – 66
Delete option, 1 – 24
DELETE_GROUP, 3 – 61
DELETE_GROUP_ROW, 3 – 62
DELETE_LIST_ELEMENT, 3 – 64
DELETE_PARAMETER, 3 – 65
DELETE_RECORD, 3 – 66, 8 – 23
DELETE_TIMER, 3 – 67
delimiters, 5 – 88
Designer component
list of options, 1 – 31
starting, 1 – 4
Designer options
Color Mode, 1 – 33
Color Palette, 1 – 34
Generate Before Run, 1 – 34
Help, 1 – 35
Module Access, 1 – 35
Module_Type, 1 – 36
Printer, 1 – 36
Run Asynchronously, 1 – 37
Save Before Generate, 1 – 37
Suppress Hints, 1 – 37
Term, 1 – 38
Use System Editor, 1 – 38
DESTROY_PARAMETER_LIST, 3 – 69
Detail Block, 5 – 66
Direction, 5 – 67
Directory, 5 – 72
DISABLE_ITEM, A – 38
Disabled
  menu items, 5 – 73, 5 – 78
text items, 5 – 104
Display Block Menu option, 1 – 11
Display Screen to Specify Logon option, 1 – 14
Display w/o Privilege, 5 – 73
Display X Position, 5 – 73
Display Y Position, 5 – 73
DISPLAY_ERROR, 3 – 70
Display_Height, 5 – 75
DISPLAY_ITEM, 3 – 70
Display_Width, 5 – 76
Displayed (Canvas–view), 5 – 74
Displayed (Item), 5 – 74
Displayed (Menu Item), 5 – 75
DO_KEY, 3 – 72, 8 – 25
Double–byte characters, 5 – 123
DOWN, 3 – 71, 8 – 26
DUPLICATE_ITEM, 3 – 73, 8 – 27
DUPLICATE_RECORD, 3 – 74, 8 – 28

E
EDIT_TEXTITEM, 3 – 75
Editor, 5 – 76
Editor Properties
  Bottom Title, 5 – 31
  Size, 5 – 182
table of Editor Properties, 5 – 8
  Top Title, 5 – 189
  Vertical Scroll Bar, 5 – 208
  Wrap Style, 5 – 222
  X Position, 5 – 222
  Y Position, 5 – 222
  Editor X Position, 5 – 77
  Editor Y Position, 5 – 77
  ENABLE_ITEM, A – 38
  Enabled (Item), 5 – 77
  Enabled (Menu Item), 5 – 78
  END_OF_GROUP constant, 3 – 20
  ENFORCE_COLUMN_SECURITY, 3 – 76
  ENTER, 3 – 77, 8 – 29
  Enter query mode, 5 – 82
  Enter the Block, 8 – 31
  Enter the Form, 8 – 32
  Enter the Item, 8 – 33
  Enter the Record, 8 – 35
  Enter the Value into an Item, 8 – 36
  ENTER_QUERY, 3 – 77, 8 – 34
  Enterable, 5 – 79
  Enterable field characteristic (V.3 term), A – 24
  Environments, portability among, B – 10
  ERASE, 3 – 79
  ERROR_CODE, 3 – 79
  ERROR_TEXT, 3 – 80
  ERROR_TYPE, 3 – 81
  EXEC IAF GET/PUT statements (V.3), A – 22
  EXEC TOOLS statements, A – 22
  EXECUTE_QUERY, 3 – 82, 8 – 37
  EXECUTE_TRIGGER, 3 – 84, 8 – 39
  Exemacro, 1 – 27
  EXIT_FORM, 3 – 85, 8 – 40
  Extensions, file formats, A – 9
  Extract option, 1 – 24

F
Fetch Records, 8 – 41
FETCH_RECORDS, 3 – 87
Field (V.3 term), A – 24
File (Menu property), 5 – 80
File formats, A – 9, A – 21
File_Name, 5 – 81
Files
  changes in file formats from V.3, A – 9
  conversion between binary and text files, A – 10
  Directory property, 5 – 72
  file extensions, B – 2
FIND_ALERT, 3 – 89
FIND_BLOCK, 3 – 90
FIND_CANVAS, 3 – 91
FIND_COLUMN, 3 – 91
FIND_EDITOR, 3 – 92
FIND_FORM, 3 – 93
FIND_GROUP, 3 – 94
FIND_ITEM, 3 – 94
FIND_LOV, 3 – 95
FIND_MENU_ITEM, 3 – 96
FIND_RELATION, 3 – 97
FIND_TIMER, 3 – 98
FIND_VIEW, 3 – 99
FIND_WINDOW, 3 – 100
Fire in Enter Query Mode, 5 – 82
First Navigation Block, 5 – 83
First_Block, 5 – 83
First_Detail_Relation, 5 – 84
First_Item, 5 – 84
First_Master_Relation, 5 – 85
FIRST_RECORD, 3 – 101, 8 – 44
Fixed Length (Item), 5 – 86
Fixed Length (Menu Substitution Param), 5 – 86
Fixed Size, 5 – 87
Flowcharts
  ABORT_QUERY, 8 – 2
  BLOCK_MENU, 8 – 3
  CALL_FORM, 8 – 4
  CALL_INPUT, 8 – 5
  Check Block for Query, 8 – 6
  Check Block for Update, 8 – 7
  Check Item for Edit, 8 – 8
  Check Record for Update, 8 – 11
  Check Record Uniqueness, 8 – 10
  CLEAR_EOL, 8 – 13
  CLEAR_FORM, 8 – 14
  CLEAR_ITEM, 8 – 15
  CLEAR_RECORD, 8 – 16
  Close the Query, 8 – 17
  COMMIT_FORM, 8 – 18
  COPY, 8 – 19
  COUNT_QUERY, 8 – 20
  CREATE_RECORD, 8 – 21
  DEFAULT_VALUE, 8 – 22
  DELETE_RECORD, 8 – 23
  DO_KEY, 8 – 25
  DOWN, 8 – 26
  DUPLICATE_ITEM, 8 – 27
  DUPLICATE_RECORD, 8 – 28
  ENTER, 8 – 29
  Enter the Block, 8 – 31
  Enter the Form, 8 – 32
  Enter the Item, 8 – 33
  Enter the Record, 8 – 35
  Enter the Value into an Item, 8 – 36
  ENTER_QUERY, 8 – 34
  EXECUTE_QUERY, 8 – 37
  EXECUTE_TRIGGER, 8 – 39
  EXIT_FORM, 8 – 40
  Fetch Records, 8 – 41
  FIRST_RECORD, 8 – 44
  Generate Sequence Number, 8 – 45
  GO_BLOCK, 8 – 46
  GO_ITEM, 8 – 47
  GO_RECORD, 8 – 48
  HOST, 8 – 49
  LAST_RECORD, 8 – 50
  Leave the Block, 8 – 51
  Leave the Form, 8 – 52
  Leave the Item, 8 – 53
  Leave the Record, 8 – 54
  Leave Unit Error Processing, 8 – 55
  Lock the Row, 8 – 58
  LOCK_RECORD, 8 – 57
  LOGON, 8 – 60
  LOGOUT, 8 – 62
  Mark Items and Records as Changed, 8 – 63
  Master–Detail Coordination, 8 – 64
  Navigate to the <Navigation Unit> Level, 8 – 66
  NEW_FORM, 8 – 67
  NEXT_BLOCK, 8 – 68
  NEXT_ITEM, 8 – 69
  NEXT_KEY, 8 – 71
  NEXT_RECORD, 8 – 73
  NEXT_SET, 8 – 74
Open the Query, 8 – 75
POST, 8 – 76
Post and Commit Transactions, 8 – 77
Prepare the Query, 8 – 86
PREVIOUS_BLOCK, 8 – 87
PREVIOUS_ITEM, 8 – 88
PREVIOUS_RECORD, 8 – 89
Process Expired Timer, 8 – 90
Process the Function Key, 8 – 91
Prompt and Answer, 8 – 92
Put Cursor At, 8 – 93
REPLACE_MENU, 8 – 96
Return for Input, 8 – 97
Rollback Form, 8 – 98
Run the Form, 8 – 99
Savepoint, 8 – 100
SCROLL_DOWN, 8 – 101
SCROLL_UP, 8 – 103
SHOW_LOV, 8 – 105
UP, 8 – 107
Validate the Block, 8 – 108
Validate the Form, 8 – 109
Validate the Item, 8 – 110
Validate the Record, 8 – 112
.FMB format[FMB], B – 4
FMB, FMT, FMX file formats, A – 9
.FMT format[FMT], B – 5
.FMX format[FMX], B – 4
Fonts
   font visual attribute, 5 – 213
   weight of, 5 – 214
Foreground color (fg color), 5 – 214
Form (V.3 term), A – 24
Form built–ins
   BELL, 3 – 26
   BREAK, 3 – 27
   CALL_FORM, 3 – 28
   CALL_INPUT, 3 – 31
   CLEAR_FORM, 3 – 36
   COMMIT_FORM, 3 – 41
   DEBUG_MODE, 3 – 59
   ENTER, 3 – 77
   ERASE, 3 – 79
   EXECUTE_TRIGGER, 3 – 84
   EXIT_FORM, 3 – 85
   FIND_FORM, 3 – 93
   FORM_FAILURE, 3 – 101
   FORM_FATAL, 3 – 103
   FORM_SUCCESS, 3 – 104
   GET_FORM_PROPERTY, 3 – 129
   HELP, 3 – 175
   ID_NULL, 3 – 180
   NEW_FORM, 3 – 205
   OPEN_FORM, 3 – 213
   POST, 3 – 220
   REDISPLAY, 3 – 228
   REPLACE_MENU, 3 – 230
   SET_FORM_PROPERTY, 3 – 251
   SHOW_KEYS, 3 – 290
   SHOW_MENU, 3 – 291
   SYNCHRONIZE, 3 – 294
Form Parameter Properties
   Data Type, 5 – 55
   Default Value, 5 – 63
   table of Form Parameter Properties, 5 – 9
Form Properties
   Character Cell WD/HT, 5 – 36
   Coordinate System, 5 – 45
   Cursor Mode, 5 – 52
   Defer_Required_Enforcement, 5 – 65
   First Navigation Block, 5 – 83
   Form_Name, 5 – 94
   Horizontal MDI Toolbar, 5 – 97
   Last_Block, 5 – 111
   Mouse Navigation Limit, 5 – 135
   Real Unit, 5 – 167
   Savepoint Mode, 5 – 176
   Starting Menu, 5 – 185
   table of Form Properties, 5 – 8
   Use 3D Controls, 5 – 201
   Use File, 5 – 199
   Validation, 5 – 203
   Validation Unit, 5 – 204
   Vertical MDI Toolbar, 5 – 207
Form storage formats
   .FMB format, B – 4
   .FMT format, B – 5
   .FMX format, B – 4
Form–level procedures, A – 8
Form/spread table (V.3 term), A – 24
FORM_FAILURE, 3 – 101
FORM_FATAL, 3 – 103
Form_Name, 5 – 94
FORM_SUCCESS, 3 – 104
Format Mask, 5 – 88
FORMS_DDL, 3 – 105
FORMS_MDI_WINDOW constant, 3 – 286
FORMS_OLE.ACTIVATE_SERVER, 3 – 109
FORMS_OLE.CLOSE_SERVER, 3 – 110
FORMS_OLE.EXEC_VERB, 3 – 111
FORMS_OLE.FIND_OLE_VERB, 3 – 112
FORMS_OLE.GET_VERB_COUNT, 3 – 114
FORMS_OLE.GET_VERB_NAME, 3 – 115
FORMS_OLE.INITIALIZE_CONTAINER, 3 – 116
FORMS_OLE INTERFACE POINTER, 3 – 113
FORMS_OLE.SERVER_ACTIVE, 3 – 117
Full-screen (Menu Style property), 5 – 130
Function keys
Runform keys, 6 – 16
static function keys, 2 – 7
typographic conventions, iii, iii
Functions and packaged procedures, A – 17

G
Generate Before Run option, 1 – 34
Generate component, configuring, 1 – 4
Generate on Upgrade option, A – 5
Generate options
Add_Triggers, 1 – 23
Batch, 1 – 23
CRT_File, 1 – 23
Debug, 1 – 24
Delete, 1 – 24
Extract, 1 – 24
Generate_on_Upgrade, 1 – 25
Help, 1 – 25
Insert, 1 – 25
Logon, 1 – 26
Module_Access, 1 – 26
Module_Type, 1 – 26
Nofail, 1 – 27
Options_Screen, 1 – 27
Output_File, 1 – 27
Parse, 1 – 28
Script, 1 – 28
Statistics, 1 – 29
Upgrade, 1 – 29
Upgrade_Roles, 1 – 30
Version, 1 – 30
Widen_Fields, 1 – 31
Generate Sequence Number, 8 – 45
Generate on Upgrade option, 1 – 25
GENERATE_SEQUENCE_NUMBER, 3 – 118
GET_APPLICATION_PROPERTY, 3 – 119
GET_BLOCK_PROPERTY, 3 – 123
GET_CANVAS_PROPERTY, 3 – 128
GET_FORM_PROPERTY, 3 – 129
GET_GROUP_CHAR_CELL, 3 – 133
GET_GROUP_DATE_CELL, 3 – 135
GET_GROUP_NUMBER_CELL, 3 – 136
GET_GROUP_RECORD_NUMBER, 3 – 138
GET_GROUP_ROW_COUNT, 3 – 139
GET_GROUP_SELECTION, 3 – 140
GET_GROUP_SELECTION_COUNT, 3 – 142
GET_ITEM_PROPERTY, 3 – 143
GET_LIST_ELEMENT_COUNT, 3 – 151
GET_LIST_ELEMENT_LABEL, 3 – 153
GET_LIST_ELEMENT_VALUE, 3 – 154
GET_LOV_PROPERTY, 3 – 155
GET_MENU_ITEM_PROPERTY, 3 – 156
GET_MESSAGE, 3 – 158
GET_PARAMETER_ATTR, 3 – 159
GET_PARAMETER_LIST, 3 – 160
GET_RADIO_BUTTON_PROPERTY, 3 – 160
GET_RECORD_PROPERTY, 3 – 163
GET_RELATION_PROPERTY, 3 – 165
GET_VIEW_PROPERTY, 3 – 167
GET_WINDOW_PROPERTY, 3 – 170
Global variables, constraints, D – 2
GO_BLOCK, 3 – 171, 8 – 46
GO_FORM, 3 – 172
GO_ITEM, 3 – 173, 8 – 47
GO_RECORD, 3 – 174, 8 – 48
Grayed–out items, Disabled, 5 – 73
Group_Name, 5 – 94
GUI
   conversion, A – 39
   functionality, A – 44
   Generate dialog, A – 5

GUI hint
   Closeable, 5 – 38
   Fixed Size, 5 – 87
   Iconifiable, 5 – 101
   Inherit Menu, 5 – 102
   Moveable, 5 – 135

H
Handles (Object IDs), 3 – 4
HELP, HELP built-in routine, 3 – 175
Help
   Designer option, 1 – 35
   Generate option, 1 – 25
   Help property (help text for item), 5 – 95
   Runform option, 1 – 13
HIDE_MENU, 3 – 175
HIDE_VIEW, 3 – 176
HIDE_WINDOW, 3 – 176
Hint (Item), 5 – 95
Hint (Menu Item), 5 – 96
Hint (Menu Substitution Parameter), 5 – 96
Horizontal MDI Toolbar, 5 – 97
Horizontal Scroll Bar, 5 – 97
HorizontalToolbar property, 5 – 98
HOST, 3 – 178, 8 – 49

I
IAPCAL command, A – 22
Icon Name, 5 – 99
Icon Title, 5 – 100
Iconic, 5 – 100
Iconifiable, 5 – 101
ID_NULL, 3 – 180
Identification, 5 – 101
IFZCAL command, A – 22
IMAGE_ZOOM, 3 – 181
In Menu/Block Description property, 5 – 102
Inherit Menu, 5 – 102
Initial Keyboard State, 5 – 103
INP format, A – 10
Insert Allowed (Block), 5 – 103
Insert Allowed (Item), 5 – 104
Insert option, 1 – 25
INSERT_RECORD, 3 – 183
INT data type, 5 – 57
Interactive option, 1 – 13
ISSUE_ROLLBACK, 3 – 184
ISSUE_SAVEPOINT, 3 – 185
Item built-ins
   CHECKBOX_CHECKED, 3 – 31
   CLEAR_EOL, 3 – 35
   CLEAR_ITEM, 3 – 37
   CONVERT_OTHER_VALUE, 3 – 43
   COPY, 3 – 44
   CUT_REGION, 3 – 56
   DEFAULT_VALUE, 3 – 60
   DISPLAY_ITEM, 3 – 70
   DUPLICATE_ITEM, 3 – 73
   EDIT_TEXTITEM, 3 – 75
   FIND_ITEM, 3 – 94
   GET_ITEM_PROPERTY, 3 – 143
   GET_RADIO_BUTTON_PROPERTY, 3 – 160
   GO_ITEM, 3 – 173
   ID_NULL, 3 – 180
   IMAGE_ZOOM, 3 – 181
   NAME_IN, 3 – 202
   NEXT_ITEM, 3 – 210
   PASTE_REGION, 3 – 215
   PREVIOUS_ITEM, 3 – 222
   READ_IMAGE_FILE, 3 – 227
   SELECT_ALL, 3 – 241
   SET_ITEM_PROPERTY, 3 – 260
   SET_RADIO_BUTTON_PROPERTY, 3 – 274
   WRITE_IMAGE_FILE, 3 – 306
Item Properties
   Alignment, 5 – 23
   Auto Hint, 5 – 24
   Auto-Skip, 5 – 28
   Base Table (Item), 5 – 29
   Bevel, 5 – 30
   Case Insensitive Query, 5 – 34
   Case Restriction, 5 – 35
   Check Box Other Values, 5 – 36
Checked Value, 5 – 37
Compression, 5 – 44
Copy Value from Item, 5 – 49
Data Type, 5 – 55
Database_Value, 5 – 60
Default Button, 5 – 62
Default Value, 5 – 63
Direction, 5 – 67
Displayed, 5 – 74
Editor, 5 – 76
Editor X Position, 5 – 77
Editor Y Position, 5 – 77
Enabled (Item), 5 – 77
Fixed Length, 5 – 86
Format Mask, 5 – 88
Hint, 5 – 95
Horizontal Scroll Bar, 5 – 97
Icon Name, 5 – 99
Iconic, 5 – 100
Initial Keyboard State, 5 – 103
Insert Allowed, 5 – 104
Item Type, 5 – 105
Item_Is_Valid, 5 – 106
Items Displayed, 5 – 106
Keep Position, 5 – 108
Label, 5 – 110
List Element, 5 – 112
List Style, 5 – 113
Lock Record, 5 – 113
LOV for Validation, 5 – 116
LOV Position, 5 – 118
Maximum Length, 5 – 123
Mirror Item, 5 – 131
Mouse Navigate, 5 – 134
Multi-line, 5 – 136
Navigable, 5 – 138
NextItem, 5 – 142
Other Values, 5 – 154
Previous Navigation Item, 5 – 157
PreviousItem, 5 – 158
Primary Key (Item), 5 – 159
Quality, 5 – 160
Query Allowed (Item), 5 – 161
Query Length, 5 – 162
Query Only, 5 – 162
Range High Value/Range Low Value, 5 – 165
Reading Order, 5 – 166
Required (Item), 5 – 174
Secure, 5 – 179
Size, 5 – 182
Sizing Style, 5 – 184
table of Item Properties, 5 – 9
Unchecked Value, 5 – 192
Update Allowed, 5 – 194
Update Only if NULL, 5 – 197
Update_Column, 5 – 196
Update_Permission, 5 – 198
Value, 5 – 205
Vertical Scroll Bar, 5 – 208
Visual Attribute Name, 5 – 213
Wrap Style, 5 – 222
X Position, 5 – 222
Y Position, 5 – 222
Item properties, validation of changes with
SET_ITEM, 3 – 267
Item Type, 5 – 105
ITEM_ENABLED, 3 – 186
Item_Is_Valid, 5 – 106
Items, 3 – 260
dynamic changes (SET_ITEM), 3 – 260
validate settings (GET_ITEM), 3 – 143
Items Displayed, 5 – 106

J
Join Condition, 5 – 107
Join Condition, constraints, D – 2

K
Keep Position, 5 – 108
Key fields, displaying system-generated, 5 – 104
Key Mode, 5 – 109
Key triggers, A – 45
  Key–F0 through Key–F9, 2 – 8
  Key–Fn trigger, 2 – 8
  Key–Others trigger, 2 – 9
  Key–Startup trigger, 2 – 80
Keyin option, 1 – 13
Keyout option, 1 – 14
L

Label (Item), 5 – 110
Label (Menu Item), 5 – 110
Label (Menu Parameter), 5 – 111
Last_Block, 5 – 111
Last_Item, 5 – 112
LAST_RECORD, 3 – 186, 8 – 50
Leave the Block, 8 – 51
Leave the Form, 8 – 52
Leave the Item, 8 – 53
Leave the Record, 8 – 54
Leave Unit Error Processing, 8 – 55
LIB format, A – 21
Library storage formats
   .PLD format, B – 9
   .PLL format, B – 8
   .PLL format stripped of source code, B – 9
database format, B – 8
List Element, 5 – 112
List Style, 5 – 113
LIST_VALUES, 3 – 187
Lock Record, 5 – 113
Lock the Row, 8 – 58
LOCK_RECORD, 3 – 188, 8 – 57
Locking, 7 – 16
   When locks are released, 7 – 18
Locking Mode, 5 – 114
Logical attributes, 5 – 214
Login to database, 1 – 8
LOGON, 3 – 189, 8 – 60
Logon option, 1 – 26
LOGON_SCREEN, 3 – 190
Logon_Screen option, 1 – 14
LOGOUT, 3 – 192, 8 – 62
LONG data type, 5 – 58
Long List, 5 – 115
LOV, 5 – 116
LOV for Validation, 5 – 116
LOV Position, 5 – 118
LOV Properties
   Auto–Confirm, 5 – 25
   Auto–Display, 5 – 25
   Auto–Refresh, 5 – 26
   Auto–Skip, 5 – 28
   Group_Name, 5 – 94
   Long List, 5 – 115
   LOV, 5 – 116
   LOV Type, 5 – 119
   Record Group, 5 – 167
   Size, 5 – 182
   table of LOV Properties, 5 – 12
   X Position, 5 – 222
   Y Position, 5 – 222
   LOV Type, 5 – 119

M

Macros, A – 17
Magic Item, 5 – 120
Main Menu, 5 – 121
MAIN_MENU, 3 – 193
Mark Items and Records as Changed, 8 – 63
Master Deletes, 5 – 122
Master–detail (Join Condition), 5 – 107
Master–Detail Coordination, 8 – 64
Maximum Length, 5 – 123
Maximum Length (Form Parameter), 5 – 124
Maximum Length (Menu Substitution Parameter), 5 – 124
Maximum length constraint, D – 2
MDI, 5 – 45
   FORMS_MDI_WINDOW constant, 3 – 286
   Widow Style property, 5 – 221
Menu built–ins
   APPLICATION_PARAMETER, 3 – 24
   BACKGROUND_MENU, 3 – 25
   FIND_MENU_ITEM, 3 – 96
   GET_MENU_ITEM_PROPERTY, 3 – 156
   HIDE_MENU, 3 – 175
   ITEM_ENABLED, 3 – 186
   MAIN_MENU, 3 – 193
   MENU_CLEAR_FIELD, 3 – 193
   MENU_NEXT_FIELD, 3 – 194
   MENU_PARAMETER, 3 – 194
   MENU_PREVIOUS_FIELD, 3 – 195
MENU_REDISPLAY, 3 – 195
MENU_SHOW_KEYS, 3 – 195
NEXT_MENU_ITEM, 3 – 211
PREVIOUS_MENU, 3 – 223
QUERY_PARAMETER, 3 – 225
SET_INPUT_FOCUS, 3 – 260
SET_MENU_ITEM_PROPERTY, 3 – 272
SHOW_BACKGROUND_MENU, 3 – 287
TERMINATE, 3 – 295
WHERE_DISPLAY, 3 – 306

Menu Item Radio Group, 5 – 125
Menu Item Roles, 5 – 125
Menu Item Type, 5 – 126

Menu items
- check, 5 – 126
- constraints, D – 2
- inaccessible, 5 – 73
- magic, 5 – 126
- plain, 5 – 126
- radio, 5 – 126
- separator, 5 – 127

Menu Module, 5 – 128
Menu Module Roles, 5 – 129
Menu nesting levels, constraints, D – 2

Menu Properties
- Accelerator property, 5 – 21
- Bottom Title, 5 – 31
- Checked, 5 – 37
- Command Text, 5 – 42
- Command Type, 5 – 43
- Default Value, 5 – 65
- Display w/o Privilege, 5 – 73
- Displayed, 5 – 75
- Enabled, 5 – 78
- File, 5 – 80
- File_Name, 5 – 81
- Fixed Length, 5 – 86
- Help, 5 – 95
- Hint (Menu Item), 5 – 96
- Hint (Menu Substitution Parameter), 5 – 96
- Icon Name, 5 – 99
- Identification, 5 – 101
- Label (Menu Item), 5 – 110
- Label (Menu Parameter), 5 – 111
- Magic Item, 5 – 120
- Main Menu, 5 – 121

Maximum Length (Menu Substitution Parameter), 5 – 124
Menu Item Radio Group, 5 – 125
Menu Item Type, 5 – 126
Menu Module, 5 – 128
Menu Role, 5 – 129
Menu Style, 5 – 130
Parameter Menus, 5 – 154
Required (Menu Substitution Parameter), 5 – 175
Secure, 5 – 179
Startup Code, 5 – 186
Subtitle, 5 – 187
table of Menu Properties, 5 – 13
 Tear-off, 5 – 188
 Use Security, 5 – 201
Menu Role, 5 – 129
Menu scroll region (Command Text), 5 – 42
Menu security group (Menu Role), 5 – 129
Menu storage formats
- .MMB format, B – 6
- .MMT format, B – 7
- .MMX format, B – 7
database format, B – 6
Menu Style, 5 – 130
MENU_CLEAR_FIELD, 3 – 193
MENU_NEXT_FIELD, 3 – 194
MENU_PARAMETER, 3 – 194
MENU_PREVIOUS_FIELD, 3 – 195
MENU_REDISPLAY, 3 – 195
MENU_SHOW_KEYS, 3 – 195

Menus
- constraints, D – 2
- Parameter Menus, 5 – 154
- standalone, A – 11, A – 18

Message Built-ins, MESSAGE, 3 – 196

Message built-ins
- CLEAR_MESSAGE, 3 – 39
- DISPLAY_ERROR, 3 – 70
- ERROR_CODE, 3 – 79
- ERROR_TEXT, 3 – 80
- GET_MESSAGE, 3 – 158
- MESSAGE_CODE, 3 – 197
- MESSAGE_TEXT, 3 – 198
MESSAGE_TYPE, 3 – 199
Message property, 5 – 130
MESSAGE_CODE, 3 – 197
MESSAGE_TEXT, 3 – 198
MESSAGE_TYPE, 3 – 199
Messages, typographic conventions, iii, iii
Migration strategies, A – 39
Mirror Item, 5 – 131
Miscellaneous object built-ins
CREATE_TIMER, 3 – 55
DELETE_TIMER, 3 – 67
FIND_EDITOR, 3 – 92
FIND_LOV, 3 – 95
FIND_TIMER, 3 – 98
ID_NULL, 3 – 180
LIST_VALUES, 3 – 187
SET_LOV_COLUMN_PROPERTY, 3 – 270
SET_LOV_PROPERTY, 3 – 271
SET_TIMER, 3 – 280
SHOW_EDITOR, 3 – 288
SHOW_LOV, 3 – 290
VALIDATE, 3 – 299
.MMB format[MMB], B – 6
MMB, MMT, MMX file formats, A – 9
.MMT format[MMT], B – 7
.MMX format[MMX], B – 7
Modal, 5 – 132
Module Access, Designer option, 1 – 35
Module_Access, Generate option, 1 – 26
Module_NLS_Lang, 5 – 133
Module_Type
   Designer option, 1 – 36
   Generate option, 1 – 26
Mouse Navigate, 5 – 134
Mouse Navigation Limit, 5 – 135
MOVE_WINDOW, 3 – 200
Moveable, 5 – 135
Multi-byte characters, 5 – 123
Multi-line, 5 – 136

N
Name, 5 – 136

NAME_IN, 3 – 202
Named Visual Attribute Properties
   Background Color, 5 – 214
   Character Mode Logical Attribute, 5 – 214
   Font, 5 – 213
   Foreground Color, 5 – 214
   Pattern, 5 – 214
   Size, 5 – 213
   Style, 5 – 214
   Weight, 5 – 214
   White on Black, 5 – 214
   Width, 5 – 214
Naming conflicts, A – 8
Navigable, 5 – 138
Navigate to the <Navigation Unit> Level, 8 – 66
Navigation (Mouse Navigation Limit), 5 – 135
Navigation Style, 5 – 139
NEW_FORM, 3 – 205, 8 – 67
Next Navigation Block, 5 – 140
Next Navigation Item, 5 – 140
NEXT_BLOCK, 3 – 208, 8 – 68
Next_Detail_Relation, 5 – 142
NEXT_FORM, 3 – 209
NEXT_ITEM, 3 – 210, 8 – 69
NEXT_KEY, 3 – 211, 8 – 71
Next_Master_Relation, 5 – 142
NEXT_MENU_ITEM, 3 – 211
NEXT_RECORD, 3 – 212, 8 – 73
NEXT_SET, 3 – 213, 8 – 74
NextBlock, 5 – 141
NextItem, 5 – 142
Nofail option, 1 – 27
Non–Isolated, 5 – 122
Non–ORACLE data sources, On–Check–Unique trigger, 2 – 11
NUMBER data type, 5 – 58
Numbers, formatting, 5 – 88

O
Object ID, 3 – 4
<table>
<thead>
<tr>
<th>Page</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OptimizeSQL option, 1 – 15</td>
</tr>
<tr>
<td>1</td>
<td>OptimizeTP option, 1 – 16</td>
</tr>
<tr>
<td>3</td>
<td>OptionsDesigner options, 1 – 31</td>
</tr>
<tr>
<td>3</td>
<td>Generate options, 1 – 21</td>
</tr>
<tr>
<td>3</td>
<td>Runform options, 1 – 9</td>
</tr>
<tr>
<td>1</td>
<td>User Preference File, 1 – 39</td>
</tr>
<tr>
<td>1</td>
<td>Options_ScreenGenerate option, 1 – 27</td>
</tr>
<tr>
<td>1</td>
<td>Runform option, 1 – 17</td>
</tr>
<tr>
<td>5</td>
<td>Oracle Terminal Resource File option, 1 – 20</td>
</tr>
<tr>
<td>5</td>
<td>Order By Clause property, 5 – 217</td>
</tr>
<tr>
<td>5</td>
<td>OS_COMMAND, A – 38</td>
</tr>
<tr>
<td>5</td>
<td>OS_COMMAND1, A – 38</td>
</tr>
<tr>
<td>5</td>
<td>Other Values, 5 – 154</td>
</tr>
<tr>
<td>3</td>
<td>Output file extensions, A – 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>OPEN_FORM, 3 – 213</td>
</tr>
<tr>
<td>5</td>
<td>Operating_System, 5 – 152</td>
</tr>
<tr>
<td>5</td>
<td>Optimizer_Hint, 5 – 152</td>
</tr>
<tr>
<td>5</td>
<td>OLE Activation Style, 5 – 143</td>
</tr>
<tr>
<td>5</td>
<td>OLE Class, 5 – 144</td>
</tr>
<tr>
<td>5</td>
<td>OLE Do In Out, 5 – 145</td>
</tr>
<tr>
<td>5</td>
<td>OLE In-place Activation, 5 – 146</td>
</tr>
<tr>
<td>5</td>
<td>OLE Popup Menu Items, 5 – 147</td>
</tr>
<tr>
<td>5</td>
<td>OLE Resize Style, 5 – 150</td>
</tr>
<tr>
<td>5</td>
<td>OLE Tenant Aspect, 5 – 150</td>
</tr>
<tr>
<td>5</td>
<td>OLE Tenant Types, 5 – 151</td>
</tr>
<tr>
<td>5</td>
<td>On–Check–Delete–Master trigger, 2 – 10</td>
</tr>
<tr>
<td>5</td>
<td>On–Check–Unique trigger, 2 – 11</td>
</tr>
<tr>
<td>5</td>
<td>On–Clear–Block trigger, 2 – 65</td>
</tr>
<tr>
<td>5</td>
<td>On–Clear–Details trigger, 2 – 12</td>
</tr>
<tr>
<td>5</td>
<td>On–Close trigger, 2 – 13</td>
</tr>
<tr>
<td>5</td>
<td>On–Column–Security trigger, 2 – 14</td>
</tr>
<tr>
<td>5</td>
<td>On–Commit trigger, 2 – 15</td>
</tr>
<tr>
<td>5</td>
<td>On–Count trigger, 2 – 16</td>
</tr>
<tr>
<td>5</td>
<td>On–Database–Record trigger, 2 – 68</td>
</tr>
<tr>
<td>5</td>
<td>On–Delete trigger, 2 – 17</td>
</tr>
<tr>
<td>5</td>
<td>On–Error trigger, 2 – 18</td>
</tr>
<tr>
<td>5</td>
<td>On–Fetch trigger, 2 – 19</td>
</tr>
<tr>
<td>5</td>
<td>On–Insert trigger, 2 – 21</td>
</tr>
<tr>
<td>5</td>
<td>On–Lock trigger, 2 – 22</td>
</tr>
<tr>
<td>5</td>
<td>On–Logon trigger, 2 – 23</td>
</tr>
<tr>
<td>5</td>
<td>On–Logout trigger, 2 – 24</td>
</tr>
<tr>
<td>5</td>
<td>On–Message trigger, 2 – 25</td>
</tr>
<tr>
<td>5</td>
<td>On–New–Field–Instance trigger, 2 – 81</td>
</tr>
<tr>
<td>5</td>
<td>On–New–Record trigger, 2 – 66</td>
</tr>
<tr>
<td>5</td>
<td>On–Populate–Details trigger, 2 – 26</td>
</tr>
<tr>
<td>5</td>
<td>On–Remove–Record trigger, 2 – 84</td>
</tr>
<tr>
<td>5</td>
<td>On–Rollback trigger, 2 – 27</td>
</tr>
<tr>
<td>5</td>
<td>On–Savepoint trigger, 2 – 28</td>
</tr>
<tr>
<td>5</td>
<td>On–Select trigger, 2 – 29</td>
</tr>
<tr>
<td>5</td>
<td>On–Sequence–Number trigger, 2 – 30</td>
</tr>
<tr>
<td>5</td>
<td>On–Update trigger, 2 – 31</td>
</tr>
<tr>
<td>5</td>
<td>On–Validate–Field trigger, 2 – 87</td>
</tr>
<tr>
<td>5</td>
<td>On–Validate–Record trigger, 2 – 89</td>
</tr>
<tr>
<td>5</td>
<td>Open the Query, 8 – 75</td>
</tr>
<tr>
<td>5</td>
<td>Open the Query, 8 – 75</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Open the Query, 8 – 75</td>
</tr>
<tr>
<td>5</td>
<td>PAGE 0 field (V.3 term), A – 24</td>
</tr>
<tr>
<td>5</td>
<td>Pages, A – 20, A – 24</td>
</tr>
<tr>
<td>5</td>
<td>Palette option, 1 – 34</td>
</tr>
</tbody>
</table>

**P**

- Packaged procedures and functions
- Oracle Forms, A – 36
- SQL*Menu V.5, A – 38
- V.4.5, A – 17
- Page 0 field (V.3 term), A – 24
- Pages, A – 20, A – 24
- Palette option, 1 – 34
- Parameter list built–ins
- ADD_PARAMETER, 3 – 23
- CREATE_PARAMETER_LIST, 3 – 51
- DELETE_PARAMETER, 3 – 65
- DESTROY_PARAMETER_LIST, 3 – 69
- GET_PARAMETER_ATTR, 3 – 159
- GET_PARAMETER_LIST, 3 – 160
- ID_NULL, 3 – 180
- RUN_PRODUCT, 3 – 235
- SET_PARAMETER_ATTR, 3 – 274
- Parameter list, creating programmatically, 3 – 51
- Parse option, 1 – 28, A – 10
- Passing parameters with a parameter list
- CALL_FORM, 3 – 28
CREATE_PARAMETER_LIST, 3 – 51
NEW_FORM, 3 – 205
RUN_PRODUCT, 3 – 235
Password, 1 – 5, 1 – 8, 5 – 155
PASTE_REGION, 3 – 215
Pattern, 5 – 214
PAUSE, 3 – 215
PECS option, 1 – 18
PL/SQL libraries, A – 21
PLA format, A – 21
.PLD format[PLD], B – 9
.PLL format stripped of source code, B – 9
.PLL format[PLL], B – 8
PLL, PLD file formats, A – 9, A – 21
Pop-up pages
   converting, A – 48
V.3 term, A – 24
POPULATE_GROUP, 3 – 216
POPULATE_GROUP_WITH_QUERY, 3 – 217
POPULATE_LIST, 3 – 218
Portability among systems, B – 10
POST, 3 – 220, 8 – 76
Post and Commit Transactions, 8 – 77
Post–Block trigger, 2 – 32
Post–Change trigger, 2 – 33
Post–Commit trigger, 2 – 35, 2 – 38
Post–Database–Commit trigger, 2 – 35
Post–Delete trigger, 2 – 36
Post–Field trigger (Post–Text–Item trigger), 2 – 46
Post–Form trigger, 2 – 37
Post–Forms–Commit trigger, 2 – 35, 2 – 38
Post–Insert trigger, 2 – 39
Post–Logon trigger, 2 – 40
Post–Logout trigger, 2 – 41
Post–Query trigger, 2 – 42
Post–Record trigger, 2 – 44
Post–Select trigger, 2 – 45
Post–Text–Item trigger, 2 – 46
Post–Update trigger, 2 – 47
Pre–Block trigger, 2 – 48
Pre–Commit trigger, 2 – 49
Pre–Delete trigger, 2 – 50
Pre–Field trigger (Pre–Text–Item trigger), 2 – 59
Pre–Form trigger, 2 – 51
Pre–Insert trigger, 2 – 52
Pre–Logon trigger, 2 – 53
Pre–Logout trigger, 2 – 54
Pre–Query trigger, 2 – 55
Pre–Record trigger, 2 – 57
Pre–Select trigger, 2 – 58
Pre–Text–Item trigger, 2 – 59
Pre–Update trigger, 2 – 60
Preferences
   Designer options, 1 – 31
   Generate options, 1 – 21
   Runform options, 1 – 9
   User Preference File, 1 – 39
Prepare the Query, 8 – 86
Prevent Masterless Operations, 5 – 155
Previous Navigation Block, 5 – 156
Previous Navigation Item, 5 – 157
PREVIOUS_BLOCK, 3 – 220, 8 – 87
PREVIOUS_FORM, 3 – 221
PREVIOUS_ITEM, 3 – 222, 8 – 88
PREVIOUS_MENU, 3 – 223
PREVIOUS_MENU_ITEM, 3 – 223
PREVIOUS_RECORD, 3 – 224, 8 – 89
PreviousBlock, 5 – 158
PreviousItem, 5 – 158
Primary Key (Block), 5 – 159
Primary Key (Item), 5 – 159
Primary key, checking programmatically, 2 – 11
PRINT, 3 – 224
Printer option, 1 – 36
Privileges, display menu items without, 5 – 73
Process Expired Timer, 8 – 90
Process the Function Key, 8 – 91
Processing
   Locking, 7 – 16
   Query processing, 7 – 2
   validation, 7 – 6
Prompt and Answer, 8 – 92
Prompts (V.3 term), A – 24
Properties
  Accelerator, 5 – 21
  Access Key, 5 – 21
  Alert Style, 5 – 22
  Alignment, 5 – 23
  Application Instance, 5 – 24
  Auto Hint, 5 – 24
  Auto–Confirm, 5 – 25
  Auto–Display, 5 – 25
  Auto–Query, 5 – 25
  Auto–Refresh, 5 – 26
  Auto–Skip (item), 5 – 28
  Auto–Skip (LOV), 5 – 28
  Base Table (Block), 5 – 29
  Base Table (Item), 5 – 29
  Bevel, 5 – 30
  Bottom Title, 5 – 31
  Button 1, Button2, Button 3, 5 – 31
  Calling_Form, 5 – 32
  Canvas, 5 – 32
  Canvas–view Type, 5 – 33
  Case Insensitive Query, 5 – 34
  Case Restriction, 5 – 35
  changing item properties, 3 – 268
  Character Cell WD/HT, 5 – 36
  Check Box Other Values, 5 – 36
  Checked, 5 – 37
  Checked Value, 5 – 37
  Class, 5 – 38
  Closeable, 5 – 38
  Column Mapping, 5 – 39
  Column Security, 5 – 40
  Column Specification, 5 – 41
  Command Text, 5 – 42
  Command Type, 5 – 43
  Compression, 5 – 44
  Connect_String, 5 – 44
  Console Window, 5 – 45
  Coordinate Information, 5 – 45
  Coordinate System, 5 – 45
  Coordination, 5 – 47
  Coordination_Status, 5 – 48
  Copy Value from Item, 5 – 49
  Current Record Attribute, 5 – 49
  Current_Form, 5 – 50
  Current_Form_Name, 5 – 51
  Current_Record, 5 – 52
  Cursor Mode, 5 – 52
  Cursor_Style, 5 – 54
  Data Type, 5 – 55
  Database_Value, 5 – 60
  Datasource, 5 – 61
  Default Alert Button, 5 – 61
  Default Button, 5 – 62
  Default Font Scaling, 5 – 62
  Default Value (Form Parameter), 5 – 63
  Default Value (Item), 5 – 63
  Default Value (Menu Substitution Parameter), 5 – 65
  Defer_Required_Enforcement, 5 – 65
  Deferred, 5 – 65
  Delete Allowed, 5 – 66
  Detail Block, 5 – 66
  Direction, 5 – 67
  Directory, 5 – 72
  Display w/o Privilege, 5 – 73
  Display X Position, 5 – 73
  Display Y Position, 5 – 73
  Display_Height, 5 – 75
  Display_Width, 5 – 76
  Displayed (Canvas–view), 5 – 74
  Displayed (Item), 5 – 74
  Displayed (Menu Item), 5 – 75
  Editor, 5 – 76
  Editor X Position, 5 – 77
  Editor Y Position, 5 – 77
  Enabled (Item), 5 – 77
  Enabled (Menu Item), 5 – 78
  Enterable, 5 – 79
  File, 5 – 80
  File_Name, 5 – 81
  Fire in Enter Query Mode, 5 – 82
  First Navigation Block, 5 – 83
  First_Block, 5 – 83
  First_Detail_Relation, 5 – 84
  First_Item, 5 – 84
  First_Master_Relation, 5 – 85
  Fixed Length (Item), 5 – 86
  Fixed Length (Menu Substitution Parameter), 5 – 86
  Fixed Size, 5 – 87
  Form_Name, 5 – 94
  Format Mask, 5 – 88
  Group_Name, 5 – 94
  Help, 5 – 95
Hint (Item), 5 – 95
Hint (Menu Item), 5 – 96
Hint (Menu Substitution Parameter), 5 – 96
Horizontal MDI Toolbar, 5 – 97
Horizontal Scroll Bar, 5 – 97
Horizontal Toolbar, 5 – 98
Icon Name, 5 – 99
Icon Title, 5 – 100
Iconic, 5 – 100
Iconifiable, 5 – 101
Identification, 5 – 101
In Menu/Block Description, 5 – 102
Inherit Menu, 5 – 102
Initial Keyboard State, 5 – 103
Insert Allowed (Block), 5 – 103
Insert Allowed (Item), 5 – 104
Item Type, 5 – 105
Item_Is_Valid, 5 – 106
Items Displayed, 5 – 106
Join Condition, 5 – 107
Key Mode, 5 – 108
Label (Item), 5 – 110
Label (Menu Item), 5 – 110
Label (Menu Parameter), 5 – 111
Last_Block, 5 – 111
Last_Item, 5 – 112
length constraint, D – 2
List Element, 5 – 112
List Style, 5 – 113
Lock Record, 5 – 113
Locking Mode, 5 – 114
Long List, 5 – 115
LOV, 5 – 116
LOV for Validation, 5 – 116
LOV Position, 5 – 118
LOV Type, 5 – 119
Magic Item, 5 – 120
Main Menu, 5 – 121
Master Deletes, 5 – 122
Maximum Length, 5 – 123
Maximum Length (Form Parameter), 5 – 124
Maximum Length (Menu Substitution Parameter), 5 – 124
Menu Item Radio Group, 5 – 125
Menu Item Roles, 5 – 125
Menu Item Type, 5 – 126
Menu Module, 5 – 128
Menu Module Roles, 5 – 129
Menu Role, 5 – 129
Menu Style, 5 – 130
Message, 5 – 130
Mirror Item, 5 – 131
Modal, 5 – 132
Module_NLS_Lang, 5 – 133
Mouse Navigate, 5 – 134
Mouse Navigation Limit, 5 – 135
Moveable, 5 – 135
Multi-line, 5 – 136
Navigable, 5 – 138
Navigation Style, 5 – 139
Next Navigation Block, 5 – 140
Next Navigation Item, 5 – 140
Next_Detail_Relation, 5 – 142
Next_Master_Relation, 5 – 142
NextBlock, 5 – 141
NextItem, 5 – 142
OLDElement, 5 – 143
OLE Class, 5 – 144
OLE Do In Out, 5 – 145
OLE In-place Activation, 5 – 146
OLE Popup Menu Items, 5 – 147
OLE Resize Style, 5 – 150
OLE Tenant Aspect, 5 – 150
OLE Tenant Types, 5 – 151
Operating_System, 5 – 152
Optimizer_Hint, 5 – 152
Order By Clause, 5 – 217
Other Values, 5 – 154
Overview, 5 – 2
Parameter Menus, 5 – 154
Password, 5 – 155
Prevent Masterless Operations, 5 – 155
Previous Navigation Block, 5 – 156
Previous Navigation Item, 5 – 157
PreviousBlock, 5 – 158
PreviousItem, 5 – 158
Primary Key (Block), 5 – 159
Primary Key (Item), 5 – 159
Quality, 5 – 160
Query Allowed (Block), 5 – 160
Query Allowed (Item), 5 – 161
Query Length, 5 – 162
Query Only, 5 – 162
Query_Hits, 5 – 163
Query_Options, 5 – 164
Raise on Entry, 5 – 164
Range High Value/Range Low Value, 5 – 165
Reading Order, 5 – 166
Real Unit, 5 – 167
Record Group, 5 – 167
Record Group Query, 5 – 168
Record Group Type, 5 – 168
Record Orientation, 5 – 169
Records Buffered, 5 – 169
Records Displayed, 5 – 170
Records Fetched, 5 – 171
Records_to_Fetch, 5 – 171
Reference Information, 5 – 173
Remove on Exit, 5 – 173
Rendered, 5 – 174
Required (Item), 5 – 174
Required (Menu Substitution Parameter), 5 – 175
Savepoint Mode, 5 – 176
Savepoint_Name, 5 – 177
Scroll Bar, 5 – 178
Secure (Item), 5 – 179
Secure (Menu Substitution Parameter), 5 – 179
Show Keys/Show Keys Description, 5 – 180
Show OLE Popup Menu, 5 – 181
Show OLE Tenant Type, 5 – 182
Size, 5 – 182.
Sizing Style, 5 – 184
Space Between Records, 5 – 184
Starting Menu, 5 – 185
Startup Code, 5 – 186
Status (Block), 5 – 186
Status (Record), 5 – 187
Subtitle, 5 – 187
tables of properties, 5 – 5
Tear-off, 5 – 188
Timer_Name, 5 – 188
Title, 5 – 188
Top Title, 5 – 189
Top_Record, 5 – 190
Transactional Triggers, 5 – 190
Trigger Style, 5 – 191
Trigger Text, 5 – 191
Trigger Type, 5 – 192
Unchecked Value, 5 – 192
Update Allowed (Block), 5 – 193
Update Allowed (Item), 5 – 194
Update Changed Columns, 5 – 195
Update Column, 5 – 196
Update Only if NULL, 5 – 197
Update_Permission, 5 – 198
Use 3D Controls, 5 – 201
Use File, 5 – 199
Use Security, 5 – 201
User_Interface, 5 – 202
User_NLS_Lang, 5 – 203
Username, 5 – 202
Validation, 5 – 203
Validation Unit, 5 – 204
Value, 5 – 205
VBOX Control File, 5 – 205
VBOX Control Name, 5 – 206
VBOX Control Value Property, 5 – 206
Vertical MDI Toolbar, 5 – 207
Vertical Scroll Bar, 5 – 208
Vertical Toolbar, 5 – 208
View, 5 – 209
View Height, 5 – 210
View Horizontal Scroll Bar, 5 – 210
View Vertical Scroll Bar, 5 – 211
View Width, 5 – 210
Visible, 5 – 211
Visual_Attribute, 5 – 216
Where Clause/Order By Clause, 5 – 217
Window, 5 – 218
Window Style, 5 – 221
Window_Handle, 5 – 219
Window_State, 5 – 220
Wrap Style, 5 – 222
X Position, 5 – 222
X Position on Canvas, 5 – 224
Y Position, 5 – 222
Y Position on Canvas, 5 – 224
Zoomable, 5 – 224
Pull–down (Menu Style property), 5 – 130
Put Cursor At, 8 – 93

Quality, 5 – 160
Query Allowed (Block), 5 – 160
Query Allowed (Item), 5 – 161
Query built–ins
ABORT_QUERY, 3 – 17
COUNT_QUERY, 3 – 46
ENTER_QUERY, 3 – 77
EXECUTE_QUERY, 3 – 82
NEXT_KEY, 3 – 211
NEXT_SET, 3 – 213
Query Length, 5 – 162
Query Only, 5 – 162
Query Only Mode option, 1 – 19
Query_Hits, 5 – 163
Query_Options, 5 – 164
QUERY_PARAMETER, 3 – 225
Queryable (Query Allowed property), 5 – 161
Quiet Mode option, 1 – 19

R
Radio button
  Label property, 5 – 110
  Position property, 5 – 222
  Value property, 5 – 205
  Visual_Attribute property, 5 – 216
Radio Group
  Label property, 5 – 110
  Menu Item Radio Group property, 5 – 125
Raise on Entry, 5 – 164
Range High Value/Range Low Value, 5 – 165
Read Input Keystrokes from File option, 1 – 13
READ_IMAGE_FILE, 3 – 227
Reading Order, 5 – 166
Real Unit, 5 – 167
Record built-ins
  CHECK_RECORD_UNIQUENESS, 3 – 33
  CLEAR_RECORD, 3 – 40
  CREATE_QUERIED_RECORD, 3 – 52
  CREATE_RECORD, 3 – 54
  DELETE_RECORD, 3 – 66
  DOWN, 3 – 71
  DUPLICATE_RECORD, 3 – 74
  FIRST_RECORD, 3 – 101
  GENERATE_SEQUENCE_NUMBER, 3 – 118
  GET_RECORD_PROPERTY, 3 – 163
  GO_RECORD, 3 – 174
  INSERT_RECORD, 3 – 183
  LAST_RECORD, 3 – 186
  LOCK_RECORD, 3 – 188
  NEXT_RECORD, 3 – 212
  NEXT_SET, 3 – 213
  PREVIOUS_RECORD, 3 – 224
  SCROLL_DOWN, 3 – 238
  SCROLL_UP, 3 – 238
  SELECT_RECORDS, 3 – 242
  SET_RECORD_PROPERTY, 3 – 276
  UP, 3 – 296
  UPDATE_RECORD, 3 – 296
Record Group, 5 – 167
Record group built-ins
  ADD_GROUP_COLUMN, 3 – 18
  ADD_GROUP_ROW, 3 – 20
  CREATE_GROUP, 3 – 47
  CREATE_GROUP_FROM_QUERY, 3 – 49
  DELETE_GROUP, 3 – 61
  DELETE_GROUP_ROW, 3 – 62
  ENFORCE_COLUMN_SECURITY, 3 – 76
  FIND_COLUMN, 3 – 91
  FIND_GROUP, 3 – 94
  GET_GROUP_CHAR_CELL, 3 – 133
  GET_GROUP_DATE_CELL, 3 – 135
  GET_GROUP_NUMBER_CELL, 3 – 136
  GET_GROUP_RECORD_NUMBER, 3 – 138
  GET_GROUP_ROW_COUNT, 3 – 139
  GET_GROUP_SELECTION, 3 – 140
  GET_GROUP_SELECTION_COUNT, 3 – 142
  ID_NULL, 3 – 180
  POPULATE_GROUP, 3 – 216
  POPULATE_GROUP_WITH_QUERY, 3 – 217
  RESET_GROUP_SELECTION, 3 – 232
  SET_GROUP_CHAR_CELL, 3 – 255
  SET_GROUP_DATE_CELL, 3 – 256
  SET_GROUP_NUMBER_CELL, 3 – 258
  SET_GROUP_SELECTION, 3 – 259
  UNSET_GROUP_SELECTION, 3 – 295
Record Group Properties
  Other Values, 5 – 154
  Record Group Type, 5 – 168
  table of Record Group Properties, 5 – 14
Record Group Query, 5 – 168
Record Group Type, 5 – 168
Record groups, constraints, D – 2
Record Orientation, 5 – 169
Record status (Status property), 5 – 187
Records, Records Fetched, 5 – 171
Records Buffered, 5 – 169
Records Displayed, 5 – 170
Records Fetched, 5 – 171
Records, checking for uniqueness, 3 – 33
Records_to_Fetch, 5 – 171
REDISPLAY, 3 – 228
Reference Information, 5 – 173
Referenced objects, A – 6
Relation built–ins
   FIND_RELATION, 3 – 97
   GET_RELATION_PROPERTY, 3 – 165
   ID_NULL, 3 – 180
   SET_RELATION_PROPERTY, 3 – 278
Relation Properties
   Auto–Query, 5 – 25
   Coordination, 5 – 47
   Coordination_Status, 5 – 48
   Deferred property, 5 – 65
   Detail Block, 5 – 66
   Master Deletes, 5 – 122
   Next_Detail_Relation, 5 – 142
   Next_Master_Relation, 5 – 142
   Prevent Masterless Operations, 5 – 155
   table of Relation Properties, 5 – 14
Remove on Exit, 5 – 173
Rendered, 5 – 174
REPLACE_CONTENT_VIEW, 3 – 229
REPLACE_MENU, 3 – 230, 8 – 96
Required (Item), 5 – 174
Required (Menu Substitution Param), 5 – 175
RESET_GROUP_SELECTION, 3 – 232
RESIZE_WINDOW, 3 – 233
Restricted built–in subprograms, 3 – 5
   BLOCK_MENU, 3 – 26
   CALL_INPUT, 3 – 31
   CLEAR_BLOCK, 3 – 34
   CLEAR_EOL, 3 – 35
   CLEAR_FORM, 3 – 36
   CLEAR_ITEM, 3 – 37
   CLEAR_MESSAGE, 3 – 39
   CLEAR_RECORD, 3 – 40
   COMMIT_FORM, 3 – 41
   CONVERT_OTHER_VALUE, 3 – 43
   COPY_REGION, 3 – 45
   COUNT_QUERY, 3 – 46
   CREATE_RECORD, 3 – 54
   CUT_REGION, 3 – 56
   DELETE_RECORD, 3 – 66
   DO_KEY, 3 – 72
   DOWN, 3 – 71
   DUPLICATE_ITEM, 3 – 73
   DUPLICATE_RECORD, 3 – 74
   EDIT_TEXTITEM, 3 – 75
   ENFORCE_COLUMN_SECURITY, 3 – 76
   ENTER, 3 – 77
   ENTER_QUERY, 3 – 77
   EXECUTE_QUERY, 3 – 82
   EXECUTE_TRIGGER, 3 – 84
   EXIT_FORM, 3 – 85
   FIRST_RECORD, 3 – 101
   GO_BLOCK, 3 – 171
   GO_ITEM, 3 – 173
   GO_RECORD, 3 – 174
   HELP, 3 – 175
   INSERT_RECORD, 3 – 183
   LAST_RECORD, 3 – 186
   NEW_FORM, 3 – 205
   NEXT_BLOCK, 3 – 208
   NEXT_ITEM, 3 – 210
   NEXT_KEY, 3 – 211
   NEXT_RECORD, 3 – 212
   NEXT_SET, 3 – 213
   PASTE_REGION, 3 – 215
   POST, 3 – 220
   PREVIOUS_BLOCK, 3 – 220
   PREVIOUS_ITEM, 3 – 222
   SCROLL_DOWN, 3 – 238
   SCROLL_UP, 3 – 238
   SHOW_LOV, 3 – 290
   UP, 3 – 296
   RETRIEVE_LIST, 3 – 234
   Return for Input, 8 – 97
   Rollback Form, 8 – 98
   Rollbacks, 3 – 37
   ROWID, 5 – 109
   Run Asynchronously option, 1 – 37
   Run the Form, 8 – 99
   RUN_PRODUCT, 3 – 235
   Runform
       Optimize Transaction Mode Processing, 1 – 16
       starting, 1 – 4

Index – 23
Runform Interface
choosing items and objects, 6 – 3
getting help, 6 – 2
interacting with form items, 6 – 4
modifying the database, 6 – 10
navigating around your form, 6 – 3
querying the database, 6 – 6
starting/exiting Runform, 6 – 2
using menus, 6 – 13
viewing the status line, 6 – 2

Runform options
Array, 1 – 11
Block_Menu, 1 – 11
Buffer_Records, 1 – 12
Debug, 1 – 12
Debug_Messages, 1 – 12
Help, 1 – 13
Interactive, 1 – 13
Keyin, 1 – 13
Keyout, 1 – 14
Logon_Screen, 1 – 14
OptimizeSQL, 1 – 15
OptimizeTP, 1 – 16
Options_Screen, 1 – 17
Output_File, 1 – 17
PECS, 1 – 18
Query_Only, 1 – 19
Quiet, 1 – 19
Session, 1 – 19
Statistics, 1 – 20
Term, 1 – 20
Window_State, 1 – 21

Secure (Item), 5 – 179
Secure (Menu Substitution Parameter), 5 – 179
Security group (Menu Role property), 5 – 129
Security Roles, Display w/out Privilege property, 5 – 73
SELECT_ALL, 3 – 241
SELECT_RECORDS, 3 – 242
Separators
delimiters (decimal, thousands), 5 – 89
menu item type, 5 – 127
NLS, 5 – 93
Session option, 1 – 19
SET_ALERT_BUTTON_PROPERTY, 3 – 243
SET_ALERT_PROPERTY, 3 – 244
SET_APPLICATION_PROPERTY, 3 – 245
SET_BLOCK_PROPERTY, 3 – 246
SET_CANVAS_PROPERTY, 3 – 250
SET_FORM_PROPERTY, 3 – 251
SET_GROUP_CHAR_CELL, 3 – 255
SET_GROUP_DATE_CELL, 3 – 256
SET_GROUP_NUMBER_CELL, 3 – 258
SET_GROUP_SELECTION, 3 – 259
SET_INPUT_FOCUS, 3 – 260
SET_ITEM_PROPERTY, 3 – 260
SET_LOV_COLUMN_PROPERTY, 3 – 270
SET_LOV_PROPERTY, 3 – 271
SET_MENU_ITEM_PROPERTY, 3 – 272
SET_PARAMETER_ATTR, 3 – 274
SET_RADIO_BUTTON_PROPERTY, 3 – 274
SET_RECORD_PROPERTY, 3 – 276
SET_RECORD_PROPERTY, 3 – 276
SET_RELATION_PROPERTY, 3 – 278
SET_TIMER, 3 – 280
SET_VIEW_PROPERTY, 3 – 282
SET_WINDOW_PROPERTY, 3 – 284
Show Help option, 1 – 13
Show Keys/Show Keys Description, 5 – 180
Show OLE Popup Menu, 5 – 181
Show OLE Tenant Type, 5 – 182
SHOW_ALERT, 3 – 286
SHOW_BACKGROUND_MENU, 3 – 287
SHOW_EDITOR, 3 – 288

S
Save Before Generate option, 1 – 37
Savepoint, 8 – 100
Savepoint Mode, 5 – 176
Savepoint_Name, 5 – 177
Screen painter (V.3 term) , A – 24
Screens, refreshing, 6 – 22
Script option, 1 – 28, A – 10
Scroll Bar, 5 – 178
SCROLL_DOWN, 3 – 238, 8 – 101
SCROLL_UP, 3 – 238, 8 – 103
SCROLL_VIEW, 3 – 239

Index – 24  Forms Reference Manual
SHOW_KEYS, 3 – 290
SHOW_LOV, 3 – 290, 8 – 105
SHOW_MENU, 3 – 291
SHOW_VIEW, 3 – 292
SHOW_WINDOW, 3 – 293
Single–user system, 2 – 22
Size, 5 – 182, 5 – 213
Sizing Style, 5 – 184
Space Between Records, 5 – 184
Spread tables (V.3 term), A – 24
Standalone menus, A – 11, A – 18
Starting Menu, 5 – 185
Startup
  of Forms components, 1 – 4
    User Preference File, 1 – 39
Startup Code, 5 – 186
Static function keys, 2 – 7
Statistics
  Generate option, 1 – 29
    Runform option, 1 – 20
Status (Block), 5 – 186
Status (Record), 5 – 187
Storage formats
  Form storage formats, B – 5
    Library storage formats, B – 8
Substitution parameters, constraints, D – 2
Subtitle, 5 – 187
Suppress Hints option, 1 – 37
Switches, A – 26
SYNCHRONIZE, 3 – 294
Synchronously, Run Modules, 1 – 37
System editor (Editor property), 5 – 76
System variables, 4 – 2
  SYSTEM.BLOCK_STATUS, 4 – 10
  SYSTEM.COORDINATION_OPERATOR, 4 – 11
  SYSTEM.CURRENT_BLOCK, 4 – 13
  SYSTEM.CURRENT_DATETIME, 4 – 14
  SYSTEM.CURRENT_FORM, 4 – 15
  SYSTEM.CURRENT_ITEM, 4 – 15
  SYSTEM.CURRENT_VALUE, 4 – 16
  SYSTEM.CURSOR_BLOCK, 4 – 16
  SYSTEM.CURSOR_ITEM, 4 – 17
  SYSTEM.CURSOR_RECORD, 4 – 18
  SYSTEM.CURSOR_VALUE, 4 – 19
  SYSTEM.CUSTOM_ITEM_EVENT, 4 – 20
  SYSTEM.CUSTOM_ITEM_EVENT_PARAMETER, 4 – 20
  SYSTEM.DATE_THRESHOLD, 4 – 21
  SYSTEM.EFFECTIVE_DATE, 4 – 22
  SYSTEM.EVENT_WINDOW, 4 – 23
  SYSTEM.FORM_STATUS, 4 – 24
  SYSTEM.LAST_QUERY, 4 – 25
  SYSTEM.LAST_RECORD, 4 – 27
  SYSTEM.MASTER_BLOCK, 4 – 27
  SYSTEM.MESSAGE_LEVEL, 4 – 28
  SYSTEM.MODE, 4 – 28
  SYSTEM.MOUSE_BUTTON_PRESSED, 4 – 29
  SYSTEM.MOUSE_BUTTON_SHIFT_PRESSED, 4 – 30
  SYSTEM.MOUSE_CANVAS, 4 – 31
  SYSTEM.MOUSE_FORM, 4 – 31
  SYSTEM.MOUSE_ITEM, 4 – 32
  SYSTEM.MOUSE_RECORD, 4 – 33
  SYSTEM.MOUSE_RECORD_OFFSET, 4 – 33
  SYSTEM.MOUSE_X_POS, 4 – 34
  SYSTEM.MOUSE_Y_POS, 4 – 34
  SYSTEM.RECORD_STATUS, 4 – 35
  SYSTEM.SUPPRESS_WORKING, 4 – 36
  SYSTEM.TRIGGER_BLOCK, 4 – 36
  SYSTEM.TRIGGER_ITEM, 4 – 37
  SYSTEM.TRIGGER_RECORD, 4 – 38

T
Tables
  Built–in subprograms, 3 – 7
    properties, 5 – 5
      triggers, 2 – 3
  Tear–off, 5 – 188
Term
  Designer option, 1 – 38
    Runform option, 1 – 20
TERMINATE, 3 – 295
Text files, converting to binary, A – 10
Time, formatting, 5 – 88
$$TIME$$, 4 – 9
Timer_Name, 5 – 188
Timers, most recently expired, 3 – 120
Title property, 5 – 188
Top Title, 5 – 189
Top_Record, 5 – 190

Transactional built-ins
CHECK_RECORD_UNIQUENESS, 3 – 33
DELETE_RECORD, 3 – 66
ENFORCE_COLUMN_SECURITY, 3 – 76
FETCH_RECORDS, 3 – 87
FORMS_DDL, 3 – 105
GENERATE_SEQUENCE_NUMBER, 3 – 118
INSERT_RECORD, 3 – 183
ISSUE_ROLLBACK, 3 – 184
ISSUE_SAVEPOINT, 3 – 185
LOGON, 3 – 189
LOGOUT, 3 – 192
SELECT_RECORDS, 3 – 242
UPDATE_RECORD, 3 – 296

Transactional options
Key_Mode, 5 – 109
Locking Mode, 5 – 114

Transactional Triggers, 5 – 190

Trigger macros, A – 17

Trigger Properties
Show Keys/Show Keys Description, 5 – 180
Table of Trigger Properties, 5 – 15
Trigger Style, 5 – 191
Trigger Text, 5 – 191
Trigger Type, 5 – 192

Trigger properties, Fire in Enter Query Mode, 5 – 82

Trigger Style, 5 – 191
Trigger Tables, 2 – 3
Trigger Text, 5 – 191
Trigger Type, 5 – 192

Triggers
Add_Triggers Generate option, 1 – 23
constraints, D – 2
Fire in Enter Query Mode property, 5 – 82
new for V.4.5, A – 27
On–Check–Delete–Master, 2 – 10
On–Check–Unique, 2 – 11
On–Clear–Details, 2 – 12
On–Close, 2 – 13
On–Column–Security, 2 – 14
On–Commit, 2 – 15
On–Count, 2 – 16
On–Delete, 2 – 17
On–Error, 2 – 18
On–Fetch, 2 – 19
On–Insert, 2 – 21
On–Lock, 2 – 22
On–Logon, 2 – 23
On–Logout, 2 – 24
On–Message, 2 – 25
On–Populate–Details, 2 – 26
On–Rollback, 2 – 27
On–Savepoint, 2 – 28
On–Select, 2 – 29
On–Sequence–Number, 2 – 30
On–Update, 2 – 31
Optimizing V2–style triggers, 1 – 15
Post–Block, 2 – 32
Post–Change, 2 – 33
Post–Database–Commit, 2 – 35
Post–Delete, 2 – 36
Post–Form, 2 – 37
Post–Forms–Commit, 2 – 38
Post–Insert, 2 – 39
Post–Logon, 2 – 40
Post–Logout, 2 – 41
Post–Query, 2 – 42
Post–Record, 2 – 44
Post–Select, 2 – 45
Post–Text–Item, 2 – 46
Post–Update, 2 – 47
Pre–Block, 2 – 48
Pre–Commit, 2 – 49
Pre–Delete, 2 – 50
Pre–Form, 2 – 51
Pre–Insert, 2 – 52
Pre–Logon, 2 – 53
Pre–Logout, 2 – 54
Pre–Query, 2 – 55
Pre–Record, 2 – 57
Pre–Select, 2 – 58
Pre–Text–Item, 2 – 59
Pre–Update, 2 – 60
renamed for V.4.5, A – 27
Triggers and Processes, 2 – 2
User–Defined, 2 – 62
When–Button–Pressed, 2 – 63
When–Checkbox–Changed, 2 – 64
When–Clear–Block, 2 – 65
When–Create–Record, 2 – 66
When–Custom–Item–Event trigger, 2 – 67
When–Database–Record, 2 – 68
When–Form–Navigate trigger, 2 – 69
When–Image–Activated, 2 – 69
When–Image–Pressed, 2 – 70
When–List–Activated, 2 – 70
When–List–Changed, 2 – 71
When–Mouse–Click, 2 – 71
When–Mouse–DoubleClick, 2 – 72
When–Mouse–Down, 2 – 74
When–Mouse–Enter, 2 – 75
When–Mouse–Leave, 2 – 76
When–Mouse–Move, 2 – 77
When–Mouse–Up, 2 – 78
When–New–Block–Instance, 2 – 79
When–New–Form–Instance, 2 – 80
When–New–Item–Instance, 2 – 81
When–New–Record–Instance, 2 – 82
When–Radio–Changed, 2 – 83
When–Remove–Record, 2 – 84
When–Timer–Expired, 2 – 85
When–Validate–Item, 2 – 87
When–Validate–Record, 2 – 89
When–Window–Activated, 2 – 91
When–Window–Closed, 2 – 92
When–Window–Deactivated, 2 – 92
When–Window–Resized, 2 – 93

Upgrading
beveling, A – 46
character-mode applications, A – 17
command line options, A – 26
conversion sequence, A – 43
denhanced functionality, A – 18
fields, A – 15
form-level procedures, A – 8
forms containing referenced objects, A – 6
from Version 2.3, A – 5
from Version 3.0, A – 4
from Version 4.0, A – 3
GUI conversion, A – 39
key triggers, A – 45
LOVs, A – 15, A – 20
master-detail relationships, A – 16
menu security roles, A – 13
menus, A – 11
migration strategies, A – 39
on a GUI platform, A – 5
pages, A – 15
Parse option, A – 10
PL/SQL libraries, A – 21
PL/SQL program units, A – 7
properties, A – 16
running after conversion, A – 5
Script option, A – 10
terminal changes, A – 24
triggers, A – 14
user exits, A – 22
using a command line on MS Windows, A – 2
using GUI functionality, A – 44
using the batch option, A – 3
V2-style triggers, A – 14
Use 3D Controls, 5 – 201, 5 – 215
Use File, 5 – 199
Use Security, 5 – 201
Use System Editor option, 1 – 38
User exits, new statements, A – 22
User interface, 3 – 120
User preferences, 1 – 39
User-defined trigger, 2 – 62

Tuning applications
Array preference, 1 – 11
OptimizeSQL option, 1 – 15
OptimizeTP option, 1 – 16

Typographic conventions, iii, iii

U
Unchecked Value, 5 – 192
Unchecked Value (Check Box), 5 – 36
Unrestricted built-in subprograms, 3 – 6
UNSET_GROUP_SELECTION, 3 – 295
UP, 3 – 296, 8 – 107
Update Allowed (Block), 5 – 193
Update Allowed (Item), 5 – 194
Update Changed Columns, 5 – 195
Update Column, 5 – 196
Update Only if NULL, 5 – 197
Update_Permission, 5 – 198
UPDATE_RECORD, 3 – 296
Upgrade option, 1 – 29, A – 2
Upgrade_Roles option, 1 – 30

Index – 27
User–Interface, 5 – 202
USER_EXIT, 3 – 297
User_NLS_Lang, 5 – 203
Userid, for database, 1 – 8
Username, 5 – 202

V
V2–style trigger (Type property), 5 – 192
V2–style triggers, A – 14
VALIDATE, 3 – 299
Validate the Block, 8 – 108
Validate the Form, 8 – 109
Validate the Item, 8 – 110
Validate the Record, 8 – 112
Validation, 5 – 203
  augmenting default validation, 7 – 6
  Item validation states, 7 – 7
  Record validation states, 7 – 9
  Standard validation checks, 7 – 10
  Status checking, 7 – 7
  Validation objects, 7 – 6
  Validation unit, 7 – 6
  When validation occurs, 7 – 6
Validation of changed item properties, 3 – 267
Validation Unit, 5 – 204
Value, 5 – 205
Variable names, constraints on, D – 2
Variables (System variables), 4 – 1
VXB Control File, 5 – 205
VXB Control Name, 5 – 206
VXB Control Value Property, 5 – 206
VXB.FIRE_EVENT, 3 – 300
VXB.GET_PROPERTY, 3 – 301
VXB.GET_VALUE_PROPERTY, 3 – 302
VXB.INVOKE_METHOD, 3 – 303
VXB_SET_PROPERTY, 3 – 304
VXB_SET_VALUE_PROPERTY, 3 – 305
Version 4.5
  concepts and functionality, A – 23
  converting forms and menus, A – 2
  file formats, A – 9
  new built–in subprograms, A – 29
  new features, A – 23
  new properties, A – 33
  new terminology, A – 24
  new triggers, A – 27
Version option, 1 – 30, A – 2
Vertical MDI Toolbar, 5 – 207
Vertical Scroll Bar, 5 – 208
Vertical Toolbar property, 5 – 208
View, 5 – 209
View Height, 5 – 210
View Horizontal Scroll Bar, 5 – 210
View Vertical Scroll Bar, 5 – 211
View Width, 5 – 210
Visible, 5 – 211
Visual Attribute Name property, 5 – 213
Visual attributes
  constraints, D – 2
  Oracle Terminal Resource File option, 1 – 20
Visual_Attribute, 5 – 216

W
Weight, 5 – 214
When–Button–Pressed trigger, 2 – 63
When–Checkbox–Changed trigger, 2 – 64
When–Clear–Block trigger, 2 – 65
When–Create–Record trigger, 2 – 66
When–Custom–Item–Event trigger, 2 – 67
When–Database–Record trigger, 2 – 68
When–Form–Navigate trigger, 2 – 69
When–Image–Activated trigger, 2 – 69
When–Image–Pressed trigger, 2 – 70
When–List–Activated trigger, 2 – 70
When–List–Changed trigger, 2 – 71
When–Mouse–Click trigger, 2 – 71
When–Mouse–DoubleClick trigger, 2 – 72
When–Mouse–Down trigger, 2 – 74
When–Mouse–Enter trigger, 2 – 75
When–Mouse–Leave trigger, 2 – 76
When–Mouse–Move trigger, 2 – 77
When–Mouse–Up trigger, 2 – 78
When–New–Block–Instance trigger, 2 – 79
When–New–Form–Instance trigger, 2 – 80
When–New–Item–Instance trigger, 2 – 81
When–New–Record–Instance trigger, 2 – 82
When–Radio–Changed trigger, 2 – 83
When–Remove–Record trigger, 2 – 84
When–Timer–Expired trigger, 2 – 85
When–Validate–Item trigger, 2 – 87
When–Validate–Record trigger, 2 – 89
When–Window–Activated trigger, 2 – 91
When–Window–Closed trigger, 2 – 92
When–Window–Deactivated trigger, 2 – 92
When–Window–Resized trigger, 2 – 93
Where clause, 5 – 217
Where Clause/Order By Clause, 5 – 217
WHERE_DISPLAY, 3 – 306
Widen Fields option, A – 46
Widen_Fields option, 1 – 31
Widow Style, 5 – 221
Window, 5 – 218
Window built-ins
  FIND_WINDOW, 3 – 100
  GET_WINDOW_PROPERTY, 3 – 170
  HIDE_WINDOW, 3 – 176
  ID_NULL, 3 – 180
  MOVE_WINDOW, 3 – 200
  REPLACE_CONTENT_VIEW, 3 – 229
  RESIZE_WINDOW, 3 – 233
  SET_WINDOW_PROPERTY, 3 – 284
  SHOW_WINDOW, 3 – 293
Window Properties
  Closeable, 5 – 38
  Fixed Size, 5 – 87
  Horizontal Scroll Bar, 5 – 97
  Horizontal Toolbar, 5 – 98
  Icon Name, 5 – 99
  Icon Title, 5 – 100
  Iconifiable, 5 – 101
  Inherit Menu, 5 – 102
  Modal, 5 – 132
  Moveable, 5 – 135
  Remove on Exit, 5 – 173
  Size, 5 – 182
  table of Window Properties, 5 – 16
Vertical Scroll Bar, 5 – 208
Vertical Toolbar, 5 – 208
View, 5 – 209
Visible, 5 – 211
Window Style, 5 – 221
Window_Handle, 5 – 219
Window_State, 5 – 220
Zoomable, 5 – 224
Window_Handle, 5 – 219
Window_State, 5 – 220
Window_State option, 1 – 21
Windows
  activating, 2 – 91
  closing, 2 – 92
  deactivating, 2 – 92
  firing triggers for, 2 – 93
  firing triggers when window activated, 2 – 91
  firing triggers when window closed, 2 – 92
  firing triggers when window deactivated, 2 – 92
FORMS_MDI_WINDOW constant, 3 – 286
MDI, 5 – 221
resizing, 2 – 93
ROOT_WINDOW, 5 – 218
Wrap Style, 5 – 222
Write Input Keystrokes to File option, 1 – 14
Write Output to Display option, 1 – 13
Write Output to File option, 1 – 17
WRITE_IMAGE_FILE, 3 – 306

X
X Position, 5 – 222
X Position on Canvas, 5 – 224

Y
Y Position, 5 – 222
Y Position on Canvas, 5 – 224
Z
Zoomable, 5 – 224
Reader’s Comment Form

Part No. A32509–2

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the topic, chapter, and page number below:

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

Please send your comments to:

Forms Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood City, CA  94065   U.S.A.
Fax: (415) 506–7200

If you would like a reply, please give your name, address, and telephone number below:

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

Thank you for helping us improve our documentation.
Forms Reference Manual

Release 4.5

Volume 2
Forms™ Reference Manual

Release 4.5
Part No. A32510-2

ORACLE®
Preface

The Forms Reference Manual, Volume 2, provides information necessary to help you use Forms 4.5. This preface includes the following topics:

- Forms Documentation Set
- Audience
- Related Publications
- Typographic Conventions
- Your Comments Are Welcome
Forms Documentation Set

The documentation set for Forms Version 4.5 consists of the following documents:

<table>
<thead>
<tr>
<th>Document</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forms Documentation Set, Version 4.5</td>
<td>A32503</td>
</tr>
<tr>
<td>Getting Started with Forms, Version 4.5</td>
<td>A32504</td>
</tr>
<tr>
<td>Forms Developer’s Guide, Version 4.5</td>
<td>A32505</td>
</tr>
<tr>
<td>Forms Advanced Techniques, Version 4.5</td>
<td>A32506</td>
</tr>
<tr>
<td>and Vol. 2</td>
<td></td>
</tr>
<tr>
<td>Forms Messages and Codes, Version 4.5</td>
<td>A32508</td>
</tr>
</tbody>
</table>

Audience

All the manuals in the Forms Version 4.5 documentation set are written for the application developer.

Related Publications

As an application designer using Version 4.5 of Forms, you should also be familiar with the following documents:

<table>
<thead>
<tr>
<th>Document</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure Builder Developer’s Guide</td>
<td>A32485</td>
</tr>
<tr>
<td>Oracle7 Server Messages and Codes Manual</td>
<td>A12379</td>
</tr>
<tr>
<td>Forms documentation for your operating system</td>
<td></td>
</tr>
</tbody>
</table>
Typographic Conventions

This manual uses the following typographic conventions to distinguish important elements from the body of the manual.

Function Keys

Forms function keys are represented by the key name enclosed in square brackets: [Next Item].

For key mappings for your particular keyboard type, refer to the following sources:

- online Forms Show Keys screen (for most keyboards, [Ctrl–K])
- the keypad diagram

For more information on the keypad diagram, refer to the Forms documentation for your operating system.

Screen Messages

Hint messages and error messages appear in a monotype font:

This is a monotype font.
Command and Example Syntax

Commands and examples appear in a monotype font, as follows:

```
SET_CANVAS_PROPERTY(canvas_name, property, value);
/*
** Built-in: SET_CANVAS_PROPERTY
** Example: Change the “background color” by setting the
canvas color dynamically at runtime to the name
** of a visual attribute you created.
*/
BEGIN
Set_Canvas_Property('my_main_cnv',VISUAL_ATTRIBUTE,'blue_text');
END;
```

Command and example syntax adhere to the following conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPPERCASE</td>
<td>Indicates a PL/SQL keyword or, if used within the parameter list for a built-in routine, a constant that you must enter exactly as spelled.</td>
</tr>
<tr>
<td>MONOTYPE</td>
<td>Used for code fragments and examples.</td>
</tr>
<tr>
<td>plain monotype</td>
<td>Indicates user-supplied items such as variables, exceptions, and actual parameters.</td>
</tr>
<tr>
<td>italic monotype</td>
<td>Indicates a default parameter. If you indicate no parameter in a parameter set, Forms applies the default parameter.</td>
</tr>
<tr>
<td>underlined monotype</td>
<td>An ellipsis shows that statements or clauses were left out. The ellipsis can appear horizontally as shown, or in vertical format.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>/*</td>
<td>A slash and asterisk begin a C-style comment.</td>
</tr>
<tr>
<td>*/</td>
<td>An asterisk and slash end a C-style comment.</td>
</tr>
<tr>
<td>—</td>
<td>Two consecutive hyphens begin an ANSI-style comment, which extends to the end of the line.</td>
</tr>
<tr>
<td>indentation</td>
<td>Indentation helps show structure within code examples, but is not required.</td>
</tr>
</tbody>
</table>
Case Sensitivity

Although neither PL/SQL nor Forms commands are case sensitive (that is, you can enter text in upper or lower case without restriction), in the documentation both upper and lower case are used for ease in reading.

In syntax examples, built-in names are shown in all caps; user-defined values are shown in lower case.

**Syntax:**

```
SET_CANVAS_PROPERTY(canvas_name, property, value);
```

**All caps.** In code examples, PL/SQL reserved words (such as BEGIN, IF, THEN, ELSE, and END) and SQL commands (such as SELECT, WHERE, ORDERBY, and INTO) are shown in all caps.

Properties, such as VISUAL_ATTRIBUTE, are also shown in upper case.

**Upper and lower case.** Names of built-in procedures (Set_Canvas_Property) and system variables (System.Suppress_Working) are shown in upper and lower case.

**Lower case.** User-defined values ('my_main_cnv') are shown in lower case.

**Example:**

```
/*
** Built-in: SET_CANVAS_PROPERTY
** Example: Change the "background color" by setting the
** canvas color dynamically at runtime to the name
** of a visual attribute you created.
*/
BEGIN
  Set_Canvas_Property('my_main_cnv', VISUAL_ATTRIBUTE, 'blue_text');
END;
```

Syntax Examples

This example illustrates first how the syntax is presented in this manual, followed by an example of how you actually enter a built-in procedure into your triggers.

**Example Syntax:**

```
SET_FORM_PROPERTY(formmodule_name, property, value);
```

**With Values:**

```
Set_Form_Property('my_form', savepoint_mode, PROPERTY_ON);
```

**Example Syntax:**

```
SET_TIMER(timer_name, milliseconds, iterate);
```

**With Values:**

```
Set_Timer('my_timer', 12000, REPEAT);
```
Your Comments Are Welcome

We value and appreciate your comments as an Oracle user and reader of the manuals. As we write, revise, and evaluate our documentation, your opinions are the most important input we receive. At the back of our printed manuals is a Reader’s Comment Form, which we encourage you to use to tell us what you like and dislike about this manual or other Oracle manuals. If the form is not available, please use the following address or FAX number.

Forms Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood City, CA  94065
U.S.A.
FAX: 415–506–7200
## Contents

### Chapter 1

<table>
<thead>
<tr>
<th>Components and Options</th>
<th>1 – 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Oracle Forms Components</td>
<td>1 – 2</td>
</tr>
<tr>
<td>Starting Oracle Forms Components</td>
<td>1 – 3</td>
</tr>
<tr>
<td>Logging In to the Database</td>
<td>1 – 8</td>
</tr>
<tr>
<td>USERID</td>
<td>1 – 8</td>
</tr>
<tr>
<td>Runform Options</td>
<td>1 – 9</td>
</tr>
<tr>
<td>Array (Runform)</td>
<td>1 – 11</td>
</tr>
<tr>
<td>Block_Menu (Runform)</td>
<td>1 – 11</td>
</tr>
<tr>
<td>Buffer_Records (Runform)</td>
<td>1 – 12</td>
</tr>
<tr>
<td>Debug (Runform)</td>
<td>1 – 12</td>
</tr>
<tr>
<td>Debug_Messages (Runform)</td>
<td>1 – 12</td>
</tr>
<tr>
<td>Help (Runform)</td>
<td>1 – 13</td>
</tr>
<tr>
<td>Interactive (Runform)</td>
<td>1 – 13</td>
</tr>
<tr>
<td>Keyin (Runform)</td>
<td>1 – 13</td>
</tr>
<tr>
<td>Keyout (Runform)</td>
<td>1 – 14</td>
</tr>
<tr>
<td>Logon_Screen (Runform)</td>
<td>1 – 14</td>
</tr>
<tr>
<td>OptimizeSQL (Runform)</td>
<td>1 – 15</td>
</tr>
<tr>
<td>OptimizeTP (Runform)</td>
<td>1 – 16</td>
</tr>
<tr>
<td>Options_Screen (Runform)</td>
<td>1 – 17</td>
</tr>
<tr>
<td>Output_File (Runform)</td>
<td>1 – 17</td>
</tr>
<tr>
<td>PECS (Runform)</td>
<td>1 – 18</td>
</tr>
<tr>
<td>Query_Only (Runform)</td>
<td>1 – 19</td>
</tr>
<tr>
<td>Quiet (Runform)</td>
<td>1 – 19</td>
</tr>
<tr>
<td>Session (Runform)</td>
<td>1 – 19</td>
</tr>
<tr>
<td>Statistics (Runform)</td>
<td>1 – 20</td>
</tr>
<tr>
<td>Term (Runform)</td>
<td>1 – 20</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Window State (Runform)</td>
<td>1–21</td>
</tr>
<tr>
<td>Setting Generate Options</td>
<td>1–21</td>
</tr>
<tr>
<td>Add Triggers (Generate)</td>
<td>1–23</td>
</tr>
<tr>
<td>Batch (Generate)</td>
<td>1–23</td>
</tr>
<tr>
<td>CRT_File (Generate)</td>
<td>1–23</td>
</tr>
<tr>
<td>Debug (Generate)</td>
<td>1–24</td>
</tr>
<tr>
<td>Delete (Generate)</td>
<td>1–24</td>
</tr>
<tr>
<td>Extract (Generate)</td>
<td>1–24</td>
</tr>
<tr>
<td>Generate_on_Upgrade (Generate)</td>
<td>1–25</td>
</tr>
<tr>
<td>Help (Generate)</td>
<td>1–25</td>
</tr>
<tr>
<td>Insert (Generate)</td>
<td>1–25</td>
</tr>
<tr>
<td>Logon (Generate)</td>
<td>1–26</td>
</tr>
<tr>
<td>Module_Access (Generate)</td>
<td>1–26</td>
</tr>
<tr>
<td>Module_Type (Generate)</td>
<td>1–26</td>
</tr>
<tr>
<td>Nofail (Generate)</td>
<td>1–27</td>
</tr>
<tr>
<td>Options_Screen (Generate)</td>
<td>1–27</td>
</tr>
<tr>
<td>Output_File (Generate)</td>
<td>1–27</td>
</tr>
<tr>
<td>Parse (Generate)</td>
<td>1–28</td>
</tr>
<tr>
<td>Script (Generate)</td>
<td>1–28</td>
</tr>
<tr>
<td>Statistics (Generate)</td>
<td>1–29</td>
</tr>
<tr>
<td>Upgrade (Generate)</td>
<td>1–29</td>
</tr>
<tr>
<td>Upgrade_Roles (Generate)</td>
<td>1–30</td>
</tr>
<tr>
<td>Version (Generate)</td>
<td>1–30</td>
</tr>
<tr>
<td>Widen_Fields (Generate)</td>
<td>1–31</td>
</tr>
<tr>
<td>Setting Designer Options</td>
<td>1–31</td>
</tr>
<tr>
<td>Designer Options</td>
<td>1–31</td>
</tr>
<tr>
<td>Runtime Options</td>
<td>1–32</td>
</tr>
<tr>
<td>Keyword Parameters</td>
<td>1–32</td>
</tr>
<tr>
<td>Color Mode</td>
<td>1–33</td>
</tr>
<tr>
<td>Color Palette</td>
<td>1–34</td>
</tr>
<tr>
<td>Generate Before Run</td>
<td>1–34</td>
</tr>
<tr>
<td>Help (Designer)</td>
<td>1–35</td>
</tr>
<tr>
<td>Module_Access (Designer)</td>
<td>1–35</td>
</tr>
<tr>
<td>Module_Type (Designer)</td>
<td>1–36</td>
</tr>
<tr>
<td>Printer</td>
<td>1–36</td>
</tr>
<tr>
<td>Run Modules Asynchronously</td>
<td>1–37</td>
</tr>
<tr>
<td>Save Before Generate</td>
<td>1–37</td>
</tr>
<tr>
<td>Suppress Hints</td>
<td>1–37</td>
</tr>
<tr>
<td>Term (Designer)</td>
<td>1–38</td>
</tr>
<tr>
<td>Use System Editor</td>
<td>1–38</td>
</tr>
<tr>
<td>User Preference File</td>
<td>1–39</td>
</tr>
<tr>
<td>Syntax for Options</td>
<td>1–39</td>
</tr>
</tbody>
</table>
Pre–Commit Trigger ................................. 2 – 49
Pre–Delete Trigger ................................. 2 – 50
Pre–Form Trigger ................................. 2 – 51
Pre–Insert Trigger ................................. 2 – 52
Pre–Logon Trigger ................................. 2 – 53
Pre–Logout Trigger ................................. 2 – 54
Pre–Query Trigger ................................. 2 – 55
Pre–Record Trigger ................................. 2 – 57
Pre–Select Trigger ................................. 2 – 58
Pre–Text–Item Trigger ............................. 2 – 59
Pre–Update Trigger ................................. 2 – 60
User–Named Trigger ................................. 2 – 62
When–Button–Pressed Trigger ................... 2 – 63
When–Checkbox–Changed Trigger ............... 2 – 64
When–Clear–Block Trigger ....................... 2 – 65
When–Create–Record Trigger ................. 2 – 66
When–Custom–Item–Event Trigger ............. 2 – 67
When–Database–Record Trigger ............... 2 – 68
When–Form–Navigate ............................. 2 – 69
When–Image–Activated Trigger ............... 2 – 69
When–Image–Pressed Trigger .................. 2 – 70
When–List–Activated Trigger .................. 2 – 70
When–List–Changed Trigger .................... 2 – 71
When–Mouse–Click Trigger ..................... 2 – 71
When–Mouse–DoubleClick Trigger ............. 2 – 72
When–Mouse–Down Trigger ..................... 2 – 74
When–Mouse–Enter Trigger ..................... 2 – 75
When–Mouse–Leave Trigger ..................... 2 – 76
When–Mouse–Move Trigger ..................... 2 – 77
When–Mouse–Up Trigger ....................... 2 – 78
When–New–Block–Instance Trigger .......... 2 – 79
When–New–Form–Instance Trigger ........... 2 – 80
When–New–Item–Instance Trigger .......... 2 – 81
When–New–Record–Instance Trigger .......... 2 – 82
When–Radio–Changed Trigger ................. 2 – 83
When–Remove–Record Trigger ............... 2 – 84
When–Timer–Expired Trigger .................. 2 – 85
When–Validate–Item Trigger ................. 2 – 87
When–Validate–Record Trigger ............... 2 – 89
When–Window–Activated Trigger ............ 2 – 91
When–Window–Closed Trigger ............... 2 – 92
When–Window–Deactivated Trigger .......... 2 – 92
When–Window–Resized Trigger ............. 2 – 93
Chapter 3: Built-in Subprograms

Overview ........................................................................... 3–1
Syntax .............................................................................. 3–2
Named Parameters ............................................................. 3–3
Code Examples ................................................................. 3–3
Object IDs .......................................................................... 3–4
Form Coordinate Units ...................................................... 3–4
Uppercase Return Values .................................................. 3–5
Restricted Built-In Subprograms ........................................... 3–5
Constants .......................................................................... 3–6
Built-in Subprograms Tables ................................................. 3–7
Individual Built-in Descriptions ............................................. 3–16
ABORT_QUERY ..................................................................... 3–17
ADD_GROUP_COLUMN .......................................................... 3–18
ADD_GROUP_ROW ............................................................... 3–20
ADD_LIST_ELEMENT ............................................................ 3–22
ADD_PARAMETER ............................................................... 3–23
APPLICATION_PARAMETER ................................................. 3–24
BACKGROUND_MENU ........................................................ 3–25
BELL ................................................................................. 3–26
BLOCK_MENU .................................................................... 3–26
BREAK .............................................................................. 3–27
CALL_FORM ....................................................................... 3–28
CALL_INPUT ....................................................................... 3–31
CHECKBOX_CHECKED ........................................................ 3–31
CHECK_RECORD_UNIQUENESS .......................................... 3–33
CLEAR_BLOCK .................................................................... 3–34
CLEAR_EOL ....................................................................... 3–35
CLEAR_FORM ..................................................................... 3–36
CLEAR_ITEM ...................................................................... 3–37
CLEAR_LIST ....................................................................... 3–38
CLEAR_MESSAGE .................................................................. 3–39
CLEAR_RECORD .................................................................. 3–40
CLOSE_FORM ..................................................................... 3–41
COMMIT_FORM .................................................................... 3–41
CONVERT_OTHER_VALUE ................................................ 3–43
COPY ................................................................................. 3–44
COPY_REGION ..................................................................... 3–45
COUNT_QUERY ..................................................................... 3–46
CREATE_GROUP ................................................................. 3–47
CREATE_GROUP_FROM_QUERY ............................................. 3–49
CREATE_PARAMETER_LIST ................................................... 3–51
CREATE_QUERIED_RECORD ............................................... 3–52
CREATE_RECORD 3 – 54
CREATE_TIMER 3 – 55
CUT_REGION 3 – 56
DBMS_ERROR_CODE 3 – 57
DBMS_ERROR_TEXT 3 – 58
DEBUG_MODE 3 – 59
DEFAULT_VALUE 3 – 60
DELETE_GROUP 3 – 61
DELETE_GROUP_ROW 3 – 62
DELETE_LIST_ELEMENT 3 – 64
DELETE_PARAMETER 3 – 65
DELETE_RECORD 3 – 66
DELETE_TIMER 3 – 67
DESTROY_PARAMETER_LIST 3 – 69
DISPLAY_ERROR 3 – 70
DISPLAY_ITEM 3 – 70
DOWN 3 – 71
DO_KEY 3 – 72
DUPLICATE_ITEM 3 – 73
DUPLICATE_RECORD 3 – 74
EDIT_TEXTITEM 3 – 75
ENFORCE_COLUMN_SECURITY 3 – 76
ENTER 3 – 77
ENTER_QUERY 3 – 77
ERASE 3 – 79
ERROR_CODE 3 – 79
ERROR_TEXT 3 – 80
ERROR_TYPE 3 – 81
EXECUTE_QUERY 3 – 82
EXECUTE_TRIGGER 3 – 84
EXIT_FORM 3 – 85
FETCH_RECORDS 3 – 87
FIND_ALERT 3 – 89
FIND_BLOCK 3 – 90
FIND_CANVAS 3 – 91
FIND_COLUMN 3 – 91
FIND_EDITOR 3 – 92
FIND_FORM 3 – 93
FIND_GROUP 3 – 94
FIND_ITEM 3 – 94
FIND_LOV 3 – 95
FIND_MENU_ITEM 3 – 96
FIND_RELATION 3 – 97
FIND_TIMER ........................................... 3 – 98
FIND_VIEW .......................................... 3 – 99
FIND_WINDOW ..................................... 3 – 100
FIRST_RECORD ...................................... 3 – 101
FORM_FAILURE ..................................... 3 – 101
FORM_FATAL ...................................... 3 – 103
FORM_SUCCESS .................................. 3 – 104
FORMS_DDL ....................................... 3 – 105
FORMS_OLE.ACTIVATE_SERVER .................. 3 – 109
FORMS_OLE.CLOSE_SERVER ...................... 3 – 110
FORMS_OLE.EXEC_VERB ......................... 3 – 111
FORMS_OLE.FIND_OLE_VERB .................... 3 – 112
FORMS_OLE.GET_INTERFACE_POINTER .......... 3 – 113
FORMS_OLE.GET_VERB_COUNT ................... 3 – 114
FORMS_OLE.GET_VERB_NAME .................... 3 – 115
FORMS_OLE.INITIALIZE_CONTAINER ............. 3 – 116
FORMS_OLE.SERVER_ACTIVE .................... 3 – 117
GENERATE_SEQUENCE_NUMBER ................ 3 – 118
GET_APPLICATION_PROPERTY ................... 3 – 119
GET_BLOCK_PROPERTY ........................... 3 – 123
GET_CANVAS_PROPERTY .......................... 3 – 128
GET_FORM_PROPERTY ................................ 3 – 129
GET_GROUP_CHAR_CELL .......................... 3 – 133
GET_GROUP_DATE_CELL .......................... 3 – 135
GET_GROUP_NUMBER_CELL ...................... 3 – 136
GET_GROUP_RECORD_NUMBER ................. 3 – 138
GET_GROUP_ROW_COUNT ......................... 3 – 139
GET_GROUP_SELECTION ........................ 3 – 140
GET_GROUP_SELECTION_COUNT .............. 3 – 142
GET_ITEM_PROPERTY ................................ 3 – 143
GET_LIST_ELEMENT_COUNT ..................... 3 – 151
GET_LIST_ELEMENT_LABEL ..................... 3 – 153
GET_LIST_ELEMENT_VALUE ..................... 3 – 154
GET_LOV_PROPERTY ................................ 3 – 155
GET_MENU_ITEM_PROPERTY ...................... 3 – 156
GET_MESSAGE ................................... 3 – 158
GET_PARAMETER_ATTR ......................... 3 – 159
GET_PARAMETER_LIST ......................... 3 – 160
GET_RADIO_BUTTON_PROPERTY ................ 3 – 160
GET_RECORD_PROPERTY ......................... 3 – 163
GET_RELATION_PROPERTY ....................... 3 – 165
GET_VIEW_PROPERTY ............................ 3 – 167
GET_WINDOW_PROPERTY ......................... 3 – 170
Chapter 4

System Variables .......................... 4 – 1

About System Variables .................. 4 – 2
  Date and Time System Default Values 4 – 3
  Local Variables .......................... 4 – 4
  Uppercase Return Values ............... 4 – 4

$$DATE$$ .................................. 4 – 5
$$DATETIME$$ .............................. 4 – 6
$$DBDATE$$ ................................ 4 – 7
$$DBDATETIME$$ ......................... 4 – 8
$$DBTIME$$ ................................ 4 – 9
$$TIME$$ ................................... 4 – 9
SYSTEM.BLOCK_STATUS .................. 4 – 10
<table>
<thead>
<tr>
<th>Property Tables</th>
<th>5 – 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item Properties</td>
<td>5 – 17</td>
</tr>
<tr>
<td>Property Descriptions</td>
<td>5 – 20</td>
</tr>
<tr>
<td>Accelerator</td>
<td>5 – 21</td>
</tr>
<tr>
<td>Access Key</td>
<td>5 – 21</td>
</tr>
<tr>
<td>Alert Style</td>
<td>5 – 22</td>
</tr>
<tr>
<td>Alignment</td>
<td>5 – 23</td>
</tr>
<tr>
<td>Application Instance</td>
<td>5 – 24</td>
</tr>
<tr>
<td>Auto Hint</td>
<td>5 – 24</td>
</tr>
<tr>
<td>Auto–Confirm</td>
<td>5 – 25</td>
</tr>
<tr>
<td>Auto–Display</td>
<td>5 – 25</td>
</tr>
<tr>
<td>Auto–Query</td>
<td>5 – 25</td>
</tr>
<tr>
<td>Auto–Refresh</td>
<td>5 – 26</td>
</tr>
<tr>
<td>Auto–Skip (Item)</td>
<td>5 – 28</td>
</tr>
<tr>
<td>Auto–Skip (LOV)</td>
<td>5 – 28</td>
</tr>
<tr>
<td>Base Table (Block)</td>
<td>5 – 29</td>
</tr>
<tr>
<td>Base Table (Item)</td>
<td>5 – 29</td>
</tr>
<tr>
<td>Bevel</td>
<td>5 – 30</td>
</tr>
<tr>
<td>Block Description</td>
<td>5 – 30</td>
</tr>
<tr>
<td>Bottom Title (Editor)</td>
<td>5 – 31</td>
</tr>
<tr>
<td>Bottom Title (Menu)</td>
<td>5 – 31</td>
</tr>
<tr>
<td>Button 1, Button 2, Button 3</td>
<td>5 – 31</td>
</tr>
<tr>
<td>Calling_Form</td>
<td>5 – 32</td>
</tr>
<tr>
<td>Canvas</td>
<td>5 – 32</td>
</tr>
<tr>
<td>Canvas–view Type</td>
<td>5 – 33</td>
</tr>
<tr>
<td>Case Insensitive Query</td>
<td>5 – 34</td>
</tr>
<tr>
<td>Case Restriction</td>
<td>5 – 35</td>
</tr>
<tr>
<td>Character Cell WD/HT</td>
<td>5 – 36</td>
</tr>
<tr>
<td>Check Box Other Values</td>
<td>5 – 36</td>
</tr>
<tr>
<td>Checked</td>
<td>5 – 37</td>
</tr>
<tr>
<td>Checked Value</td>
<td>5 – 37</td>
</tr>
<tr>
<td>Class</td>
<td>5 – 38</td>
</tr>
<tr>
<td>Closeable</td>
<td>5 – 38</td>
</tr>
<tr>
<td>Column Mapping</td>
<td>5 – 39</td>
</tr>
<tr>
<td>Column Name</td>
<td>5 – 39</td>
</tr>
<tr>
<td>Column Title</td>
<td>5 – 39</td>
</tr>
<tr>
<td>Display Width</td>
<td>5 – 39</td>
</tr>
<tr>
<td>Return Item</td>
<td>5 – 39</td>
</tr>
<tr>
<td>Column Name (LOV)</td>
<td>5 – 40</td>
</tr>
<tr>
<td>Column Name (Record Group)</td>
<td>5 – 40</td>
</tr>
<tr>
<td>Column Security</td>
<td>5 – 40</td>
</tr>
<tr>
<td>Column Specification</td>
<td>5 – 41</td>
</tr>
<tr>
<td>Column Name</td>
<td>5 – 41</td>
</tr>
</tbody>
</table>
Direction (Radio Group) .............................. 5 – 71
Direction (Windows) ............................... 5 – 71
Directory ............................................ 5 – 72
Display Width (LOV) ............................ 5 – 72
Display w/o Privilege ............................ 5 – 73
Display X Position, Display Y Position ........ 5 – 73
Displayed (Item) .................................. 5 – 74
Displayed (Canvas–view) ......................... 5 – 74
Displayed (Menu Item) ......................... 5 – 75
Display_Height ..................................... 5 – 75
Display_Width ....................................... 5 – 76
Editor ................................................ 5 – 76
Editor X Position, Editor Y Position .......... 5 – 77
Enabled (Item) ..................................... 5 – 77
Enabled (Menu Item) ............................. 5 – 78
Enterable ............................................ 5 – 79
Execution Style .................................... 5 – 79
File ................................................ 5 – 80
File_Name .......................................... 5 – 81
Fire in Enter Query Mode ....................... 5 – 82
First Navigation Block ........................... 5 – 83
First_Block ......................................... 5 – 83
First_Detail_Relation ......................... 5 – 84
First_Item ......................................... 5 – 84
First_Master_Relation ......................... 5 – 85
Fixed Length(Item) .............................. 5 – 86
Fixed Length (Menu Substitution Parameter) ... 5 – 86
Fixed Size .......................................... 5 – 87
Format Mask ....................................... 5 – 88
Form_Name ......................................... 5 – 94
Group_Name ....................................... 5 – 94
Help ................................................ 5 – 95
Hint (Item) ......................................... 5 – 95
Hint (Menu Item) ................................ 5 – 96
Hint (Menu Substitution Parameter) .......... 5 – 96
Horizontal MDI Toolbar ....................... 5 – 97
Horizontal Scroll Bar ........................... 5 – 97
Horizontal Toolbar ............................. 5 – 98
Icon Name ......................................... 5 – 99
Icon Title .......................................... 5 – 100
Iconic ............................................ 5 – 100
Iconifiable ........................................ 5 – 101
Identification .................................... 5 – 101
In Menu/Block Description .......................................................... 5 – 102
Inherit Menu ................................................................. 5 – 102
Initial Keyboard State .............................................................. 5 – 103
Insert Allowed (Block) ........................................................... 5 – 103
Insert Allowed (Item) .............................................................. 5 – 104
Item Type ............................................................... 5 – 105
Items Displayed ................................................................. 5 – 106
Item_Is_Valid ................................................................. 5 – 106
Join Condition ................................................................. 5 – 107
Keep Position ................................................................. 5 – 108
Key Mode ................................................................. 5 – 109
Label (Item) ................................................................. 5 – 110
Label (Menu Item) ............................................................ 5 – 110
Label (Menu Substitution Parameter) ..................................... 5 – 111
Last_Block ................................................................. 5 – 111
Last_Item ................................................................. 5 – 112
Length (Record Group) .......................................................... 5 – 112
List Elements ................................................................. 5 – 112
  List Item ................................................................. 5 – 112
  List Item Value ........................................................... 5 – 112
List Style ................................................................. 5 – 113
Lock Record ................................................................. 5 – 113
Locking Mode ................................................................. 5 – 114
Long List ................................................................. 5 – 115
LOV ................................................................. 5 – 116
LOV for Validation ............................................................. 5 – 116
LOV Position ................................................................. 5 – 118
LOV Type ................................................................. 5 – 119
Magic Item ................................................................. 5 – 120
Main Menu ................................................................. 5 – 121
Master Deletes ............................................................... 5 – 122
Maximum Length .............................................................. 5 – 123
Maximum Length (Form Parameter) .......................................... 5 – 124
Maximum Length (Menu Substitution Parameter) ...................... 5 – 124
Menu Item Radio Group ......................................................... 5 – 125
Menu Item Roles .............................................................. 5 – 125
Menu Item Type .............................................................. 5 – 126
Menu Module ............................................................... 5 – 128
Menu Module Roles .......................................................... 5 – 129
Menu Role ............................................................... 5 – 129
Menu Style ............................................................... 5 – 130
Message ................................................................. 5 – 130
Mirror Item ................................................................. 5 – 131
Modal .......................................................... 5 – 132
Module_NLS_Lang .............................................. 5 – 133
Mouse Navigate ............................................... 5 – 134
Mouse Navigation Limit ..................................... 5 – 135
Moveable ....................................................... 5 – 135
Multi-Line ...................................................... 5 – 136
Name .......................................................... 5 – 136
Navigable ...................................................... 5 – 138
Navigation Style ............................................. 5 – 139
Next Navigation Block ....................................... 5 – 140
Next Navigation Item ......................................... 5 – 140
NextBlock ..................................................... 5 – 141
NextItem ....................................................... 5 – 142
Next_Detail_Relation ........................................ 5 – 142
Next_Master_Relation ........................................ 5 – 142
OLE Activation Style ........................................ 5 – 143
OLE Class ...................................................... 5 – 144
OLE Do In Out ............................................... 5 – 145
OLE In–place Activation ..................................... 5 – 146
OLE Popup Menu Items ...................................... 5 – 147
OLE Resize Style ............................................. 5 – 150
OLE Tenant Aspect .......................................... 5 – 150
OLE Tenant Types ........................................... 5 – 151
Operating_System .......................................... 5 – 152
Optimizer_Hint ............................................... 5 – 152
Order By ....................................................... 5 – 153
Other Values .................................................. 5 – 154
Parameter Menus ............................................. 5 – 154
Password ....................................................... 5 – 155
Prevent Masterless Operations ............................ 5 – 155
Previous Navigation Block ................................ 5 – 156
Previous Navigation Item ................................... 5 – 157
PreviousBlock ............................................... 5 – 158
PreviousItem .................................................. 5 – 158
Primary Key (Block) ......................................... 5 – 159
Primary Key (Item) ........................................... 5 – 159
Quality ......................................................... 5 – 160
Query Allowed (Block) ...................................... 5 – 160
Query Allowed (Item) ....................................... 5 – 161
Query Length ................................................ 5 – 162
Query Only ................................................... 5 – 162
Query_Hits .................................................... 5 – 163
Query_Options ............................................... 5 – 164
Raise on Entry .......................................................... 5 – 164
Range High Value/Range Low Value ............................ 5 – 165
Reading Order ......................................................... 5 – 166
Real Unit .................................................................. 5 – 167
Record Group ............................................................ 5 – 167
Record Group Query .................................................. 5 – 168
Record Group Type .................................................... 5 – 168
Record Orientation .................................................... 5 – 169
Records Buffered ...................................................... 5 – 169
Records Displayed ..................................................... 5 – 170
Records Fetched ....................................................... 5 – 171
Records_to_Fetch ...................................................... 5 – 171
Reference Information .............................................. 5 – 173
Remove on Exit ...................................................... 5 – 173
Rendered .................................................................. 5 – 174
Required (Item) ........................................................ 5 – 174
Required (Menu Parameter) ...................................... 5 – 175
Return Item (LOV) ..................................................... 5 – 175
Savepoint Mode ....................................................... 5 – 176
Savepoint_Name ....................................................... 5 – 177
Scroll Bar .................................................................. 5 – 178
Secure (Item) ............................................................ 5 – 179
Secure (Menu Parameter) ......................................... 5 – 179
Show Keys/Show Keys Description .............................. 5 – 180
Show OLE Popup Menu .............................................. 5 – 181
Show OLE Tenant Type ............................................. 5 – 182
Size ....................................................................... 5 – 182
  Size (Canvas–view) .................................................. 5 – 182
  Size (Editor) ........................................................... 5 – 182
  Size (Item) ............................................................... 5 – 183
  Size (LOV) .............................................................. 5 – 183
  Size (Window) ........................................................ 5 – 184
Sizing Style ............................................................. 5 – 184
Space Between Records ............................................ 5 – 184
Starting Menu .......................................................... 5 – 185
Startup Code ............................................................ 5 – 186
Status (Block) .......................................................... 5 – 186
Status (Record) ........................................................ 5 – 187
Subtitle ................................................................ 5 – 187
Tear–off ................................................................ 5 – 188
Timer_Name ............................................................. 5 – 188
Title ...................................................................... 5 – 188
Title (LOV) ............................................................... 5 – 188
| Title (Menu) | 5 – 189 |
| Title (Window) | 5 – 189 |
| Top Title | 5 – 189 |
| Top_Record | 5 – 190 |
| Transactional Triggers | 5 – 190 |
| Trigger Style | 5 – 191 |
| Trigger Text | 5 – 191 |
| Trigger Type | 5 – 192 |
| Unchecked Value | 5 – 192 |
| Update Allowed (Block) | 5 – 193 |
| Update Allowed (Item) | 5 – 194 |
| Update Changed Columns | 5 – 195 |
| Update_Column | 5 – 196 |
| Update Only if NULL | 5 – 197 |
| Update_Permission | 5 – 198 |
| Use File | 5 – 199 |
| Use Security | 5 – 201 |
| Use 3D Controls | 5 – 201 |
| Username | 5 – 202 |
| User_Interface | 5 – 202 |
| User_NLS_Lang | 5 – 203 |
| Validation | 5 – 203 |
| Validation Unit | 5 – 204 |
| Value | 5 – 205 |
| VBX Control File | 5 – 205 |
| VBX Control Name | 5 – 206 |
| VBX Control Value Property | 5 – 206 |
| Vertical MDI Toolbar | 5 – 207 |
| Vertical Scroll Bar | 5 – 208 |
| Vertical Toolbar | 5 – 208 |
| View | 5 – 209 |
| View Height, View Width | 5 – 210 |
| View Horizontal Scroll Bar | 5 – 210 |
| View Vertical Scroll Bar | 5 – 211 |
| Visible | 5 – 211 |
| Visual Attribute Name | 5 – 213 |
| Visual_Attribute | 5 – 216 |
| WHERE Clause/ORDER BY Clause | 5 – 217 |
| Width/Height (WD, HT) | 5 – 218 |
| Window | 5 – 218 |
| Window_Handle | 5 – 219 |
| Window_State | 5 – 220 |
| Window_Style | 5 – 221 |
EXECUTE_TRIGGER .............................. 8 – 39
EXIT/Form .................................... 8 – 40
Fetch Records ................................. 8 – 41
FIRST_RECORD .................................. 8 – 44
Generate Sequence Number .................. 8 – 45
GO_BLOCK ...................................... 8 – 46
GO_ITEM ........................................ 8 – 47
GO_RECORD ..................................... 8 – 48
HOST ............................................. 8 – 49
LAST_RECORD ................................... 8 – 50
Leave the Block ............................... 8 – 51
Leave the Form ............................... 8 – 52
Leave the Item ................................. 8 – 53
Leave the Record ............................. 8 – 54
Leave Unit Error Processing ............... 8 – 55
LOCK_RECORD .................................. 8 – 57
Lock the Row ................................. 8 – 58
LOGON .......................................... 8 – 60
LOGOUT ......................................... 8 – 62
Mark Items and Records as Changed .... 8 – 63
Master–Detail Coordination ............... 8 – 64
Navigate to the <Navigation Unit> Level 8 – 66
NEW_FORM ..................................... 8 – 67
NEXT_BLOCK .................................. 8 – 68
NEXT_ITEM ..................................... 8 – 69
NEXT_KEY ...................................... 8 – 71
NEXT_RECORD ................................ 8 – 73
NEXT_SET ...................................... 8 – 74
Open the Query .............................. 8 – 75
POST ............................................. 8 – 76
Post and Commit Transactions ............ 8 – 77
Prepare the Query ......................... 8 – 86
PREVIOUS_BLOCK ............................ 8 – 87
PREVIOUS_ITEM ............................. 8 – 88
PREVIOUS_RECORD .......................... 8 – 89
Process Expired Timer ....................... 8 – 90
Process the Function Key ................. 8 – 91
Prompt and Answer ......................... 8 – 92
Put Cursor At ............................... 8 – 93
REPLACE_MENU .............................. 8 – 96
Return for Input ............................ 8 – 97
Rollback Form .............................. 8 – 98
Run the Form ............................... 8 – 99
Savepoint ......................................................... 8 – 100
SCROLL_DOWN .................................................. 8 – 101
SCROLL_UP ......................................................... 8 – 103
SHOW_LOV ......................................................... 8 – 105
UP ................................................................. 8 – 107
Validate the Block ........................................... 8 – 108
Validate the Form ............................................. 8 – 109
Validate the Item ............................................... 8 – 110
Validate the Record .......................................... 8 – 112

Appendix A

Compatibility with Prior Versions ......................... A – 1
About Upgrading Forms and Menus to Version 4.5 .... A – 2
Upgrading a Form ............................................ A – 2
  Upgrading from Version 4.0 to Version 4.5 .......... A – 3
  Upgrading from Version 3.0 to Version 4.5 .......... A – 4
  Upgrading from Version 2.3 to Version 4.5 .......... A – 5
  Running an Application After Upgrading .......... A – 5
  GUI Platforms ............................................. A – 5
Upgrading Forms Containing Referenced Objects .... A – 6
Resolving Naming Conflicts in Form–Level Procedures A – 8
Upgrading File Formats .................................... A – 9
  Version 3.0 .INP Files .................................. A – 10
Converting File Formats .................................. A – 10
Upgrading a Menu .......................................... A – 11
  Upgrading from SQL*Menu 5.0 ....................... A – 11
  Creating an .MMB File Only ......................... A – 12
Upgrading Menu Security Roles ....................... A – 13
Output When Upgrading from Version 3.0 to Version 4.5 A – 14
  Triggers .................................................... A – 14
  Fields ..................................................... A – 15
  Pages ..................................................... A – 15
  LOVs ....................................................... A – 15
  Master–Detail Block Relationships ................. A – 16
  Properties ............................................... A – 16
  Packaged Procedures and Functions .............. A – 17
  Character–mode Applications ...................... A – 17
Enhanced Version 3.0 Functionality ..................... A – 18
  Integrated Forms and Menus ....................... A – 18
  New Block Window ................................. A – 19
  Master–Detail Block Relationship .................. A – 19
  List of Values (LOV) .................. A – 20
  Pages ................................................. A – 20
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/SQL Libraries</td>
<td>A – 21</td>
</tr>
<tr>
<td>User Exits</td>
<td>A – 22</td>
</tr>
<tr>
<td>Calling Oracle Forms from a C Program</td>
<td>A – 22</td>
</tr>
<tr>
<td>New Features in Version 4.5</td>
<td>A – 23</td>
</tr>
<tr>
<td>Terminology Changes</td>
<td>A – 24</td>
</tr>
<tr>
<td>Terminology Related to Properties</td>
<td>A – 25</td>
</tr>
<tr>
<td>Terminology Related to System Variables</td>
<td>A – 25</td>
</tr>
<tr>
<td>Command Line Options</td>
<td>A – 26</td>
</tr>
<tr>
<td>Triggers, Built–Ins, and Properties</td>
<td>A – 27</td>
</tr>
<tr>
<td>Triggers</td>
<td>A – 27</td>
</tr>
<tr>
<td>Built–In Subprograms</td>
<td>A – 29</td>
</tr>
<tr>
<td>Properties</td>
<td>A – 33</td>
</tr>
<tr>
<td>3.0 Packaged Procedures and Functions</td>
<td>A – 36</td>
</tr>
<tr>
<td>SQL*Menu Version 5.0 Packaged Procedures and Functions</td>
<td>A – 38</td>
</tr>
<tr>
<td>Moving from Character Mode to GUI</td>
<td>A – 39</td>
</tr>
<tr>
<td>User Expectations</td>
<td>A – 41</td>
</tr>
<tr>
<td>Migration Strategies</td>
<td>A – 41</td>
</tr>
<tr>
<td>Conversion Sequence</td>
<td>A – 43</td>
</tr>
<tr>
<td>Adding GUI Functionality</td>
<td>A – 44</td>
</tr>
<tr>
<td>Converting Key Triggers</td>
<td>A – 45</td>
</tr>
<tr>
<td>Widening Fields</td>
<td>A – 46</td>
</tr>
<tr>
<td>Creating a Checkbox from a Version 3.0 Checkbox Field</td>
<td>A – 47</td>
</tr>
<tr>
<td>Converting Pop–Up Pages</td>
<td>A – 48</td>
</tr>
</tbody>
</table>

**Appendix B**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module Storage</td>
<td>B – 1</td>
</tr>
<tr>
<td>Managing Modules</td>
<td>B – 2</td>
</tr>
<tr>
<td>Form Storage Formats</td>
<td>B – 3</td>
</tr>
<tr>
<td>Database Format</td>
<td>B – 3</td>
</tr>
<tr>
<td>.FMB Format</td>
<td>B – 4</td>
</tr>
<tr>
<td>.FMX Format</td>
<td>B – 4</td>
</tr>
<tr>
<td>.FMT Format</td>
<td>B – 5</td>
</tr>
<tr>
<td>Menu Storage Formats</td>
<td>B – 5</td>
</tr>
<tr>
<td>Database Format</td>
<td>B – 6</td>
</tr>
<tr>
<td>.MMB Format</td>
<td>B – 6</td>
</tr>
<tr>
<td>.MMX Format</td>
<td>B – 7</td>
</tr>
<tr>
<td>.MMT Format</td>
<td>B – 7</td>
</tr>
<tr>
<td>Library Storage Formats</td>
<td>B – 8</td>
</tr>
<tr>
<td>Database Format</td>
<td>B – 8</td>
</tr>
<tr>
<td>.PLL Format</td>
<td>B – 8</td>
</tr>
<tr>
<td>.PLL Format Stripped of Source Code</td>
<td>B – 9</td>
</tr>
<tr>
<td>.PLD Format</td>
<td>B – 9</td>
</tr>
<tr>
<td>Portability Among Systems</td>
<td>B – 10</td>
</tr>
</tbody>
</table>
### Appendix C
Reserved Words ........................................... C – 1

### Appendix D
Constraints ............................................... D – 1
Object Constraints ...................................... D – 2
Cursor Constraints ...................................... D – 3
This chapter describes Oracle Forms object properties, and includes information on the following topics:

- Setting and modifying properties  5 – 3
- Reading the property descriptions  5 – 4
- Property tables  5 – 5
- Property descriptions  5 – 20
Overview

Each object in an Oracle Forms application possesses characteristics known as properties. The properties of an object determine its appearance and functionality.

This chapter includes descriptions of all the object properties, arranged alphabetically. The subheadings within each description highlight information specific to that property, such as any default value, restrictions, and usage notes. In some cases, where a subheading is not meaningful for a property, the subheading is omitted.

Several tables are included immediately following this overview. These tables group properties by object type, and include the page number where you can find the description of the property.
Setting Properties Programmatically

Each property description includes a Set: heading that describes how the property can be set; either declaratively in the Designer, programmatically at runtime, or both.

For information on defining object properties in the Designer, refer to the *Oracle Forms Developer’s Guide*.

You can modify object properties programmatically by using built-ins. The following built-ins can be used to dynamically modify the properties of existing objects during a runtime session:

- `SET_APPLICATIONPROPERTY`
- `SET_BLOCKPROPERTY`
- `SET_CANVASPROPERTY`
- `SET_FORMPROPERTY`
- `SET_ITEMPROPERTY`
- `SET_LOVPROPERTY`
- `SET_MENUITEMPROPERTY`
- `SET_PARAMETERATTR`
- `SET_RADIOBUTTONPROPERTY`
- `SET_RECORDPROPERTY`
- `SET_RELATIONPROPERTY`
- `SET_VIEWPROPERTY`
- `SET_WINDOWPROPERTY`

You can programmatically determine the settings of most properties by using the set of corresponding built-ins to get properties.
**Reading the Property Descriptions**

The property descriptions follow a general pattern. The property name is printed in a bold typeface and is followed by a brief description. Some properties have multiple descriptions, depending on the object to which they are applied. In these cases, the property may appear more than once, with the object listed as part of the property heading, as shown here:

**Name (Record Group)**

The headings in the following table are included for those properties to which they apply. Only headings and information that apply are included.

<table>
<thead>
<tr>
<th><strong>Heading</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Applies to:</td>
<td>Names the object class or classes for which this property is meaningful.</td>
</tr>
<tr>
<td>Set:</td>
<td>Indicates where you can set the property: in the Designer, programmatically, or both places.</td>
</tr>
<tr>
<td>Refer to Built-in:</td>
<td>Indicates the built-in that you can use to set the property, if it can be set programmatically.</td>
</tr>
<tr>
<td>Default:</td>
<td>Indicates a default value for the property.</td>
</tr>
<tr>
<td>Required/Optional:</td>
<td>Indicates whether the property is required or optional.</td>
</tr>
<tr>
<td>Restrictions:</td>
<td>Lists any restrictions for usage of the property.</td>
</tr>
<tr>
<td>Usage Notes:</td>
<td>Describes any particular usage considerations that you should keep in mind when using the property.</td>
</tr>
<tr>
<td>See Also:</td>
<td>Refers you to a similar or dependent property.</td>
</tr>
</tbody>
</table>
The following lists provide all of the available properties and group them by object class. The object classes appear in order of functionality. The lists also include the page number where you can find the property description. A few properties apply to more than one object class. For example, the Name property applies to several object classes, and is sometimes different. Be sure to refer to the property for the correct object.

<table>
<thead>
<tr>
<th>Alert Properties</th>
<th>Property</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Alert Style</td>
<td>5 – 22</td>
</tr>
<tr>
<td></td>
<td>Button 1, Button 2, Button 3</td>
<td>5 – 31</td>
</tr>
<tr>
<td></td>
<td>Class</td>
<td>5 – 38</td>
</tr>
<tr>
<td></td>
<td>Comment</td>
<td>5 – 44</td>
</tr>
<tr>
<td></td>
<td>Default Alert Button</td>
<td>5 – 61</td>
</tr>
<tr>
<td></td>
<td>Default Button</td>
<td>5 – 62</td>
</tr>
<tr>
<td></td>
<td>Message</td>
<td>5 – 130</td>
</tr>
<tr>
<td></td>
<td>Name</td>
<td>5 – 136</td>
</tr>
<tr>
<td></td>
<td>Title</td>
<td>5 – 188</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application Properties</th>
<th>Property</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Calling Form</td>
<td>5 – 32</td>
</tr>
<tr>
<td></td>
<td>Class</td>
<td>5 – 38</td>
</tr>
<tr>
<td></td>
<td>Comment</td>
<td>5 – 44</td>
</tr>
<tr>
<td></td>
<td>Connect_String</td>
<td>5 – 44</td>
</tr>
<tr>
<td></td>
<td>Current Form</td>
<td>5 – 50</td>
</tr>
<tr>
<td></td>
<td>Current Form Name</td>
<td>5 – 51</td>
</tr>
<tr>
<td></td>
<td>Cursor_Style</td>
<td>5 – 54</td>
</tr>
<tr>
<td></td>
<td>Datasource</td>
<td>5 – 61</td>
</tr>
<tr>
<td></td>
<td>Display_Height</td>
<td>5 – 75</td>
</tr>
<tr>
<td></td>
<td>Display_Width</td>
<td>5 – 76</td>
</tr>
<tr>
<td></td>
<td>Name</td>
<td>5 – 136</td>
</tr>
<tr>
<td></td>
<td>Operating System</td>
<td>5 – 152</td>
</tr>
<tr>
<td></td>
<td>Password</td>
<td>5 – 155</td>
</tr>
<tr>
<td></td>
<td>Savepoint_Name</td>
<td>5 – 177</td>
</tr>
<tr>
<td></td>
<td>Timer Name</td>
<td>5 – 188</td>
</tr>
<tr>
<td></td>
<td>Username</td>
<td>5 – 202</td>
</tr>
</tbody>
</table>
### User Interface Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>User_Interface</td>
<td>5 – 202</td>
</tr>
<tr>
<td>User_NLS_Lang</td>
<td>5 – 203</td>
</tr>
</tbody>
</table>

### Block Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Instance</td>
<td>5 – 24</td>
</tr>
<tr>
<td>Base Table</td>
<td>5 – 29</td>
</tr>
<tr>
<td>Class</td>
<td>5 – 38</td>
</tr>
<tr>
<td>Column Security</td>
<td>5 – 40</td>
</tr>
<tr>
<td>Comment</td>
<td>5 – 44</td>
</tr>
<tr>
<td>Current Record</td>
<td>5 – 52</td>
</tr>
<tr>
<td>Current Record Attribute</td>
<td>5 – 49</td>
</tr>
<tr>
<td>Delete Allowed</td>
<td>5 – 66</td>
</tr>
<tr>
<td>Enterable</td>
<td>5 – 79</td>
</tr>
<tr>
<td>First_Block</td>
<td>5 – 83</td>
</tr>
<tr>
<td>First_Detial_Relation</td>
<td>5 – 84</td>
</tr>
<tr>
<td>First_Item</td>
<td>5 – 84</td>
</tr>
<tr>
<td>First_Master_Relation</td>
<td>5 – 85</td>
</tr>
<tr>
<td>In Menu/Block Description</td>
<td>5 – 102</td>
</tr>
<tr>
<td>Insert Allowed (Block)</td>
<td>5 – 103</td>
</tr>
<tr>
<td>Key Mode</td>
<td>5 – 109</td>
</tr>
<tr>
<td>Last_Item</td>
<td>5 – 112</td>
</tr>
<tr>
<td>Locking Mode</td>
<td>5 – 114</td>
</tr>
<tr>
<td>Name</td>
<td>5 – 136</td>
</tr>
<tr>
<td>Navigation Style</td>
<td>5 – 139</td>
</tr>
<tr>
<td>Next_Block</td>
<td>5 – 141</td>
</tr>
<tr>
<td>Next_Navigation Block</td>
<td>5 – 140</td>
</tr>
<tr>
<td>Optimizer Hint</td>
<td>5 – 152</td>
</tr>
<tr>
<td>Previous Navigation Block</td>
<td>5 – 156</td>
</tr>
<tr>
<td>PreviousBlock</td>
<td>5 – 158</td>
</tr>
<tr>
<td>Primary Key (Block)</td>
<td>5 – 159</td>
</tr>
<tr>
<td>Query Allowed (Block)</td>
<td>5 – 160</td>
</tr>
<tr>
<td>Query Hits</td>
<td>5 – 163</td>
</tr>
<tr>
<td>Query Options</td>
<td>5 – 164</td>
</tr>
<tr>
<td>Record Orientation</td>
<td>5 – 169</td>
</tr>
<tr>
<td>Property</td>
<td>Page Number</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Records Buffered</td>
<td>5 – 169</td>
</tr>
<tr>
<td>Records Displayed</td>
<td>5 – 170</td>
</tr>
<tr>
<td>Records Fetched</td>
<td>5 – 171</td>
</tr>
<tr>
<td>Records_to_Fetch</td>
<td>5 – 171</td>
</tr>
<tr>
<td>Scroll Bar</td>
<td>5 – 178</td>
</tr>
<tr>
<td>Status (Block)</td>
<td>5 – 186</td>
</tr>
<tr>
<td>Top Record</td>
<td>5 – 190</td>
</tr>
<tr>
<td>Transactional Triggers</td>
<td>5 – 190</td>
</tr>
<tr>
<td>Update Allowed (Block)</td>
<td>5 – 193</td>
</tr>
<tr>
<td>Update Changed Columns</td>
<td>5 – 195</td>
</tr>
<tr>
<td>Visual Attribute Name</td>
<td>5 – 213</td>
</tr>
<tr>
<td>WHERE Clause/ORDER BY Clause</td>
<td>5 – 217</td>
</tr>
<tr>
<td>Window_Handle</td>
<td>5 – 219</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Canvas–view Properties</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Property</strong></td>
<td><strong>Page Number</strong></td>
</tr>
<tr>
<td>Bevel</td>
<td>5 – 30</td>
</tr>
<tr>
<td>Canvas</td>
<td>5 – 32</td>
</tr>
<tr>
<td>Canvas–view Type</td>
<td>5 – 33</td>
</tr>
<tr>
<td>Class</td>
<td>5 – 38</td>
</tr>
<tr>
<td>Comment</td>
<td>5 – 44</td>
</tr>
<tr>
<td>Display X Position</td>
<td>5 – 73</td>
</tr>
<tr>
<td>Display Y Position</td>
<td>5 – 73</td>
</tr>
<tr>
<td>Displayed (Canvas–view)</td>
<td>5 – 74</td>
</tr>
<tr>
<td>Name</td>
<td>5 – 136</td>
</tr>
<tr>
<td>Raise on Entry</td>
<td>5 – 164</td>
</tr>
<tr>
<td>Size</td>
<td>5 – 182</td>
</tr>
<tr>
<td>View Height</td>
<td>5 – 210</td>
</tr>
<tr>
<td>View Horizontal Scroll Bar</td>
<td>5 – 210</td>
</tr>
<tr>
<td>View Width</td>
<td>5 – 210</td>
</tr>
<tr>
<td>View Vertical Scroll Bar</td>
<td>5 – 211</td>
</tr>
<tr>
<td>Visible</td>
<td>5 – 211</td>
</tr>
<tr>
<td>Visual Attribute Name</td>
<td>5 – 213</td>
</tr>
<tr>
<td>Visual_Attribute</td>
<td>5 – 216</td>
</tr>
<tr>
<td>Window</td>
<td>5 – 218</td>
</tr>
<tr>
<td>Property</td>
<td>Page Number</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>X Position</td>
<td>5 – 222</td>
</tr>
<tr>
<td>Y Position</td>
<td>5 – 222</td>
</tr>
<tr>
<td>X Position on Canvas</td>
<td>5 – 224</td>
</tr>
<tr>
<td>Y Position on Canvas</td>
<td>5 – 224</td>
</tr>
</tbody>
</table>

**Editor Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottom Title (Editor)</td>
<td>5 – 31</td>
</tr>
<tr>
<td>Class</td>
<td>5 – 38</td>
</tr>
<tr>
<td>Comment</td>
<td>5 – 44</td>
</tr>
<tr>
<td>Editor Position</td>
<td>5 – 77</td>
</tr>
<tr>
<td>Horizontal Scroll Bar</td>
<td>5 – 97</td>
</tr>
<tr>
<td>Name</td>
<td>5 – 136</td>
</tr>
<tr>
<td>Size</td>
<td>5 – 182</td>
</tr>
<tr>
<td>Top Title</td>
<td>5 – 189</td>
</tr>
<tr>
<td>Vertical Scroll Bar</td>
<td>5 – 208</td>
</tr>
<tr>
<td>Wrap Style</td>
<td>5 – 222</td>
</tr>
<tr>
<td>X Position</td>
<td>5 – 222</td>
</tr>
<tr>
<td>Y Position</td>
<td>5 – 222</td>
</tr>
</tbody>
</table>

**Form Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Instance</td>
<td>5 – 24</td>
</tr>
<tr>
<td>Character Cell WD/HT</td>
<td>5 – 36</td>
</tr>
<tr>
<td>Class</td>
<td>5 – 38</td>
</tr>
<tr>
<td>Comment</td>
<td>5 – 44</td>
</tr>
<tr>
<td>Console Window</td>
<td>5 – 45</td>
</tr>
<tr>
<td>Coordinate System</td>
<td>5 – 45</td>
</tr>
<tr>
<td>Current Record Attribute</td>
<td>5 – 49</td>
</tr>
<tr>
<td>Cursor Mode</td>
<td>5 – 52</td>
</tr>
<tr>
<td>Default Font Scaling</td>
<td>5 – 62</td>
</tr>
<tr>
<td>Defer_Required_Enforcement</td>
<td>5 – 65</td>
</tr>
<tr>
<td>File Name</td>
<td>5 – 81</td>
</tr>
<tr>
<td>First Navigation Block</td>
<td>5 – 83</td>
</tr>
<tr>
<td>First_Block</td>
<td>5 – 83</td>
</tr>
<tr>
<td>Form Name</td>
<td>5 – 94</td>
</tr>
<tr>
<td>Horizontal MDI Toolbar</td>
<td>5 – 97</td>
</tr>
<tr>
<td>Property</td>
<td>Page Number</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Last Block</td>
<td>5 – 111</td>
</tr>
<tr>
<td>Menu Module</td>
<td>5 – 128</td>
</tr>
<tr>
<td>Menu Role</td>
<td>5 – 129</td>
</tr>
<tr>
<td>Menu Style</td>
<td>5 – 130</td>
</tr>
<tr>
<td>Module NLS Lang</td>
<td>5 – 133</td>
</tr>
<tr>
<td>Mouse Navigation Limit</td>
<td>5 – 135</td>
</tr>
<tr>
<td>Name</td>
<td>5 – 136</td>
</tr>
<tr>
<td>Real Unit</td>
<td>5 – 167</td>
</tr>
<tr>
<td>Savepoint Mode</td>
<td>5 – 176</td>
</tr>
<tr>
<td>Starting Menu</td>
<td>5 – 185</td>
</tr>
<tr>
<td>Title</td>
<td>5 – 188</td>
</tr>
<tr>
<td>Use File</td>
<td>5 – 199</td>
</tr>
<tr>
<td>Use 3D Controls</td>
<td>5 – 201</td>
</tr>
<tr>
<td>Validation</td>
<td>5 – 203</td>
</tr>
<tr>
<td>Validation Unit</td>
<td>5 – 204</td>
</tr>
<tr>
<td>Vertical MDI Toolbar</td>
<td>5 – 207</td>
</tr>
<tr>
<td>Visual Attribute Name</td>
<td>5 – 213</td>
</tr>
<tr>
<td>Window_Handle</td>
<td>5 – 219</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Form Parameter Properties</th>
<th>Property</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Class</td>
<td>5 – 38</td>
</tr>
<tr>
<td></td>
<td>Comment</td>
<td>5 – 44</td>
</tr>
<tr>
<td></td>
<td>Data Type</td>
<td>5 – 55</td>
</tr>
<tr>
<td></td>
<td>Default Value (Form Parameter)</td>
<td>5 – 63</td>
</tr>
<tr>
<td></td>
<td>Maximum Length (Form Parameter)</td>
<td>5 – 124</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item Properties</th>
<th>Property</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Access Key</td>
<td>5 – 21</td>
</tr>
<tr>
<td></td>
<td>Alignment</td>
<td>5 – 23</td>
</tr>
<tr>
<td></td>
<td>Application Instance</td>
<td>5 – 24</td>
</tr>
<tr>
<td></td>
<td>Auto Hint</td>
<td>5 – 24</td>
</tr>
<tr>
<td></td>
<td>Auto–Skip (Item)</td>
<td>5 – 28</td>
</tr>
<tr>
<td></td>
<td>Base Table (Item)</td>
<td>5 – 29</td>
</tr>
<tr>
<td></td>
<td>Bevel</td>
<td>5 – 30</td>
</tr>
<tr>
<td></td>
<td>Canvas</td>
<td>5 – 32</td>
</tr>
<tr>
<td>Property</td>
<td>Page Number</td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>Case Insensitive Query</td>
<td>5 – 34</td>
<td></td>
</tr>
<tr>
<td>Case Restriction</td>
<td>5 – 35</td>
<td></td>
</tr>
<tr>
<td>Check Box Other Values</td>
<td>5 – 36</td>
<td></td>
</tr>
<tr>
<td>Checked Value</td>
<td>5 – 37</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>5 – 38</td>
<td></td>
</tr>
<tr>
<td>Comment</td>
<td>5 – 44</td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td>5 – 44</td>
<td></td>
</tr>
<tr>
<td>Copy Value from Item</td>
<td>5 – 49</td>
<td></td>
</tr>
<tr>
<td>Current Record Attribute</td>
<td>5 – 49</td>
<td></td>
</tr>
<tr>
<td>Data Type</td>
<td>5 – 55</td>
<td></td>
</tr>
<tr>
<td>Database_Value</td>
<td>5 – 60</td>
<td></td>
</tr>
<tr>
<td>Default Button</td>
<td>5 – 62</td>
<td></td>
</tr>
<tr>
<td>Default Value (Item)</td>
<td>5 – 63</td>
<td></td>
</tr>
<tr>
<td>Direction</td>
<td>5 – 67</td>
<td></td>
</tr>
<tr>
<td>Displayed (Item)</td>
<td>5 – 74</td>
<td></td>
</tr>
<tr>
<td>Editor</td>
<td>5 – 76</td>
<td></td>
</tr>
<tr>
<td>Editor X Position</td>
<td>5 – 77</td>
<td></td>
</tr>
<tr>
<td>Editor Y Position</td>
<td>5 – 77</td>
<td></td>
</tr>
<tr>
<td>Enabled (Item)</td>
<td>5 – 77</td>
<td></td>
</tr>
<tr>
<td>Fixed Length (Item)</td>
<td>5 – 86</td>
<td></td>
</tr>
<tr>
<td>Format Mask</td>
<td>5 – 88</td>
<td></td>
</tr>
<tr>
<td>Hint (Item)</td>
<td>5 – 95</td>
<td></td>
</tr>
<tr>
<td>Horizontal Scroll Bar</td>
<td>5 – 97</td>
<td></td>
</tr>
<tr>
<td>Icon Name</td>
<td>5 – 99</td>
<td></td>
</tr>
<tr>
<td>Iconic</td>
<td>5 – 100</td>
<td></td>
</tr>
<tr>
<td>Initial Keyboard State</td>
<td>5 – 103</td>
<td></td>
</tr>
<tr>
<td>Insert Allowed (Item)</td>
<td>5 – 104</td>
<td></td>
</tr>
<tr>
<td>Item Type</td>
<td>5 – 105</td>
<td></td>
</tr>
<tr>
<td>Items Displayed</td>
<td>5 – 106</td>
<td></td>
</tr>
<tr>
<td>Item_Is_Valid</td>
<td>5 – 106</td>
<td></td>
</tr>
<tr>
<td>Keep Position</td>
<td>5 – 108</td>
<td></td>
</tr>
<tr>
<td>Label (Item)</td>
<td>5 – 110</td>
<td></td>
</tr>
<tr>
<td>List Elements</td>
<td>5 – 112</td>
<td></td>
</tr>
<tr>
<td>List Style</td>
<td>5 – 113</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Page Number</td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>Lock Record</td>
<td>5 – 113</td>
<td></td>
</tr>
<tr>
<td>LOV</td>
<td>5 – 116</td>
<td></td>
</tr>
<tr>
<td>LOV for Validation</td>
<td>5 – 116</td>
<td></td>
</tr>
<tr>
<td>LOV Position</td>
<td>5 – 118</td>
<td></td>
</tr>
<tr>
<td>Maximum Length</td>
<td>5 – 123</td>
<td></td>
</tr>
<tr>
<td>Mirror Item</td>
<td>5 – 131</td>
<td></td>
</tr>
<tr>
<td>Mouse Navigate</td>
<td>5 – 134</td>
<td></td>
</tr>
<tr>
<td>Multi-Line</td>
<td>5 – 136</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>5 – 136</td>
<td></td>
</tr>
<tr>
<td>Navigable</td>
<td>5 – 138</td>
<td></td>
</tr>
<tr>
<td>Next Item</td>
<td>5 – 142</td>
<td></td>
</tr>
<tr>
<td>Next Navigation Item</td>
<td>5 – 140</td>
<td></td>
</tr>
<tr>
<td>OLE Activation Style</td>
<td>5 – 143</td>
<td></td>
</tr>
<tr>
<td>OLE Class</td>
<td>5 – 144</td>
<td></td>
</tr>
<tr>
<td>OLE Do In Out</td>
<td>5 – 145</td>
<td></td>
</tr>
<tr>
<td>OLE In-place Activation</td>
<td>5 – 146</td>
<td></td>
</tr>
<tr>
<td>OLE Popup Menu Items</td>
<td>5 – 147</td>
<td></td>
</tr>
<tr>
<td>OLE Resize Style</td>
<td>5 – 150</td>
<td></td>
</tr>
<tr>
<td>OLE Tenant Aspects</td>
<td>5 – 150</td>
<td></td>
</tr>
<tr>
<td>OLE Tenant Types</td>
<td>5 – 151</td>
<td></td>
</tr>
<tr>
<td>Other Values</td>
<td>5 – 154</td>
<td></td>
</tr>
<tr>
<td>Previous Navigation Item</td>
<td>5 – 157</td>
<td></td>
</tr>
<tr>
<td>PreviousItem</td>
<td>5 – 158</td>
<td></td>
</tr>
<tr>
<td>Primary Key (Item)</td>
<td>5 – 159</td>
<td></td>
</tr>
<tr>
<td>Quality</td>
<td>5 – 160</td>
<td></td>
</tr>
<tr>
<td>Query Allowed (Item)</td>
<td>5 – 161</td>
<td></td>
</tr>
<tr>
<td>Query Length</td>
<td>5 – 162</td>
<td></td>
</tr>
<tr>
<td>Query Only</td>
<td>5 – 162</td>
<td></td>
</tr>
<tr>
<td>Range High Value</td>
<td>5 – 165</td>
<td></td>
</tr>
<tr>
<td>Range Low Value</td>
<td>5 – 165</td>
<td></td>
</tr>
<tr>
<td>Reading Order</td>
<td>5 – 166</td>
<td></td>
</tr>
<tr>
<td>Rendered</td>
<td>5 – 174</td>
<td></td>
</tr>
<tr>
<td>Required (Item)</td>
<td>5 – 174</td>
<td></td>
</tr>
<tr>
<td>Secure (Item)</td>
<td>5 – 179</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Page Number</td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>Show OLE Popup Menu</td>
<td>5 – 181</td>
<td></td>
</tr>
<tr>
<td>Show OLE Tenant</td>
<td>5 – 182</td>
<td></td>
</tr>
<tr>
<td>SHOW_POPUPMENU</td>
<td>5 – 181</td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>5 – 182</td>
<td></td>
</tr>
<tr>
<td>Sizing Style</td>
<td>5 – 184</td>
<td></td>
</tr>
<tr>
<td>Space Between Records</td>
<td>5 – 184</td>
<td></td>
</tr>
<tr>
<td>Unchecked (Value)</td>
<td>5 – 192</td>
<td></td>
</tr>
<tr>
<td>Update Allowed (Item)</td>
<td>5 – 194</td>
<td></td>
</tr>
<tr>
<td>Update Only if NULL</td>
<td>5 – 197</td>
<td></td>
</tr>
<tr>
<td>Update_Permission</td>
<td>5 – 198</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>5 – 205</td>
<td></td>
</tr>
<tr>
<td>VBX Control File</td>
<td>5 – 205</td>
<td></td>
</tr>
<tr>
<td>VBX Control Name</td>
<td>5 – 206</td>
<td></td>
</tr>
<tr>
<td>VBX Control Value Property</td>
<td>5 – 206</td>
<td></td>
</tr>
<tr>
<td>Vertical Scroll Bar</td>
<td>5 – 208</td>
<td></td>
</tr>
<tr>
<td>Visual Attribute</td>
<td>5 – 216</td>
<td></td>
</tr>
<tr>
<td>Visual Attribute Name</td>
<td>5 – 213</td>
<td></td>
</tr>
<tr>
<td>Window_Handle</td>
<td>5 – 219</td>
<td></td>
</tr>
<tr>
<td>Wrap Style</td>
<td>5 – 222</td>
<td></td>
</tr>
<tr>
<td>X Position</td>
<td>5 – 222</td>
<td></td>
</tr>
<tr>
<td>Y Position</td>
<td>5 – 222</td>
<td></td>
</tr>
</tbody>
</table>

**List of Values (LOV) Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto–Confirm</td>
<td>5 – 25</td>
</tr>
<tr>
<td>Auto–Display</td>
<td>5 – 25</td>
</tr>
<tr>
<td>Auto–Refresh</td>
<td>5 – 26</td>
</tr>
<tr>
<td>Auto–Skip (LOV)</td>
<td>5 – 28</td>
</tr>
<tr>
<td>Class</td>
<td>5 – 38</td>
</tr>
<tr>
<td>Column Mapping</td>
<td>5 – 39</td>
</tr>
<tr>
<td>Column Specification</td>
<td>5 – 41</td>
</tr>
<tr>
<td>Comment</td>
<td>5 – 44</td>
</tr>
<tr>
<td>Group Name</td>
<td>5 – 94</td>
</tr>
<tr>
<td>Long List</td>
<td>5 – 115</td>
</tr>
<tr>
<td>LOV Position</td>
<td>5 – 118</td>
</tr>
<tr>
<td>Property</td>
<td>Page Number</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Name</td>
<td>5 – 136</td>
</tr>
<tr>
<td>Record Group</td>
<td>5 – 167</td>
</tr>
<tr>
<td>Size</td>
<td>5 – 182</td>
</tr>
<tr>
<td>Title</td>
<td>5 – 188</td>
</tr>
<tr>
<td>Visual Attribute Name</td>
<td>5 – 213</td>
</tr>
<tr>
<td>X Position</td>
<td>5 – 222</td>
</tr>
<tr>
<td>Y Position</td>
<td>5 – 222</td>
</tr>
</tbody>
</table>

### Menu Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerator</td>
<td>5 – 21</td>
</tr>
<tr>
<td>Bottom Title (Menu)</td>
<td>5 – 31</td>
</tr>
<tr>
<td>Case Restriction</td>
<td>5 – 35</td>
</tr>
<tr>
<td>Checked</td>
<td>5 – 37</td>
</tr>
<tr>
<td>Class</td>
<td>5 – 38</td>
</tr>
<tr>
<td>Command Text</td>
<td>5 – 42</td>
</tr>
<tr>
<td>Command Type</td>
<td>5 – 43</td>
</tr>
<tr>
<td>Comment</td>
<td>5 – 44</td>
</tr>
<tr>
<td>Data Type</td>
<td>5 – 55</td>
</tr>
<tr>
<td>Default Value (Menu Substitution Parameter)</td>
<td>5 – 63</td>
</tr>
<tr>
<td>Directory</td>
<td>5 – 72</td>
</tr>
<tr>
<td>Display w/o Priv</td>
<td>5 – 73</td>
</tr>
<tr>
<td>Displayed (Menu Item)</td>
<td>5 – 74</td>
</tr>
<tr>
<td>Enabled (Menu Item)</td>
<td>5 – 78</td>
</tr>
<tr>
<td>File</td>
<td>5 – 80</td>
</tr>
<tr>
<td>Fixed Length (Menu Substitution Parameter)</td>
<td>5 – 86</td>
</tr>
<tr>
<td>Help</td>
<td>5 – 95</td>
</tr>
<tr>
<td>Hint (Menu Item)</td>
<td>5 – 96</td>
</tr>
<tr>
<td>Hint (Menu Substitution Parameter)</td>
<td>5 – 96</td>
</tr>
<tr>
<td>Icon Name</td>
<td>5 – 99</td>
</tr>
<tr>
<td>Identification</td>
<td>5 – 101</td>
</tr>
<tr>
<td>Label (Menu Item)</td>
<td>5 – 110</td>
</tr>
<tr>
<td>Label (Menu Parameter)</td>
<td>5 – 111</td>
</tr>
<tr>
<td>Magic Item</td>
<td>5 – 120</td>
</tr>
<tr>
<td>Property</td>
<td>Page Number</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Main Menu</td>
<td>5 – 121</td>
</tr>
<tr>
<td>Maximum Length (Menu Substitution Parameter)</td>
<td>5 – 124</td>
</tr>
<tr>
<td>Menu Item Radio Group</td>
<td>5 – 125</td>
</tr>
<tr>
<td>Menu Item Roles</td>
<td>5 – 125</td>
</tr>
<tr>
<td>Menu Item Type</td>
<td>5 – 126</td>
</tr>
<tr>
<td>Menu Module Roles</td>
<td>5 – 129</td>
</tr>
<tr>
<td>Name</td>
<td>5 – 136</td>
</tr>
<tr>
<td>Parameter Menus</td>
<td>5 – 154</td>
</tr>
<tr>
<td>Required (Menu Substitution Parameter)</td>
<td>5 – 175</td>
</tr>
<tr>
<td>Secure (Menu Substitution Parameter)</td>
<td>5 – 179</td>
</tr>
<tr>
<td>Startup Code</td>
<td>5 – 186</td>
</tr>
<tr>
<td>Subtitle</td>
<td>5 – 187</td>
</tr>
<tr>
<td>Tear–off</td>
<td>5 – 188</td>
</tr>
<tr>
<td>Title</td>
<td>5 – 188</td>
</tr>
<tr>
<td>Use Security</td>
<td>5 – 201</td>
</tr>
<tr>
<td>Visual Attribute Name</td>
<td>5 – 213</td>
</tr>
</tbody>
</table>

### Record Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status (Record)</td>
<td>5 – 187</td>
</tr>
</tbody>
</table>

### Record Group Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>5 – 38</td>
</tr>
<tr>
<td>Column Specification</td>
<td>5 – 41</td>
</tr>
<tr>
<td>Comment</td>
<td>5 – 44</td>
</tr>
<tr>
<td>Name</td>
<td>5 – 136</td>
</tr>
<tr>
<td>Other Values</td>
<td>5 – 154</td>
</tr>
<tr>
<td>Record Group Query</td>
<td>5 – 168</td>
</tr>
<tr>
<td>Record Group Type</td>
<td>5 – 168</td>
</tr>
</tbody>
</table>

### Relation Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>5 – 38</td>
</tr>
<tr>
<td>Comment</td>
<td>5 – 44</td>
</tr>
<tr>
<td>Coordination</td>
<td>5 – 47</td>
</tr>
<tr>
<td>Property</td>
<td>Page Number</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Coordination Status</td>
<td>5 – 48</td>
</tr>
<tr>
<td>Detail</td>
<td>5 – 66</td>
</tr>
<tr>
<td>Join Condition</td>
<td>5 – 107</td>
</tr>
<tr>
<td>Master Deletes</td>
<td>5 – 122</td>
</tr>
<tr>
<td>Name</td>
<td>5 – 136</td>
</tr>
<tr>
<td>Next Detail Relation</td>
<td>5 – 142</td>
</tr>
<tr>
<td>Next Master Relation</td>
<td>5 – 142</td>
</tr>
<tr>
<td>Prevent Masterless Operation</td>
<td>5 – 155</td>
</tr>
</tbody>
</table>

**Trigger Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>5 – 38</td>
</tr>
<tr>
<td>Comment</td>
<td>5 – 44</td>
</tr>
<tr>
<td>Execution Style</td>
<td>5 – 79</td>
</tr>
<tr>
<td>Fire in Enter Query Mode</td>
<td>5 – 82</td>
</tr>
<tr>
<td>Name</td>
<td>5 – 136</td>
</tr>
<tr>
<td>Show Keys/Show Keys Description</td>
<td>5 – 180</td>
</tr>
<tr>
<td>Trigger Style</td>
<td>5 – 191</td>
</tr>
<tr>
<td>Trigger Text</td>
<td>5 – 191</td>
</tr>
<tr>
<td>Trigger Type</td>
<td>5 – 192</td>
</tr>
<tr>
<td>Window Properties</td>
<td>Property</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td></td>
<td>Closeable</td>
</tr>
<tr>
<td></td>
<td>Comment</td>
</tr>
<tr>
<td></td>
<td>Fixed Size</td>
</tr>
<tr>
<td></td>
<td>Horizontal Scroll Bar</td>
</tr>
<tr>
<td></td>
<td>Horizontal Toolbar</td>
</tr>
<tr>
<td></td>
<td>Icon Name</td>
</tr>
<tr>
<td></td>
<td>Icon Title</td>
</tr>
<tr>
<td></td>
<td>Iconifiable</td>
</tr>
<tr>
<td></td>
<td>Inherit Menu</td>
</tr>
<tr>
<td></td>
<td>Modal</td>
</tr>
<tr>
<td></td>
<td>Moveable</td>
</tr>
<tr>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td>Remove on Exit</td>
</tr>
<tr>
<td></td>
<td>Size</td>
</tr>
<tr>
<td></td>
<td>Title</td>
</tr>
<tr>
<td></td>
<td>Vertical Scroll Bar</td>
</tr>
<tr>
<td></td>
<td>Vertical Toolbar</td>
</tr>
<tr>
<td></td>
<td>View</td>
</tr>
<tr>
<td></td>
<td>Visible</td>
</tr>
<tr>
<td></td>
<td>Visual Attribute Name</td>
</tr>
<tr>
<td></td>
<td>Window Style</td>
</tr>
<tr>
<td></td>
<td>Window_State</td>
</tr>
<tr>
<td></td>
<td>X Position</td>
</tr>
<tr>
<td></td>
<td>Y Position</td>
</tr>
<tr>
<td></td>
<td>Zoomable (GUI Hint)</td>
</tr>
</tbody>
</table>
Item Properties

Each item has a number of properties that determine the way the item appears or the behavior of the item. Many of these properties are common among items.

The following table lists the valid item properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Button</th>
<th>Chart</th>
<th>CheckBox</th>
<th>Container</th>
<th>Display</th>
<th>Item</th>
<th>List</th>
<th>Image</th>
<th>List Item</th>
<th>Radio Button</th>
<th>Radio Group</th>
<th>Text Item</th>
<th>VBX Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Key</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alignment</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application Instance</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Auto-Skip (Item)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Auto Hint</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base Table (Item)</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bevel</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Canvas (Item)</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case Insensitive Query</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case Restriction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Check Box Other Values</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checked Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comment</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current Record Attribute</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Copy Value from Item</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Type</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Database Value</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default Button</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default Value (Item)</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Displayed</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Editor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Editor X Position</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Editor Y Position</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enabled (Item)</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Button</td>
<td>Chart</td>
<td>Check Box</td>
<td>OLE Container</td>
<td>Display Item</td>
<td>Image</td>
<td>List Item</td>
<td>Radio Button</td>
<td>Radio Group</td>
<td>Text Item</td>
<td>VRX Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------</td>
<td>-------</td>
<td>-----------</td>
<td>---------------</td>
<td>--------------</td>
<td>-------</td>
<td>-----------</td>
<td>--------------</td>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed Length (Item)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Format Mask</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hint (Item)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Horizontal Scroll Bar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Icon Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iconic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert Allowed (Item)</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Item_Displayed</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Item_is_Valid</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Item Type</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keep Position</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Label (Item)</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>List Element</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>List Style</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lock Record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOV for Validation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOV Position</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum Length</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mirror Item</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mouse Navigate</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi–Line</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Navigable</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Next Item</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Next Navigation Item</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OLE Activation Style</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OLE Classes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OLE Do In Out</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OLE In–place Activation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OLE Popup Menu Item</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Button</td>
<td>Chart</td>
<td>Check Box</td>
<td>OLE Container</td>
<td>Display Item</td>
<td>Image Item</td>
<td>List Item</td>
<td>Radio Button</td>
<td>Radio Group</td>
<td>Text Item</td>
<td>VBX Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------</td>
<td>-------</td>
<td>-----------</td>
<td>---------------</td>
<td>--------------</td>
<td>------------</td>
<td>-----------</td>
<td>--------------</td>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OLE Resize Style</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OLE Tenant Aspects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OLE Tenant Types</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other Values</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Previous Navigation Item</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PreviousItem</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Key (Item)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query Allowed (Item)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query Length</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query Only</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Range High Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Range Low Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rendered (Item)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Required (Item)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Show OLE Popup Menu</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Show OLE Tenant</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHOW_POPUP_MENU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sizing Style</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Space Between Records</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unchecked (Value)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update Allowed (Item)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update Only if NULL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update Permission</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VBX Control File</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VBX Control Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VBX Control Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Button</td>
<td>Chart</td>
<td>Check Box</td>
<td>OLE Container</td>
<td>Display Item</td>
<td>Image Item</td>
<td>List Item</td>
<td>Radio Button</td>
<td>Radio Group</td>
<td>Text Item</td>
<td>VBX Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------</td>
<td>-------</td>
<td>-----------</td>
<td>---------------</td>
<td>--------------</td>
<td>------------</td>
<td>-----------</td>
<td>--------------</td>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Property Descriptions

The following descriptions supply pertinent information about individual properties. Property names are shown as they appear in the Properties window in the Designer.
Accelerator

Specifies a logical function key to be associated with a menu item. Accelerator keys are named ACCELERATOR1, ACCELERATOR2, and so on, through ACCELERATOR5. Operators can select the menu item by pressing the key or key combination that is mapped to the logical accelerator key.

**Applies to:** menu item

**Set:** Designer

**Required/Optional:** optional

**Restrictions:**
- Not valid for a menu running in full-screen display style.
- Not valid for separator menu items.
- Key mappings must not interfere with standard Oracle Forms key mappings.
- When running with bar-style menus, accelerator keys can be used only for items on the menu that is currently displayed.

**Usage Notes:** The mappings of logical accelerator keys to physical device keys is defined in the runtime resource file. You must edit the resource file in Oracle Terminal to change the key mappings. You can also create additional accelerator keys in Oracle Terminal (ACCELERATOR6, ACCELERATOR7, and so on), which you can then associate with menu items in a menu module.

Access Key

Specifies the character that will be the access key, allowing the operator to select or execute an item by pressing a key combination, such as Alt-C.

The access key character is displayed with an underscore in the item label.

For example, assume that Push_Button1’s label is “Commit” and the access key is defined as “c”. When the operator presses Alt-C (on Microsoft Windows), Oracle Forms executes the “Commit” command.

**Applies to:** button, radio button, and check box
The character selected for an access key must be a character in the item’s label, in order for the access key to be displayed to the operator.

• Buttons with the Iconic property set to True cannot have an access key.

When the operator initiates an action via an access key, any triggers associated with the action fire. For example, assume that Push_Button1 has an access key assigned to it. Assume also that there is a When–Button–Pressed trigger associated with Push_Button1. When the operator presses the access key, the When–Button–Pressed trigger fires for Push_Button1.

Alert Style

Specifies the style of the alert, whether caution, warning, or informational. On GUI platforms, the alert style determines which bitmap icon is displayed in the alert.

Applies to: alert
Set: Designer
Default: warning
Alignment

Specifies the text alignment within the item. The allowable values for this property are as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Left-justified, regardless of Reading Order property.</td>
</tr>
<tr>
<td>Center</td>
<td>Centered, regardless of Reading Order property.</td>
</tr>
<tr>
<td>Right</td>
<td>Right-justified, regardless of Reading Order property.</td>
</tr>
<tr>
<td>Start</td>
<td>Item text is aligned with the starting edge of the item bounding box. The starting edge depends on the value of the item’s Reading Order property. Start is evaluated as Right alignment when the reading order is Right To Left, and as Left alignment when the reading order is Left to Right.</td>
</tr>
<tr>
<td>End</td>
<td>Item text is aligned with the ending edge of the item bounding box. The ending edge depends on the value of the item’s Reading Order property. End is evaluated as Left alignment when the reading order is Right To Left, and as Right alignment when the reading order is Left to Right.</td>
</tr>
</tbody>
</table>

Applies to: display item, text item
Set: Designer, programmatically
Refer to Built-in: • GET_ITEMPROPERTY
• SET_ITEMPROPERTY
Default: Start
Usage Notes: • In unidirectional applications (reading order Left to Right), accept the default, Start, in most cases. For unidirectional applications, Start gives exactly the same results as Left and End gives the same results as Right.
• In bidirectional applications:
  • If your data must be aligned with the item’s Reading Order, choose Start (the default).
  • If your data must be aligned opposite to the item’s Reading Order, choose End.
• Unsupported by some window managers.
**Application Instance**

Specifies a reference to an instance of an application on the Microsoft Windows platform. Other platforms always return the NULL value.

- **Applies to:** form, block, or item
- **Refer to Built-in:** GET_APPLICATIONPROPERTY
- **Default:** NULL
- **Restrictions:** Valid only on Microsoft Windows (Returns NULL on other platforms).
- **Usage Notes:** Specify the APPLICATION_INSTANCE property in GET_APPLICATIONPROPERTY to obtain the instance handle.

**Auto Hint**

Determines when the help text specified by the item property, Hint, is displayed:

- Set Auto Hint to True to have Oracle Forms display the hint text whenever the input focus enters the item.
- Set Auto Hint to False to have Oracle Forms display the hint text only when the input focus is in the item and the operator presses [Help] or selects the Help command on the default menu.

- **Applies to:** all items except chart item, display item, and custom item
- **Set:** Designer, programmatically
- **Refer to Built-in:**
  - GET_ITEMPROPERTY
  - SET_ITEMPROPERTY
- **Default:** False
- **Restrictions:** Not applicable when the Hint property is NULL.
- **Usage Notes:** If a trigger causes Oracle Forms to navigate through several items before stopping at the target item, the help text does not display for the traversed items, but only for the target item.
**Auto–Confirm**

Specifies what happens when an operator reduces the list to a single choice when using auto-reduction or searching:

- When Auto–Confirm is set to True, the LOV is dismissed automatically and column values from the single row are assigned to their corresponding return items.
- When Auto–Confirm is set to False, the LOV remains displayed, giving the operator the option to explicitly select the remaining choice or dismiss the LOV.

| Applies to: | LOV |
| Set:        | Designer |
| Default:    | False |

**Auto–Display**

Specifies whether Oracle Forms displays the LOV automatically when the operator or the application navigates into a text item to which the LOV is attached.

| Applies to: | LOV |
| Set:        | Designer |
| Default:    | False |

**Auto–Query**

See Coordination.
Auto-Refresh

Determines whether Oracle Forms re-executes the query to populate an LOV that is based on a query record group. By default, Oracle Forms executes the query to populate an LOV’s underlying record group whenever the LOV is invoked; that is, whenever the LOV is displayed, or whenever Oracle Forms validates a text item that has the Use LOV for Validation property set to True.

- When Auto-Refresh is set to True (the default), Oracle Forms executes the query each time the LOV is invoked. This behavior ensures that the LOV’s underlying record group contains the most recent database values.

- When Auto-Refresh is set to False, Oracle Forms executes the query only if the LOV’s underlying record group is not flagged as having been populated by a query that occurred because this or any other LOV was invoked. (Remember that more than one LOV can be based on the same record group.) If the LOV’s underlying record group has already been populated as a result of an LOV displaying, Oracle Forms does not re-execute the query, but instead displays the LOV using the records currently stored in the record group.

The Auto-Refresh property also determines how long records retrieved by the query remain stored in the underlying record group:

- When Auto-Refresh is set to True, records returned by the query are stored in the underlying record group only as long as the LOV is needed. Once the operator dismisses the LOV, or validation is completed, the record cache is destroyed.

- When Auto-Refresh is set to False, records from the initial query remain stored in the LOV’s underlying record group until they are removed or replaced. You can manipulate these records programmatically. For example, you can explicitly replace the records in an LOV’s underlying record group by calling the POPULATE_GROUP built-in. Other record group built-ins allow you to get and set the values of cells in a record group.

Applies to: list of values (LOV)
Set: Designer, programmatically
Refer to Built-in: • GET_LOV_PROPERTY • SET_LOV_PROPERTY
Default: True

Restrictions: Valid only for an LOV based on a query record group, rather than a static or non–query record group.

Usage Notes:

- When multiple LOVs are based on the same record group, it is usually appropriate to use the same Auto–Refresh setting for each one. This is not, however, a strict requirement; the following scenario describes refresh behavior when one LOV has Auto–Refresh set to True and another has Auto–Refresh set to False.

  LOV1 and LOV2 are based on the same record group; LOV1 has Auto–Refresh set to True, LOV2 has Auto–Refresh set to False. When LOV1 is invoked, Oracle Forms executes the query to populate the underlying record group. When the operator dismisses LOV1, Oracle Forms destroys the record cache, and clears the record group.

  When LOV2 is subsequently invoked, Oracle Forms again executes the query to populate the record group, even though LOV2 has Auto–Refresh set to False. Because LOV2’s underlying record group was cleared when LOV1 was dismissed, Oracle Forms does not consider it to have been queried by an LOV invocation, and so re–executes the query.

  If, on the other hand, both LOV1 and LOV2 had Auto–Refresh set to False, Oracle Forms would execute the query when LOV1 was invoked, but would not re–execute the query for LOV2. This is true even if the initial query returned no rows.

- When Auto–Refresh is set to False, you can programmatically replace the rows that were returned by the initial query with POPULATE_GROUP. Oracle Forms ignores this operation when deciding whether to re–execute the query. (Oracle Forms looks only at the internal flag that indicates whether a query has occurred, not at the actual rows returned by that query.)
Auto-Skip (Item)

Moves the cursor to the next navigable item when adding or changing data in the last character of the current item. The last character is defined by the Maximum Length property.

Applies to: text item
Set: Designer, programmatically
Refer to Built-in: • GET_ITEM_PROPERTY
• SET_ITEM_PROPERTY
Default: False
Restrictions: • The text item must have the Displayed and Enabled properties set to True at runtime.
• Valid only for single-line text items.
• The Key–NXT–ITEM trigger does not fire when the cursor moves as a result of this property. This behavior is consistent with the fact that the operator did not press [Next Item].

Usage Notes: Combine the Auto-Skip property with the Fixed Length property to move the cursor to the next applicable text item when an operator enters the last required character.

Auto-Skip (LOV)

Moves the cursor to the next navigable item when the operator makes a selection from an LOV to a text item. When Auto-Skip is set to False, the focus remains in the text item after the operator makes a selection from the LOV.

Applies to: LOV
Set: Designer, programmatically
Refer to Built-in: SET_ITEM_PROPERTY
Default: False
Restrictions:  
- The text item must have the Displayed and Enabled properties set to True at runtime.
- The Key–NXT–ITEM trigger does not fire when the cursor moves as a result of this property. This behavior is consistent with the fact that the operator did not press [Next Item].

Base Table (Block)

Specifies the name of the database table or view to which the block corresponds. Items in a base table block can correspond to columns in the block’s base table.

Applies to: block
Set: Designer
Refer to Built-in: GET_BLOCK_PROPERTY
Required/Optional: Optional
Usage Notes:  
- A block is not required to correspond to any table.
- A base table block can contain control items that do not correspond to columns in the base table.

Base Table (Item)

Establishes that an item corresponds to a column in the base table of the owning block.

Applies to: any item except button, chart, or custom item (VBX)
Set: Designer
Refer to Built-in: GET_ITEM_PROPERTY
Default: True
Required/Optional: optional
Bevel

Specifies the appearance of the object border, either RAISED, LOWERED, or NONE.

Applies to: chart item, image item, custom item, stacked canvas–views, text items (Microsoft Windows only)

Set: Designer

Refer to Built-in: GET_ITEMPROPERTY

Default: LOWERED

Restrictions: On window managers that do not support beveling, the RAISED and LOWERED options are equivalent, and simply specify that the item should have a border.

Block Description

See In Menu/Block Description.
Bottom Title (Editor)

Specifies a title of up to 72 characters to appear at the bottom of the editor window.

Applies to: editor

Set: Designer

Required/Optional: optional

Bottom Title (Menu)

Specifies a title of up to 72 characters to appear at the bottom of the full-screen menu, above the message line and status line.

Applies to: menu

Set: Designer

Required/Optional: optional

Restrictions: Valid only for full-screen menu display style.

Button 1, Button 2, Button 3

Specifies the text labels for the three available alert buttons.

Applies to: alert

Set: Designer, programmatically

Refer to Built-in: SET_ALERT_BUTTON_PROPERTY

Required/Optional: At least one button must have a label.

Default: Button 1: OK, Button 2: Cancel, Button 3: NULL

See also: SET_ALERT_BUTTON_PROPERTY
### Calling_Form

Specifies the name of the calling form, as indicated by the form module Name property.

**Applies to:** application

**Set:** not settable

**Refer to Built-in:** GET_APPLICATION_PROPERTY

**Default:** NULL

**Usage Notes:** Only valid in a called form; that is, a form that was invoked from a calling form by the execution of the CALL_FORM built-in procedure.

### Canvas

Specifies the canvas-view on which you want the item to be displayed.

**Applies to:** item

**Set:** Designer

**Default:** The item’s current canvas assignment.

**Required/Optional:** optional

**Restrictions:** The canvas-view specified must already exist in the form.

**Usage Notes:**
- Items are assigned to a specific canvas, which in turn is assigned to a specific window.
- If you leave the Canvas property blank, the item is a NULL-canvas item; that is, an item that is not assigned to any canvas and so cannot be displayed in the Layout Editor or at runtime.
- If you change the name of a canvas in the Designer, Oracle Forms automatically updates the Canvas property for all items assigned to that canvas.
## Canvas-view Type

Specifies the type of canvas-view, either Content, Stacked, Vertical Toolbar, or Horizontal Toolbar. The type determines how the canvas-view is displayed in the window to which it is assigned, and determines which properties make sense for the canvas-view.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content</td>
<td>The default. Specifies that the canvas-view should occupy the entire content area of the window to which it is assigned. Most canvas-views are content canvas-views.</td>
</tr>
<tr>
<td>Stacked</td>
<td>Specifies that the canvas-view should be displayed in its window at the same time as the window's content canvas-view. Stacked views are usually displayed programmatically and overlay some portion of the content view displayed in the same window.</td>
</tr>
<tr>
<td>Vertical Toolbar</td>
<td>Specifies that the canvas-view should be displayed as a vertical toolbar under the menu bar of the window. You can define iconic buttons, pop-lists, and other items on the toolbar as desired.</td>
</tr>
<tr>
<td>Horizontal Toolbar</td>
<td>Specifies that the canvas-view should be displayed as a horizontal toolbar at the left side of the window to which it is assigned.</td>
</tr>
</tbody>
</table>

**Applies to:** canvas-view  
**Set:** Designer  
**Default:** Content  
**Usage Notes:** In the Properties window, the properties listed under the Stacked View heading are valid only for a canvas-view with the Canvas-view type property set to Stacked.
Case Insensitive Query

Determines whether the operator can perform case-insensitive queries on the text item.

**Applies to:** text item

**Set:** Designer, programmatically

**Refer to Built-in:**
- GET_ITEMPROPERTY
- SET_ITEMPROPERTY

**Default:** False

**Restrictions:** Setting this property to True can cause queries to take longer to execute.

**Usage Notes:** Case-insensitive queries are optimized to take advantage of an index. For example, assume you perform the following steps:

- Create an index on the EMP table.
- Set the Case Insensitive Query property on ENAME to True.
- In Enter Query mode, enter the name ‘BLAKE’ into :ENAME.
- Execute the query.

Oracle Forms constructs the following statement:

```
SELECT * FROM EMP WHERE UPPER(ENAME) = 'BLAKE' AND 
(ENAME LIKE 'bl%' OR ENAME LIKE 'bL%' OR 
ENAME LIKE 'BL%' OR ENAME LIKE 'bl%');
```

The last part of the WHERE clause is performed first, making use of the index. Once the database finds an entry that begins with bl, it checks the UPPER(ENAME) = ‘BLAKE’ part of the statement, and makes the exact match.
Case Restriction

Specifies the case for text entered in the text item or menu substitution parameter. The allowable values for this property are as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>Text appears as typed.</td>
</tr>
<tr>
<td>UPPER</td>
<td>Lower case text converted to upper case as it is typed.</td>
</tr>
<tr>
<td>LOWER</td>
<td>Upper case text converted to lower case as it is typed.</td>
</tr>
</tbody>
</table>

 Applies to:  text item, menu substitution parameters

 Set:  Designer, programmatically

Refer to Built-in:

- GET_ITEM_PROPERTY
- SET_ITEM_PROPERTY

Restrictions:

- Oracle Forms automatically sets the case restriction property to UPPER if you create the text item through the New Block window and the corresponding column in the database is of the DATE data type.
- Values assigned to the text item through triggers are not affected.
- Queried records display as they are stored in the database.
- Case Restriction governs the display of all strings, whether they are entered by an operator or assigned programmatically, because Case Restriction serves as both an input and output format mask enforced by the user interface.

If you programmatically assign string values that conflict with the setting for Case Restriction, you will not see the effect in the text item because its display will be forced to conform to the current setting of Case Restriction. This also means that if data that violates the Case Restriction setting is queried into or programmatically assigned to an item, then the internal value of that text item may differ from what the operator sees on the screen.
Character Cell WD/HT

Specifies the width and height of a character cell when the Coordinate System property is set to Real, rather than Character. The width and height are expressed in the current real units (centimeters, inches, or points) indicated by the Real Unit property setting.

Applies to: form module
Set: Designer
Required/Optional: optional
See also: Coordinate System property
Real Unit property

Check Box Other Values

Specifies how any fetched or assigned value that is not one of the pre-defined "checked" or "unchecked" values should be interpreted.

Applies to: check box
Set: Designer
Default: NOT ALLOWED
Usage Notes: The following settings are valid for this property:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Allowed</td>
<td>Any queried record that contains a value other than the user-defined checked and unchecked values is rejected and no error is raised. Any attempt to assign an other value is disallowed.</td>
</tr>
<tr>
<td>Checked</td>
<td>Any value other than the user-defined unchecked value is interpreted as the checked state.</td>
</tr>
<tr>
<td>Unchecked</td>
<td>Any value other than the user-defined checked value is interpreted as the unchecked state.</td>
</tr>
</tbody>
</table>
### Checked

Specifies the state of a check box– or radio–style menu item, either CHECKED or UNCHECKED.

**Applies to:** menu item  
**Set:** programmatically  
**Refer to Built–in:**  
- GET_MENU_ITEM_PROPERTY  
- SET_MENU_ITEM_PROPERTY  
**Default:** NULL  
**Required/Optional:** optional  
**Restrictions:** Valid only for a menu item with the Menu Item Type property set to Check or Radio.

### Checked Value

Specifies the value you want the check box to display as the checked state. For example, Y, 1, MANAGER, or 1992. When a value that matches the checked value is fetched or assigned to the check box, the check box is displayed checked. Similarly, when the operator toggles the check box to the checked state, the value of the check box becomes the checked value.

**Applies to:** check box  
**Set:** Designer  
**Default:** NULL  
**Required/Optional:** optional  
**Restrictions:** The value must be compatible with the datatype specified by the Data Type property.
Class

Specifies the name of the property class from which the object can inherit property settings.

Applies to: all objects
Set: Designer
Default: Null
Required/Optional: optional

Closeable

Specifies whether the window manager–specific Close command is enabled or disabled for a window. On GUI window managers, the Close command is available on the window’s system menu, or by double–clicking the box in the upper–left corner of the window.

Applies to: window
Set: Designer
Default: True
Restrictions: Cannot be set for a root window: a root window is always closeable.
Usage Notes:

• Setting Closeable to True enables the Close command so that the Close Window event can be sent to Oracle Forms when the operator issues the Close command. However, to actually close the window in response to this event, you must write a When–Window–Closed trigger that explicitly closes the window. You can close a window programmatically by calling HIDE_WINDOW, SET_WINDOWPROPERTY, or EXIT_FORM.

• On Microsoft Windows, if the operator closes the MDI parent window, Oracle Forms by executes DO_KEY('Exit_Form') by default.

See also: SYSTEM.EVENT_WINDOW system variable
When–Window–Closed trigger
The Column Mapping group of properties include Column Name, Column Title, Display Width, and Return Item.

**Applies to:** LOV

**Set:** Designer

**Column Name**
Specifies the names of the columns in an LOV.

**Required/Optional:** At least one column must be defined.

**Default:** Names of the columns in the underlying record group.

**Usage Notes:** The column names must adhere to object naming standards.

**Column Title**
Specifies the title that displays above the column currently selected in the column name list.

**Display Width**
Specifies the width to be reserved in the LOV for displaying values from the column currently selected in the Column Name list.

**Required/Optional:** optional

**Usage Notes:**
- If you want the column values to be displayed in the LOV, enter the number of characters you want Oracle Forms to reserve for the column in the LOV window.
- To make the column a hidden column, set Display Width to 0. (You can specify a return item for a hidden column, just as you would for a displayed column.)
- To add extra space between columns in the LOV window, increase the Display Width setting.
- The actual display width reserved for a column is based on the width of an average character in the LOV font. When a column value contains many wide characters, truncation can occur. To avoid this situation, increase the Display Width for the column.

**Return Item**
Specifies the name of the form item or variable to which Oracle Forms should assign the column’s value whenever the operator selects an LOV record.

**Default:** NULL

**Required/Optional:** optional
Usage Notes: The Return Item can be any of the following entries:
  • form item (block_name.item_name)
  • form parameter (PARAMETER.my_parameter)
  • global parameter (GLOBAL.my_global)

Do not put a colon in front of the object name.

---

Column Name (LOV)

See Column Mapping.

---

Column Name (Record Group)

See Column Specification.

---

Column Security

Specifies when Oracle Forms should enforce update privileges on a column-by-column basis for the block’s base table. If an operator does not have update privileges on a particular column in the base table, Oracle Forms makes the corresponding item non-updateable for this operator only, by turning off the Update Allowed item property at form startup.

The following table describes the effects of the allowable values for this property:

<table>
<thead>
<tr>
<th>State</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Oracle Forms enforces the update privileges that are defined in the database for the current operator.</td>
</tr>
<tr>
<td>False</td>
<td>Oracle Forms does not enforce the defined update privileges.</td>
</tr>
</tbody>
</table>

Applies to: block

Set: Designer

Refer to Built-in: GET_BLOCKPROPERTY

Default: False
See also:  Update Allowed (Block) property  
Update Allowed (Item) property  
Update_Permission property

---

**Column Specification**

The Column Specification group of properties include Column Name, Column Value, Data Type, Length.

**Applies to:** record group

**Set:** Designer

**Column Name**

Specifies the names of the columns in a record group.

**Required/Optional:** At least one column must be defined.

**Default:** Names of the columns in the underlying record group.

**Usage Notes:** The column names must adhere to object naming standards. There can be up to 255 columns in a record group.

**Column Value**

For a static record group, specifies the row values for the column currently selected in the Column Name list.

**Default:** NULL

**Restrictions:**

- Column value only applies to static record groups.
- You cannot reference an uninitialized variable or an item for this property, as that action constitutes a forward reference that Oracle Forms is unable to validate at design time.
- The data type of the value must correspond to the data type of its associated column, as indicated in the Column Name property.

**Data Type**

Specifies the data type for a given record group column.

**Default:** CHAR, except when you define a query record group, in which case, the data type of each column defaults to the data type of the corresponding database column.

**Restrictions:** The data type of a record group column can only be CHAR, LONG, NUMBER, or DATE.
**Length**

Specifies the length, in characters, of the record group column currently selected in the Column Name list.

**Default:** For a query record group, the default is the width specified for the column in the database. For a static record group, the default is 30.

**Required/Optional:** required

---

**Column Title (LOV)**

See Column Mapping.

---

**Column Value (Record Group)**

See Column Specification.

---

**Command Text**

Specifies menu item command text for the current menu item. Valid values depend on the current setting of the menu item Command Type property. For instance, when the command type is MENU, valid command text is the name of a submenu in the menu module. When the command type is PL/SQL, valid command text is any valid PL/SQL statements.

**Applies to:** menu item

**Set:** Designer

**Required/Optional:** Required for all command types except NULL.

**Restrictions:** The value can be up to 240 characters in length.

**See also:** Command Type property
Command Type

Specifies the nature of the menu item command. This property determines how Oracle Forms interprets the text in the Command Text property.

Applies to: menu item

Set: Designer

Default: NULL

Required/Optional: required

Usage Notes:

<table>
<thead>
<tr>
<th>Command Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null</td>
<td>Specifies that the menu item does not issue a command. The NULL command is required for separator menu items and optional for all other types of items.</td>
</tr>
<tr>
<td>Menu</td>
<td>Invokes a submenu. Valid command text is the name of the submenu to be invoked.</td>
</tr>
<tr>
<td>PL/SQL</td>
<td>The default command type. Executes a PL/SQL command. Valid command text is PL/SQL statements, including calls to built-in and user-named subprograms. Note: PL/SQL in a menu module cannot refer directly to the values of items, variables, or parameters in a form module. Instead, use the built-ins NAME_IN and COPY to indirectly reference such values.</td>
</tr>
<tr>
<td>Plus*</td>
<td>Avoid. To invoke SQL<em>Plus, use the PL/SQL command type, and execute the HOST built-in to launch SQL</em>Plus. (On Windows platforms, use plus31.exe as the executable name.)</td>
</tr>
<tr>
<td>Current Forms*</td>
<td>Avoid. To invoke Oracle Forms, use the PL/SQL command type, and execute the HOST or RUN_PRODUCT built-ins to execute a valid Oracle Forms login.</td>
</tr>
<tr>
<td>Macro*</td>
<td>Avoid. Executes a SQL*Menu macro.</td>
</tr>
</tbody>
</table>

*This command type is included for compatibility with previous versions. Do not use this command type in new applications.

See also: Command Text property
Comment

The Comment property specifies general information about any Oracle Forms object that you create. Use comments to record information that will be useful to you or to other designers who develop, maintain, and debug your applications.

Applies to: all objects
Set: Designer
Required/Optional: optional

Compression

Specifies whether an image being read into a form from a file should be compressed when converting to the Oracle internal format.

Applies to: image item
Set: Designer
Default: True

Connect_String

The Connect String property specifies the form operator’s SQL*NET connect string.

If the current operator does not have a SQL*NET connect string, Oracle Forms returns NULL.

Applies to: application
Refer to Built-in: GET_APPLICATION_PROPERTY
See also: Username property
Password property
Console Window

Specifies the name of the window that should display the Oracle Forms console. The console includes the status line and message line, and is displayed at the bottom of the window.

On Microsoft Windows, the console is always displayed on the MDI application window, rather than on any particular window in the form; however, you must still set this property to the name of a form window to indicate that you want the console to be displayed.

If you do not want a form to have a console, set this property to <Null>.

Applies to: form
Set: Designer
Default: WINDOW0
Required/Optional: optional

Coordinate Information

For more information about the Coordinate Information group of properties, refer to the individual property descriptions for Coordinate System, Real Unit, Character Cell, and Default Font Scaling.

Coordinate System

Specifies whether object size and position values should be interpreted as character cell values, or as real units (centimeters, inches, pixels, or points). The following settings are valid for this property:

Character Sets the coordinate system to a character cell–based measurement. The actual size and position of objects will depend on the size of a default character on your particular platform.

Real Sets the coordinate system to the unit of measure specified by the Real Unit property (centimeters, inches, pixels, or points.)
Changing the coordinate system for the form changes the ruler units displayed on Layout Editor rulers, but does not change the grid spacing and snap-points settings.

**Applies to:** form

**Set:** Designer

**Default:** Centimeter

**Usage Notes:** The coordinate system you select is enforced at design time and at runtime. For example, if you programmatically move a window with SET_WINDOW_PROPERTY, the position coordinates you pass to the built-in are interpreted in the current form coordinate units.

When you convert from one coordinate system to another, Oracle Forms automatically converts object size and position values that were specified declaratively at design time. Loss of precision can occur when you convert to less precise units.

If portability is a concern, setting the Coordinate System to Character provides the most portable unit across platforms, but sets a coarse grid that reduces the ability to fine-tune the layout. If your application runs in both character-mode and GUI, the decision about which coordinate system to use depends on which interface style you want to optimize.

If you want to optimize for GUIs, the Real setting provides maximum flexibility for proportional fonts, but may require some fine-tuning to avoid overlapping fields on the character-mode side.

If you want to optimize for character-mode, choose the Character setting. This setting provides less flexibility for the proportional fonts used on GUIs, but lets you line up character cell boundaries exactly.

<table>
<thead>
<tr>
<th>For this type of application...</th>
<th>Set Coordinate System to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI only</td>
<td>Real: inches, centimeters, or points</td>
</tr>
<tr>
<td>Character-mode only</td>
<td>Character</td>
</tr>
<tr>
<td>Mixed character-mode and GUI:</td>
<td></td>
</tr>
<tr>
<td>Optimize for GUI</td>
<td>Real</td>
</tr>
<tr>
<td>Optimize for character-mode</td>
<td>Character</td>
</tr>
</tbody>
</table>

**See also:** Real Unit property
Coordination

Specifies how and when the population phase of block coordination should occur. Specify the coordination desired by setting the Deferred and Auto-Query properties. When you set these properties at design time, Oracle Forms creates or modifies the appropriate master-detail triggers to enforce the coordination setting you choose.

Applies to: relation

Set: Designer, programmatically

Refer to Built-in:
- GET_RELATION_PROPERTY
- SET_RELATION_PROPERTY

Default: Immediate coordination (Deferred False, Auto-Query False)

Restrictions: The ability to set and get these properties programmatically is included only for applications that require a custom master-detail scheme. For a default master-detail relation created at design time, Oracle Forms generates the appropriate triggers to enforce coordination, and setting the coordination properties at runtime has no effect on the default trigger text.

Usage Notes: Whenever the current record in the master block changes at runtime (a coordination-causing event), Oracle Forms needs to populate the detail block with a new set of records. You can specify exactly how and when that population should occur by setting this property to one of three valid settings:

- **Deferred=False, Auto-Query ignored**
  The default setting. When a coordination-causing event occurs in the master block, the detail records are fetched immediately.

- **Deferred=True, Auto-Query=True**
  When a coordination-causing event occurs, Oracle Forms defers fetching the associated detail records until the operator navigates to the detail block.

- **Deferred=True, Auto-Query=False**
  When a coordination-causing event occurs, Oracle Forms defers fetching the associated detail records until the operator navigates to the detail block and explicitly executes a query.

- **Deferred=False, Auto-Query=True**
  Not a valid setting.
**Coordination_Status**

For a block that is a detail block in a master–detail block relation, this property specifies the current coordination status of the block with respect to its master block(s). This property is set to the value COORDINATED when the block is coordinated with all of its master blocks. When the block is not coordinated with all of its master blocks, Coordination_Status is set to NON_COORDINATED.

Immediately after records are fetched to the detail block, the status of a detail block is COORDINATED. When a different record becomes the current record in the master block, the status of the detail block again becomes NON_COORDINATED.

**Applies to:** relation

**Set:** programatically

**Refer to Built-in:**
- GET_BLOCK_PROPERTY
- SET_BLOCK_PROPERTY

**Usage Notes:** This property is included for designers who are programmatically enforcing a custom master–detail block coordination scheme. Its use is not required when you are using Oracle Forms declarative master–detail coordination.
Copy Value from Item

Specifies the source of the value that Oracle Forms uses to populate the item. When you define a master–detail relation, Oracle Forms sets this property automatically on the foreign key item(s) in the detail block. In such cases, the Copy Value from Item property names the primary key item in the master block whose value gets copied to the foreign key item in the detail block whenever a detail record is created or queried.

Applies to: all items except buttons, chart items, and image items
Set: Designer
Refer to Built-in: GET_ITEM_PROPERTY
Required/Optional: optional
Usage Notes:
- Specify this property in the form `<block_name>.<block_item_name>`.
- Setting the Copy Value from Item property does not affect record status at runtime, because the copying occurs during default record processing.
- To prevent operators from de–enforcing the foreign key relationship, set the Enabled property to False for the foreign key items.
- To get the Copy Value from Item property programmatically with GET_ITEMPROPERTY, use the constant ENFORCE_KEY.

Current Record Attribute

Specifies the named visual attribute used when an item is part of the current record.

Applies to: form, block, item
Set: Designer, programmatically
Refer to Built-in:
- GET_FORM_PROPERTY
- SET_FORM_PROPERTY
- GET_BLOCK_PROPERTY
- SET_BLOCK_PROPERTY
- GET_ITEM_PROPERTY
• *SET_ITEM_PROPERTY*

**Required/Optional:** optional

**Usage Notes:** This property can be set at the form, block, or item level, or at any combination of levels. If you specify named visual attributes at each level, the item-level attribute overrides all others, and the block-level overrides the form-level.

Current Record Attribute is frequently used at the block level to display the current row in a multi-record block in a special color. For example, if you define Vis_Att_Blue for the Emp block which displays four detail records, the current record will display as blue, because it contains the item that is part of the current record.

If you define an item-level Current Record Attribute, you can display a pre-determined item in a special color when it is part of the current record, but you cannot dynamically highlight the current item, as the input focus changes. For example, if you set the Current Record Attribute for EmpNo to Vis_Att_Green, the EmpNo item in the current record would display as green. When the input focus moved to EmpName, EmpNo would still be green and EmpName would not change.

---

**Current_Form**

Specifies the name of the .FMX file of the form currently being executed.

**Applies to:** application

**Set:** not settable

**Refer to Built-in:** GET_APPLICATION_PROPERTY

**Usage Notes:** Get the value of this property to determine the name of the file the current form came from in an application that has multiple called forms.

Current_Form at the application level corresponds to File_Name at the form level. File_Name is gettable with GET_FORM_PROPERTY.

**See also:** File_Name property
GET_FORM_PROPERTY built-in
Current_Form_Name

Specifies the name of the current form, as indicated by the form module Name property.

Applies to: application

Set: not settable

Refer to Built-in: GET_APPLICATION_PROPERTY

Usage Notes: Get the value of this property to determine the name of the current form in an application that has multiple called forms. Current_Form_Name at the application level corresponds to Form_Name at the form level. Form_Name is gettable with GET_FORM_PROPERTY.

See also: Form_Name property
GET_FORM_PROPERTY built-in
Current_Record

Specifies the number of the current record in the block’s list of records.

**Applies to:** block

**Set:** not settable

**Refer to Built-in:** GET_BLOCKPROPERTY

---

Cursor Mode

Defines the cursor state across transactions. The cursor refers to the memory work area in which SQL statements are executed. For more information on cursors, refer to the *Oracle RDBMS Database Administrator’s Guide*. This property is useful for applications running against a non-ORACLE data source.

The following settings are valid for the Cursor_Mode property:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open (the default)</td>
<td>Specifies that cursors should remain open across transactions.</td>
</tr>
<tr>
<td>Close</td>
<td>Specifies that cursors should be closed when a commit is issued.</td>
</tr>
</tbody>
</table>

**Applies to:** form

**Set:** Designer, programmatically

**Refer to Built-in:**
- GET_FORM_PROPERTY
- SET_FORM_PROPERTY

**Default:** OPEN_AT_COMMIT

**Usage Notes:**
- Because ORACLE allows the database state to be maintained across transactions, Oracle Forms allows cursors to remain open across COMMIT operations. This reduces overhead for subsequent execution of the same SQL statement because the cursor does not need to be re-opened and the SQL statement does not always need to be re-parsed.
• Some non-ORACLE databases do not allow database state to be maintained across transactions. Therefore, you can specify the CLOSE_AT_COMMIT parameter of the Cursor_Mode option to satisfy those requirements.

• Closing cursors at commit time and re-opening them at execute time can degrade performance in three areas:
  • during the COMMIT operation
  • during future execution of other SQL statements against the same records
  • during execution of queries

• Oracle Forms does not explicitly close cursors following commit processing if you set the property to CLOSE_AT_COMMIT. This setting is primarily a hint to Oracle Forms that the cursor state can be undefined after a commit.

Oracle Forms maintains a transaction ID during all types of transaction processing. For instance, Oracle Forms increments the transaction ID each time it opens a cursor, performs a commit, or performs a rollback.

When Oracle Forms attempts to re-execute a cursor, it checks the transaction ID. If it is not the current transaction ID, then Oracle Forms opens, parses, and executes a new cursor. Only the last transaction ID is maintained.

• If you query, change data, then commit, Oracle Forms increments the transaction ID. Subsequent fetches do not re-open and execute the cursor, for the following reasons:
  • Oracle Forms does not attempt to handle read consistency issues, nor does it handle re-positioning in the cursor.
  • Oracle Forms expects ORACLE or the connect to return an end-of-fetch error when trying to fetch from an implicitly closed cursor.

On a subsequent execution of the query, Oracle Forms opens a new cursor.

• When using this property in conjunction with transactional triggers, you, the designer, must manage your cursors. For example, you might want to close any open queries on the block whenever you perform a commit. For more information about transactional triggers, see Appendix D, “Connecting to Non-ORACLE Data Source” in the Oracle Forms Advanced Techniques Manual.
Cursor_Style

Specifies the mouse cursor style. Use this property to dynamically change the shape of the cursor.

The following settings are valid for the Cursor Style property:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSY</td>
<td>Displays a GUI–specific busy symbol.</td>
</tr>
<tr>
<td>CROSSHAIR</td>
<td>Displays a GUI–specific crosshair symbol.</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>Displays a GUI–specific arrow symbol.</td>
</tr>
<tr>
<td>HELP</td>
<td>Displays a GUI–specific help symbol.</td>
</tr>
<tr>
<td>INSERTION</td>
<td>Displays a GUI–specific insertion symbol.</td>
</tr>
</tbody>
</table>

**Applies to:** application

**Set:** Programmatically

**Refer to Built-in:**
- GET_APPLICATION_PROPERTY
- SET_APPLICATION_PROPERTY

**Default:** Arrow symbol

**Usage Notes:** When Oracle Forms is performing a long operation, it displays the “Working” message and replaces any cursor style specified with the BUSY cursor.

For example, if you set the cursor style to “HELP” and the operator executes a large query, the HELP cursor is replaced by the BUSY cursor while the query is being executed. After Oracle Forms executes the query, the BUSY cursor reverts to the HELP cursor.

Note, however, if you change the cursor style while Oracle Forms is displaying the BUSY cursor, the cursor style changes immediately rather than waiting for Oracle Forms to complete the operation before changing cursor styles.
Data Type

Specifies what kinds of values Oracle Forms allows as input and how Oracle Forms displays those values.

Applies to: check box, display item, list item, radio group, text item, custom item, form parameter

Note: All data types do not apply to each item type.

Set: Designer

Usage Notes:

- In Oracle Forms V4.5, only the standard data types CHAR, DATE, LONG, and NUMBER are recommended. These data types are based on native ORACLE data types, and offer better performance and application portability. The other data types are valid only for text items, and are included primarily for compatibility with previous versions. You can achieve the same formatting characteristics by using a standard data type with an appropriate format mask.

- The data type of a base table item must be compatible with the data type of the corresponding database column. Use the CHAR data type for items that correspond to ORACLE VARCHAR2 database columns.

- Do not create items that correspond to database CHAR columns if those items will be used in queries or as the join condition for a master–detail relation; use VARCHAR2 database columns instead.

- Oracle Forms will perform the following actions on items, as appropriate:
  - remove any trailing blanks
  - change the item to NULL if it consists of all blanks
  - remove leading zeros if the data type is NUMBER, INT, MONEY, RINT, RMONEY, or RNUMBER (unless the item’s format mask permits leading zeros)

- The form parameter Data Type property supports the data types CHAR, DATE, and NUMBER.
**ALPHA**
Contains any combination of letters (upper and/or lower case).

**Default:** Blanks

**Example:** "Employee", "SMITH"

**CHAR**
Supports VARCHAR2 up to 2000 characters. Contains any combination of the following characters:
- Letters (upper and/or lower case)
- Digits
- Blank spaces
- Special characters ($, #, @, and _)

**Default:** Blanks

**Example:** "100 Main Street", "CHAR_EXAMPLE_2"

**DATE**
Contains a valid date. You can display a DATE item in any other valid format by changing the item’s format mask.

**Default:** DD–MON–YY

**Restrictions:**
Refers to a DATE column in the database and is processed as a true date, not a character string.

The DATE data type contains a ZERO time component.

**Example:** 01–JAN–92

**DATETIME**
Contains a valid date and time.

**Default:** DD–MON–YY HH24:MI[:SS]

**Restrictions:**
Refers to a DATE column in the database and is processed as a true date, not a character string.

The DATETIME data type contains a four digit year. If the year input to a DATETIME data type is two digits, the year is interpreted as 00YY.

**Example:** 31–DEC–88 23:59:59
EDATE
Contains a valid European date.

Default: DD/MM/YY
Restrictions: V3 data type.
Must refer to a NUMBER column in the database.
Included for backward compatibility. Instead, follow these recommendations:
- Use the DATE data type.
- Apply a format mask to produce the European date format.
- Reference a DATE column in the database, rather than a NUMBER column.

Example: 23/10/92 (October 23, 1992)
01/06/93 (June 1, 1993)

INT
Contains any integer (signed or unsigned whole number).

Default: 0
Example: 1, 100, −1000

JDATE
Contains a valid Julian date.

Default: MM/DD/YY
Restrictions: V3 data type.
Must refer to a NUMBER column in the database.
Included for backward compatibility. Instead, follow these recommendations:
- Use the DATE data type.
- Apply a format mask to produce the Julian date format.
- Reference a DATE column in the database, rather than a NUMBER column.

Example: 10/23/92 (October 23, 1992)
06/01/93 (June 1, 1993)
LONG
Contains any combination of up to 65,534 characters. Stored in ORACLE as variable–length character strings.

Default: Blanks
Restrictions: Not allowed as a reference in the WHERE or ORDER BY clauses of any SELECT statement. LONG items are not queryable in Enter Query mode.

MONEY
Contains a signed or unsigned number to represent a sum of money.

Restrictions: V3 data type. Included for backward compatibility. Instead, use a format mask with a number to produce the same result.

Example: 10.95, 0.99, –15.47

NUMBER
Contains fixed or floating point numbers, in the range of 1.0x10^{-129} to 9.99x10^{124}, with one or more of the following characteristics:

• signed
• unsigned
• containing a decimal point
• in regular notation
• in scientific notation
• up to 38 digits of precision

NUMBER items refer to NUMBER columns in the database and Oracle Forms processes their values as true numbers (not character strings).

Default: 0
Restrictions: Commas cannot be entered into a number item (e.g., 99,999). Use a format mask instead.

Example: –1, 1, 1.01, 10.001, 1.85E3

RINT
Displays integer values as right–justified.

Restrictions: V3 data type. Included for backward compatibility. Instead, follow these recommendations:

Use the NUMBER data type.
RMONEY
Displays MONEY values as right-justified.

**Restrictions:**
V3 data type.
Included for backward compatibility. Instead, follow these recommendations:
Use the NUMBER data type.
Apply a format mask such as $999.99 to produce a right-justified number.

RNUMBER
Displays NUMBER values as right-justified.

**Restrictions:**
V3 data type.
Included for backward compatibility. Instead, follow these recommendations:
Use the NUMBER data type.
Apply a format mask such as 999.999 to produce a right-justified number.

TIME
Contains numbers and colons that refer to NUMBER columns in the database.

**Default:**
HH24:MI[:SS]

**Restrictions:**
V3 data type.
Included for backward compatibility. Instead, follow these recommendations:
Use the DATETIME data type.
Apply a format mask to produce only the time.
Not allowed as a reference to DATE columns in the database.

**Example:**
:10:23:05
21:07:13

**See also:**
Format Mask property
Data Type (Record Group)

See Column Specification.

Database_Value

For a base table item that is part of a database record whose status is QUERY or UPDATE, Database_Value returns the value that was originally fetched from the database. When a fetched value has been updated and then subsequently committed, Database_Value returns the committed value.

For a control item that is part of a database record, Database_Value returns the value that was originally assigned to the item when the record was fetched from the database.

For any item that is part of a non-database record whose status is NEW or INSERT, Database_Value returns the current value of the item.

Note: You can examine the Database_Value property to determine what the value of an item in a database record was before it was modified by the operator.

Note: You can examine the SYSTEM.RECORD_STATUS system variable or use the GET_RECORD_PROPERTY built-in to determine if a record has been queried from the database.

Applies to: all items except buttons, chart items, and image items

Set: not settable

Refer to Built-in: GET_ITEMPROPERTY
Datasource

Specifies the name of the database currently in use.

Applies to: application
Set: not settable
Refer to Built-in: GET_APPLICATION_PROPERTY
Default: ORACLE
Usage Notes: This property is used in connection with non-Oracle data sources. It returns the name of the database for connections established by Oracle Forms, not for connections established by On-Logon triggers. The following settings are valid for this property:

- ORACLE
- DB2
- NULL (Unspecified database, or not logged on)
- NONSTOP
- TERADATA
- NCR/3600
- NCR/3700
- SQLSERVER

Default Alert Button

Specifies which of three possible alert buttons is to be the default alert button. The default alert button is normally bordered uniquely or highlighted in some specific manner to visually distinguish it from other buttons.

Applies to: alert
Set: Designer
Default: Button 1
Required/Optional: optional
Default Button

Specifies that the button should be identified as the default button. At runtime, the operator can invoke the default button by pressing [Select] if focus is within the block that contains the default button.

On some window managers, the default button is bordered or highlighted in a unique fashion to distinguish it from other buttons in the interface.

**Applies to:** button  
**Set:** Designer  
**Default:** False  
**Required/Optional:** optional

Default Font Scaling

Specifies that the font indicated for use in a form defaults to the relative character scale of the display device in use.

**Applies to:** form module  
**Set:** Designer  
**Default:** True  
**Restrictions:** Valid only when the Coordinate System property is set to Character Cell.  
**See also:** Coordinate System property
### Default Value (Form Parameter)

Specifies the value that Oracle Forms assigns the parameter at form startup.

- **Applies to:** Form Parameter
- **Set:** Designer
- **Default:** NULL
- **Required/Optional:** optional
- **Usage Notes:** Any valid constant is a valid value for this property.
- **See also:**
  - $$DATE$$ system variable
  - $$DATETIME$$ system variable
  - $$TIME$$ system variable

### Default Value (Item)

Specifies the default value that Oracle Forms should assign to the item whenever a record is created. The default value can be one of the following:

- raw value (216, ‘TOKYO’)
- form item (:block_name.item_name)
- global variable (:GLOBAL.my_global)
- form parameter (:PARAMETER.my_param)
- a sequence (:SEQUENCE.my_seq.NEXTVAL)

- **Applies to:** all items except buttons, chart items, and image items
- **Set:** Designer
- **Default:** NULL
- **Required/Optional:** Optional for all items except radio groups, check boxes, and list items. For a radio group, a valid Default Value is required unless a) the radio group accepts Other Values or b) the value associated with one of the radio buttons in the group is NULL.
For a list item, a valid Default Value is required unless a) the list item accepts Other Values or b) the value associated with one of the list elements is NULL.

For a check box, a valid Default Value is required unless a) the check box accepts Other Values or b) the value associated with Checked or Unchecked is NULL.

**Restrictions:**

- The value you specify must be compatible with the data type of the item.
- For a text item, the value cannot be outside the range defined by the Range Low Value and Range High Value properties.
- For a radio group, the default value must be either the name (not the label) of one of the radio buttons, or the value associated with one of the radio buttons. Oracle Forms checks first for a radio button name.
- For a list item, the default value must be either the name of one of the list elements, or the value associated with one of the list elements. Oracle Forms checks first for a list element name.

**Usage Notes:**

If you are using the default value to initialize the state of items such as check boxes, radio groups, or list items, keep in mind that the default value does not get assigned until Oracle Forms creates a record in the block.
Default Value (Menu Substitution Parameter)

Specifies the value that Oracle Forms assigns the parameter at form startup.

Set: Designer

Required/Optional: required

Deferred

See Coordination.

Defer_Required_Enforcement

For an item that has the Required property set to True, it specifies whether Oracle Forms should defer enforcement of the Required item attribute until the record is validated.

By default, when an item has Required set to True, Oracle Forms will not allow navigation out of the item until a valid value is entered. Set Defer_Required_Enforcement to True to allow the operator to move freely among the items in the record, postponing enforcement of the Required attribute until the record is validated.

Applies to: form

Set: programmatically

Default: False

Refer to Built-in:
- GET_FORM_PROPERTY
- SET_FORM_PROPERTY

See also: Required (Item) property
Delete Allowed

Specifies whether records can be deleted from the block.

| Applies to: | block |
| Set:        | Designer, programmatically |
| Default:    | True |
| Refer to Built-in: |
|     • GET_BLOCK_PROPERTY |
|     • SET_BLOCK_PROPERTY |

Detail Block

Specifies the name of the detail block in a master-detail block relation.

| Applies to: | relation |
| Set:        | Designer |
| Refer to Built-in: | GET_RELATION_PROPERTY (Detail_Name) |
| Default:    | NULL |
| Required/Optional: | required |
| Restrictions: | The block specified must exist in the active form. |
Direction

**Note:** This property is specific to bidirectional National Language Support (NLS) applications.

Specifies the layout direction for bidirectional objects.

For the purposes of this property, assume that Local refers to languages displayed Right–To–Left, and Roman refers to languages displayed Left–To–Right.

Direction is an umbrella property that provides as much functionality for each object as possible. For all objects except text items and display items, the Direction property is the only bidirectional property, and its setting controls the other aspects of bidirectional function. (List items, however, have both a Direction property and an Initial Keyboard State property.)

The form–level Direction property is the highest level setting of the property. When you accept the Default setting for the form–level Direction property, the layout direction for the form is inherited from the natural writing direction specified by the NLS language environment variable.

In most cases, leaving all the other Direction properties set to Default will provide the desired functionality—that is, the NLS language environment variable layout direction will ripple down to each subsequent level. You only need to specify the bidirectional properties when you want to override the inherited default values.

This chart summarizes inheritance for the Direction property.

<table>
<thead>
<tr>
<th>Default Setting Derives Value From This Object</th>
<th>From This Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>NLS environment variable</td>
</tr>
<tr>
<td>All objects, such as Alert, Block, LOV, Window, and Canvas–view</td>
<td>Form</td>
</tr>
<tr>
<td>All items, such as Text Item, Display Item, Check Box, Button, Radio Group, and List Item</td>
<td>Canvas–view</td>
</tr>
</tbody>
</table>

Properties 5 – 67
This table summarizes the functions controlled by the Direction property for each object type. (Text items and display items do not have a Direction property; instead, in the Designer, you can specifically set Alignment, Reading Order, and Initial Keyboard State properties for these items. However, programmatically, you can get and set the Direction property only for all items, including text items and display items.)

<table>
<thead>
<tr>
<th>Layout Direction</th>
<th>Text Reading Order</th>
<th>Text Alignment</th>
<th>Scrollbar Position</th>
<th>Initial Keyboard State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alert</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block (for future use)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOV (for future use)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Window (of menu)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Canvas–view (also point of origin)</td>
<td>X</td>
<td>X (boilerplate text)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Check Box</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Button</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radio Group</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>List Item</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Note: The headings listed above represent functions, not properties: for example, the Direction property for alerts does not set the Initial Keyboard State property, it controls the initial keyboard state function.
The allowable values for this property are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Direction based on the property shown in the table.</td>
</tr>
<tr>
<td>Right–To–Left</td>
<td>Direction is right–to–left.</td>
</tr>
<tr>
<td>Left–To–Right</td>
<td>Direction is left–to–right.</td>
</tr>
</tbody>
</table>

**Applies to:** all objects listed in the table  
**Set:** Designer, programmatically  
**Refer to Built-in**  
- GET_FORM_PROPERTY  
- GET_WINDOW_PROPERTY  
- GET_VIEW_PROPERTY  
- GET_ITEM_PROPERTY  
- SET_FORM_PROPERTY  
- SET_WINDOW_PROPERTY  
- SET_VIEW_PROPERTY  
- SET_ITEM_PROPERTY

**General Usage Notes:**  
- If you want all items on your form to default to the natural writing direction specified by the language environment variable, set Direction at the Form level to Default, and allow all other Direction properties to be Default, as well.  
- In most cases, the Default setting will provide the functionality you need. Occasionally, however, you may want to override the default by setting the Direction property for a specific object that needs to be displayed differently from the higher-level Direction property. For example, you may want to have most items on a canvas inherit their Direction from the canvas-view Direction property, but in the case of a specific text item, you might set the Direction property to override the default.  
- If you are developing a bilingual application and need to display both Local and Roman menus, create a trigger to display the correct version of the menu based on the USER_NLS_LANG property of the GET_APPLICATION_PROPERTY built-in.
• Follow these guidelines when choosing a Direction property value:
  • If you are developing a bilingual application and want to display a Local object in Right–To–Left mode and a Roman object in Left–To–Right, use the Default value.
  • If the object is normally composed of Local text, choose the Right–To–Left value.
  • If the object is normally composed of Roman text, choose the Left–To–Right value.

**Direction (Alert)**
Specifies the layout direction of the alert interface items, including the reading order of the text displayed within the alert window.

**Direction (Button)**
Specifies the reading order of button text and the initial keyboard state when the button receives input focus.

**Direction (Canvas–view)**
Specifies the layout direction of the canvas–view, including:
  • layout direction used in the Layout Editor
  • point of origin (for Right–to–Left, point of origin is top right corner; for Left–to–Right, point of origin is top left corner)
  • display of rulers and scrollbars
  • reading order of boilerplate text

**Canvas–view Usage Notes:**
• Refer to the Usage Notes for the form–level Direction property to determine which value to choose.
• To develop an application with blocks laid out in different directions, place each block on a different canvas–view. This will provide:
  • automatic layout of blocks in the canvas–view Direction property
  • boilerplate text reading order will default to the canvas–view Direction property
• If a block spans multiple canvas–views, keep the canvas–view Direction property the same for all canvas–views, unless you intend to have part of the block displayed with a different Direction.
• In the Designer, if you change the canvas–view Direction property while the Layout Editor is open, the change will not take place until you reopen the Layout Editor.
**Direction (Check Box)**
Specifies the layout direction of a check box, including:
- the position of the box relative to the text
- reading order of check box label
- initial keyboard state when the check box receives input focus

**Direction (Form)**
Specifies the layout direction of a form. Setting the form-level Direction property to Default lets the form inherit layout direction from the natural writing direction of the language specified in the NLS environment variable.

**Form Usage Notes:**
- If you are developing a bilingual application that must run in both Right–To–Left and Left–To–Right directions, use the Default value.
- During testing, set Direction to either Right–To–Left or Left–To–Right, to test your form in Local or Roman direction. Before generating the final executable, return the setting to Default.
- If your application must run in one direction only, choose the corresponding value.

**Direction (List Item)**
Specifies the layout direction of the list items in both popup lists and combo boxes, including:
- position of the scroll bar
- alignment of list text
- reading order of list text
- initial keyboard state when the list item gains input focus

**Direction (Radio Group)**
Specifies layout direction of the radio buttons of a group (position of the circle relative to the text), including:
- reading order of text
- initial keyboard state when the radio group gains input focus

**Direction (Windows)**
Specifies layout direction of the window object, including:
- layout direction of the menu
- reading order of any text displayed within the window area that is not part of an object that has its own Direction property (for example, the window title)
Directory

Specifies the directory in which Oracle Forms should look for the .MMX runtime menu file. This property is applicable only when you want Oracle Forms to locate the menu .MMX runfile through database lookup, rather than direct reference.

When using database lookup, the menu module must be stored in the database. At runtime, Oracle Forms queries the menu module definition stored in the database to find out the directory and filename of the menu .MMX runfile. The Directory and File menu module properties specify the path where Oracle Forms should look for the .MMX menu file.

Applies to: menu module
Set: Designer
Default: blank
Required/Optional: optional
Restrictions: Not valid when using direct reference to specify the location of the menu .MMX runfile. You use direct reference when you attach a menu to a form by setting the Use File form module property to True.
Usage Notes: If you leave the directory path unspecified, Oracle Forms first searches the default directory for the file, then searches any predefined search paths. For more information on search paths, refer to the Oracle Forms documentation for your platform.
See also: File property
Use File property

Display Width (LOV)

See Column Mapping.
**Display w/o Privilege**

Determines whether the current menu item is displayed when the current form operator is not a member of a security role that has access privileges to the item:

- When Display without Privilege is False, Oracle Forms does not display the item if the operator does not have access to it.
- When Display without Privilege is True, Oracle Forms displays the item as a disabled (grayed) menu item. The operator can see the item on the menu, but cannot execute the command associated with the item.

You can only grant access to members of those roles displayed in the roles list. To add a database role to this list, set the menu module property, Menu Module Roles. For more information on establishing the roles list and assigning a role access to menu items, see the Oracle Forms Developer's Guide, Chapter 23, “Customizing Menus.”

**Applies to:** menu item  
**Set:** Designer  
**Default:** False  
**Restrictions:** Valid only when the name of at least one database role has been specified in the roles list.  
**See also:** Menu Item Roles property  
Menu Module Roles property

**Display X Position, Display Y Position**

Specifies the x,y coordinates for the stacked canvas–view’s upper left corner relative to the upper left corner of the window’s current content view.

**Applies to:** canvas–view  
**Set:** Designer, programmatically  
**Refer to Built-in:**  
- GET_VIEW_PROPERTY  
- SET_VIEW_PROPERTY
**Displayed (Item)**

Determines whether an item that is assigned to a canvas is shown or hidden at runtime.

- ** Applies to:** all items
- ** Set:** Designer, programmatically
- ** Refer to Built-in:**
  - GET_ITEM_PROPERTY
  - SET_ITEM_PROPERTY
  - GET_RADIO_BUTTONPROPERTY
  - SET_RADIO_BUTTONPROPERTY
- ** Default:** True
- ** Restrictions:** When the item is part of the foreign key in a default master-detail relation, the default is False.

**Displayed (Canvas-view)**

Determines whether a stacked canvas-view is initially shown or hidden in the window to which it is assigned.

- ** Applies to:** stacked canvas-view
- ** Set:** Designer, programmatically
- ** Refer to Built-in:**
  - GET_VIEW_PROPERTY (VISIBLE)
  - SET_VIEW_PROPERTY (VISIBLE)
Default: True

Restrictions: • Valid only for a stacked canvas-view (Canvas-view Type property set to Stacked).
• A displayed view may not be visible if it is behind the content view or another stacked view assigned to the same window.

See also: Canvas-view Type property

**Displayed (Menu Item)**

Determines whether the menu item is shown or hidden at runtime.

Applies to: menu item

Set: programmatically

Refer to Built-in: • GET_MENU_ITEMPROPERTY
• SET_MENU_ITEMPROPERTY

Default: True

**Display_Height**

Specifies the height of the display device, in the units specified by the current setting of the Coordinate Units form property. Use this property to dynamically calculate the optimum display position for windows on the screen.

Applies to: application

Set: not settable

Refer to Built-in: GET_APPLICATIONPROPERTY
Display_Width

Specifies the width of the display device, in the units specified by the current setting of the Coordinate Units form property. Use this property to dynamically calculate the optimum display position for windows on the screen.

Applies to: application

Set: not settable

Refer to Built-in: GET_APPLICATIONPROPERTY

Editor

Specifies that one of the following editors should be used as the default editor for this text item:

- a user–named editor that you defined in the form or
- a system editor outside of Oracle Forms that you specified by setting the SYSTEM_EDITOR environment variable

Applies to: text item

Set: Designer

Refer to Built-in: GET_ITEMPROPERTY

Default: blank, indicating the default Oracle Forms editor

Required/Optional: optional

Restrictions: The editor specified must exist in the active form.

Usage Notes: To specify a system editor:

- Define the system editor by setting the SYSTEM_EDITOR environment variable.
- Enter the value SYSTEM_EDITOR in the Editor Name field.
Editor X Position, Editor Y Position

Specifies the horizontal (x) and vertical (y) coordinates of the upper left corner of the editor relative to the upper left corner of the window's content canvas. When you set the Editor property, you can set the Editor position properties to override the default display coordinates specified for the editor.

Applies to: text item

Set: Designer

Refer to Built-in: GET_ITEM_PROPERTY

Default: 0, 0; indicating that Oracle Forms should use the default editor display coordinates, as specified by the editor Position property.

Required/Optional: optional

Enabled (Item)

Determines whether operators can use the mouse to manipulate an item.

On most window managers, Enabled set to False grays out the item.

Applies to: all items except buttons, chart items, and display items

Set: Designer, programmatically

Refer to Built-in:
- GET_ITEMPROPERTY
- SET_ITEMPROPERTY
- GET_RADIO_BUTTONPROPERTY
- SET_RADIO_BUTTONPROPERTY

Default: True

Usage Notes:
- When Enabled is set to True, Navigable can be set to True or False. When Enabled is False, an item is always non-Navigable.
- Enabled set to False grays out the item. If you want to have the item appear normally, so the user can inspect it without being able to change it, set the following properties, instead:
Enabled (Menu Item)

Specifies whether the menu item should be displayed as an enabled (normal) item or disabled (grayed) item.

Applies to: menu item

Set: programmatically

Refer to Built-in:
- GET_MENU_ITEM_PROPERTY
- SET_MENU_ITEM_PROPERTY

Default: True

Restrictions: You cannot programmatically enable or disable a menu item that is hidden as a result of the following conditions:
- The menu module Use Security property is True.
- The menu item Display w/o Privilege property is set to False.
- The current operator is not a member of a role that has access to the menu item.

See also: Use Security property
Display w/o Privilege built-in

• Insert Allowed (Item) to False
• Update Allowed (Item) to False
• Enabled to True
## Enterable

Specifies whether the block is enterable. A block is enterable when any item in the block has its Enabled and Navigable properties set to True.

- **Applies to:** block
- **Set:** not settable
- **Refer to Built-in:** GET_BLOCK_PROPERTY
- **See also:** Enabled property, Navigable property

## Execution Style

Specifies how the current trigger code should execute if there is a trigger with the same name defined at a higher level in the object hierarchy.

The following settings are valid for this property:

- **Override**
  - Specifies that the current trigger fire *instead* of any trigger by the same name at any higher scope. This is known as “override parent” behavior.

- **Before**
  - Specifies that the current trigger fire *before* firing the same trigger at the next–higher scope. This is known as “fire before parent” behavior.

- **After**
  - Specifies that the current trigger fire *after* firing the same trigger at the next–higher scope. This is known as “fire after parent” behavior.

- **Applies to:** trigger
- **Set:** Designer
- **Default:** Override
File

Specifies the name of the .MMX runtime menu file that Oracle Forms should look for at form startup. This property is applicable only when you want Oracle Forms to locate the menu runfile through database lookup, rather than direct reference.

To use database lookup, the menu module must be stored in the database. At runtime, Oracle Forms queries the menu module definition stored in the database to find out the directory and filename of the menu .MMX runfile. The Directory and File menu module properties specify the path where Oracle Forms should look for the .MMX menu file.

Applies to: menu module

Set: Designer

Default: Module Name property

Required/Optional: optional

Restrictions: • Not valid when using direct reference to specify the location of the menu .MMX runfile. You use direct reference when you attach a menu to a form by setting the Use File form module property to True.

• The .MMX file extension is not required.

Usage Notes: If you leave the directory unspecified, Oracle Forms first searches the default directory for the file, then searches any predefined search paths. For more information on search paths, refer to the Oracle Forms documentation for your platform.

See also: Directory property
Use File property
File_Name

Specifies the name of the file where the named form is stored.

Applies to: form
Set: not settable
Refer to Built-in: GET_FORMPROPERTY
Required/Optional: optional
Restrictions: If two or more forms share the same name, File_Name supplies the name of the file where the most recently-accessed form is stored.
Usage Notes: File_Name at the form level corresponds to Current_Form at the application level. Current_Form is gettable with GET_APPLICATIONPROPERTY.
See also: Current_Form property
Fire in Enter Query Mode

Specifies that the trigger should fire when the form is in Enter Query mode, as well as in Normal mode.

**Applies to:** trigger

**Set:** Designer

**Default:** no

**Usage Notes:** Only applicable to the following triggers:
- Key
- On–Error
- On–Message
- When– triggers, except:
  - When–Database–Record
  - When–Image–Activated
  - When–New–Block–Instance
  - When–New–Form–Instance
  - When–Create–Record
  - When–Remove–Record
  - When–Validate–Record
  - When–Validate–Item

**See also:** SYSTEM.MODE system variable
First Navigation Block

Specifies the name of the block to which Oracle Forms should navigate at form startup and after a CLEAR_FORM operation. By default, the First_Navigation_Block is the first block in the form’s commit sequence, as indicated by the sequence of blocks in the Object Navigator. You can set the First_Navigation_Block property programmatically to specify a different block as the first navigation block.

**Applies to:** form module

**Set:** Designer, programmatic

**Refer to Built–in:**
- GET_FORM_PROPERTY
- SET_FORM_PROPERTY

**Default:** The first block in the form; that is, the block that is listed first in the Object Navigator.

**Required/Optional:** optional

**Usage Notes:** You can set this property from a When–New–Form–Instance trigger, which fires at form startup, before Oracle Forms navigates internally to the first block in the form.

**See also:** When–New–Form–Instance trigger

First_Block

Specifies the block that is the first block in the form, as indicated by the sequence of blocks in the Object Navigator. At startup, Oracle Forms navigates to the first item in the first block.

**Applies to:** form

**Set:** not settable

**Refer to Built–in:** GET_FORM_PROPERTY
First_Detail_Relation

Specifies the name of the first master-detail block relation in which the given block is the detail block.

**Applies to:** block

**Set:** not settable

**Refer to Built-in:** GET_BLOCK_PROPERTY

**Usage Notes:** This property is useful when you are writing your own master-detail coordination scheme. It can be used in conjunction with the Next_Master_Relation and Next_Detail_Relation properties to traverse a list of relations.

**See also:** First_Master_Relation property
Next_Detail_Relation property
Next_Master_Relation property

First_Item

Specifies the item that is the first item in the block, as indicated by the sequence of items in the Object Navigator. At startup, Oracle Forms navigates to the first item in the first block.

**Applies to:** block

**Set:** not settable

**Refer to Built-in:** GET_BLOCK_PROPERTY

**See also:** Last_Item property
First_Master_Relation

Specifies the name of the first master-detail block relation in which the given block is the master block.

Applies to: block

Set: not settable

Refer to Built-in: GET_BLOCKPROPERTY

Usage Notes: This property is useful when you are writing your own master-detail coordination scheme. It can be used in conjunction with the Next_Master_Relation and Next_Detail_Relation properties to traverse a list of relations.

See also: First_Detail_Relation property

Next_Detail_Relation property

Next_Master_Relation property
**Fixed Length(Item)**

When set to True, Fixed Length specifies that the item should be considered valid only when it contains the maximum number of characters allowed. The maximum number of characters allowed is determined by the Maximum Length property setting.

- **Applies to:** text item
- **Set:** Designer, programmatically
- **Refer to Built-in:**
  - GET_ITEM_PROPERTY
  - SET_ITEM_PROPERTY
- **Default:** False
- **Restrictions:**
  - When the Required property is False, Oracle Forms accepts NULL values as valid, even if the Fixed Length property is True. If Required is True, NULL values are not accepted as valid.
  - The Displayed and Enabled properties must be set to True.
  - A text item value of the NUMBER data type cannot contain leading zeroes. Oracle Forms automatically removes leading zeroes and interprets the text item as “not filled.”

**Fixed Length (Menu Substitution Parameter)**

When set to True, Fixed Length specifies that the parameter should be considered valid only when it contains the maximum number of characters allowed. The maximum number of characters allowed is determined by the Maximum Length property setting.

- **Applies to:** menu substitution parameter
- **Set:** Designer
- **Default:** False
- **See also:** Maximum Length property
Fixed Size

Specifies that the window is to be a fixed size and cannot be resized at runtime. This property is a GUI hint, and may not be supported on all platforms.

Applies to: window
Set: Designer
Default: False

Restrictions:
- Cannot be set for a root window: a root window can always be resized at runtime.
- Fixed Size is only valid when the Zoomable property is set to False

Usage Notes: The Fixed Size property prevents an operator from resizing the window, but it does not prevent you from resizing the window programmatically with RESIZE_WINDOW or SET_WINDOW_PROPERTY.

See also: Resize_Window property
SET_WINDOW_PROPERTY built-in
Zoomable property
Format Mask

Specifies the display format and input accepted for data in text items.

Applies to:  text item
Set:  Designer
Refer to Built-in:  • GET_ITEM_PROPERTY
• SET_ITEM_PROPERTY
Required/Optional:  optional
Restrictions:  • Valid only for single-line text items (Multi-line property set to False).
• When setting the Maximum Length property for a text item, include space for any embedded characters inserted by the format mask you specify.
• Format masks can contain a maximum of 30 characters.
• Oracle Forms supports only ORACLE format masks that are used for both input and output. Input-only format masks, such as WW, are not supported.
Usage Notes:  Valid format masks for character strings, numbers and dates are described in the following tables.

Character Strings

The following table describes valid format masks for character strings.

<table>
<thead>
<tr>
<th>Element</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FM</td>
<td>FMXX99</td>
<td>Fill mode: accept string as typed, do not right justify. Allows operator input string to be shorter than the format mask.</td>
</tr>
<tr>
<td>X</td>
<td>XXXX</td>
<td>Any alphabetic, numeric, or special character. Operator input string must be exact length specified by format mask.</td>
</tr>
<tr>
<td>9</td>
<td>9999</td>
<td>Numeric characters only. Operator input string must be exact length specified by format mask.</td>
</tr>
<tr>
<td>A</td>
<td>AAAAA</td>
<td>Alphabetic characters only. Operator input string must be exact length specified by format mask.</td>
</tr>
</tbody>
</table>
Example:

<table>
<thead>
<tr>
<th>Format Mask</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXAA</td>
<td>Will accept: —ab, abcd, 11ab; will not accept: —11, ab11, or ab— (must use XX to accept hyphens and other special characters).</td>
</tr>
<tr>
<td>XXXX</td>
<td>Will accept any combination of alphabetic, numeric, or special characters: —ab, abcd, 11ab, —11, ab11, or ab—. Will accept 1234 or abcd; will not accept 123 or abc. (To accept input string shorter than mask, use FMXXXX.)</td>
</tr>
<tr>
<td>FMXX99</td>
<td>Will accept ab12, ab1, ab followed by two spaces; will not accept 12ab or abcd. (To produce the Oracle Forms Version 3.0 Alpha datatype, use FMAAAAAA.)</td>
</tr>
</tbody>
</table>

- To embed additional characters such as a hyphen (–) or a comma (,), surround the character with double-quotes (").
- Embedded characters are separate from text item values and are not collated along with text item values, even when the operator enters them.

Numbers

The following table describes valid format masks for numbers.

<table>
<thead>
<tr>
<th>Element</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>9999</td>
<td>Number of nines determines display width.</td>
</tr>
<tr>
<td>0</td>
<td>0999</td>
<td>Display leading zeros.</td>
</tr>
<tr>
<td>0</td>
<td>9990</td>
<td>Display zero value as zero, not blank.</td>
</tr>
<tr>
<td>$</td>
<td>$9999</td>
<td>Prefix value with dollar sign.</td>
</tr>
<tr>
<td>B</td>
<td>B9999</td>
<td>Display zero value as blank, not &quot;0&quot;.</td>
</tr>
<tr>
<td>MI</td>
<td>9999MI</td>
<td>Display “—” after a negative value.</td>
</tr>
<tr>
<td>PR</td>
<td>9999PR</td>
<td>Display a negative value in &lt;angle brackets&gt;.</td>
</tr>
<tr>
<td>comma</td>
<td>9,999</td>
<td>Display a comma in this position. For correct behavior in multilingual applications, substitute G to return the appropriate group (thousands) separator.</td>
</tr>
<tr>
<td>period</td>
<td>99.99</td>
<td>Display a decimal point in this position. For correct behavior in multilingual applications, substitute D to return the appropriate decimal separator.</td>
</tr>
<tr>
<td>E</td>
<td>9.999EEE</td>
<td>Display in scientific notation (format must contain exactly four “E”s).</td>
</tr>
<tr>
<td>FM</td>
<td>FM999</td>
<td>Fill mode: accept string as typed, do not right justify.</td>
</tr>
</tbody>
</table>
• When you mask a number with nines (9), Oracle Forms adds a space in front of the number to accommodate the plus (+) or minus (–) sign. However, since the plus sign is not displayed, it appears as if Oracle Forms adds a space in front of the number. (The minus sign is displayed.)

• To embed additional characters such as a hyphen (–) or a comma (,), surround the character with double–quotes (”).

• Embedded characters are separate from text item values and are not collated along with text item values, even when the operator enters them.

Example:

<table>
<thead>
<tr>
<th>Format Mask</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FM099&quot;–&quot;99&quot;–&quot;9999</td>
<td>Displays the social security number as formatted, including hyphens, even if operator enters only nine digits. To create a Social Security column, create an 11–character column, set to fixed length, with a format mask of 099”–”99”–”9999. This mask will accommodate Social Security numbers that begin with zero, accepting 012–34–5678 or 012345678 (both stored as 012345678).</td>
</tr>
<tr>
<td>99999PR</td>
<td>Accepts –123; reformats as &lt;123&gt;.</td>
</tr>
<tr>
<td>999MI</td>
<td>Accepts –678; reformats as 678–.</td>
</tr>
<tr>
<td>9.999EEEE</td>
<td>Displays as 1.00E+20.</td>
</tr>
</tbody>
</table>
Dates

The following table describes valid format masks for dates.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYYY or SYYYY</td>
<td>4-digit year; “S” prefixes “BC” date with “–”.</td>
</tr>
<tr>
<td>YYY or YY or Y</td>
<td>Last 3, 2, or 1 digits of year.</td>
</tr>
<tr>
<td>Y,YYY</td>
<td>Year with comma in this position.</td>
</tr>
<tr>
<td>BC or AD</td>
<td>BC/AD indicator.</td>
</tr>
<tr>
<td>B.C. or A.D.</td>
<td>BD/AD indicator with periods.</td>
</tr>
<tr>
<td>RR</td>
<td>Defaults to correct century.</td>
</tr>
<tr>
<td>MM</td>
<td>Month (01–12; JAN = 01).</td>
</tr>
<tr>
<td>MONTH</td>
<td>Name of month, padded with blanks to length of 9 characters.</td>
</tr>
<tr>
<td>MON</td>
<td>Name of month, 3-letter abbreviation.</td>
</tr>
<tr>
<td>DDD</td>
<td>Day of year (1–366).</td>
</tr>
<tr>
<td>DD</td>
<td>Day of month (1–31).</td>
</tr>
<tr>
<td>D</td>
<td>Day of week (1–7; Sunday=1).</td>
</tr>
<tr>
<td>DAY</td>
<td>Name of day, padded with blanks to length of 9 characters.</td>
</tr>
<tr>
<td>DY</td>
<td>Name of day, 3-letter abbreviation.</td>
</tr>
<tr>
<td>J</td>
<td>Julian day; the number of days since January 1, 4712 BC.</td>
</tr>
<tr>
<td>AM or PM</td>
<td>Meridian indicator.</td>
</tr>
<tr>
<td>A.M. or P.M.</td>
<td>Meridian indicator with periods.</td>
</tr>
<tr>
<td>HH or HH12</td>
<td>Hour of day (1–12).</td>
</tr>
<tr>
<td>HH24</td>
<td>Hour of day (0–23).</td>
</tr>
<tr>
<td>MI</td>
<td>Minute (0–59).</td>
</tr>
<tr>
<td>SS</td>
<td>Second (0–59).</td>
</tr>
<tr>
<td>SSSSS</td>
<td>Seconds past midnight (0–86399).</td>
</tr>
<tr>
<td>. , .</td>
<td>Punctuation is reproduced in the result.</td>
</tr>
<tr>
<td>...</td>
<td>Quoted string is reproduced in the result.</td>
</tr>
<tr>
<td>FM</td>
<td>Fill mode: assumes implied characters such as O or space; displays significant characters left justified. Allows operator input to be shorter than the format mask. (Use in conjunction with FX to require specific delimiters.)</td>
</tr>
<tr>
<td>FX</td>
<td>All date literals must match the format mask exactly, including delimiters.</td>
</tr>
</tbody>
</table>
• When you prefix a date mask with FX, the operator must enter the date exactly as you define the mask, including the specified delimiters:

Example:

<table>
<thead>
<tr>
<th>Format Mask</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FXDD–MON–YY</td>
<td>Will accept 12–JAN–94 but will not accept 12JAN94 or 12JAN94, because the delimiters do not match the mask. Will accept 01–JAN–94 but will not accept 1–JAN–94.</td>
</tr>
<tr>
<td>FMFXDD–MON–Y</td>
<td>FM: will accept 1–JAN–94, converting it to 01–JAN–94. FX: will accept only hyphen delimiters ((-)).</td>
</tr>
<tr>
<td>DD–MON–YY</td>
<td>Will accept 12JAN94, but will not accept 12–JAN–94. Note: Any delimiter characters will be accepted, but if delimiters are omitted by the operator, this mask will interpret date characters as a delimiters. Will accept 12–JAN–94, (but will erroneously interpret as 12–JAN–04); but will not accept 12JAN94, because &quot;AN&quot; is not a valid month name.</td>
</tr>
</tbody>
</table>

• Use of a format mask only affects how the data looks. Oracle Forms stores full precision, regardless of how the data is presented.

• Embedded characters are separate from text item values and are not collated along with text item values, even when the operator enters them.

• To embed additional characters such as a hyphen (\(\-\)) or a comma (\(,\)), surround the character with double quotes (\"\).
When you use day of the week formats, be sure that the data includes day of the week information. To avoid illogical masks, display also either the day of the year (1–366) or the month in some format.

Example:

<table>
<thead>
<tr>
<th>Format Mask</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DY–DD–MON–YY</td>
<td>Displays as WED–12–JAN–94. Be sure to include month. Avoid masks such as DY–DD–YY, which could generate an error.</td>
</tr>
</tbody>
</table>

NLS Format Masks

The following table describes valid National Language Support (NLS) format masks. For more information, see Appendix B, “Native Language Support” in the Oracle Forms Advanced Techniques Manual.

<table>
<thead>
<tr>
<th>Element</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>C999</td>
<td>Returns the international currency symbol.</td>
</tr>
<tr>
<td>L</td>
<td>L9999</td>
<td>Returns the local currency symbol.</td>
</tr>
<tr>
<td>D</td>
<td>99D99</td>
<td>Returns the decimal separator.</td>
</tr>
<tr>
<td>G</td>
<td>9G999</td>
<td>Returns the group (thousands) separator.</td>
</tr>
<tr>
<td></td>
<td>comma</td>
<td>Displays a comma in this position.</td>
</tr>
<tr>
<td></td>
<td>period</td>
<td>Displays a decimal point in this position.</td>
</tr>
</tbody>
</table>

Example:

<table>
<thead>
<tr>
<th>Format Mask</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L99G999D99</td>
<td>Displays the local currency symbol, group, and decimal separators: if NLS_LANG=American, this item displays as $1,600.00; if NLS_LANG=Norwegian, this item displays as Kr.1,600,00.</td>
</tr>
<tr>
<td>C99G999D99</td>
<td>Displays the appropriate international currency symbol: if NLS_LANG=American, this item displays as USD1,600.00; if NLS_LANG=French, this item displays as FRF1,600.00.</td>
</tr>
</tbody>
</table>
Form_Name

Specifies the name of the form.

Applies to: form
Set: not settable
Refer to Built-in: GET_FORM_PROPERTY
Usage Notes: Form_Name at the form level corresponds to Current_Form_Name at the application level. Current_Form_Name is gettable with GET_APPLICATION_PROPERTY.
See also: Current_Form_Name property

Group_Name

Specifies the name of the record group on which an LOV is based.

Applies to: LOV
Set: programmatically
Refer to Built-in:
  • GET_LOV_PROPERTY
  • SET_LOV_PROPERTY
Default: Name of the underlying record group.
Usage Notes: Set Group_Name to replace the LOV’s current record group with another record group at runtime. The column names and types in the new record group must match the column names and types in the record group you are replacing.
Help

On character mode platform specifies help text for the menu item. Help text is displayed in a window when the operator presses [Help] while the menu item is selected.

**Applies to:** menu item

**Set:** Designer

**Required/Optional:** optional

**Restrictions:** Applies to character mode applications only.

**See also:** Hint (Menu item) property

---

Hint (Item)

Specifies item-specific help text that can be displayed on the message line of the root window at runtime. Hint text is available when the input focus is in the item.

**Applies to:** all items except chart items, display items, and custom items

**Set:** Designer

**Refer to Built-in:** • GET_ITEM_PROPERTY (HINT_TEXT)

**Default:** For an item that was created through the New Block window, “Enter value for: <item name>”. For all other items, NULL.

**Required/Optional:** optional

**Usage Notes:** Leave the Hint property NULL if you do not want the item to have hint text.
Hint (Menu Item)

For a character mode application, specifies hint text for a menu item. In pull-down and bar menu display styles, hint text is displayed on the message line when the input focus is in the menu item.

In full-screen display style, hint text, if specified, is displayed as the item descriptor, and the menu item name is ignored. (When no hint text is specified, Oracle Forms displays the item name as the descriptor.)

Applies to: menu item
Set: Designer
Required/Optional: optional
Restrictions:
- Maximum of 70 characters.
- Applies to character mode applications only.
See also: Help property

Hint (Menu Substitution Parameter)

Specifies a description or instruction to appear on the message line when the operator enters a value for the menu substitution parameter.

Applies to: menu substitution parameter
Set: Designer
Required/Optional: optional
Horizontal MDI Toolbar

On Microsoft Windows, specifies the canvas-view that should be displayed as a horizontal toolbar on the MDI application window. The canvas-view specified must have the Canvas-view Type property set to Horizontal Toolbar.

- **Applies to:** form
- **Set:** Designer
- **Default:** Null
- **Required/Optional:** optional
- **Restrictions:** Valid only on Microsoft Windows. On other platforms, the Horizontal MDI Toolbar property is ignored and the canvas-view is mapped to the window indicated by its Window property setting.
- **See also:** Canvas-view Type property
  Vertical MDI Toolbar property

Horizontal Scroll Bar

Determines whether a secondary window or image item is displayed with a scroll bar.

- **Applies to:** window, editor, image item
- **Set:** Designer
- **Default:** False
- **Required/Optional:** optional
- **Restrictions:**
  - Valid on window managers that support horizontal scroll bars.
  - For a window, only valid when the Modal property set to False
- **See also:** Modal property
  Vertical Scroll Bar property
**Horizontal Toolbar**

Specifies the canvas–view that should be displayed as a horizontal toolbar on the window. The canvas–view specified must be a horizontal toolbar canvas–view (Canvas–view Type property set to Horizontal Toolbar) and must be assigned to the current window by setting the Window property.

**Applies to:** window

**Set:** Designer

**Default:** Null

**Required/Optional:** required if you are creating a horizontal toolbar

**Usage Notes:**
- In the Properties window, the poplist for this property shows only canvas–views that have the Canvas–view Type property set to Horizontal Toolbar.
- At runtime, Oracle Forms attempts to display the specified horizontal toolbar on the window. However, if more than one toolbar of the same type has been assigned to the same window (by setting the canvas–view Window property to point to the specified window), Oracle Forms may display a different toolbar in response to navigation events or programmatic control.
- On Microsoft Windows, the specified horizontal toolbar canvas–view will not be displayed on the window if you have specified that it should be displayed on the MDI application window by setting the Horizontal MDI Toolbar form property.

**See also:** Canvas–view Type property

Vertical Toolbar property
**Icon Name**

Specifies the name of the icon resource that you want to represent the iconic button, menu item, or window.

**Applies to:** button, menu item, window

**Set:** Designer, programmatically

**Refer to Built-in:**
- GET_ITEM_PROPERTY
- SET_ITEM_PROPERTY

**Default:** NULL

**Required/Optional:** optional

**Restrictions:**
- For buttons, it is only applicable when the Iconic property is set to True, indicating that the button is an iconic button, rather than a text button.
- For menu items, it is not valid for menu running in full-screen display style.
- For a window, it is only valid when Iconifiable property set to True.
- Icon resources must exist in the runtime operating system, and are not incorporated in the form definition. For this reason, icon resource files are not portable across platforms.

**Usage Notes:**
When defining the Iconic property, do not include the icon file extension. For example, enter `MY_ICON`, not `MY_ICON.ICO`.

Use the platform-specific environment variable to indicate the directory where icon resources are located. For example, the Microsoft Windows name for this variable is `TK21_ICON`.

For more information on this variable name, refer to the Oracle Forms documentation for your operating system.
Icon Title

Specifies the text string that should appear below an iconified window.

- **Applies to:** window
- **Set:** Designer
- **Default:** False
- **Required/Optional:** optional
- **Restrictions:** Only applicable when the Iconifiable property is set to True.
- **See also:** Iconifiable property

Iconic

Specifies that a button is to be an iconic button.

- **Applies to:** button
- **Set:** Designer
- **Refer to Built-in:** GET_ITEM_PROPERTY
- **Default:** False
- **Required/Optional:** optional
- **Restrictions:** A valid icon resource file name must be supplied.
- **Usage Notes:** When Iconic is True, the button’s Icon Name property specifies the icon resource that Oracle Forms should display for the button.
Iconifiable

Specifies that a window can be iconified on window managers that support this feature.

**Applies to:** window

**Set:** Designer

**Default:** True

**Required/Optional:** optional

**Restrictions:** Cannot be set for a root window: a root window is always iconifiable.

**See also:** Icon Name property

Identification

For applications running on character mode in the pull-down and bar menu display styles, this property specifies the string that displays on the message line when the operator navigates to the menu. In full-screen display style, this property specifies the string that identifies the menu module.

**Applies to:** menu module

**Set:** Designer

**Default:** The default module name

**Required/Optional:** optional

**Restrictions:**
- Up to a 40-character description only.
- Applicable for character mode applications only.
In Menu/Block Description

Specifies whether the block should be listed in the block menu and, if so, the description that should be used for the block.

Oracle Forms has a built-in block menu that allows operators to invoke a list of blocks in the current form by pressing [Block Menu]. When the operator selects a block from the list, Oracle Forms navigates to the first enterable item in the block.

**Applies to:** block

**Set:** Designer

**Default:** True. Block Description: For a new block, NULL; For an existing block, the block name at the time the block was created.

**Required/Optional:** optional

**Restrictions:** The block does not appear in the Block Menu if you set the In Menu property to True but leave the Block Description property blank.

Inherit Menu

Specifies whether the window should display the current form menu on window managers that support this feature.

**Applies to:** window

**Set:** Designer

**Default:** True

**Required/Optional:** optional

**Restrictions:**
- Cannot be set for a root window: a root window always displays the current form menu.
- Not valid on Microsoft Windows.
Initial Keyboard State

**Note:** This property is specific to bidirectional National Language Support (NLS) applications.

Initial Keyboard State sets the keyboard to generate either Local or Roman characters when the item receives input focus, so the operator can begin to type immediately, without switching the keyboard state.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Initial keyboard state is based on the value of the Reading Order property.</td>
</tr>
<tr>
<td>Local</td>
<td>Initial keyboard state is Local (Right To Left language).</td>
</tr>
<tr>
<td>Roman</td>
<td>Initial keyboard state is Roman (Left To Right language).</td>
</tr>
</tbody>
</table>

**Applies to:** display item, text item

**Set:** Designer

**Usage Notes:**
- Most of the time, you will use this property only for text items.
- The operator can override the setting for Initial Keyboard State by pressing the keyboard state toggle key.

Insert Allowed (Block)

Specifies whether records can be inserted in the block.

**Applies to:** block

**Set:** Designer

**Refer to Built-in:**
- GET_BLOCK_PROPERTY
- SET_BLOCK_PROPERTY

**Default:** True

**See also:** Insert Allowed (Item) property
Query Allowed (Block) property
Query Allowed (Item) property
Update Allowed (Block) property
Update Allowed (Item) property
Insert Allowed (Item)

Determines whether an operator can modify the value of an item in a new record (i.e., when the Record_Status is NEW or INSERT).

If you set Insert Allowed to False for an item, the user will not be able to manipulate the item in a new record. For example, the user will not be able to type into a text item, check a check box, or select a radio button.

**Applies to:** text item, check box, list item, radio button, image item, custom items

**Set:** Designer, programmatically

**Refer to Built-in:**
- GET_ITEMPROPERTY
- SET_ITEMPROPERTY

**Default:** True

**Restrictions:** Item properties must be set as follows:
- Enabled set to True
- Displayed set to True

Insert Allowed at the block level takes precedence, so when you set Insert Allowed (Item) to True, also set Insert Allowed (Block) to True.

**Usage Notes:** Set Insert Allowed to False when you want the user to be able to inspect an item without being able to modify it. For example, for a system–generated key field, you might set Insert Allowed to False to prevent modification of the key while still displaying it normally (not grayed out).

Set the **Enabled** property to False if you want to prevent an item from responding to mouse events. Disabled items are grayed out to emphasize the fact that they are not currently applicable, while enabled items with Insert Allowed set to False allow the user to browse an item’s value with the mouse or keyboard, but not to modify the item’s value.

Insert Allowed resembles Update Allowed, which applies to records with a Record_Status of QUERY or CHANGED. For items in database blocks, Insert Allowed, in combination with Update Allowed, lets you control whether the operator can enter or change the value displayed by an item. For items in non–database blocks, setting Insert Allowed to False lets you create a display–only item without disabling it.
Item Type

Specifies the type of item. An item can be one of the following types:

- Button
- Chart Item
- Check box
- Display Item
- Image
- List Item
- OLE Container
- Radio Group
- Radio Group
- Test Item
- VBX Control
- User Area

Applies to: items
Set: Designer
Default: text item
Required/Optional: required
Restrictions: When you set Item Type from the New Block window, the Button and Chart item types are not available. These items do not store values and so cannot relate directly to database columns.
**Items Displayed**

Specifies the number of item instances displayed for the item when the item is in a multi-record block.

Setting Items Displayed > 0 overrides the Records Displayed block property.

**Applies to:** item

**Set:** Designer

**Restrictions:** Items Displayed must be <= Records Displayed block property setting.

**Default:** Zero. Zero indicates that the item should display the number of instances specified by the Records Displayed block property.

**Required/Optional:** optional

**Usage:** Use Items Displayed to create a single button, chart, OLE item, VBX control, or image as part of a multi-record block. For instance, if Records Displayed is set to 5 to create a multi-record block and you create a button, by default you will get 5 buttons, one per record. To get only one button, set Items Displayed to 1.

**Item_Is_Valid**

Specifies whether an item is marked internally as valid.

**Applies to:** item

**Set:** programmatically

**Refer to Built-in:**
- GET_ITEM_PROPERTY
- SET_ITEM_PROPERTY

**Default:** item in a new record: False; item in a queried record: True
Usage Notes: • Use Item_Is_Valid to check whether the current status of a text item is valid.

• Set Item_Is_Valid to True to instruct Oracle Forms to treat any current data in an item as *valid* and skip any subsequent validation. Set Item_Is_Valid to False to instruct Oracle Forms to treat any current data in a text item as *invalid* and subject it to subsequent validation.

---

Join Condition

Defines the relationship that links a record in the detail block with a record in the master block.

Applies to: relation

Set: Designer

Required/Optional: • required for a relation object

• optional in the New Block window

Restrictions: • You cannot specify a constant in the join condition.

• Maximum length for a join condition is 255 characters.

Usage Notes: You can specify a join condition with the following entries:

• an item name that exists in both the master block and the detail block (block_2.item_3)

• an equating condition of two item names, where one item exists in the master block and the other item exists in the detail block

• a combination of item names and equating conditions
Examples: To link a detail block to its master block through the ORDID text item that is common to both blocks, define the following join condition:
ORDID

To link the detail block to its master block through a number of text items, define the join condition as follows:
block1.item1 = block2.item1 AND block1.item2 = block2.item2

Keep in mind that the join condition specifies the relationship between the items in each block, not between the columns in the tables on which those blocks are based. Thus, the items specified must actually exist in the form for the join condition to be valid.

---

Keep Position

Specifies that the cursor position be the same upon re-entering the text item as when last exited.

Applies to: text item
Set: Designer, programmatically
Refer to Built-in:
- GET_ITEM_PROPERTY
- SET_ITEM_PROPERTY
Default: False
Restrictions: Unsupported on some window managers.
Usage Notes: Use this property if you want to give the operator the flexibility to move the cursor to an item, then back to the partially filled item, and have the cursor reposition itself to the end of the partial text.
Key Mode

Specifies how Oracle Forms uniquely identifies rows in the database. This property is included for applications that will run against non-ORACLE data sources. For applications that will run against ORACLE, use the default setting.

By default, the ORACLE database uses unique ROWID values to identify each row. Non-ORACLE databases do not include the ROWID construct, but instead rely solely on unique primary key values to identify unique rows. If you are creating a form to run against a non-ORACLE data source, you must use primary keys, and set the Key Mode block property accordingly.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique</td>
<td>The default setting. Instructs Oracle Forms to use ROWID constructs to identify unique rows in an ORACLE database.</td>
</tr>
<tr>
<td>Updateable</td>
<td>Specifies that Oracle Forms should issue UPDATE statements that include primary key values. Use this setting if your database allows primary key columns to be updated and you intend for the application to update primary key values.</td>
</tr>
<tr>
<td>Non-Updateable</td>
<td>Specifies that Oracle Forms should not include primary key columns in any UPDATE statements. Use this setting if your database does not allow primary key values to be updated.</td>
</tr>
</tbody>
</table>

Applies to: block

Set: Designer, programmatically

Refer to Built-in:
- GET_BLOCK_PROPERTY
- SET_BLOCK_PROPERTY

Default: Unique

Usage Notes: When the Key Mode property is set to one of the primary key modes, you must identify the primary key items in your form by setting the Primary Key block property to True for the block, and the Primary Key item property to True for at least one item in the block.
**Label (Item)**

Specifies the text label that displays for a button, check box, or radio button in a radio group.

**Applies to:** button, check box, radio group button  
**Set:** Designer, programmatically  
**Refer to Built-in:**  
- GET_ITEM_PROPERTY  
- SET_ITEM_PROPERTY  
- GET_RADIO_BUTTON_PROPERTY  
- SET_RADIO_BUTTON_PROPERTY  
**Default:** blank  
**Required/Optional:** optional

**Label (Menu Item)**

Specifies the text label for each menu item.

**Applies to:** menu item  
**Set:** Designer, programmatically  
**Refer to Built-in:**  
- GET_MENU_ITEM_PROPERTY  
- SET_MENU_ITEM_PROPERTY  
**Required/Optional:** optional
Usage Notes: Each menu item has both a name and a label. The label, used only in the runtime GUI, may differ from the name, which can be used programmatically.

Unlike the name, which must follow PL/SQL naming conventions, the label can include multiple words and punctuation. For example, More Info... is an acceptable label, while the corresponding name would be more_info.

When you create a new menu item in the Menu editor, Oracle Forms gives it a default name, like ITEM2, and a default label, <New Item>. When you edit the item label in the Menu editor, making it, for instance, “Show Keys,” the menu item name remains ITEM2 until you change it in either the Navigator or the Properties palette.

Label (Menu Substitution Parameter)

Specifies the label that will prompt the operator to supply a value for the substitution parameter.

Applies to: menu substitution parameter
Set: Designer
Required/Optional: optional
Restrictions: none

Last_Block

Specifies the name of the block with the highest sequence number in the form, as indicated by the sequence of blocks listed in the Object Navigator.

Applies to: form module
Set: not settable
Refer to Built-in: GET_FORM_PROPERTY
See also: First_Block property
**Last_Item**

Specifies the name of the item with the highest sequence number in the indicated block, as indicated by the sequence of items listed in the Object Navigator.

- **Applies to:** block
- **Set:** not settable
- **Refer to Built-in:** GET_BLOCKPROPERTY
- **See also:** First_Item property

---

**Length (Record Group)**

See Column Specification.

---

**List Elements**

The List Elements property group includes the List Item and List Item Value properties.

- **Applies to:** list item
- **Set:** Designer

**List Item**

Specifies the text label for each element in a list item.

- **Required/Optional:** required

**List Item Value**

Specifies the value associated with a specific element in a list item.

- **Default:** NULL
- **Required/Optional:** required
- **Restrictions:**
  - Must be compatible with the item’s data type.
  - Must be unique among values associated with element values.
- **Usage Notes:** When you leave the List Item Value field blank, the value associated with the element is NULL.
List Style

Specifies the display style for the list item, either poplist, combo box, or T-list. The poplist and combo box styles take up less space than a T-list, but operators must open the poplist or combo box to see list elements. A T-list remains "open," and operators can see more than one value at a time if the list is large enough to display multiple values.

Applies to: list item
Set: Designer
Default: Poplist
Usage Notes: The display style you select for the list item has no effect on the data structure of the list item.

Lock Record

Specifies that Oracle Forms should attempt to lock the row in the database that corresponds to the current record in the block whenever the text item’s value is modified, either by the operator or programmatically.

Applies to: text item
Set: Designer, programmatically
Refer to Built-in: • GET_ITEM_PROPERTY  
• SET_ITEM_PROPERTY
Default: False
Restrictions: Valid only when the item is a control item (Base Table Item property set to False) in a base table block.
Usage Notes:

- Set this property to True when the text item is a control item (an item not associated with a base table column), but you still want Oracle Forms to lock the row in the database that corresponds to the current record in the block.
- Useful for lookup text items where locking underlying record is required.
- To set the Lock Record property with SET_ITEM_PROPERTY, use the constant LOCK_RECORD_ON_CHANGE.

---

Locking Mode

Specifies when Oracle Forms should attempt to obtain database locks on rows that correspond to queried records in the form.

The following table describes the allowable settings for the Locking_Mode property.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMMEDIATE</td>
<td>Specifies that Oracle Forms should attempt to lock the corresponding row immediately after an operator modifies an item value in a queried record. With this setting, Oracle Forms locks the record as soon as the operator presses a key to enter or edit the value in a text item.</td>
</tr>
<tr>
<td>DELAYED</td>
<td>Specifies that Oracle Forms should wait to lock the corresponding row in the database until the transaction is about to be committed. With this setting, the record is locked only while the transaction is being posted to the database, not while the operator is editing the record.</td>
</tr>
</tbody>
</table>

Applies to: block

Set: Designer, programmatically

Refer to Built-in:

- GET_BLOCK_PROPERTY
- SET_BLOCK_PROPERTY

Default: Immediate

Restrictions: none

Usage Notes: This property is included primarily for applications that will run against non–ORACLE data sources. For applications that will run against ORACLE, use the default setting.
Long List

When Long List is set to True, Oracle Forms displays a query criteria dialog before displaying the LOV. Operators can enter a value in the dialog to further restrict the rows returned by the default SELECT statement that populates the LOV’s underlying record group. Oracle Forms uses the value entered by the operator to construct the WHERE clause for the SELECT statement. The value is applied to the first column displayed in the LOV (a hidden LOV column is not displayed).

The WHERE clause constructed by Oracle Forms appends the wildcard symbol to the value entered by the operator. So, for instance, if the operator enters 7, the WHERE clause reads LIKE ‘7%’ and returns 7, 712, and 7290.

Apply to: LOV

Set: Designer

Default: False

Restrictions:
- Not applicable for an LOV based on a static record group. (A static record group does not have an associated query.)
- If the SELECT statement for the LOV’s underlying record group joins tables, the name of the first column displayed in the LOV must be unique among all columns in all joined tables. If it is not, an error occurs when the operator attempts to use the Long List feature. For example, when joining the EMP and DEPT tables, the DEPTNO column would not be unique because it occurs in both tables. An alternative is to create a view in the database, and assign a unique name to the column you want operators to reference in the query criteria dialog.
- When a long-list LOV is used for item validation, the query criteria dialog is not displayed so that LOV validation is transparent to the forms operator. Rather, Oracle Forms uses the current value of the text item to construct the WHERE clause used to reduce the size of the list by applying the wildcard criteria to the first visible column in the LOV.
LOV

Specifies the list of values (LOV) to attach to the text item. When an LOV is attached to a text item, operators can navigate to the item and press [List of Values] to invoke the LOV. To alert operators that an LOV is available, Oracle Forms displays the LOV list lamp on the message line when the input focus is in a text item that has an attached LOV.

- Applies to: text item
- Set: Designer
- Refer to Built-in: GET_ITEM_PROPERTY
- Required/Optional: optional
- Restrictions: The LOV must exist in the active form module.

LOV for Validation

Specifies whether Oracle Forms should validate the value of the text item against the values in the attached LOV.

- Applies to: text item
- Set: Designer
- Default: False
- Required/Optional: optional
- Restrictions: LOV property must be specified.
Usage Notes: When LOV for Validation is True, Oracle Forms compares the current value of the text item to the values in the first column displayed in the LOV whenever the validation event occurs:

- If the value in the text item matches one of the values in the first column of the LOV, validation succeeds, the LOV is not displayed, and processing continues normally.

- If the value in the text item does not match one of the values in the first column of the LOV, Oracle Forms displays the LOV and uses the text item value as the search criteria to automatically reduce the list.

For example, if the operator enters the first three digits of a 6-digit product code and then tries to navigate to the next item, Oracle Forms displays the LOV and auto-reduces the list to display all of the codes that have the same first three digits.

- If the operator selects a value from the LOV, Oracle Forms dismisses the LOV and assigns the selected values to their corresponding return items.

When you use an LOV for validation, Oracle Forms always marks a text item as Valid if the operator selects a choice from the LOV. Thus, it is your responsibility to ensure that:

- the text item to which the LOV is attached is defined as a return item for the first column displayed in the LOV and

- the values in the LOV are valid

Note, however, that a When-Validate-Item trigger on the item still fires, and any validation checks you perform in the trigger still occur.

Note also that the first column displayed in the LOV may not be the first column in the LOV’s underlying record group, as some record group columns may not have been included in the LOV structure, or may be hidden columns.
LOV Position

Specifies the horizontal (X) and vertical (Y) coordinates of the upper left corner of the LOV relative to the screen. When you attach an LOV to a text item by setting the LOV property, you can also set the LOV Position property to override the default display coordinates specified for the LOV.

Applies to: text item
Set: Designer
Refer to Built-in: GET_ITEM_PROPERTY
Default: 0, 0; indicating that Oracle Forms should use the default LOV display coordinates, as specified by the LOV Position property.
Required/Optional: required
Usage Notes:
- If you leave the LOV position property set to 0,0, Oracle Forms displays the LOV at the display coordinates you specified when you created the LOV. If you specify position coordinates, the coordinates you specify override the LOV’s default position coordinates.
- The LOV property must be specified.
LOV Type

Specifies how you intend to reference the record group object on which the LOV will be based. Every LOV has an associated record group from which it derives its values at runtime.

**Applies to:** List of Values (LOV)

**Set:** Designer

**Default:** Query

**Required/Optional:** required

**Restrictions:** none

**Usage Notes:** The following settings are valid for this property:

- **Record Group**
  Indicates that you intend to base the LOV on an existing record group. When you select this option, you must enter the name of the record group in the Record Group field, then choose Apply to retrieve the record group column names. The record group you specify can be either a static record group or a query record group, and must already exist in the active form.

- **Query**
  Indicates that you intend to create a record group for the LOV by entering a SELECT statement. When you select this option, enter the SELECT statement in the multi-line field provided, then choose Apply. Oracle Forms creates an underlying record group having the same name as the LOV itself. You can edit the record group definition in the Record Group property sheet.

- **V2-Style**
  This option is included for compatibility with previous versions of Oracle Forms. Its use is not recommended in new applications.
### Magic Item

Specifies one of the following predefined menu items for custom menus: Cut, Copy, Paste, Clear, Undo, About, Help, Quit, or Window. Magic menu items are automatically displayed in the native style for the platform on which the form is being executed, with the appropriate accelerator key assigned.

Cut, Copy, Paste, Clear, Window, and Quit have built-in functionality supplied by Oracle Forms, while the other magic menu items can have commands associated with them.

**Applies to:** menu item

**Set:** Designer

**Default:** Cut

**Required/Optional:** optional

**Restrictions:**
- Magic menu items should be used only in pull-down menus.
- Any given magic menu item may appear only once in the entire menu hierarchy for a given menu module. For example, a menu containing the magic menu item Cut cannot be a submenu of two different options in the menu module.
- Leave the magic menu item's Icon, Accelerator, and Hint properties blank.

**Usage Notes:** The following settings are valid for this property:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut, Copy, Paste, Clear</td>
<td>These items perform the usual text-manipulation operations. Oracle Forms supplies their functionality, so the designer may not enter a command for these items.</td>
</tr>
<tr>
<td>Undo, About</td>
<td>These items have no native functionality, so the designer must enter a command for these items. Any type of command can be used for these items, except Menu.</td>
</tr>
<tr>
<td>Help</td>
<td>The command for the Help menu item must be Menu. The designer provides the functionality of items on this submenu.</td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>Quit</td>
<td>Quit also has built-in functionality, so the designer may not assign a command to this item.</td>
</tr>
<tr>
<td>Window</td>
<td>The Window magic menu item presents a submenu of all open windows, allowing the user to activate any of them. If the Window magic menu item has a command that invokes a submenu, that submenu will contain both the list of open widows and the user-defined submenu items, in an order determined by Oracle Forms. The command type for a magic Window item is Null or Menu.</td>
</tr>
</tbody>
</table>

**Main Menu**

Specifies the name of the individual menu in the module that is to be the main or starting menu at runtime.

If you are creating a pulldown menu, you will not need to change this property: it is automatically set to the name of the first menu you create, and updated thereafter if you change the name of that menu.

The Main Menu property is used mostly with full-screen menus, to limit the menus to which operators have access. Operators cannot navigate to any menu that is above the main menu in the menu module hierarchy.

### Applies to:
- menu module

### Set:
- Designer

### Default:
- blank

### Required/Optional:
- required

### Usage Notes:
When you attach the menu module to a form by setting the appropriate properties in the Form Module property sheet, you can specify a different menu in the module to be the main menu by setting the Starting Menu property.

### See also:
- Starting Menu property
Master Deletes

Specifies how the deletion of a record in the master block should affect records in the detail block:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non–Isolated</td>
<td>The default setting. Prevents the deletion of a master record when associated detail records exist in the database.</td>
</tr>
<tr>
<td>Isolated</td>
<td>Allows the master record to be deleted and does not affect associated detail records in the database.</td>
</tr>
<tr>
<td>Cascading</td>
<td>Allows the master record to be deleted and automatically deletes any associated detail records in the detail block’s base table at commit time. In a master-detail-detail relation, where relations are nested, only records in the immediate detail block are deleted (deletions do not cascade to multiple levels of a relation chain automatically).</td>
</tr>
</tbody>
</table>

Applies to: relation

Set: Designer, programmatically

Refer to Built-in:
- GET_RELATION_PROPERTY
- SET_RELATION_PROPERTY

Default: Non–Isolated

Restrictions:
- Do not set the Master Deletes property to Cascading if the application is running against an Oracle7 Server using the cascading deletes feature.
- Setting this property at runtime has no effect for a default master-detail relation. At design time, Oracle Forms creates the appropriate triggers to enforce the relation, and changing the Master Deletes property at runtime does not alter the default trigger text. The ability to set and get this property programmatically is included only for designers who are coding custom master-detail coordination.
Maximum Length

Specifies the maximum length of the data value that can be stored in the item.

** Applies to:** all items except buttons, image items, and chart items

** Set:** Designer

Refer to Built-in: GET_ITEMPROPERTY

** Default:**
- For a base table item, the length of the corresponding column in the base table.
- For a control item, 30.
- For a LONG item, 240 bytes.

** Required/Optional:** required

** Usage Notes:** For CHAR values, the Maximum Length is 2,048 characters.

** Note:** Bear these considerations in mind if you are writing applications for a multi-byte character set:

- Oracle Forms allows operators to enter the full number of single-byte characters, up to the Maximum Length specified.
- If the operator enters a combination of single-byte and multi-byte characters that produce a string whose total length in bytes exceeds the item’s Maximum Length, the string will be truncated to the nearest whole character and a warning will be displayed. To avoid this situation, consider raising the Maximum Length for the item. (If Maximum Length exceeds the display width of the item, Oracle Forms will automatically allow the operator to scroll the contents of the item.)
Maximum Length (Form Parameter)

Specifies the maximum length, in characters, of a form parameter of type CHAR.

**Applies to:** form parameter

**Set:** Designer

**Default:** For a parameter of type CHAR, 30

**Required/Optional:** required

**Restrictions:**
- Valid only for parameters of type CHAR. For NUMBER and DATE parameters, Oracle Forms uses the standard ORACLE widths for these data types.
- Maximum length of a CHAR parameter is 2,048 bytes.

Maximum Length (Menu Substitution Parameter)

Specifies the maximum length, in characters, of a menu substitution parameter.

**Applies to:** menu substitution parameter

**Set:** Designer

**Default:** 30

**Required/Optional:** required
Menu Item Radio Group

Specifies the name of the radio group to which the current radio menu item belongs.

Applies to: menu item
Set: Designer
Required/Optional: required
Restrictions: • The menu item’s Menu Item Type property must be set to Radio.
• Radio items must be adjacent to each other on the same menu.
• Only one radio group per individual menu is allowed.
Usage Notes: Specify the same Radio Group for all radio items that belong to the same logical set.
See also: Menu Item Type property

Menu Item Roles

Specifies which menu roles have access to a menu item.

Applies to: menu item
Set: Designer
Required/Optional: optional
Usage Notes: You can only grant access to members of those roles displayed in the roles list. To add a role to this list, set the menu module property, Menu Module Roles.
Restrictions: Valid only when the name of at least one role has been specified in the menu module roles list.
See also: Menu Module Roles property
Use Security property
Menu Item Type

Specifies the type of menu item: Plain, Check, Magic, Radio, or Separator. The type determines how the item is displayed and whether the item can have an associated command.

Applies to: menu items

Set: Designer

Default: Plain

Restrictions: The top-level menu should only have plain or magic menu items.

Usage Notes: The following menu item types are available:

Plain
Default. Standard text menu item.

Check
Indicates a Boolean menu item that is either True or False, checked or unchecked.

Whenever the operator selects a Check menu item Oracle Forms toggles the state of that item and executes the command associated with that menu item, if there is one.

Magic
Indicates one of the the following predefined menu items: Cut, Copy, Paste, Clear, Undo, About, Help, Quit, and Window. Magic menu items are automatically displayed in the native style of the platform on which the form is executed, in the position determined by the platform’s conventions, with the appropriate accelerator key assigned.

Cut, Copy, Paste, Clear, Windows, and Quit have built-in functionality supplied by Oracle Forms, while the other magic menu items require that commands be associated with them.

Radio
Indicates a BOOLEAN menu item that is part of a radio group. Enter a radio group name in the Radio Group property field. One and only one Radio menu item in a group is selected at any given time.

When the operator selects a Radio menu item, Oracle Forms toggles the selection state of the item and executes its command, if there is one.
Separator  A line or other cosmetic item. You specify a Separator menu item for the purpose of separating other menu items on the menu. A Separator menu item cannot be selected and therefore it cannot have a command associated with it.

- You can use GET_MENU_ITEM_PROPERTY and SET_MENU_ITEM_PROPERTY to get and set the state of check and radio menu items.

- Magic menu items Cut, Copy, Clear, and Paste are automatically enabled and disabled by Oracle Forms. You can also use GET_MENU_ITEM_PROPERTY and SET_MENU_ITEM_PROPERTY to get and set the state of magic menu items programmatically, but the result of disabling magic menu items will vary, depending on the behavior of the native platform.

See also:  Command Type property
Command Text property
Magic Item property
Menu Module

Specifies the name of the menu to use with this form. When this property is set to Default, Oracle Forms runs the form with the default menu that is built in to every form. When left NULL, Oracle Forms runs the form without any menu.

When any value other than Default or Null is specified, the Use File property determines how the Menu Module property setting is interpreted:

- When the Use File property is True, the Menu Module property specifies the name of the menu .MMX runfile that Oracle Forms should use with this form.
- When the Use File property is False, it specifies the name of the menu module in the database that Oracle Forms should query at form startup to find out the name of the menu .MMX file to use with this form.

**Applies to:** form module

**Set:** Designer

**Default:** Default, indicating that Oracle Forms should run the form with the default form menu.

**Required/Optional:** optional

**See also:** Use File property
**Menu Module Roles**

Specifies which database are available for items in this menu module.

**Applies to:** menu module  
**Set:** Designer  
**Required/Optional:** optional  
**Usage Notes:** Use Menu Module Roles to construct the entire list of roles with access to this menu module, then use Menu Item Roles to specify which of these roles should have access to a specific menu item.

**See also:** Menu Item Roles property  
Use Security property

---

**Menu Role**

Specifies the security role that Oracle Forms should use to run the menu. When the Menu Role property is specified, Oracle Forms runs the indicated menu as if the current operator were a member of the security role specified.

**Applies to:** form module  
**Set:** Designer  
**Required/Optional:** optional  
**Usage Notes:** The Menu Role property is included for backward compatibility only. Its use is not recommended in current applications.

In previous versions of Oracle Forms, the Menu Role property allowed designers to test a menu by creating a master role that had access to all of the items in the menu, and then running the menu under that role. You can obtain the same result by setting the menu module property Use Security to False. When Use Security is False, all operators have access to all menu items, and you do not have to be a member of any particular role to test your application.

**See also:** Use Security property
Menu Style

Specifies the menu display style Oracle Forms should use to run the custom menu specified by the Menu Module property. Display style options include pull-down, full-screen, or bar.

** Applies to:** form module
** Set:** Designer
** Default:** Pull-down
** Required/Optional:** optional
** Restrictions:** Not valid when the Menu Module property is set to DEFAULT. (The default menu runs only in pull-down display style.)
** See also:** Menu Module property

Message

Specifies the message that is to be displayed in an alert.

** Applies to:** alert
** Set:** Designer, programmatically
** Refer to Built-in:** SET_ALERT_PROPERTY
** Required/Optional:** optional
** Restrictions:** Maximum of 200 characters. Note, however, that several factors affect the maximum number of characters displayed, including the font chosen and the limitations of the runtime window manager.
Mirror Item

Specifies the name of the item from which the current item should derive its value. Setting this property synchronizes the values of the two items, so that they effectively mirror each other. When the operator or the application changes the value of either item, the value of the other item changes also.

 Applies to: all items except OLE containers

 Set: Designer

 Required/Optional: Optional

 Default: NULL

 Restrictions: • The item specified must be in the same block as the current item. In the Properties window, the poplist for this property shows only items in the same block as the current item.

 • The maximum number of items in a form that can point to the same mirror item is 100.

 Usage Notes: • You can set Mirror Item for base table or control blocks. When Mirror Item is specified, the current item’s Base Table Item property is ignored, and the item derives its value from the mirror item specified, rather than from a column in the database.

 • You can use mirror item to create more than one item in a block that display the same database column value.
Modal

Specifies whether a window is to be modal. Modal windows require an operator response to dismiss the window before processing can continue.

Applies to: window

Set: Designer

Default: False

Restrictions:
- Valid only for a secondary window. Cannot be set for the root window: the root window is always modeless.
- When Modal is set to True, the following window properties are ignored:
  - Closeable
  - Icon Name
  - Iconifiable
  - Moveable
  - Zoomable
  - Fixed Size
  - Icon Title
  - Inherit Menu
  - Vertical/Horizontal Scroll Bar
Module_NLS_Lang

Specifies the complete current value of the NLS_LANG environment variable defined for the form, for national language support. MODULE_NLS_LANG is the equivalent of concatenating the following properties:

- MODULE_NLS_LANGUAGE (language only)
- MODULE_NLS_TERRITORY (territory only)
- MODULE_NLS_CHARACTER_SET (character set only)

For more information, see Oracle Forms Advanced Techniques, Appendix B, "National Language Support.”

Applies to: form

Set: Not settable from within Oracle Forms. Set at your operating system level.

Refer to Built-in: GET_FORMPROPERTY

Default: Default is usually “America_American.WE8ISO8859P1,” but all the defaults can be port-specific.
Mouse Navigate

Specifies whether Oracle Forms should perform navigation to the item when the operator activates the item with a mouse.

**Applies to:** button, check box, list item, radio group

**Set:** Designer

**Refer to Built-in:**
- GET_ITEM_PROPERTY
- SET_ITEM_PROPERTY

**Default:** True

**Restrictions:** Applies only in mouse-driven environments.

**Usage Notes:**
When Mouse Navigate is False, Oracle Forms does not perform navigation to the item when the operator activates it with the mouse. For example, a mouse click in a button or check box is not treated as a navigational event. Oracle Forms fires any triggers defined for the button or check box (such as When–Button–Pressed), but the input focus remains in the current item.

When Mouse Navigate is True, Oracle Forms navigates to the item, firing any appropriate navigation and validation triggers on the way.
Mouse Navigation Limit

Determines how far outside the current item an operator can navigate with the mouse. Mouse Navigation Limit can be set to the following settings:

- **Form**: (The default.) Allows operators to navigate to any item in the current form.
- **Block**: Allows operators to navigate only to items that are within the current block.
- **Record**: Allows operators to navigate only to items that are within the current record.
- **Item**: Prevents operators from navigating out of the current item. This setting prevents operators from navigating with the mouse at all.

**Applies to:** form  
**Set:** Designer  
**Default:** Form

Moveable

Specifies that the operator can move the window from one location to another on the screen, or that the window can be moved programmatically by way of the appropriate built-in subprogram.

**Applies to:** window  
**Set:** Designer  
**Default:** True  
**Required/Optional:** optional  
**Restrictions:** Cannot be set for a root window: a root window is always movable.  
**See also:** MOVE_WINDOW built-in  
SET_WINDOW_PROPERTY built-in
Multi–Line

Determines whether the text item is a single–line or multi–line editing region.

**Applies to:** text item

**Set:** Designer

**Refer to Built–in:** GET_ITEM_PROPERTY

**Default:** False

**Restrictions:** Valid only for text items with data type CHAR, ALPHA, or LONG.

**Usage Notes:** Setting the Multi–line property True allows a text item to store multiple lines of text, but it does not automatically make the item large enough to display multiple lines. It is up to you to set the Width, Height, Font Size, and Maximum Length properties to ensure that the desired number of lines and characters are displayed.

- **Single–line** Pressing the carriage return key while the input focus is in a single–line text item initiates a [Next Item] function.
- **Multi–line** Pressing the carriage return key while the input focus is in a multi–line text item starts a new line in the item.

Name

Specifies the internal name of the object. Every object must have a valid name that conforms to ORACLE naming conventions.

**Applies to:** all objects

**Set:** Designer

**Default:** $OBJECT\_CLASS \, n$, where $OBJECT\_CLASS$ is the type of object, and $n$ is the next available number in the module; for example, BLOCK5 or EDITOR3.

**Required/Optional:** required

**Restrictions:**

- Must adhere to ORACLE naming restrictions:
• Can be up to 30 characters long
• Must begin with a letter
• Can contain letters, numbers, and the special characters $, #, @ and _ (underscore)
• Are not case sensitive
• Must uniquely identify the object:
  • Item names must be unique among item names in the same block
  • Relation names must be unique among relations that have the same master block
• Cannot be set for the root window

**Example:** ENAME, ADDRESS1, PHONE_NO1

**Usage Notes:**
• For menu items and radio buttons, the Name property has unique characteristics:
  • The Name property specifies an internal handle that does not display at runtime.
  • The Name property is used to refer to the menu item or radio button in PL/SQL code.
  • The Label property specifies the text label that displays for the menu item or current radio button.
• For menu substitution parameters, the following restrictions apply:
  • Restricted to a two–character identifier for the substitution parameter.
  • Must be alphanumeric.
  • Must start with an alphabetic character.
  • When referencing the parameter in a menu command line, the parameter must be preceded by an ampersand (&xx)
  • In a PL/SQL reference, the parameter must be preceded by a colon (:xx).
Navigable

Determines whether the operator or the application can place the input focus in the item during default navigation. When set to True for an item, the item is navigable. When set to False, Oracle Forms skips over the item and enters the next navigable item in the default navigation sequence. The default navigation sequence for items is defined by the order of items in the Object Navigator.

Applies to: all items except chart items and display items

Set: Designer, programmatically

Refer to Built-in:
- GET_ITEM_PROPERTY
- SET_ITEM_PROPERTY

Default: True

Restrictions:
- On some window managers, items such as buttons, check boxes, radio groups, and list items do not accept input focus. On these platforms, the Navigable property has no effect, and operators can only interact with the item by using a mouse.
- You can use the GO_ITEM built-in procedure to navigate to an item that has the Navigable property set to False.

Usage Notes: This property works in conjunction with the Enabled property to determine whether or not an operator can modify data in an item. When Enabled is True, Navigable can be set to True or False. When Enabled is False, an item is always non-Navigable.
Navigation Style

Determines how a Next Item or Previous Item operation is processed when the input focus is in the last navigable item or first navigable item in the block, respectively.

Applies to: block

Set: Designer, programmatically

Refer to Built-in: • GET_BLOCK_PROPERTY
• SET_BLOCK_PROPERTY

Default: Same Record

Usage Notes: The following settings are valid for this property:

Same Record The default navigation style. A Next Item operation from the block’s last navigable item moves the input focus to the first navigable item in the block, in that same record.

Change Record A Next Item operation from the block’s last navigable item moves the input focus to the first navigable item in the block, in the next record. If the current record is the last record in the block and there is no open query, Oracle Forms creates a new record. If there is an open query in the block (the block contains queried records), Oracle Forms retrieves additional records as needed.

Change Block A Next Item operation from the block’s last navigable item moves the input focus to the first navigable item in the first record of the next block. Similarly, a Previous Item operation from the first navigable item in the block moves the input focus to the last item in the current record of the previous block. The Next Navigation Block and Previous Navigation Block properties can be set to redefine a block’s “next” or “previous” navigation block.

See also: Enabled property
Navigable property
Next Navigation Block

Specifies the name of the block that is defined as the “next navigation block” with respect to this block. By default, this is the block with the next higher sequence number in the form, as indicated by the order of blocks listed in the Object Navigator. However, you can set this property to redefine a block’s “next” block for navigation purposes.

**Applies to:** block  
**Set:** Designer, programmatically  
**Refer to Built-in:**  
- GET_BLOCK_PROPERTY  
- SET_BLOCK_PROPERTY  
**Default:** The name of the block with the next higher sequence number, as indicated by the order of blocks listed in the Object Navigator.  
**Required/Optional:** optional  
**Usage Notes:** Setting this property does not change the value of the NEXTBLOCK property.  
**See also:** First Navigation Block property  
NEXT_BLOCK built-in  
Previous Navigation Block property

Next Navigation Item

Specifies the name of the item that is defined as the “next navigation item” with respect to this current item. By default, the next navigation item is the item with the next higher sequence as indicated by the order of items in the Object Navigator. However, you can set this property to redefine the “next item” for navigation purposes.

**Applies to:** item  
**Set:** Designer, programmatically  
**Refer to Built-in:**  
- GET_ITEM_PROPERTY  
- SET_ITEM_PROPERTY
Restrictions: The item specified as Next Navigation Item must be in the same block as the current item.

Default: NULL. NULL indicates the default sequence, which is the name of the item with the next higher sequence number.

NextBlock

Specifies the name of the block with the next higher sequence number in the form, as indicated by the order of blocks listed in the Object Navigator.

Applies to: block

Set: not settable

Refer to Built-in: GET_BLOCK_PROPERTY

Usage Notes: • You can programmatically visit all of the blocks in a form by using GET_BLOCK_PROPERTY to determine the First_Block and NextBlock values.
• The value of NextBlock is NULL when there is no block with a higher sequence number than the current block.
• Setting the Next Navigation Block property has no effect on the value of NextBlock.

See also: First_Block property
Last_Block property
Next Navigation Block property
NextItem

Specifies the name of the item with the next higher sequence number in the block, as indicated by the order of items listed in the Object Navigator.

**Applies to:** item

**Set:** not settable

**Refer to Built-in:** GET_ITEM_PROPERTY

Next_Detail_Relation

Returns the name of the relation that has the same detail block as the specified relation. If no such relation exists, returns NULL.

**Applies to:** relation

**Set:** not settable

**Refer to Built-in:** GET_RELATION_PROPERTY

**Usage Notes:** Use this property with the FIRST_DETAIL_RELATION property to traverse a list of relations for a given master block.

**See also:** First_Detail_Relation property

Next_Master_Relation

Returns the name of the next relation that has the same master block as the specified relation. If no such relation exists, returns NULL.

**Applies to:** relation

**Set:** not settable

**Refer to Built-in:** GET_RELATION_PROPERTY

**Usage Notes:** Use this property with the FIRST_MASTER_RELATION property to traverse a list of relations for a given master block.

**See also:** First_Master_Relation property
**OLE Activation Style**

Specifies the event that will activate the OLE containing item.

** Applies to:** OLE Container

** Set:** Designer

** Default:** Double Click

** Usage Notes:** The following settings are valid for this property:

- **Double Click**
  - The default OLE activation style. An OLE object becomes active by double-clicking anywhere on the OLE object.

- **Focus–in**
  - Navigating to the OLE object causes the OLE object to become active.

- **Manual**
  - An OLE object becomes active by selecting Edit or Open from the Object submenu of the OLE popup menu. The Show OLE Popup Menu property must be set to TRUE and the Object menu item must be set to displayed and enabled. The OLE popup menu is accessible when the mouse cursor is on the OLE object and the right mouse button is pressed.

  If the Show OLE Popup Menu property is TRUE and the Object menu item is displayed and enabled, it is also possible to manually activate the OLE object through the OLE popup menu when the OLE Activation Style is Double Click or Focus–in.

** Restrictions:** Valid only on Microsoft Windows and Macintosh.

** See also:**
- OLE Do In Out property
- OLE In-place Activation property
OLE Class

Determines what class of OLE objects can reside in an OLE container. The following settings are valid for this property:

- **NULL**: The default OLE class. You can insert any kind of OLE object class specified in the registration database in an OLE container.
- **Other than NULL**: Only OLE objects from the specified class can be inserted in an OLE container at runtime. The OLE object classes that are available for selection depend on information contained in the registration database. The content of the registration database is determined by the OLE server applications installed on your computer.

**Applies to**: OLE Container

**Set**: Designer

**Default**: NULL

**Usage Notes**: You select a specific class if you want to create an application that allows operators to change the current OLE object in the OLE container, but want to restrict the operators to creating OLE objects from a particular class.

**Restrictions**: Valid only on Microsoft Windows and Macintosh.

**See also**: OLE Tenant Aspect

OLE Tenant Types property

Show OLE Tenant Type property
OLE Do In Out

Specifies if the OLE server of the embedded object allows inside–out object support during in–place activation. Inside–out activation allows for more than one embedded object to have an active editing window within an OLE container. The following settings are valid for this property:

- **TRUE**: Turns on inside–out object support for embedded objects that have the OLE In–place Activation property set to True.
- **FALSE**: Turns off inside–out object support for embedded objects that have the OLE in–place Activation property set to True.

**Applies to:** OLE Container  
**Set:** Designer  
**Restrictions:**  
- OLE Do In Out only applies to embedded OLE objects that support in–place activation.  
- Valid only on Microsoft Windows and Macintosh.  
**Default:** TRUE  
**See also:** OLE Activation Style property  
OLE In–place Activation property
OLE In-place Activation

Specifies if OLE in-place activation is used for editing embedded OLE objects. The following settings are valid for this property:

TRUE  
Turns on OLE in-place activation. OLE in-place activation is used for editing embedded OLE objects; linked objects are activated with external activation.

FALSE  
Turns off OLE in-place activation and turns on external activation. External Activation is used for editing embedded or linked OLE objects.

**Applies to:** OLE Container

**Set:** Designer

**Restrictions:**
- OLE In-place Activation can only be used on embedded OLE objects, not linked OLE objects.
- Valid only on Microsoft Windows and Macintosh.

**Default:** FALSE

**See also:** OLE Activation Style property
OLE Do In Out property
OLE Popup Menu Items

Determined which OLE popup menu commands are displayed and enabled when the mouse cursor is on the OLE object and the right mouse button is pressed. The OLE popup menu commands manipulate OLE objects. OLE popup menu commands and their actions include:

<table>
<thead>
<tr>
<th>OLE Popup Menu Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUT</td>
<td>Cuts an OLE object and places the content on the clipboard.</td>
</tr>
<tr>
<td>COPY</td>
<td>Copies an OLE object and places the content on the clipboard.</td>
</tr>
<tr>
<td>PASTE</td>
<td>Pastes the content from the clipboard to an OLE container.</td>
</tr>
<tr>
<td>PASTE SPECIAL</td>
<td>Pastes an OLE object from the clipboard to an OLE container in a format other than the original format.</td>
</tr>
<tr>
<td>INSERT OBJECT</td>
<td>Inserts an OLE object in an OLE container.</td>
</tr>
<tr>
<td>DELETE OBJECT</td>
<td>Deletes an OLE object from an OLE container.</td>
</tr>
<tr>
<td>LINKS</td>
<td>Invokes a dialog that has settings to determine how links are updated, edit linked source files, and change links from one source file to another source file.</td>
</tr>
<tr>
<td>OBJECT</td>
<td>Depending on the OLE server, it is possible to perform various operations on an OLE object. Some examples include opening an OLE object, editing an OLE object, and converting an OLE object from one format to another.</td>
</tr>
</tbody>
</table>

Applies to: OLE Container

Set: Designer, programmatically

Refer to Built-in: • GET_ITEM_PROPERTY
• SET_ITEM_PROPERTY

Default: Display On and Enable On for all menu commands

Required/Optional: required
Usage Notes:

- In the Designer, you can set each OLE popup menu command to exhibit the following characteristics by selecting the appropriate check box:

  **Display**
  Specifies whether the selected menu command is displayed.

  **Enable**
  Specifies whether a menu command that has Display On is enabled or disabled. A disabled item appears dimmed or grayed.

- In addition to setting OLE popup menu command properties in the Designer, you can set and get OLE popup menu command properties programmatically. To set or get the OLE popup menu commands programmatically, use a programmatic property name that corresponds to a menu command. The following list includes each of the OLE popup menu commands and a corresponding programmatic property name:

<table>
<thead>
<tr>
<th>Menu Command</th>
<th>Programmatic Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>POPUPMENU_CUT_ITEM</td>
</tr>
<tr>
<td>Copy</td>
<td>POPUPMENU_COPYITEM</td>
</tr>
<tr>
<td>Paste</td>
<td>POPUPMENU_PASTEITEM</td>
</tr>
<tr>
<td>Paste Special</td>
<td>POPUPMENU_PASTESPECIALITEM</td>
</tr>
<tr>
<td>Insert Object</td>
<td>POPUPMENU_INSOBJITEM</td>
</tr>
<tr>
<td>Delete Object</td>
<td>POPUPMENU_DELOBJITEM</td>
</tr>
<tr>
<td>Links</td>
<td>POPUPMENU_LINKSITEM</td>
</tr>
<tr>
<td>Object</td>
<td>POPUPMENU_OBJECTITEM</td>
</tr>
</tbody>
</table>
• You can programmatically set the OLE popup menu command properties to any of the following values:

- **DISPLAYED** Specifies that an OLE popup menu command is displayed and enabled.
- **ENABLED** Specifies that an OLE popup menu command is displayed and disabled. A disabled item appears dimmed or grayed.
- **HIDDEN** Specifies that an OLE popup menu command is not displayed on the OLE popup menu. A command that is not displayed is not enabled.

• In addition to the values that you can set programmatically, you can programmatically get the following values from each of the OLE popup menu commands:

- **DISPLAYED** Return value when an OLE popup menu command is displayed and enabled.
- **ENABLED** Return value when an OLE popup menu command is displayed and disabled. A disabled item appears dimmed or grayed.
- **HIDDEN** Return value when an OLE popup menu command is not displayed on the OLE popup menu. A command that is not displayed is not enabled.
- **UNSupported** Return value when the OLE popup menu is not supported. This is the return value for every platform except Microsoft Windows.

**Restrictions:** Valid only on Microsoft Windows and Macintosh.

**See also:** Show OLE Popup Menu property
OLE Resize Style

Determines how an OLE object is displayed in an OLE container. The following settings are valid for this property:

- **CLIP**
  The default OLE resize style. An OLE object is cropped to fit into an OLE container.

- **SCALE**
  An OLE object is scaled to fit into an OLE container.

- **INITIAL**
  An OLE container is resized to fit an OLE object at creation time only.

- **DYNAMIC**
  An OLE container is resized to fit an OLE object whenever the OLE object size changes.

**Applies to:** OLE Container  
**Set:** Designer  
**Required/Optional:** required  
**Default:** CLIP  
**Restrictions:** Valid only on Microsoft Windows and Macintosh.

OLE Tenant Aspect

Determines how an OLE object appears in an OLE container.

**Applies to:** OLE Container  
**Set:** Designer  
**Default:** CONTENT  
**Usage Notes:** The following settings are valid for this property:

- **CONTENT**
  The default OLE tenant aspect. The content of an OLE object is displayed in an OLE container. The content of the OLE object depends on the value of the OLE Resize Style property and can either be clipped, scaled, or full size.

- **ICON**
  An icon of an OLE object is displayed in an OLE container. The default icon is one that represents...
the OLE server application that created the OLE object. You can choose which icon to use from the insert object dialog.

**THUMBNAIL**  A reduced view of the OLE object is displayed in an OLE container.

**Restrictions:**  Valid only on Microsoft Windows.

**See also:**  OLE Class property
OLE Resize Style property
OLE Tenant Types property

## OLE Tenant Types

Specifies the type of OLE objects that can be tenants of the OLE container. The following settings are valid for this property:

**ANY**  The default OLE tenant type. Any OLE object can be a tenant of the OLE container.

**NONE**  No object can reside in the OLE container.

**STATIC**  Only static OLE objects can be a tenant of the OLE container. A static OLE object is a snapshot image of a linked OLE object that has a broken link to its source. A static OLE object cannot be modified.

**EMBEDDED**  Only an embedded OLE object can be a tenant of the OLE container.

**LINKED**  Only a linked OLE object can be a tenant of the OLE container.

**Applies to:**  OLE Container

**Set:**  Designer

**Default:**  ANY

**Restrictions:**  Valid only on Microsoft Windows and Macintosh.

**See also:**  OLE Class property
OLE Tenant Aspect property
### Operating_System

Specifies the name of the current operating system, such as Microsoft WINDOWS, UNIX, Sun OS, MACINTOSH, VMS, and HP–UX.

**Applies to:** application  
**Set:** not settable  
**Refer to Built-in:** GET_APPLICATION_PROPERTY  
**Usage Notes:** Because the value returned by this property is platform-specific, refer to the Oracle Forms documentation for your operating system if the platform you are using is not listed above.  
**See also:** User_Interface property

### Optimizer_Hint

Specifies a hint string that Oracle Forms passes on to the RDBMS optimizer when constructing queries. Using the optimizer can improve the performance of database transactions.

**Applies to:** block  
**Set:** programmatically  
**Refer to Built-in:**  
- GET_BLOCK_PROPERTY  
- SET_BLOCK_PROPERTY  
**Restrictions:** Valid only for applications running against the Oracle7 Server.
Usage Notes: Consider a form that contains a block named DeptBlock based on the DEPT table. If the operator enters a criteria of " > 25 " for the DEPTNO column and executes the query, the default SELECT statement that Oracle Forms generates to query the appropriate rows from the database is as follows:

```
SELECT DEPTNO, DNAME, LOC, ROWID
FROM DEPT
WHERE (DEPTNO > 25)
```

The designer can use SET_BLOCK_PROPERTY to set the Optimizer_Hint property to request that the Oracle7 Server attempt to optimize the SQL statement for best response time:

```
Set_Block_Property('DeptBlock', OPTIMIZER_HINT, 'FIRST_ROWS');
SELECT /*+ FIRST_ROWS */ DEPTNO, DNAME, LOC, ROWID
FROM DEPT
WHERE (DEPTNO > 25)
```

For more information on how to use this feature with Oracle7, refer to the following sources:

- *Oracle7 Server Application Developer's Guide*, Chapter 5, "Tuning SQL Statements"
- *Oracle7 Server Concepts Manual*, Chapter 13, "The Optimizer"

Order By

See WHERE CLAUSE/ORDER BY CLAUSE.
Other Values

Specifies how any fetched or assigned value that is not one of the pre-defined values associated with a specific list element or radio button should be interpreted.

Applies to: list item, radio group
Set: Designer
Default: blank
Required/Optional: optional

Usage Notes:
• Leave this property blank to indicate that other values are not allowed for this item or radio group. Any queried record that contains a value other than the user-defined element value is silently rejected. Any attempt to assign an other value is disallowed.
• Any value you specify must evaluate to one of the following references:
  • the value associated with one of the list elements or radio groups
  • the name (not the label) of one of the list elements

Parameter Menus

Indicates the name(s) of the individual menu(s) in the module with which the parameter is associated. When the operator navigates to a menu that has an associated parameter, Oracle Forms prompts the operator to enter a value in the Enter Parameter Values dialog.

Applies to: menu parameter
Set: Designer
Required/Optional: optional
Restrictions: Applies only to full-screen menus.
**Password**

Specifies the password of the current operator.

**Applies to:** application  
**Set:** not settable  
**Refer to Built-in:** GET_APPLICATION_PROPERTY  
**Usage Notes:** The Password property returns only the password. If you want a connect string as well, examine the Connect_String property.  
**See also:** Connect_String property  
Username property

**Prevent Masterless Operations**

Specifies whether operators should be allowed to query or insert records in a block that is a detail block in a master–detail relation. When set to True, Oracle Forms does not allow records to be inserted in the detail block when there is no master record in the master block, and does not allow querying in the detail block when there is no master record that came from the database.

When Prevent Masterless Operation is True, Oracle Forms displays an appropriate message when operators attempt to insert or query a record:

- FRM-41105: Cannot create records without a parent record.  
- FRM-41106: Cannot query records without a parent record.

**Applies to:** relation  
**Set:** Designer, programmatically  
**Refer to Built-in:**  
- GET_RELATION_PROPERTY  
- SET_RELATION_PROPERTY  
**Default:** False
Previous Navigation Block

Specifies the name of the block that is defined as the “previous navigation block” with respect to this block. By default, this is the block with the next lower sequence in the form, as indicated by the order of blocks in the Object Navigator. However, you can set this property to redefine a block’s “previous” block for navigation purposes.

Applies to: block

Set: Designer, programmatically

Refer to Built-in
- GET_BLOCKPROPERTY
- SET_BLOCKPROPERTY

Default: The name of the block with the next lower sequence in the form.

Required/Optional: optional

Usage Notes: Setting this property has no effect on the value of the PreviousBlock property.

See also: Next Navigation Block property,
PREVIOUS_BLOCK property
Previous Navigation Item

Specifies the name of the item that is defined as the “previous navigation item” with respect to the current item. By default, this is the item with the next lower sequence in the form, as indicated by the order of items in the Object Navigator. However, you can set this property to redefine the “previous item” for navigation purposes.

Applies to: item

Set: Designer, programmatically

Refer to Built-in
- GET_ITEM_PROPERTY
- SET_ITEM_PROPERTY

Restrictions: The item specified as Previous Navigation Item must be in the same block as the current item.

Default: NULL. NULL indicates the default, which is the name of the item with the next lower sequence in the form.

Required/Optional: optional
### PreviousBlock

Specifies the name of the block with the next lower sequence in the form, as indicated by the order of blocks in the Object Navigator.

- **Applies to:** block
- **Set:** not settable
- **Refer to Built-in:** GET_BLOCK_PROPERTY
- **Required/Optional:** optional
- **Usage Notes:**
  - You may use this property with the First_Block or Last_Block form properties to traverse a list of blocks.
  - The value of PreviousBlock is NULL when there is no block with a lower sequence number than the current block.
  - Setting the Previous Navigation Block property has no effect on the value of PreviousBlock.

- **See also:** Previous Navigation Block property

### PreviousItem

Specifies the name of the item with the next lower sequence number in the block, as indicated by the order of items in the Object Navigator.

- **Applies to:** item
- **Set:** not settable
- **Refer to Built-in:** GET_ITEM_PROPERTY
- **Required/Optional:** optional
Primary Key (Block)

Indicates that any record inserted or updated in the block must have a unique key in order to avoid committing duplicate rows to the block’s base table.

Applies to: block
Set: Designer, programmatically
Refer to Built-in
• GET_BLOCK_PROPERTY
• SET_BLOCK_PROPERTY
Default: False
Restrictions: • The block must be a base table block (Base Table property specified).
• The Primary Key item property must be set to True for one or more items in the block.
See also: Primary Key (Item) property

Primary Key (Item)

Indicates that the item is a base table item in a base table block and that it corresponds to a primary key column in the base table. Oracle Forms requires values in primary key items to be unique.

Applies to: all items except buttons, chart items, and image items
Set: Designer, programmatically
Refer to Built-in
• GET_ITEM_PROPERTY
• SET_ITEM_PROPERTY
Default: False
Required/Optional: optional
Restrictions: The Primary Key block property must be set to True for the item’s owning block.
Quality

Determines the level of quality used to display an image item, allowing you to control the tradeoff between image quality and memory/performance.

The following settings are valid for this property:

- High: Displays the image with high quality, which requires more resources.
- Medium: Displays the image with medium quality.
- Low: Displays the image with low quality, which requires fewer resources.

Applies to: image item
Set: Designer
Default: High
Restrictions: none

Query Allowed (Block)

Specifies whether Oracle Forms should allow the operator or the application to execute a query in the block. When Query Allowed is False, Oracle Forms displays the following message if the operator attempts to query the block:

FRM-40360: Cannot query records here.

Applies to: block
Set: Designer, programmatically
Refer to Built-in
- GET_BLOCK_PROPERTY
- SET_BLOCK_PROPERTY
Default: True
Restrictions: When the Query Allowed block property is True, the Query Allowed item property must be set to True for at least one item in the block.
**Query Allowed (Item)**

Determines if the item can be included in a query against the base table of the owning block.

**Applies to:** all items except buttons, chart items, and image items

**Set:** Designer, programmatically

**Refer to Built-in**
- GET_ITEMPROPERTY
- SETITEMPROPERTY

**Default:** True; however if the item is part of the foreign key in the detail block of a master–detail block relation, Oracle Forms sets this property to False.

**Restrictions:**
- When the Query Allowed item property is set to True, the Query Allowed block property must be set to True.
- The Displayed property must also be set to True.
- Items with the data type LONG cannot be directly queried.

**Usage Notes:** To set the Query Allowed (Item) property programmatically, use the constant QUERYABLE.
Query Length

Specifies the number of characters an operator is allowed to enter in the text item when the form is Enter Query mode.

Applies to: text item
Set: Designer
Default: The value of the item’s Maximum Length property.
Restrictions:  
• A text item’s Query Length can be 0; if it is not 0, it cannot be less than the Maximum Length.
• The maximum query length is 255 characters.
Usage Notes: You can make the query length greater than the Maximum Length when you want to allow the operator to enter complex query conditions. For example, a query length of 5 allows an operator to enter the query condition !=500 in a text item with a Maximum Length of 3.

Query Only

Specifies that an item can be queried but that it should not be included in any INSERT or UPDATE statement that Oracle Forms issues for the block at runtime.

Applies to: check box, radio group, list item, image item, text item, custom item (OLE)
Set: programmatically
Default: False
Refer to Built-in  
• GET_ITEM_PROPERTY
• SET_ITEM_PROPERTY
**Query_Hits**

Specifies the NUMBER value that indicates the number of records identified by the COUNT_QUERY operation. If this value is examined while records are being retrieved from a query, QUERY_HITS specifies the number of records that have been retrieved.

This property is included primarily for applications that will run against non-ORACLE data sources.

**Applies to:** block

**Set:** programmatically

**Refer to Built-in**
- GET_BLOCK_PROPERTY
- SET_BLOCK_PROPERTY

**Restrictions:** Set this property greater than or equal to 0.

**Usage Notes:** This property can be used in several ways:

- In an application that runs against a non-ORACLE data source, use SET_BLOCK_PROPERTY(QUERY_HITS) in an On-Count trigger to inform Oracle Forms of the number of records that a query will return. This allows you to implement count query processing equivalent to Oracle Forms default Count Query processing.

- Use GET_BLOCK_PROPERTY(QUERY_HITS) during Count Query processing to examine the number of records a query will potentially retrieve.

- Use GET_BLOCK_PROPERTY(QUERY_HITS) during fetch processing to examine the number of records that have been retrieved by the query so far and placed on the block’s list of records.

**See also:** Count_Query built-in
Query_Options

Specifies the type of query operation Oracle Forms would be doing by default if you had not circumvented default processing. This property is included for applications that will run against non-ORACLE data sources.

Values for this property include:

- VIEW
- FOR_UPDATE
- COUNT_QUERY
- NULL

Applies to: block
Set: not settable
Refer to Built-in: GET_BLOCKPROPERTY

Raise on Entry

For a canvas-view that is displayed in the same window with one or more other canvas-views, Raise on Entry specifies how Oracle Forms should display the canvas-view when the operator or the application navigates to an item on the canvas-view.

- When Raise on Entry is False, Oracle Forms raises the view in front of all other views in the window only if the target item is behind another view.
- When Raise on Entry is True, Oracle Forms always raises the view to the front of the window when the operator or the application navigates to any item on the view.

Applies to: canvas-view
Set: Designer
Default: False
Restrictions: Applicable only when more than one canvas-view is assigned to the same window.
Range High Value/Range Low Value

Determines the maximum value or minimum value, inclusive, that Oracle Forms allows in the text item.

Applies to: text item
Set: Designer
Refer to Built-in: GET_ITEMPROPERTY
Required/Optional: optional

Usage Notes:
- The following values are valid for range settings:
  - any valid constant
  - form item (:block_name.item_name)
  - global variable (:GLOBAL.my_global)
  - form parameter (:PARAMETER.my_param)
- Oracle Forms evaluates the values in items by data type, as follows:
  - ALPHA alphabetical according to your system’s collating sequence
  - CHAR alphabetical according to your system’s collating sequence
  - DATE chronological
  - DATETIME chronological
  - INT numerical ascending
  - NUMBER numerical ascending
- For all items, you can enter dates in either:
  - the default format for your NLS LANG setting or
  - the format you specified as a format mask
Reading Order

Note: This property is specific to bidirectional National Language Support (NLS) applications.

Specifies the reading order for groups of words (segments) in the same language within a single text item.

Reading Order allows you to control the display of bilingual text items, text items that include segments in both Roman and Local languages. (The Reading Order property has no effect on text items composed of a single language.)

The allowable values for this property are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Text item inherits the reading order specified by its canvas–view Direction property setting.</td>
</tr>
<tr>
<td>Right–To–Left</td>
<td>Item reading order is right-to-left.</td>
</tr>
<tr>
<td>Left–To–Right</td>
<td>Item reading order is left-to-right.</td>
</tr>
</tbody>
</table>

Applies to: display item, text item

Set: Designer

Refer to Built-in
- GET_ITEMPROPERTY
- SET_ITEMPROPERTY

Default: Default

Usage Notes:
- In most cases, you will not need to explicitly set the Reading Order property (the Default setting will provide the functionality you need). Use the Reading Order property only when you need to override the default reading order for an item.

- To get or set the Reading Order property programmatically, use the Direction property.
- To display a Local segment in Right–To–Left mode and a Roman segment in Left–To–Right, use the Default value.
- If your item text is mostly Local, choose the Right–To–Left value.
- If your item text is mostly Roman, choose the Left–To–Right value.
Real Unit

When the Coordinate System property is set to Real, the Real Unit property specifies the real units to be used for specifying size and position coordinates in the form. Real units can be centimeters, inches, pixels or points. (A point is 1/72nd of an inch.)

Oracle Forms interprets all size and position coordinates specified in the form in the real units you specify here. When you convert from one real unit to another, some loss of precision may occur for existing object size and position values.

Applies to: form module
Set: Designer
Default: Centimeter
Required/Optional: optional
Restrictions: Valid only when the coordinate system property is set to Real.

Record Group

Specifies the name of the record group from which the LOV derives its values.

Applies to: LOV
Set: Designer, programmatically
Refer to Built-in: GET_LOVPROPERTY (GROUP_NAME)
SET_LOVPROPERTY (GROUP_NAME)
Default: Null
Required/Optional: Required
Usage Notes: An LOV displays the records stored in its underlying record group. Each LOV must be based on a record group. A record group can be populated by a query (query record group) or by fixed values (static record group).
Record Group Query

Specifies the SELECT statement for query associated with the record group.

Applies to: record group
Set: Designer, programmatically
Refer to Built-in: POPULATE_GROUP_WITH_QUERY
Required/Optional: optional

Record Group Type

Specifies the type of record group, either Static or Query:

Static Specifies that the record group is constructed of explicitly defined column names and column values. The values of a static record group are specified at design time and cannot be changed at runtime.

Query Specifies that the record group is associated with a SELECT statement, and thus can be populated dynamically at runtime. When you select this option, enter the SELECT statement in the multi-line field provided, then choose Apply.

Applies to: record group
Set: Designer
Default: Query
Record Orientation

Determines the orientation of records in the block, either horizontal records or vertical records. When you set this property, Oracle Forms adjusts the display position of items in the block accordingly.

Applies to: block
Set: Designer
Default: Vertical records
Required/Optional: optional
Restrictions: Valid only for a multi-record block (Records Displayed property set greater than 1).
Usage Notes: You can also set this property when you create a block in the New Block window by setting the Orientation option to either Vertical or Horizontal.
See also: Items Displayed property
Records Displayed property

Records Buffered

Specifies the minimum number of records buffered in memory during a query in the block.

Applies to: block
Set: Designer
Default: NULL; which indicates the minimum setting allowed (the value set for the Records Displayed property plus a constant of 3).
Required/Optional: optional
Restrictions: • The minimum value for Buffered equals the value set for the Records Displayed block property plus 3. The maximum number buffered is determined by Oracle Forms at runtime.
• If you specify a number lower than the minimum, Oracle Forms returns an error when you attempt to accept the value.
Usage Notes:

- Oracle Forms buffers any additional records beyond the maximum to a temporary file on disk.
- Improve processing speed by increasing the number of records buffered.
- Save memory by decreasing the number of records buffered. This can, however, result in slower disk I/O.
- If you anticipate that the block may contain a large number of records either as the result of a query or of heavy data entry, consider raising the Records Buffered property to increase performance.
- Consider lowering the Records Buffered property if you anticipate retrieving large items, such as image items, because of the amount of memory each item buffered may require.

See also:
Records Displayed property
Records Fetched property

---

Records Displayed

Specifies the maximum number of records that the block can display at one time. The default is 1 record. Setting Records Displayed greater than 1 creates a multi-record block.

Applies to: block
Set: Designer
Refer to Built-in: GET_BLOCK_PROPERTY
Default: 1
Required/Optional: required
See also: Items Displayed property
Records Fetched

Specifies the maximum number of records that Oracle Forms should fetch from the database at one time.

Applies to: block

Set: Designer

Refer to Built-in: GET_BLOCK_PROPERTY

Default: The number of records the block can display, as indicated by the Records Displayed block property.

Required/Optional: required

Restrictions: • Minimum number of records is 1.

• There is no maximum.

Usage Notes: A size of 1 provides the fastest perceived response time, because Oracle Forms fetches and displays only 1 record at a time. By contrast, a size of 10 fetches up to 10 records before displaying any of them, however, the larger size reduces overall processing time by making fewer calls to the database for records.

See also: Records Buffered property

Records_to_Fetch

Returns the number of records Oracle Forms expects an On-Fetch trigger to fetch and create as queried records.

You can programmatically examine the value of Records_To_Fetch when you are using transactional triggers to replace default Oracle Forms transaction processing when running against a non-ORACLE data source.

Applies to: block

Set: not settable

Refer to Built-in: GET_BLOCK_PROPERTY
**Usage Notes:** Records_To_Fetch is defined only within the scope of an On-Fetch trigger.

The first time the On-Fetch trigger fires, the value of Records_To_Fetch is either the array size (as specified by the Records Fetched block property) or the number of records displayed + 1, whichever is larger.

If the On-Fetch trigger creates this many queried records, the next time the On-Fetch trigger fires, the value of Records_To_Fetch will be the same number.

If, however, the On-Fetch trigger creates fewer records than the value of Records_To_Fetch and returns without raising Form_Trigger_Failure, Oracle Forms will fire the On-Fetch trigger again. Records_To_Fetch will be set to its previous value minus the number of queried records created by the previous firing of the On-Fetch trigger.

This behavior continues until one of the following events occurs:

- The trigger does not create a single queried record (signaling a successful end of fetch).
- The expected number of queried records gets created.
- The trigger raises a Form_Trigger_Failure (signaling that the fetch aborted with an error and fetch processing should halt).

**Example:**

```sql
/*
** Call a client-side package function to retrieve
** the proper number of rows from a package cursor.
*/
DECLARE
  j NUMBER := Get_Block_Property(RECORDS_TO_FETCH);
  emprow emp%ROWTYPE;
BEGIN
  FOR ctr IN 1..j LOOP
    /* Try to get the next row.*/
    EXIT WHEN NOT MyPackage.Get_Next_Row(emprow);
    Create_Queried_Record;
    :Emp.rowid := emprow.ROWID;
    :Emp.empno := emprow.EMPNO;
    :Emp.ename := emprow.ENAME;
  END LOOP;
END;
```

**See also:** Records Fetched property
Records Displayed property
Reference Information

Specifies the following information about the source object and source module for a referenced objects.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>The name of the source module.</td>
</tr>
<tr>
<td>Storage</td>
<td>The source module type (Form or Menu) and location (File System or Database)</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the source object in the source module. (The name of a reference object can be different than the name of its source object.)</td>
</tr>
</tbody>
</table>

**Applies to:** any reference object

**Set:** Designer

**Required/Optional:** optional

Remove on Exit

For a modeless window, determines whether Oracle Forms hides the window automatically when the operator navigates to an item in another window.

**Applies to:** window

**Set:** Designer, programmatically

**Refer to Built-in:**
- GET_WINDOW_PROPERTY
- SET_WINDOWPROPERTY

**Default:** False

**Restrictions:**
- Not valid for a modal window (a window that has the Modal property set to True).
- Cannot be set for a root window: a root window always remains visible when the operator navigates to an item in another window.
Rendered

Specifies that the item is to be displayed as a rendered object when it does not have focus.

**Applies to:** text item, display item

**Set:** Designer

**Default:** True

**Usage Notes:** Use the Rendered property to conserve system resources. A rendered item does not require system resources until it receives focus. When a rendered item no longer has focus, the resources required to display it are released.

Required (Item)

When a new record is being entered, specifies that the item is invalid when its value is NULL.

**Applies to:** list item, text item

**Set:** Designer, programmatically

**Refer to Built-in:**
- GET_ITEM_PROPERTY
- SET_ITEM_PROPERTY

**Default:** False

**Restrictions:**
- Oracle Forms sets Required to True automatically when you create an item with the New Block window and the corresponding column in the database has the NOT NULL constraint.
- When the Required property is True for a list item, the item is considered valid as long as any element in the list is selected. This is true even if the valid value associated with the selected element is NULL. When no element in the list is selected, the list item is considered invalid.

**Usage Notes:** When an item has Required set to True, by default Oracle Forms will not allow navigation out of the item until a valid value is entered. To allow the operator to move freely among the items in the record, set the Defer_Required_Enforcement property to True. This will postpone
enforcement of the Required attribute from item validation to record validation.

Even when Required is set to True, there are circumstances when an item's value could be NULL. Oracle Forms checks for required items as part of its validation process: each item in a new record is subject to validation, but queried data is presumed to be valid and an item is not validated unless it is changed. For example, if the record already exists and is queried from the database, the item that would be Required could come in as NULL.

---

**Required (Menu Parameter)**

Specifies that the operator is required to enter a value for the menu substitution parameter.

- **Applies to:** menu substitution parameter
- **Set:** Designer
- **Default:** False

---

**Return Item (LOV)**

See Column Mapping.
Savepoint Mode

Specifies whether Oracle Forms should issue savepoints during a session. This property is included primarily for applications that will run against non–ORACLE data sources. For applications that will run against ORACLE, use the default setting.

The following table describes the settings for this property:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True (the default)</td>
<td>Specifies that Oracle Forms should issue a savepoint at form startup and at the start of each Post and Commit process.</td>
</tr>
<tr>
<td>False</td>
<td>Specifies that Oracle Forms is to issue no savepoints, and that no rollbacks to savepoints are to be performed.</td>
</tr>
</tbody>
</table>

Applies to: form module

Set: Designer, programmatically

Refer to Built-in: • GET_FORM_PROPERTY
• SET_FORM_PROPERTY

Default: True

Required/Optional: optional

Restrictions: When Savepoint Mode is False, Oracle Forms does not allow a form that has uncommitted changes to invoke another form with the CALL_FORM procedure.
Savepoint_Name

Specifies the name of the savepoint Oracle Forms is expecting to be set or rolled back to.

**Applies to:** application

**Set:** not settable

**Refer to Built-in:** GET_APPLICATION_PROPERTY

**Usage Notes:** The value of this property should be examined only within an On-Savepoint or On-Rollback trigger:

- Use Savepoint_Name in an On-Savepoint trigger to determine the savepoint to be set by a call to ISSUE_SAVEPOINT.

- In an On-Rollback trigger, examine Savepoint_Name to determine the savepoint to which Oracle Forms should roll back by way of a call to ISSUE_ROLLBACK. A NULL savepoint name implies that a full rollback is expected.

**See also:** ISSUE_ROLLBACK built-in

ISSUE_SAVEPOINT built-in
Scroll Bar

The Scroll Bar option specifies whether Oracle Forms should create a block scroll bar for the block you are defining. When Scroll Bar is set to True, Oracle Forms creates the scroll bar on the canvas–view specified by the Scroll Bar Canvas property.

When you create a block scroll bar, you can set the properties of the scroll bar object itself, including Scroll Bar Canvas, Scroll Bar Orientation, Scroll Bar X Position, Scroll Bar Y Position, Scroll Bar Width, Scroll Bar Height, Reverse Direction, and Visual Attribute Name.

**Applies to:** block

**Set:** Designer

**Default:** False

**Required/Optional:** optional

**Usage Notes:** Setting Reverse Direction to True causes Oracle Forms to fetch the next set of records when the operator scrolls upward. If the operator scrolls downward, Oracle Forms displays already fetched records.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scroll Bar Canvas</td>
<td>Specifies the canvas–view on which the block’s scroll bar should be displayed. The specified canvas–view must exist in the form.</td>
</tr>
<tr>
<td>Scroll Bar Orientation</td>
<td>Specifies whether the block scroll bar should be displayed horizontally or vertically.</td>
</tr>
<tr>
<td>Scroll Bar X Position</td>
<td>Specifies the x position of a block scroll bar. The default value is 0.</td>
</tr>
<tr>
<td>Scroll Bar Y Position</td>
<td>Specifies the width of a block scroll bar. The default value is 0.</td>
</tr>
<tr>
<td>Scroll Bar Width</td>
<td>Specifies the width of a block scroll bar. The default value is 2.</td>
</tr>
<tr>
<td>Scroll Bar Height</td>
<td>Specifies the height of a block scroll bar. The default value is 10.</td>
</tr>
<tr>
<td>Reverse Direction</td>
<td>Specifies that the scroll bar scrolls in reverse. The default value is False.</td>
</tr>
<tr>
<td>Visual Attribute Name</td>
<td>Specifies the font, color, and pattern attributes to use for scroll bar. Refer to the Visual Attribute Name property for more information. The default setting is determined by the platform and resource file definition.</td>
</tr>
</tbody>
</table>
Secure (Item)

Hides characters that the operator types into the text item. This setting is typically used for password protection.

The following list describes the allowable values for this property:

- **True**: Disables the echoing back of data entered by the operator.
- **False**: Enables echoing of data entered by the operator.

**Applies to:** text item

**Set:** Designer, programmatically

**Refer to Built-in:**
- GET_ITEM_PROPERTY
- SET_ITEMPROPERTY

**Default:** False

**Restrictions:** Valid only for single-line text items.

Secure (Menu Parameter)

Hides characters that the operator enters for the substitution parameter.

**Applies to:** menu substitution parameter

**Set:** Designer

**Default:** False

**Required/Optional:** optional
Show Keys/Show Keys Description

Specifies whether a key trigger description is displayed in the runtime Show Keys help screen and, if so, what description should display. An entry in the Show Keys screen includes a text description for the key name and the physical keystroke associated with it, for example, Ctrl–S.

**Applies to:** trigger

**Set:** Designer

**Default:** Show Keys: False, Show Keys Description: blank

**Restrictions:** Valid only for key triggers.

**Usage Notes:**
- If you do not want the name or the description to appear in the Show Keys window, set the Show Keys property to False. This is the default setting.
- If you want the name of the key that corresponds to the trigger and its default description to be displayed in the Show Keys window, set the Show Keys property True and leave the Show Keys description blank.
- If you want to replace the default key description, set the Show Keys property True, then enter the desired description in the Show Keys Description field.

**See also:** Function Key built–in
Show OLE Popup Menu

Determines whether the right mouse button displays a popup menu of commands for interacting with the OLE object. The following settings are valid for this property:

TRUE The default OLE popup menu selection. The OLE popup menu is displayed when the mouse cursor is on the OLE object and the right mouse button is pressed.

FALSE The OLE popup menu is not displayed when mouse cursor is on the OLE object and the right mouse button is pressed.

Applies to: OLE Container

Set: Designer, programmatically

Refer to Built-in:
- GET_ITEM_PROPERTY
- SET_ITEM_PROPERTY

Default: True

Required/Optional: required

Usage Notes:
- In addition to the Designer, you can programmatically set and get the OLE popup menu value by using the SHOW_POPUPMENU property. For the SET_ITEM_PROPERTY built-in, the OLE popup menu is shown when the SHOW_POPUPMENU property is set to PROPERTY_TRUE. When the SHOW_POPUPMENU property is set to PROPERTY_FALSE, the OLE popup menu is not shown. You can also use the SHOW_POPUPMENU property with the GET_ITEM_PROPERTY built-in to obtain the current OLE popup menu setting. The GET_ITEM_PROPERTY built-in returns TRUE when the OLE popup menu is shown, and GET_ITEM_PROPERTY returns FALSE when the OLE popup menu is not shown.
- Valid only on Microsoft Windows and Macintosh.

See also: OLE Popup Menu Items property
Show OLE Tenant Type

Determines whether a border defining the OLE object type surrounds the OLE container. The type of border varies according to the object type.

Applies to: OLE Container
Set: Designer
Default: True
Restrictions: Valid only on Microsoft Windows and Macintosh.
See also: OLE Tenant Types property

Size

Size (Canvas–view)

Specifies the width and height of the canvas in the current form coordinate units specified by the Coordinate System form property.

Applies to: canvas–view
Set: Designer, programmatically
Refer to Built-in:
• GET_CANVASPROPERTY
• SET_CANVASPROPERTY

Size (Editor)

Specifies the width and height of the editor in the current form coordinate units specified by the Coordinate System form property.

Applies to: editor
Set: Designer, programmatically
Refer to Built-in:
• EDIT_TEXTITEM
• SHOW_EDITOR

Restrictions: Oracle Forms will ensure that the minimum width of the editor is set wide enough to display the buttons at the bottom of the editor. (On platforms that allow editors to be resized, you can resize the editor to a minimum that will not display all the buttons.)
**Size (Item)**

Specifies the width and height of the item in the current form coordinate units specified by the Coordinate System form property.

**Applies to:** item

**Set:** Designer, programmatically

**Refer to Built-in:**
- GET_ITEM_PROPERTY
- SET_ITEM_PROPERTY
- GET_RADIO_BUTTON_PROPERTY
- SET_RADIO_BUTTON_PROPERTY

**Usage Notes:**
- For a text item or display item, the number of characters the item can store is determined by the Max Length property, and is not affected by the size property.
- In applications that will run on character mode platforms, the height of items that display text labels must be at least 2 character cells for the text to display properly.

**Size (LOV)**

Specifies the width and height of the LOV in the current form coordinate units specified by the Coordinate System form property.

**Applies to:** LOV

**Set:** Designer, programmatically

**Refer to Built-in:**
- GET_LOV_PROPERTY
- SET_LOV_PROPERTY

**Restrictions:** Oracle Forms will ensure that the minimum width of the LOV is set wide enough to display the buttons at the bottom of the LOV. (On platforms that allow LOVs to be resized, you can resize the LOV to a minimum that will not display all the buttons.)
Size (Window)  
Specifies the width and height of the window in the current form coordinate units specified by the Coordinate System form property.

Applies to: window  
Set: Designer, programmatically  
Default: 80 characters by 24 characters  
Refer to Built-in:  
• GET_WINDOW_PROPERTY  
• SET_WINDOW_PROPERTY  

Sizing Style
Determines the display style of an image when the image size does not match the size of the image item.

The following settings are valid for this property:

Crop      Displays only the portion of the full image that fits in the display rectangle.
Adjust    Scales the image to fit within the display rectangle.

Applies to: image item  
Set: Designer  
Default: Crop  

Space Between Records
Specifies the amount of space between instances of items in a multi-record block. A multi-record block is a block that has the Records Displayed property set to greater than 1.

Applies to: item  
Set: Designer  
Default: 0  
Required/Optional: optional
Usage Notes: If you are working in character cell ruler units, the amount of space between item instances must be at least as large as the height of a single cell.

For example, to increase the amount of space between item instances in a 5 record item, you must set the Space Between Records property to at least 4—one cell for each space between item instances.

Starting Menu

Specifies the name of the individual menu in the menu module that Oracle Forms should use as the main, or top-level, menu for this invocation. Operators cannot navigate above the menu specified as the starting menu.

By default, the starting menu is the menu named in the menu module property, Main Menu. The Starting Menu property allows you to override the Main Menu property.

Applies to: form module
Set: Designer
Default: blank (Oracle Forms uses the default main menu as the starting menu)
Required/Optional: optional
Restrictions: • Not valid when the Menu Module property is set to DEFAULT.
• The menu specified must exist in the menu module.
See also: Main Menu property
**Startup Code**

Specifies optional PL/SQL code that Oracle Forms executes when the menu module is loaded in memory at form startup. Think of startup code as a trigger that fires when the menu module is loaded.

**Applies to:** menu module

**Set:** Designer

**Required/Optional:** optional

**Usage Notes:** Startup code does not execute when Oracle Forms is returning from a called form.

**See also:** When–New–Instance trigger

---

**Status (Block)**

Specifies the current status of an indicated block. Block status can be New, Changed, or Query.

**Applies to:** block

**Set:** not settable

**Refer to Built–in:** GET_BLOCK_PROPERTY

**Usage Notes:** You can determine the status of the current block in the form by examining the SYSTEM.BLOCK_STATUS system variable. Form status can be examined by way of the SYSTEM.FORM_STATUS system variable.

**See also:** SYSTEM.FORM_STATUS system variable
SYSTEM.BLOCK_STATUS system variable
SYSTEM.RECORD_STATUS system variable
Status (Record) system variable
Status (Record)

Specifies the current status of the indicated record. Record status can be New, Changed, Query, or Insert.

- **Applies to:** record
- **Set:** programmatically
- **Refer to Built-in:**
  - GET_RECORD_PROPERTY
  - SET_RECORD_PROPERTY
- **Usage Notes:** The status property allows you to examine the status of any indicated record. You can also examine the status of the current record in the form with the SYSTEM.RECORD_STATUS system variable.
- **See also:** Status (Block) property
  - SYSTEM.RECORD_STATUS system variable
  - SYSTEM.BLOCK_STATUS
  - SYSTEM.FORM_STATUS system variable

Subtitle

Specifies a subtitle of up to 40 characters to appear below the Title in full-screen menus.

- **Applies to:** menu
- **Set:** Designer
- **Required/Optional:** optional
- **Restrictions:** Valid only for full-screen menus.
### Tear-off

Defines a menu as a tear-off menu.

- **Applies to:** menu
- **Set:** Designer
- **Default:** False
- **Restrictions:** Only supported in the pull-down menu style, on window managers that support this feature.

### Timer_Name

Specifies the name of the most recently expired timer.

- **Applies to:** application
- **Set:** not settable
- **Refer to Built-in:** GET_APPLICATION_PROPERTY
- **Required/Optional:** optional
- **Restrictions:** Only valid when examined in a When-Timer-Expired trigger.

### Title

Specifies the title to be displayed for the object.

- **Applies to:** alert, form module, LOV, menu, window
- **Set:** Designer
- **Required/Optional:** optional

*Title (LOV)*

- **Default:** NULL
- **Required/Optional:** optional
**Title (Menu)**

**Restrictions:** Valid only for full-screen menus.

**Usage Notes:**
- Up to 40 characters in length.
- Appears at the top of the full-screen menu.

**Title (Window)**

**Refer to Built-in:**
- GET_WINDOW_PROPERTY
- SET_WINDOWPROPERTY

**Required/Optional:** optional

**Usage Notes:**
- If you do not specify a title for a window that is not a root window, Oracle Forms uses the window’s object name, as indicated by the window Name property.
- If you do not specify a title for a root window, and the current menu is the Default menu, Oracle Forms uses the name of the form module for the root window title, as indicated by the form module Name property. When the current menu is a custom menu running in Pull-down or Bar display style, Oracle Forms uses the name of the main menu in the module for the root window title, as indicated by the menu module Main property.

**Top Title**

Specifies a title of up to 72 characters to appear at the top of the editor window.

**Applies to:** editor

**Set:** Designer

**Required/Optional:** optional
Top_Record

Specifies the record number of the topmost record that is visible in the block. (Records are numbered in the order they appear on the block’s internal list of records.)

**Applies to:** block

**Set:** not settable

**Refer to Built-in:** GET_BLOCK_PROPERTY

**Usage Notes:** Together, the TOP_RECORD and RECORDS_DISPLAYED properties allow you to determine the number of the bottom record in the display, that is, the record having the highest record number among records that are currently displayed in the block.

**See also:** Records Displayed built-in

---

Transactional Triggers

Identifies a block as a *transactional control block*; that is, a non–database block that Oracle Forms should manage as a transactional block. This property is included for applications that will run against non–ORACLE data sources, and that will include transactional triggers. If your application will run against ORACLE, leave this property set to False.

When you create a non–ORACLE data source application, you are essentially simulating the functionality of a base table block by creating a transactional control block. Such a block is a control block because its base table is not specified at design time (the Base Table block property is NULL), but it is transactional because there are transactional triggers present that cause it to function as if it were a base table block.

For more information, see *Oracle Forms Advanced Techniques*, Chapter 4, “Connecting to Non–ORACLE Data Sources.”

**Applies to:** block

**Set:** Designer

**Default:** False
Usage Notes:

- Transactional Triggers applies only when the Base Table property is NULL.
- Setting Transactional Triggers to True enables the Primary Key and Column Security properties.

### Trigger Style

Specifies whether the trigger is a PL/SQL trigger or a V2-style trigger. Oracle Corporation recommends that you write PL/SQL triggers only. V2-style trigger support is included only for compatibility with previous versions.

**Applies to:** trigger

**Set:** Designer

**Default:** PL/SQL

**Usage Notes:** Choosing V2–Style Trigger enables the Zoom button, which opens the Trigger Step property sheet.

### Trigger Text

Specifies the PL/SQL code that Oracle Forms executes when the trigger fires.

**Applies to:** trigger

**Set:** Designer

**Required/Optional:** required
Trigger Type

Specifies the type of trigger, either built-in or user-named. User-named triggers are appropriate only in special situations, and are not required for most applications.

- **Applies to:** trigger
- **Set:** Designer
- **Default:** PL/SQL
- **Required/Optional:** required
- **Usage Notes:** Trigger type can be one of the following:
  - **Built-in**
    - Specifies that the trigger is one provided by Oracle Forms and corresponds to a specific, pre-defined runtime event.
  - **User-named**
    - Specifies that the trigger is not provided by Oracle Forms. A user-named trigger can only be executed by a call to the EXECUTE_TRIGGER built-in procedure.

Unchecked Value

Specifies the value you want the check box to display as the unchecked state. For example, Y, 1, MANAGER, or 1992. When a value that matches the unchecked value is fetched or assigned to the check box, the check box is displayed unchecked. Similarly, when the operator toggles the check box to the unchecked state, the value of the check box becomes the unchecked value.

- **Applies to:** check box
- **Set:** Designer
- **Default:** NULL
- **Required/Optional:** Optional; leaving this property blank makes the Unchecked value NULL.
- **Restrictions:** The value must be compatible with the datatype specified by the Data Type property.
**Update Allowed (Block)**

Determines whether operators can modify the values of items in the block that have the Update Allowed item property set to True. (Setting Update Allowed to False for the block overrides the Update Allowed setting of any items in the block.)

**Applies to:** block

**Set:** Designer, programmatically

**Refer to Built-in:**
- GET_BLOCK_PROPERTY
- SET_BLOCK_PROPERTY

**Default:** True

**Restrictions:** When the Update Allowed block property is set to True, at least one item in the block must have the Update Allowed item property set to True for the block to be updateable.

**Usage Notes:** To set the Update Allowed (Block) property programmatically with SET_BLOCK_PROPERTY, use the constant UPDATE_ALLOWED.

**See also:**
- Update Allowed (Item)
- Update Changed Columns
- Update Only if Null
- Insert Allowed (Block)
- Insert Allowed (Item)
- Query Allowed (Block)
- Query Allowed (Item)
Update Allowed (Item)

Specifies whether operators should be allowed to change the value of the base table item in a queried record. When Update Allowed is set to False, operators can navigate to the item in a queried record, but if they attempt to change its value, Oracle Forms displays error FRM-40200: Field is protected against update.

Setting Update Allowed to True does not prevent operators from entering values in a NEW (INSERT) record.

Applies to: all items except buttons, chart items, and image items

Set: Designer, programmatically

Refer to Built-in:
- GET_ITEM_PROPERTY
- SET_ITEM_PROPERTY

Default: True

Restrictions: Item properties must be set as follows:
- Base Table Item set to True
- Enabled set to True
- Displayed set to True
- Update Only if NULL set to False
- When the Update Allowed item property is True, the Update Allowed block property must also be True for the item to be updateable

Usage Notes: To set the Update Allowed (Item) property programmatically, use the constant UPDATEABLE.
Update Changed Columns

When queried records have been marked as updates, specifies that only columns whose values were actually changed should be included in the SQL UPDATE statement that is sent to the database during a COMMIT. By default, Update Changed Columns is set to False, and all columns are included in the UPDATE statement.

**Applies to:** block

**Set:** Designer, programmatically

**Refer to Built-in:**
- GET_BLOCK_PROPERTY
- SET_BLOCK_PROPERTY

**Default:** False

**Required/Optional:** optional

**Usage Notes:**
- When Update Changed Columns is False, Oracle Forms can reuse the same SQL statement for multiple updates, without having to reparse each time. Setting Update Changed Columns to True can degrade performance because the UPDATE statement must be reparsed each time. In general, you should only set Update Changed Columns to True when you know that operators will seldom update column values that will take a long time to transfer over the network, such as LONGs.

- Set Update Changed Columns to True in the following circumstances:
  - To save on network traffic, if you know an operator will primarily update only one or two columns.
  - To avoid re-sending large items that are not updated, such as images or LONGs.
  - To fire database triggers on changed columns only. For example, if you implement a security scheme with a database trigger that fires when a column has been updated and writes the userid of the person performing the update to a table.

**See also:**
- Update Allowed (Block) property
- Update Allowed (item) property
- Update Only if Null property
Update_Column

When set to True, forces Oracle Forms to treat this item as updated.

If the Update Changed Columns block property is set to True, setting Update Column to Property_True specifies that the item has been updated and its corresponding column should be included in the UPDATE statement sent to the database.

If the Update Changed Columns block property is set to True, and Update Column is set to Property_False, the item’s column will not be included in the UPDATE statement sent to the database.

If the Updated Changed Columns block property is set to False, the Update Column setting is ignored, and all base table columns are included in the UPDATE statement.

Applies to: item

Set: programmatically

Refer to Built-in:
- GET_ITEM_PROPERTY
- SET_ITEM_PROPERTY

Required/Optional: optional

Usage Notes: The main use for this property is in conjunction with Update Changed Columns. However, whether or not Update Changed Columns is set to True, you can use this property to check whether a given column was updated.

Note: Although Update Column affects Record Status, setting this property to Property_Off for all columns will not return Record Status to QUERY. If you want Record Status to revert to QUERY, you must set it explicitly with SET_RECORD_PROPERTY.
**Update Only if NULL**

Indicates that operators can modify the value of the item only when the current value of the item is NULL.

**Applies to:** all items except buttons, chart items, and image items

**Set:** Designer, programmatically

**Refer to Built-in:**
- GET_ITEM_PROPERTY
- SET_ITEM_PROPERTY

**Default:** False

**Required/Optional:** optional

**Restrictions:** Item properties must be set as follows:
- Enabled set to True
- Displayed set to True
- Update Allowed set to False

**Usage Notes:** To set the Update Only if NULL property programmatically, use the constant UPDATE_NULL.
Update_Permission

Setting Update_Permission to False performs the following three actions:

- Sets the Update_If_Null property to False.
- Sets the Update Allowed property to False.
- Specifies that this column should not be included in any UPDATE statements issued by Oracle Forms, by removing that column from the SET clause of the UPDATE statements.

**Applies to:** all items except buttons and chart items

**Set:** programmatically

**Refer to Built-in:**
- GET_ITEM_PROPERTY
- SET_ITEM_PROPERTY

**Default:** True

**Required/Optional:** optional

**Usage Notes:** Update_Permission allows form developers to implement their own security mechanism, overriding the Oracle Forms default Column Security property. This property is included primarily for applications that will run against non-ORACLE data sources. Use Update_Permission when you want to exclude certain columns from any UPDATE statements: for example, when using an On-Column-Security trigger to implement a custom security scheme.
Use File

Use File allows you to specify the location of the .MMX runfile when you attach a custom menu to a form module. Oracle Forms loads the .MMX file at form startup.

**Applies to:** form module

**Set:** Designer

**Default:** True

**Required/Optional:** optional

**Usage Notes:** Setting the Use File property allows you to specify the location of the menu .MMX runfile through either direct reference or through database lookup. In most cases, you will want to use direct reference to the file system. Database lookup is included for backward compatibility.

**Direct Reference**
To refer directly to the .MMX file, set the Use File property to True, then enter the path/filename of the .MMX file in the Menu Module field.

**Database Lookup**
To refer to the menu by way of database lookup, set the Use File property to False, then enter the name of the menu module stored in the database. At form startup, Oracle Forms queries the menu module definition to look up the name of the .MMX runfile it needs. (The Menu Module File and Directory define the path to the .MMX file in the file system.)

When the form is loaded at runtime, Oracle Forms locates the .MMX file by querying the database to look up the pointer to the .MMX file defined by the menu module File and Directory properties.
The following table compares the property settings and database conditions required when attaching a menu to a form through direct reference to those required for database lookup.

<table>
<thead>
<tr>
<th>Condition or Property</th>
<th>Direct Reference</th>
<th>Database Lookup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form Module Property: “Use File”</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>Form Module Property: “Menu Module”</td>
<td>Name of .MMX runfile</td>
<td>Name of .MMB menu design module in database</td>
</tr>
<tr>
<td>Menu Module Property: “Directory/File”</td>
<td>n/a</td>
<td>Path/filename of .MMX file in file system</td>
</tr>
<tr>
<td>Database Connection</td>
<td>Not required</td>
<td>Required at form startup</td>
</tr>
<tr>
<td>Location of Menu .MMB at Load Time</td>
<td>n/a</td>
<td>Must be stored in database</td>
</tr>
</tbody>
</table>

The following diagrams compare using direct reference and database lookup when attaching a custom menu to a form.

**File System**

File System

Form1.FMX file

Use File Property:
MENU1.MMX

MENU1.MMX file
File System

Attaching a Menu to a Form through Direct Reference

**Database**

File System

Form1.FMX file

Module Name:
MENU1.MMB

DATABASE

Menu1
Directory/File:
MENU1.MMX

DATABASE

MENU1.MMX file
File System

Attaching a Menu to a Form through Database Lookup

**See also:** Menu Module property
Use Security

Specifies that Oracle Forms should enforce the security scheme defined for the menu module, using the Menu Module Roles property.

**Applies to:** menu module

**Set:** Designer

**Default:** False

**Restrictions:** none

**Usage Notes:** This property can be set to False so that developers can test a menu module without having to be members of any database role. Use Security can then be set to True at production to enforce those roles.

**See also:** Menu Module Roles property

Use 3D Controls

On Microsoft Windows, specifies that Oracle Forms displays items with a 3-dimensional, beveled look.

When Use 3D Controls is set to True, any canvas-view that has Visual Attribute Name set to Default will automatically be displayed with background color grey.

In addition, when Use 3D Controls is set to True, the bevel for each item automatically appears lowered, even if an item-level property is set, for example, to raised.

**Applies to:** form

**Set:** Designer

**Default:** For a new form, True. For a form upgraded from a previous version of Oracle Forms, False.

**Restrictions:** Valid only on Microsoft Windows.
Username

Specifies the username of the current operator.

**Applies to:** application

**Set:** not settable

**Refer to Built-in:** GET_APPLICATION_PROPERTY

**Usage Notes:** May be used with the LOGON built-in in an On-Logon trigger or for connecting to a non-ORACLE data source.

The Username property returns only the username. If you want a connect string as well, examine the Connect_String property.

**See also:** Password property
Connect_String property

User_Interface

Specifies the name of the user interface currently in use.

**Applies to:** application

**Set:** not settable

**Refer to Built-in:** GET_APPLICATION_PROPERTY

**Usage Notes:** This property returns one of the following values:

- MOTIF
- MACINTOSH
- MSWINDOWS
- PM
- CHARMODE
- BLOCKMODE
- X

**See also:** Operating_System property
**User_NLS_Lang**

Specifies the complete value of the NLS_LANG environment variable defined for the current Runform session, for national language support. USER_NLS_LANG is the equivalent of concatenating the following properties:

- USER_NLS_LANGUAGE (language only)
- USER_NLS_TERRITORY (territory only)
- USER_NLS_CHARACTER_SET (character set only)

For more information, see Appendix B, “National Language Support” in the *Oracle Forms Advanced Techniques Manual*.

**Applies to:** application

**Set:** Not settable from within Oracle Forms. Set at your operating system level.

**Refer to Built-in:** GET_APPLICATION_PROPERTY

**Default:** Default is usually “America_American.WE8ISO8859P1,” but all the defaults can be port-specific.

---

**Validation**

Specifies whether default Oracle Forms validation processing has been enabled or disabled for a form.

**Applies to:** form module

**Set:** programmatically

**Refer to Built-in:** GET_FORM_PROPERTY

**Default:** True

**Usage Notes:** Use this property with caution, because when you set Validation to False all internal form validation will be bypassed and no WHEN-VALIDATE triggers will fire.

You can programmatically set Validation to False for only brief periods of time when you specifically want to avoid all default Oracle Forms
validation behavior. Once you set Validation to True again, any text items left in an unvalidated state will be validated according to normal processing rules.

When Validation is set to False, the Post–Change trigger will fire during query processing but will not fire elsewhere.

### Validation Unit

Specifies the scope of form validation at runtime. Specifically, the validation unit defines the maximum amount of data that an operator can enter in the form before Oracle Forms initiates validation. For most applications, the Validation Unit is Item (default setting on most platforms), which means that Oracle Forms validates data in an item as soon as the operator attempts to navigate out of the item.

**Applies to:** form module

**Set:** Designer, programmatically

**Refer to Built-in:**
- GET_FORMPROPERTY
- SET_FORMPROPERTY

**Default:** Default

**Usage Notes:** The following settings are valid for this property:
- Default
- Form
- Block
- Record
- Item

In block mode environments, the Default setting is equivalent to selecting Block. In all other environments, Default is equivalent to selecting Item.
Value

Specifies the value associated with a specific radio.

**Applies to:** radio button

**Set:** Designer

**Default:** NULL

**Required/Optional:** required

**Restrictions:**
- Must be compatible with the item’s data type.
- Must be unique among values associated with radio button.

**Usage Notes:** When you leave the Value field blank, the value associated with the radio button is NULL.

VBX Control File

Specifies the VBX file selection.

**Applies to:** VBX Control

**Set:** Designer

**Default:** none

**Required/Optional:** required

**Usage Notes:** The selection of a VBX file determines which VBX controls are available for use. The number and type of VBX files available for selection depends on the third-party VBX controls that are installed on your system.

Because moving a form module with hard-coded paths to another computer system can make the VBX file and location invalid, you should avoid specifying an absolute path for the VBX Control File property.

For a VBX control file that is not associated with an absolute path, the search criteria is the system default search path. If all default search paths fail to locate the specified VBX control file, the FORMS45_PATH parameter in the ORACLE.INI file becomes the search criteria for
finding the VBX control file. If all search paths in the FORMS45_PATH parameter fail to locate the VBX control file, a runtime error message informs you that the VBX control cannot be found.

**Restrictions:** Valid only on Microsoft Windows.

---

**VBX Control Name**

Specifies the VBX control selection from a VBX file. Some VBX files contain more than a single VBX control. You must specify which VBX control to use even when a VBX file contains on a single VBX control.

**Applies to:** VBX Control

**Set:** Designer

**Default:** none

**Restrictions:** Valid only on Microsoft Windows.

---

**VBX Control Value Property**

Specifies the value property of a VBX control. This property determines the value of the VBX custom item in Oracle Forms.

**Applies to:** VBX Control

**Set:** Designer

**Refer to Built-in:**
- `VBX.GET_VALUE_PROPERTY`
- `VBX.SET_VALUE_PROPERTY`

**Default:** Most VBX controls have a default value property. If the default value property exists, it is the default Oracle Forms VBX Control Value Property. If the VBX control does not have a default value property, the Oracle Forms VBX Control Value Property is the VBX property named “value”. If the VBX property “value” does not exist, a default value property is not assigned to the Oracle Forms VBX Control Value Property.

**Required/Optional:** required
Usage Notes: The VBX CONTROL VALUE PROPERTY automatically synchronizes itself with a VBX property. Changes to the VBX property are reflected in the VBX CONTROL VALUE PROPERTY.

Restrictions: Valid only on Microsoft Windows.

---

**Vertical MDI Toolbar**

On Microsoft Windows, specifies the toolbar canvas-view that should be displayed as a vertical toolbar on the MDI application window. The canvas-view specified must have the Canvas-view Type property set to Vertical Toolbar.

**Applies to:** form

**Set:** Designer

**Default:** Null

**Required/Optional:** optional

**Restrictions:** Valid only on Microsoft Windows. On other platforms, the Vertical MDI Toolbar property is ignored and the toolbar canvas-view is mapped to the window indicated by its Window property setting.

**See also:** Horizontal MDI Toolbar property
Vertical Scroll Bar

Specifies that a vertical scroll bar is to appear on the side of the window.

Applies to: window, image item, editor, item
Set: Designer
Default: False
Required/Optional: optional
Restrictions:
- Not valid for a root window: a root window cannot have scroll bars.
- Valid on window managers that support vertical scroll bars.
- For text item, the Multi–Line property must be TRUE.

Vertical Toolbar

Specifies the canvas–view that should be displayed as a vertical toolbar on the window. The canvas–view you specify must be a vertical toolbar canvas–view (Canvas–view Type property set to Vertical Toolbar) and must be assigned to the current window by setting the Window property.

Applies to: window
Set: Designer
Default: Null
Required/Optional: required if you are creating a vertical toolbar
Usage Notes:
- In the Properties window, the poplist for this property shows only canvas–views that have the Canvas–view Type property set to Vertical Toolbar.
- At runtime, Oracle Forms attempts to display the specified vertical toolbar on the window. However, if more than one toolbar of the same type has been assigned to the same window (by setting the canvas–view Window property to point to the specified window), Oracle Forms may display a different toolbar in response to navigation events or programmatic control.
• On Microsoft Windows, the specified vertical toolbar canvas–view will not be displayed on the window if you have specified that it should instead be displayed on the MDI application window by setting the Vertical MDI Toolbar form property.

See also: Horizontal MDI Toolbar property
          Canvas–view Type property

View

Specifies the canvas–view that is to be the window’s primary content view. At runtime, Oracle Forms always attempts to display the primary view in the window. For example, when you display a window for the first time during a session by executing the SHOW_WINDOW built–in procedure, Oracle Forms displays the window with its primary content view.

If, however, Oracle Forms needs to display a different content view because of navigation to an item on that view, the primary content view is superseded by the target view.

Applies to: window

Set: Designer

Default: NULL

Required/Optional: Required only for a window that will be shown programmatically, rather than in response to navigation to an item on a canvas assigned to the window.

Restrictions: The specified view must be a content view (Canvas–view Type property set to Content), and must be assigned to the indicated window (by setting the Window canvas property).

See also: Canvas–view Type property
          Window property
**View Height, View Width**

Specifies the width and height of the view for a stacked canvas. The size and position of the view define the part of the canvas that is actually displayed in the window at runtime.

**Note:** For a content or toolbar canvas, the view is represented by the window to which the canvas–view is assigned, and so the View Height and View Width properties do not apply.

- **Applies to:** canvas–view
- **Set:** Designer, programmatically
- **Refer to Built–in:** SET_VIEW_PROPERTY
- **Default:** 0,0
- **Required/Optional:** optional
- **Restrictions:** Valid only for a stacked view (Canvas Type property set to Stacked). For a content view, the viewport size is determined by the runtime size of the window in which the content view is displayed.

**View Horizontal Scroll Bar**

Determines whether a canvas–view is displayed with a scroll bar.

- **Applies to:** canvas view
- **Set:** Designer
- **Default:** window default: True
- **Required/Optional:** optional
- **Restrictions:**
  - Applies only to a canvas–view that has the Type property set to Stacked.
  - Valid on window managers that support horizontal scroll bars.
**View Vertical Scroll Bar**

Specifies that a vertical scroll bar is to appear on the side of a canvas–view.

**Applies to:** canvas–view

**Set:** Designer

**Default:** True

**Required/Optional:** optional

**Restrictions:**
- Applies only to a canvas–view that has the Type property set to Stacked.
- Valid on window managers that support vertical scroll bars.

**Visible**

Indicates whether the object is currently displayed or visible. Set VISIBLE to True or False to show or hide a canvas–view or window.

**Applies to:** canvas–view, window

**Set:** programmatically

**Refer to Built–in:**
- GET_VIEWPROPERTY
- GET_WINDOWPROPERTY
- SET_VIEWPROPERTY
- SET_WINDOWPROPERTY

**Default:** TRUE

**Usage Notes:**
- You cannot hide the canvas–view that contains the current item.
- You can hide a window that contains the current item.

**NOTE:** In some operating systems, it is possible to hide the only window in the form.

- When you use GET_WINDOWPROPERTY to determine window visibility, Oracle Forms uses the following rules:
• A window is considered visible if it is displayed, even if it is entirely hidden behind another window.

• A window that has been iconified (minimized) is reported as visible to the operator because even though it has a minimal representation, it is still mapped to the screen.

• When you use GET_VIEWPROPERTY to determine canvas–view visibility, Oracle Forms uses the following rules:

  • A view is reported as visible when it is a) in front of all other views in the window or b) only partially obscured by another view.

  • A view is reported as not visible when it is a) a stacked view that is behind the content view in the window or b) completely obscured by a single stacked view. Note that a view is reported as visible even if it is completely obscured by a combination of two or more stacked views.

  • The display state of the window does not affect the setting of the canvas–view VISIBLE property. That is, a canvas may be reported visible even if the window in which it is displayed is not currently mapped to the screen.

See also: Displayed (Canvas–view) property
Visual Attribute Name

Specifies how the object’s individual attribute settings (Font Name, Background Color, Fill Pattern, etc.) are derived. The following settings are valid for this property:

Default
Specifies that the object should be displayed with default color, pattern, and font settings. When Visual Attribute Name is set to Default, the individual attribute settings reflect the current system defaults. The actual settings are determined by a combination of factors, including the type of object, the resource file in use, and the platform.

Custom
Specifies that the object should be displayed with the attribute settings that you specify for the object explicitly at design time, either by setting them in the Properties window or in the Layout Editor.

Named visual attribute
Specifies a named visual attribute that should be applied to the object. Named visual attributes are separate objects that you create in the Object Navigator and then apply to interface objects, much like styles in a word processing program. When Visual Attribute Name is set to a named visual attribute, the individual attribute settings reflect the attribute settings defined for the named visual attribute object. When the current form does not contain any named visual attributes, the poplist for this property will show only Default and Custom.

Applies to: all interface objects
Set: Designer
Default: Default

Usage Notes:
- Setting individual attributes (Font Name, Background Color, Fill Pattern, etc.) explicitly, either in the Properties window or the Layout Editor, always converts Default or Named to Custom.
- Default, Custom, and named visual attributes include the following individual attributes, listed in the order they appear in the Properties window:
  - Font Name The font family, or typeface, that should be used for text in the object. The list of fonts available is system-dependent.
  - Font Size The size of the font, specified in points.
**Font Style**  The style of the font.

**Font Width**  The width of the font, that is, the amount of space between characters (kerning).

**Font Weight**  The weight of the font.

**Foreground Color**  The color of the object’s foreground region. For items, the Foreground Color attribute defines the color of text displayed in the item.

**Background Color**  The color of the object’s background region.

**Fill Pattern**  The pattern to be used for the object’s fill region. Patterns are rendered in the two colors specified by Background Color and Foreground Color.

**Character Mode Logical Attribute**  Specifies the name of a character mode logical attribute defined in an Oracle Terminal resource file that is to be used as the basis of device attributes for a character mode version of your application.

**White on Black**  Specifies that the object is to appear on a monochrome bitmap display device as white text on a black background.

Not all attributes are valid for each object type. For example, setting font attributes for a window object has no effect. (The font used in a window’s title bar is derived from the system.)

- A new object in a new form has Default visual attributes. The default settings are defined internally. You can override the default font for new items and boilerplate by setting the optional FORMS45_DEFAULTFONT environment variable. For example, On Microsoft Windows, you can set this variable in the ORACLE.INI file, as follows: FORMS45_DEFAULTFONT="COURIER.10". The default font specified determines the font used for new boilerplate text generated by the New Block window, and for any items that have Visual Attribute Name set to Default.

- When you create an item in the Layout Editor, its initial visual attribute settings are determined by the current Layout Editor settings for fonts, colors, and patterns, as indicated by the Font dialog and Color and Pattern palettes.
• On Microsoft Windows, the colors of buttons, window title bars, and window borders are controlled by the Windows Control Panel color settings specified for these elements. You cannot override these colors in Oracle Forms.

• When the Use 3D Controls form property is set to True on Microsoft Windows (the default), items are rendered with shading that provides a sculpted, three-dimensional look. A side effect of setting this property is that any canvas-views that have Visual Attribute Name set to Default derive their color setting from the Windows Control Panel (gray for most color schemes). You can override this setting by explicitly applying custom or named visual attributes to the canvas-view.

• An item that has Visual Attribute Name set to Default, or that has individual attribute settings left unspecified, inherits those settings from the canvas-view to which it is assigned. Similarly, a canvas-view that has Visual Attribute Name set to Default, or that has individual attribute settings left unspecified, inherits those settings from the window in which it is displayed. For example, if you set a window’s Background Color to CYAN, and then leave Background Color unspecified for the canvas-view assigned to the window, at runtime, that canvas-view will inherit the CYAN background from its window. Visual attribute settings derived through window—canvas or canvas—item inheritance are apparent only at runtime, not at design time.

• You can apply property classes to objects to specify visual attribute settings. A property class can contain either the Visual Attribute Name property, or one or more of the individual attribute properties. (If a property class contains both Visual Attribute Name and individual attributes, the Visual Attribute Name property takes precedence.)

• If you apply both a named visual attribute and a property class that contains visual attribute settings to the same object, the named visual attribute settings take precedence, and the property class visual attribute settings are ignored.

• Logical attribute definitions defined in the resource file take precedence over visual attributes specified in the Oracle Forms Designer, local environment variable definitions, and default Oracle Forms attributes. To edit the resource file, use the Oracle Terminal utility.
Visual_Attribute

Specifies the named visual attribute that should be applied to the object at runtime. A visual attribute defines a collection of font, color, and pattern attributes that determine the appearance of the object.

Applies to:  canvas–view, item, radio button

Set:  programmatically

Refer to Built-in:  
- GET_ITEM_PROPERTY
- GET_RADIO_BUTTON_PROPERTY
- SET_CANVAS_PROPERTY
- SET_ITEM_PROPERTY
- SET_RADIO_BUTTON_PROPERTY

Restrictions:  The visual attribute must be a named visual attribute defined in the form module or a logical attribute defined in the runtime resource file.

Usage Notes:  When you execute the appropriate GET_ function to determine the setting of this property at runtime, the return value is one of the following:

- the name of a named visual attribute
- the name of a logical attribute defined in the resource file
- CUSTOM (indicating that the item has custom visual attributes)
- DEFAULT (indicating that the item uses the default attributes defined in the resource file)
WHERE Clause/ORDER BY Clause

The default WHERE Clause and default ORDER BY Clause properties specify standard SQL clauses for the default SELECT statement associated with a base table block. These clauses are automatically appended to the SELECT statement that Oracle Forms constructs and issues whenever the operator or the application executes a query in the block.

Applies to: block

Set: Designer, programmatically

Refer to Built-in: • GET_BLOCK_PROPERTY
  • SET_BLOCK_PROPERTY

Required/Optional: optional

Restrictions: • WHERE Clause and ORDER BY Clause are standard SQL clauses, and must follow the rules for such constructs as specified in the SQL Language Reference Manual.
• Maximum length for WHERE Clause is 32,000 bytes.

Usage Notes: • The reserved words WHERE and ORDER BY are optional. If you do not include them, Oracle Forms automatically prefixes the statement with these words.
• Embedded comments are not supported in WHERE Clause and ORDER BY Clause.
• WHERE Clause can reference the following objects:
  • columns in the block’s base table (except LONG columns)
  • form items (:block_name.item_name)
  • global variables (:GLOBAL.my_global)
  • form parameters (:PARAMETER.my_parameter)
• ORDER BY Clause can reference the following objects:
  • columns in the block’s base table (except LONG columns)
  • form items (:block_name.item_name)
Example: In the following example from an order tracking system, the WHERE Clause limits the retrieved records to those whose \textit{shipdate} column is NULL. The ORDER BY Clause arranges the selected records from the lowest (earliest) date to the highest (latest) date.

\begin{verbatim}
WHERE shipdate IS NULL
ORDER BY orderdate
\end{verbatim}

This WHERE Clause/ORDER BY Clause statement specifies the base conditions for record retrieval. The operator can further restrict the records retrieved by placing the form in Enter Query mode and entering ad hoc query conditions.

See also: Query Allowed (Block) property
Query Allowed (Item) property
Optimizer\_Hint property

---

**Width/Height (WD, HT)**

See Size

---

**Window**

Specifies the window in which the canvas–view will be displayed at runtime.

\begin{itemize}
  \item \textbf{Applies to:} canvas–view
  \item \textbf{Set:} Designer
  \item \textbf{Refer to Built-in:} \texttt{GET\_VIEW\_PROPERTY}
  \item \textbf{Default:} \texttt{ROOT\_WINDOW}, if there is a root window in the form, else the first window listed under the Windows node in the Object Navigator.
  \item \textbf{Required/Optional:} required for the canvas–view to be displayed at runtime
\end{itemize}
Window_Handle

On Microsoft Windows, a window handle is a unique internal character constant that can be used to refer to objects. It is possible to obtain a window handle for any item or window.

**Applies to:** form, block, item

**Refer to Built-in:**
- GET_ITEM_PROPERTY
- GET_WINDOW_PROPERTY
- GET_RADIO_BUTTON_PROPERTY

**Default:** NULL

**Restrictions:** Valid only on Microsoft Windows. (Returns NULL on other platforms.)

**Usage Notes:**
- Specify the name of the item and the WINDOW_HANDLE property in GET_ITEM_PROPERTY to obtain the window handle to an item.
- Specify the name of the window and the WINDOW_HANDLE property in GET_WINDOW_PROPERTY to obtain the window handle to a window. If the name of the window of GET_WINDOW_PROPERTY is FORMS_MDI_WINDOW, the return value is a handle to the MDI client window. The handle to a MDI client window is used to create child MDI windows and controls.
- Specify the item name or item id of the radio group, the name of the radio button, and the WINDOW_HANDLE property in GET_RADIO_BUTTON_PROPERTY to obtain a window handle to a radio button.
- To obtain a window handle to a radio group, use the name of the radio group as the item name in GET_ITEM_PROPERTY. A window handle to the button that is in focus is returned. If no button is in focus, the window handle to the button that is selected is returned. If neither a focused or selected button exists, the window handle to the first button is returned.
**Window_State**

Specifies the current display state of the window:

- **NORMAL**: Specifies that the window should be displayed normally, according to its current Width, Height, X Position, and Y Position property settings.
- **MINIMIZE**: Specifies that the window should be minimized, or iconified so that it is visible on the desktop as a bitmap graphic.
- **MAXIMIZE**: Specifies that the window should be enlarged to fill the screen according to the display style of the window manager.

**Applies to:** window

**Set:** Programmatically

**Refer to Built-in:**
- GET_WINDOW_PROPERTY
- SET_WINDOW_PROPERTY

**Default:** NORMAL

**Restrictions:** Setting Window_State to MAXIMIZE is not supported on Motif.

**Usage Notes:**

The minimize and maximize display states are managed by the window manager and do not affect the window’s current width and height settings, as defined by the Width and Height properties. Thus, if a window display state is currently minimized or maximized, any call to SET_WINDOW_PROPERTY or RESIZE_WINDOW that changes the Width or Height properties will be applied, but will not become apparent to the operator until the window is returned to the Normal state.

Similarly, GET_WINDOW_PROPERTY always returns the window’s current Width and Height property settings, even if the window is currently in the minimized or maximized display state.
Window Style

Specifies whether the window is a Document window or a Dialog window. Document and dialog windows are displayed differently on window managers that support a Multiple Document Interface (MDI) system of window management.

Applies to: window
Set: Designer
Default: Document
Restrictions: Valid only for a secondary window. (A root window is always a document window.)
Usage Notes: MDI applications display a default parent window, called the application window. All other windows in the application are either document windows or dialog windows.

Document windows always remain within the application window frame. If the operator resizes the application window so that it is smaller than a document window, the document window is clipped. An operator can maximize a document window so that it occupies the entire workspace of the application window.

Dialog windows are free-floating, and the operator can move them outside the application window if they were defined as Movable. If the operator resizes the application window so that it is smaller than a dialog window, the dialog window is not clipped.
Wrap Style

Specifies how text is displayed when a line of text exceeds the width of a text item or editor window.

The following list describes the allowable values for this property:

- NONE: No wrapping: text exceeding the right border is not shown.
- CHARACTER: Text breaks following the last visible character, and wraps to the next line.
- WORD: Text breaks following last visible complete word, and wraps to the next line.

Applies to: text item, editor
Set: Designer
Refer to Built-in: GET_ITEM_PROPERTY
Default: WORD
Restrictions: Valid only for multi-line text items.

X Position, Y Position

For an object, specifies where it appears on the screen. For an item, specifies the position of the item’s upper left corner relative to the upper left corner of the item’s canvas. The values you specify are interpreted in the current form coordinate units (character cells, centimeters, inches, pixels, or points), as specified by the Coordinate System form property.

Applies to: all items, editors, LOVs, windows, canvas-views
Set: Designer, programmatically
Restrictions:
- Values for windows and canvas-views may be negative.
- Values for all items, editors, and LOVs must be non-negative.
- Precision allowed is based on the current form coordinate units. Rounding may occur when necessary.

Usage Notes: The following information is specific to the current object.
ITEM

Determines where the item appears on the owning canvas.

Refer to Built-in:

- GET_ITEMPROPERTY
- SET_ITEMPROPERTY
- GET_RADIO_BUTTONPROPERTY
- SET_RADIO_BUTTONPROPERTY

Default: x,y(0,0)

LOV

Determines where the LOV appears on the screen: (0,0) is the upper left corner of the entire screen, regardless of where the root window appears on the screen. The LOV can be displayed anywhere on the screen, including locations outside the form.

Refer to Built-in:

- GET_LOVPROPERTY
- SET_LOVPROPERTY

Default: x,y(0,0)

WINDOW

Determines where the window appears on the screen: (0,0) is the upper left corner of the entire screen.

Refer to Built-in:

- GET_WINDOWPROPERTY
- SET_WINDOWPROPERTY

Default: x,y(0,0)
X Position on Canvas, Y Position on Canvas

Specifies the location of the view’s upper left corner relative to the upper left corner of the canvas. The size and location of the viewport define the view; that is, the part of the canvas that is actually visible in the window to which the canvas is assigned.

**Applies to:** canvas-view

**Set:** Designer, programmatically

**Refer to Built-in:**
- GET_VIEW_PROPERTY
- SET_VIEW_PROPERTY

**Default:** 0,0

**See also:** View Height property
View Width property

Zoomable

Specifies that operators can resize the window by using the zooming capabilities provided by the runtime window manager.

**Applies to:** window

**Set:** Designer

**Default:** True

**Restrictions:**
- Cannot be set for a root window: a root window is always zoomable.
- Only valid when Fixed Size is set to False
Runform Interface

This chapter describes the Runform interface. Among the topics in this chapter are these:

- Starting and Exiting Runform 6 – 2
- Getting Help 6 – 2
- Viewing the Status Line 6 – 2
- Navigating Around Your Form at Runtime 6 – 3
- Choosing Items and Objects 6 – 3
- Interacting with Form Items 6 – 4
- Querying the Database 6 – 6
- Modifying the Database 6 – 10
- Using Menus 6 – 11
- Printing 6 – 14
- Invoking the Debugger 6 – 14
- Runform Function Keys 6 – 15
Starting/Exiting Runform

<table>
<thead>
<tr>
<th>Action</th>
<th>Command Line</th>
<th>Menu</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting Runform</td>
<td><code>f45run &lt;module_name&gt;&lt;username/password&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exiting Runform</td>
<td></td>
<td>Action-&gt;Exit</td>
<td>[Exit]</td>
</tr>
</tbody>
</table>

Getting Help

<table>
<thead>
<tr>
<th>Action</th>
<th>Menu</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard mapping</td>
<td>Help-&gt;Keys</td>
<td>[Keys]</td>
</tr>
<tr>
<td>List of Values</td>
<td>Help-&gt;List</td>
<td>[List of Values]</td>
</tr>
<tr>
<td>Display Error</td>
<td>Help-&gt;display Error</td>
<td>[Display Error]</td>
</tr>
<tr>
<td>Debugger</td>
<td>Help-&gt;Debug</td>
<td>[Debug Mode]</td>
</tr>
</tbody>
</table>

Viewing the Status Line

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>Indicates that there are records above the current record in the block.</td>
</tr>
<tr>
<td>v</td>
<td>Indicates that there are records following the current record in the block.</td>
</tr>
<tr>
<td>&lt;BGM&gt;</td>
<td>The &lt;BGM&gt; lamp displays when there is a background menu available for use with the current application. The background menu can provide shortcuts and general functions for running an application.</td>
</tr>
<tr>
<td>Count</td>
<td>Indicates the number of records retrieved and displayed by a query. Each time the operator displays a record fetched by a query, Oracle Forms increases the count. After the operator fetches the last record, Oracle Forms displays an asterisk (*) before the count.</td>
</tr>
<tr>
<td>Enter Query</td>
<td>Indicates that the operator is in Enter Query mode. While in Enter Query mode, the operator can retrieve records by specifying query retrieval conditions.</td>
</tr>
<tr>
<td>&lt;Insert&gt; or &lt;Replace&gt;</td>
<td>The &lt;Insert&gt; or &lt;Replace&gt; lamp displays the current character mode, either insert or replace.</td>
</tr>
</tbody>
</table>
Navigating Around Your Form

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;List&gt;</td>
<td>The &lt;List&gt; lamp displays when there is a list of values associated with the current item.</td>
</tr>
<tr>
<td>&lt;OSC&gt;</td>
<td>The &lt;OSC&gt; lamp indicates that the operator has been granted authority to issue operating system commands from within Oracle Forms.</td>
</tr>
</tbody>
</table>

### Action Menu

<table>
<thead>
<tr>
<th>Action</th>
<th>Menu</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block navigation</td>
<td>Block-&gt;Next</td>
<td>[Next Block]</td>
</tr>
<tr>
<td></td>
<td>Block-&gt;Previous</td>
<td>[Previous Block]</td>
</tr>
<tr>
<td>Record navigation</td>
<td>Record-&gt;Next</td>
<td>[Next Record]</td>
</tr>
<tr>
<td></td>
<td>Record-&gt;Previous</td>
<td>[Previous Record]</td>
</tr>
<tr>
<td></td>
<td>Record-&gt;scroll Up</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Record-&gt;scroll down</td>
<td></td>
</tr>
<tr>
<td>Item navigation</td>
<td>Item-&gt;Next</td>
<td>[Next Item]</td>
</tr>
<tr>
<td></td>
<td>Item-&gt;Previous</td>
<td>[Previous Item]</td>
</tr>
<tr>
<td>Radio button navigation</td>
<td></td>
<td>[Left] or [Right]</td>
</tr>
</tbody>
</table>

Choosing Items and Objects

<table>
<thead>
<tr>
<th>Action</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choosing an item or object</td>
<td>Navigate to the desired item and then press [Return]</td>
</tr>
</tbody>
</table>
## Interacting with Form Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Action</th>
<th>Menu</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editor</td>
<td>Invoke editor</td>
<td>Navigate to the text item and then choose</td>
<td>[Accept]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Edit→Edit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Accept modifications</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cancel modifications</td>
<td>[Cancel]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Search</td>
<td>[Search]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cut text</td>
<td>Edit→Cut</td>
<td>[Cut]</td>
</tr>
<tr>
<td></td>
<td>Paste text</td>
<td>Edit→Paste</td>
<td>[Paste]</td>
</tr>
<tr>
<td></td>
<td>Copy</td>
<td>Edit→Copy</td>
<td>[Copy]</td>
</tr>
<tr>
<td></td>
<td>Move right</td>
<td></td>
<td>[Right]</td>
</tr>
<tr>
<td></td>
<td>Move left</td>
<td></td>
<td>[Left]</td>
</tr>
<tr>
<td></td>
<td>Move up</td>
<td></td>
<td>[Up]</td>
</tr>
<tr>
<td></td>
<td>Move down</td>
<td></td>
<td>[Down]</td>
</tr>
<tr>
<td></td>
<td>Move to beginning of line</td>
<td></td>
<td>[Beginning of Line]</td>
</tr>
<tr>
<td></td>
<td>Move to end of line</td>
<td></td>
<td>[End of Line]</td>
</tr>
<tr>
<td></td>
<td>Move to first line</td>
<td></td>
<td>[First Line]</td>
</tr>
<tr>
<td></td>
<td>Move to last line</td>
<td></td>
<td>[Last Line]</td>
</tr>
<tr>
<td></td>
<td>Delete backwards</td>
<td></td>
<td>[Delete Backward]</td>
</tr>
<tr>
<td></td>
<td>Delete character</td>
<td></td>
<td>[Delete Character]</td>
</tr>
<tr>
<td></td>
<td>Delete line</td>
<td></td>
<td>[Delete Line]</td>
</tr>
<tr>
<td>List Item</td>
<td>Reveal pop list values</td>
<td></td>
<td>[Down]</td>
</tr>
<tr>
<td></td>
<td>Scroll up</td>
<td></td>
<td>[Up]</td>
</tr>
<tr>
<td></td>
<td>Scroll down</td>
<td></td>
<td>[Down]</td>
</tr>
<tr>
<td></td>
<td>Choose a value</td>
<td></td>
<td>[Return]</td>
</tr>
<tr>
<td>Item</td>
<td>Action</td>
<td>Menu</td>
<td>Keyboard</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------</td>
<td>-----------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>List of Values</td>
<td>Invoke LOV</td>
<td>Help-&gt;List</td>
<td>[List of Values]</td>
</tr>
<tr>
<td></td>
<td>Scroll up</td>
<td></td>
<td>[Up]</td>
</tr>
<tr>
<td></td>
<td>Scroll down</td>
<td></td>
<td>[Down]</td>
</tr>
<tr>
<td></td>
<td>Choose a value and dismiss LOV</td>
<td></td>
<td>[Return]</td>
</tr>
<tr>
<td></td>
<td>Search for a specific value</td>
<td></td>
<td>Press [Search] or navigate to the Find pane, then enter &quot;%&quot; followed by your search criteria.</td>
</tr>
<tr>
<td></td>
<td>across all columns</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Auto-reduce the list of values</td>
<td></td>
<td>Type the letters of the word or words you want to search for in the List pane. Press [Backspace] to return to the previous reduction criteria.</td>
</tr>
<tr>
<td></td>
<td>(first column only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cancel</td>
<td></td>
<td>[Cancel]</td>
</tr>
<tr>
<td>OLE Item</td>
<td>Activate OLE item</td>
<td>Choose Object-&gt;Edit</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>from the OLE popup menu if available. Alternatively, you can use your mouse to activate an OLE item by either double clicking it or by navigating to it.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Return to form</td>
<td>Choose Exit from the OLE application menu or click on a form item.</td>
<td>None</td>
</tr>
<tr>
<td>Radio Group</td>
<td>Choose a radio button</td>
<td></td>
<td>[Select]</td>
</tr>
<tr>
<td></td>
<td>Navigate between buttons</td>
<td></td>
<td>[Left] or [Right]</td>
</tr>
<tr>
<td>Text Item</td>
<td>Scroll up in a multi–line text item</td>
<td></td>
<td>[Up]</td>
</tr>
<tr>
<td></td>
<td>Invoke editor</td>
<td>Edit-&gt;Edit</td>
<td>[Edit]</td>
</tr>
</tbody>
</table>
## Querying the Database

<table>
<thead>
<tr>
<th>Task</th>
<th>Method</th>
<th>Menu</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieve database records</td>
<td>Retrieve all records</td>
<td>Execute-&gt;eXecute</td>
<td>[Execute Query], or select the default Query button twice if it is available.</td>
</tr>
<tr>
<td>Retrieve specific records</td>
<td>Choose Query-&gt;Enter, specify your query retrieval criteria, then choose Query-&gt;eXecute.</td>
<td>Press [Enter Query], specify your query retrieval criteria, then press [Execute Query]. Alternatively, if the default query button is available, you could select the Query button, enter your query retrieval criteria, and then select the Query button to execute your query.</td>
<td></td>
</tr>
<tr>
<td>Retrieve next set of records</td>
<td>Query-&gt;Fetch next set</td>
<td>[Next Set of Records]</td>
<td></td>
</tr>
<tr>
<td>Cancel an outstanding/long query</td>
<td></td>
<td>[Cancel Query]</td>
<td>(On some platforms Ctrl+C)</td>
</tr>
<tr>
<td>Include an item in a query while in Enter Query mode</td>
<td>Check box</td>
<td></td>
<td>Navigate to the desired check box and press Shift-click simultaneously.</td>
</tr>
<tr>
<td></td>
<td>Radio button</td>
<td></td>
<td>Select the desired radio button. To exclude a radio button, deselect the selected value.</td>
</tr>
<tr>
<td></td>
<td>List item value</td>
<td></td>
<td>Select the desired radio button value. To exclude a text list style list item, deselect the selected value. To exclude a pop list style list item, select the “blank” value.</td>
</tr>
<tr>
<td></td>
<td>Text item</td>
<td></td>
<td>Navigate to the text item, enter your query retrieval criteria, then press [Execute Query].</td>
</tr>
<tr>
<td>Task</td>
<td>Method</td>
<td>Menu</td>
<td>Keyboard</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>Reuse your previous search criteria</td>
<td></td>
<td>Choose Query→Enter, Query→Last criteria, and then choose Query→Execute.</td>
<td>Press [Enter Query] twice and then press [Execute Query].</td>
</tr>
<tr>
<td>Count query records</td>
<td></td>
<td>Query→count Hits</td>
<td>[Count Query Hits]</td>
</tr>
<tr>
<td>Cancel Enter Query mode</td>
<td></td>
<td>Query→Cancel</td>
<td>[Cancel]</td>
</tr>
</tbody>
</table>

**Note:** While in Enter Query mode, all function keys are valid except for the following:

- [Block Menu]  [Next Block]
- [Clear Form]  [Next Record]
- [Commit]      [Next Set of Records]
- [Down]        [Previous Block]
- [Duplicate Record]  [Scroll Down]
- [Insert Record]  [Scroll Up]
- [Lock Record]   [Up]
## Methods for Retrieving Specific Records

<table>
<thead>
<tr>
<th>Task</th>
<th>Method</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieve specific records:</td>
<td>by exact values</td>
<td>Press [Enter Query], enter the desired values, then press [Execute Query].</td>
</tr>
<tr>
<td>by pattern matching</td>
<td>Press [Enter Query], enter the desired pattern, then press [Execute Query]. Use an underscore to represent any character. Use % to search for any combination of characters. For example, specifying _IN%S will retrieve: BINS, FINES, WINNERS, WINEMAKERS, etc.</td>
<td></td>
</tr>
<tr>
<td>by entering variable conditions (relational operators)</td>
<td>Press [Enter Query], include a relational operator with your retrieval criteria, then press [Execute Query]. For example, &lt;=100. For more information about relational operators, see the chart below, &quot;Using Relational Operators.&quot;</td>
<td></td>
</tr>
<tr>
<td>by using the Query Where feature</td>
<td>Press [Enter Query], include a colon in the desired fields, specify your query where criteria, then press [Execute Query]. For more information about how to use the Query Where feature, see the chart below, &quot;Ways to use the Query Where Clause.&quot;</td>
<td></td>
</tr>
<tr>
<td>by using SQL</td>
<td>Press [Enter Query], include a colon in the desired fields, include the desired SQL statements, then press [Execute Query]. For example, you could enter: :ORDER &gt; 500 AND :CLIENT&lt;110 ORDER BY :CLIENT</td>
<td></td>
</tr>
</tbody>
</table>
Using Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>equal to</td>
<td>=&quot;SMITH&quot;</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
<td>!=19.5</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>&gt;100.00</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or</td>
<td>&gt;=2000</td>
</tr>
<tr>
<td></td>
<td>equal to</td>
<td></td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>&lt;&quot;DAVIS&quot;</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or</td>
<td>&lt;=100.00</td>
</tr>
<tr>
<td></td>
<td>equal to</td>
<td></td>
</tr>
<tr>
<td>BETWEEN</td>
<td>between two</td>
<td>#BETWEEN 100 AND 110</td>
</tr>
<tr>
<td></td>
<td>values</td>
<td></td>
</tr>
<tr>
<td>IS NULL</td>
<td>is NULL</td>
<td>:var IS NULL</td>
</tr>
<tr>
<td>LIKE</td>
<td>is like</td>
<td>#LIKE 'SMITH'</td>
</tr>
<tr>
<td>IN</td>
<td>equal to any</td>
<td>#IN 'KING'</td>
</tr>
<tr>
<td></td>
<td>member of</td>
<td></td>
</tr>
</tbody>
</table>

Note: When using relational operators be aware of the following restrictions:

- Relational operators do not work in items with time values.
  To retrieve records from items with time values, perform a search using the Query WHERE dialog.
- Whenever you use a relational operator that is a word (such as BETWEEN), you must precede the operator with a “#”.

Note: Because Oracle Forms considers the # symbol to be a special character in Enter Query mode, it is not possible to query records with actual data values that contain the # (#1, #2, etc.) without writing additional code.
Ways to Use the Query
Where Clause

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Method</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>To retrieve records that have a value:</td>
<td>greater than (&gt;)</td>
<td>:CLIENTID&gt;106</td>
</tr>
<tr>
<td></td>
<td>greater than or equal to (&gt;=)</td>
<td>:ITEMTOT&gt;=100</td>
</tr>
<tr>
<td></td>
<td>less than (&lt;)</td>
<td>:ORDERID&lt;305</td>
</tr>
<tr>
<td></td>
<td>less than or equal to (&lt;=)</td>
<td>:ITEMTOT&lt;=100</td>
</tr>
<tr>
<td></td>
<td>equal to (=)</td>
<td>:CLIENTID=106</td>
</tr>
<tr>
<td></td>
<td>not equal to (!=)</td>
<td>:NAME !='SHAPE UP'</td>
</tr>
<tr>
<td>To express a query that can be satisfied by either of two conditions:</td>
<td>Use the OR operator</td>
<td>:CODE=100860 OR :CODE=100861</td>
</tr>
<tr>
<td>To express a query with two conditions:</td>
<td>Use the AND operator</td>
<td>:ORDERDATE='14–JUL–86' AND :CLIENTID=106</td>
</tr>
</tbody>
</table>

Modifying the Database

<table>
<thead>
<tr>
<th>Action</th>
<th>Menu</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit a fetched record</td>
<td>Edit-&gt;Edit</td>
<td>[Edit]</td>
</tr>
<tr>
<td>Create a new record</td>
<td>Record-&gt;Insert</td>
<td>[Insert Record]</td>
</tr>
<tr>
<td>Lock a fetched record</td>
<td>Record-&gt;Lock</td>
<td>[Lock Record]</td>
</tr>
<tr>
<td>Delete a record</td>
<td>Record-&gt;Remove</td>
<td>[Delete Record]</td>
</tr>
<tr>
<td>Save your changes</td>
<td>Action-&gt;Save</td>
<td>[Save]</td>
</tr>
<tr>
<td>Revert changes</td>
<td>Action-&gt;Clear All</td>
<td>[Clear All]</td>
</tr>
<tr>
<td>Duplicate record</td>
<td>Record-&gt;Duplicate</td>
<td>[Duplicate Record]</td>
</tr>
<tr>
<td>Duplicate item</td>
<td>Duplicate-&gt;Item</td>
<td>[Duplicate Item]</td>
</tr>
<tr>
<td>Clear Form</td>
<td></td>
<td>[Clear Form]</td>
</tr>
<tr>
<td>Clear Block</td>
<td>Block-&gt;Clear</td>
<td>[Clear Block]</td>
</tr>
<tr>
<td>Clear Record</td>
<td>Record-&gt;Clear</td>
<td>[Clear Record]</td>
</tr>
<tr>
<td>Clear Item</td>
<td>Item-&gt;Clear</td>
<td>[Clear Item]</td>
</tr>
</tbody>
</table>
## Using the Default Menu

<table>
<thead>
<tr>
<th>Menu</th>
<th>Menu Item</th>
<th>Function Key</th>
<th>Built-in</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Action Menu</strong></td>
<td>Revert</td>
<td>[Clear Form/ Rollback]</td>
<td>CLEAR_FORM</td>
</tr>
<tr>
<td></td>
<td>Save</td>
<td>[Commit]</td>
<td>COMMIT_FORM</td>
</tr>
<tr>
<td></td>
<td>Refresh</td>
<td>[Refresh]</td>
<td>REDISPLAY</td>
</tr>
<tr>
<td></td>
<td>Print</td>
<td>[Print]</td>
<td>PRINT</td>
</tr>
<tr>
<td></td>
<td>Exit</td>
<td>[Exit/Cancel]</td>
<td>EXIT_FORM</td>
</tr>
<tr>
<td><strong>Edit Menu</strong></td>
<td>Cut</td>
<td>[Cut]</td>
<td>CUT_REGION</td>
</tr>
<tr>
<td></td>
<td>Copy</td>
<td>[Copy]</td>
<td>COPY_REGION</td>
</tr>
<tr>
<td></td>
<td>Paste</td>
<td>[Paste]</td>
<td>PASTE_REGION</td>
</tr>
<tr>
<td></td>
<td>Edit</td>
<td>[Edit]</td>
<td>EDIT_FIELD</td>
</tr>
<tr>
<td><strong>Block Menu</strong></td>
<td>Previous</td>
<td>[Previous Block]</td>
<td>PREVIOUS_BLOCK</td>
</tr>
<tr>
<td></td>
<td>Next</td>
<td>[Next Block]</td>
<td>NEXT_BLOCK</td>
</tr>
<tr>
<td></td>
<td>Clear</td>
<td>[Clear Block]</td>
<td>CLEAR_BLOCK</td>
</tr>
<tr>
<td><strong>Record Menu</strong></td>
<td>Previous</td>
<td>[Previous Record]</td>
<td>PREVIOUS_RECORD</td>
</tr>
<tr>
<td></td>
<td>Next</td>
<td>[Next Record]</td>
<td>NEXT_RECORD</td>
</tr>
<tr>
<td></td>
<td>Scroll Up</td>
<td>[Scroll Up]</td>
<td>SCROLL_UP</td>
</tr>
<tr>
<td></td>
<td>Scroll down</td>
<td>[Scroll Down]</td>
<td>SCROLL_DOWN</td>
</tr>
<tr>
<td></td>
<td>Clear</td>
<td>[Clear Record]</td>
<td>CLEAR_RECORD</td>
</tr>
<tr>
<td></td>
<td>Remove</td>
<td>[Delete Record]</td>
<td>DELETE_RECORD</td>
</tr>
<tr>
<td></td>
<td>Insert</td>
<td>[Insert Record]</td>
<td>CREATE_RECORD</td>
</tr>
<tr>
<td></td>
<td>Duplicate</td>
<td>[Duplicate Record]</td>
<td>DUPLICATE_RECORD</td>
</tr>
<tr>
<td></td>
<td>Lock</td>
<td>[Lock Record]</td>
<td>LOCK_RECORD</td>
</tr>
<tr>
<td><strong>Item Menu</strong></td>
<td>Previous</td>
<td>[Previous Item]</td>
<td>PREVIOUS_ITEM</td>
</tr>
<tr>
<td></td>
<td>Next</td>
<td>[Next Item]</td>
<td>NEXT_ITEM</td>
</tr>
<tr>
<td></td>
<td>Clear</td>
<td>[Clear Item]</td>
<td>CLEAR_ITEM</td>
</tr>
<tr>
<td></td>
<td>Duplicate</td>
<td>[Duplicate Item]</td>
<td>DUPLICATE_ITEM</td>
</tr>
<tr>
<td>Menu</td>
<td>Menu Item</td>
<td>Function Key</td>
<td>Built-in</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------</td>
<td>-----------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Query Menu</td>
<td>Enter</td>
<td>[Enter Query]</td>
<td>ENTER_QUERY</td>
</tr>
<tr>
<td></td>
<td>eXecute</td>
<td>[Execute Query]</td>
<td>EXECUTE_QUERY</td>
</tr>
<tr>
<td></td>
<td>Last criteria</td>
<td>[Enter Query]</td>
<td>ENTER_QUERY</td>
</tr>
<tr>
<td></td>
<td>Cancel</td>
<td>[Exit/Cancel]</td>
<td>EXIT_FORM</td>
</tr>
<tr>
<td></td>
<td>count Hits</td>
<td>[Count Query Hits]</td>
<td>COUNT_QUERY</td>
</tr>
<tr>
<td></td>
<td>Fetch next set</td>
<td>[Next Set of Records]</td>
<td>NEXT_SET</td>
</tr>
<tr>
<td>Help Menu</td>
<td>Help</td>
<td>[Help]</td>
<td>HELP</td>
</tr>
<tr>
<td></td>
<td>Keys</td>
<td>[Show Keys]</td>
<td>SHOW_KEYS</td>
</tr>
<tr>
<td></td>
<td>List</td>
<td>[List]</td>
<td>LIST_VALUES</td>
</tr>
<tr>
<td></td>
<td>Error</td>
<td>[Display Error]</td>
<td>DISPLAY_ERROR</td>
</tr>
<tr>
<td></td>
<td>Debug</td>
<td>[Debug Mode]</td>
<td>BREAK</td>
</tr>
</tbody>
</table>

### Other Menu Features

<table>
<thead>
<tr>
<th>Menu Feature</th>
<th>Task</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block menu</td>
<td>Select block menu</td>
<td>[Block Menu]</td>
</tr>
<tr>
<td></td>
<td>Select a block</td>
<td>[Up] or [Down]</td>
</tr>
<tr>
<td></td>
<td>Navigate to the selected block</td>
<td>[Return]</td>
</tr>
<tr>
<td></td>
<td>Search for a value</td>
<td>[Search]</td>
</tr>
<tr>
<td></td>
<td>Exit block menu</td>
<td>[Cancel] or [Exit]</td>
</tr>
<tr>
<td>Background Menu</td>
<td>Select background menu</td>
<td>[Show BGM]</td>
</tr>
<tr>
<td></td>
<td>Choose background menu item</td>
<td>[Background Menu n]</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>n</em> represents a number between 1 and 10 and corresponds to a specific command on the background menu.</td>
</tr>
<tr>
<td></td>
<td>Exit background menu</td>
<td>[Exit]</td>
</tr>
</tbody>
</table>
### Menu Feature

<table>
<thead>
<tr>
<th>Menu Feature</th>
<th>Task</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerator keys</td>
<td>Choose an accelerator key</td>
<td>Press the appropriate key combination (if available)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Accelerator keys are displayed beside menu items.</td>
</tr>
<tr>
<td>Operating system command</td>
<td>Issue operating system</td>
<td>[Enter&gt;1 OS Command]</td>
</tr>
<tr>
<td></td>
<td>command (if available to</td>
<td>To return to Oracle Forms, press [Return].</td>
</tr>
<tr>
<td></td>
<td>user role the &lt;OSC&gt; lamp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>is displayed at the status</td>
<td></td>
</tr>
<tr>
<td></td>
<td>line)</td>
<td></td>
</tr>
<tr>
<td>Menu parameter value</td>
<td>Accept entered values</td>
<td>[Return]</td>
</tr>
</tbody>
</table>

### Menu Style

<table>
<thead>
<tr>
<th>Menu Style</th>
<th>Task</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pull-down</td>
<td>Select pull-down menu</td>
<td>[Alt–Menu Accelerator]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example, press Alt–A on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MS Windows to select the Action</td>
</tr>
<tr>
<td></td>
<td></td>
<td>menu.</td>
</tr>
<tr>
<td></td>
<td>Scroll up</td>
<td>[Up]</td>
</tr>
<tr>
<td></td>
<td>Scroll down</td>
<td>[Down]</td>
</tr>
<tr>
<td></td>
<td>Choose menu item</td>
<td>[Return]</td>
</tr>
<tr>
<td></td>
<td>Exit menu</td>
<td>[Escape]</td>
</tr>
<tr>
<td>Bar</td>
<td>Select bar menu</td>
<td>[Application Menu]</td>
</tr>
<tr>
<td></td>
<td>Scroll between menu items</td>
<td>[Left]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Right]</td>
</tr>
<tr>
<td></td>
<td>Choose menu item</td>
<td>[Return]</td>
</tr>
<tr>
<td></td>
<td>Return to previous menu</td>
<td>[Previous Menu]</td>
</tr>
<tr>
<td></td>
<td>Return to main menu</td>
<td>[Main Menu]</td>
</tr>
<tr>
<td></td>
<td>Exit menu</td>
<td>[Exit]</td>
</tr>
<tr>
<td>Full-screen</td>
<td>Select full-screen menu</td>
<td>[Display Full-screen Menu]</td>
</tr>
<tr>
<td></td>
<td>Scroll between menu items</td>
<td>[Up]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Down]</td>
</tr>
<tr>
<td></td>
<td>Choose menu item</td>
<td>[Return]</td>
</tr>
<tr>
<td></td>
<td>Return to previous menu</td>
<td>[Previous Menu]</td>
</tr>
<tr>
<td></td>
<td>Return to main menu</td>
<td>[Main Menu]</td>
</tr>
</tbody>
</table>
### Printing

<table>
<thead>
<tr>
<th>Menu Style</th>
<th>Task</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display current menu location in menu hierarchy</td>
<td>[Where Display]</td>
<td></td>
</tr>
<tr>
<td>Exit menu</td>
<td>[Exit]</td>
<td></td>
</tr>
</tbody>
</table>

### Invoking the Debugger

<table>
<thead>
<tr>
<th>Action</th>
<th>Menu</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print the contents of the current screen</td>
<td>Action→Print</td>
<td>[Print]</td>
</tr>
<tr>
<td>Invoke the debugger</td>
<td>Help→Debug</td>
<td>[Debug Mode]</td>
</tr>
</tbody>
</table>
Runform Function Keys

This section lists the Runform function keys for the operator’s interface.

For key mappings for a particular keyboard type, refer to the following sources:

- online Oracle Forms Show Keys screen (for most keyboards, [Ctrl–K])
- the keyboard map

A keyboard map shows which physical key or key sequence corresponds to each Oracle Forms function key on a particular system. To obtain platform-specific key mappings, refer to the Oracle Forms documentation for your operating system.

**Note:** There are at least two Oracle Forms keyboard mappings for each system—one for designing a form (Designer keyboard map) and one for running a form (Runform keyboard map).

Many of the Runform function keys correspond to built-in subprogram routines that you use to execute equivalent functions programmatically. For more information about the processes invoked by these keys, refer to the Oracle Forms Reference Manual, Vol. 1, Chapter 3, “Built-in Routines.”

The function keys in the operator’s interface fall into the following categories:

- Oracle Forms Common Keys
- Editor
- List Values
- Normal Mode (all modes except Enter Query mode)
  - Navigation
  - Editing
  - Miscellaneous

Oracle Forms function keys are represented by the key name enclosed in square brackets: [Next Item].

**Note:** Some keys appear in more than one group.
The Oracle Forms Common Keys include:

- [Accept]
- [Cancel]
- [Exit]
- [Help]
- [Refresh]
- [Show Keys]

Oracle Forms Common Keys plus:

- [Search]
- [Toggle Insert/Replace]

Oracle Forms Common Keys plus:

- [Delete Backward]
- [Delete Character]
- [Down]
- [Left]

Oracle Forms Common Keys plus:

- [Beginning of Line]
- [Down]
- [First Line]
- [Last Line]
- [Left]
- [Next Block]
- [Next Item]
- [Next Primary Key]
- [Next Record]
- [Next Set of Records]
- [Previous Block]
• [Previous Item]
• [Previous Record]
• [Right]
• [Scroll Down]
• [Scroll Left]
• [Scroll Right]
• [Scroll Up]
• [Up]

Editing

Oracle Forms Common Keys plus:
• [Clear Block]
• [Clear Item]
• [Clear Form]
• [Clear Record]
• [Copy]
• [Cut]
• [Delete Backward]
• [Delete Line]
• [Delete Record]
• [Duplicate Item]
• [Duplicate Record]
• [Edit]
• [Insert Record]
• [Paste]
• [Search]
• [Select]
• [Toggle Insert/Replace]
Miscellaneous

Oracle Forms Common Keys plus:

- [Application Menu]
- [Background Menu n]
- [Block Menu]
- [Count Query Hits]
- [Debug Mode]
- [Display Error]
- [Enter 1 Operating System Command]
- [Enter >1 Operating System Command]
- [Enter Query]
- [Execute Query]
- [List of Values]
- [Menu]
- [Print]
- [Redefine User Password]
- [Show Background Menu]
- [Where Display]

[Accept] Accepts your modifications and closes the dialog box.
[Accept] enters into the database all changes made since the last [Accept] or [Clear Form].

[Application Menu] Allows the operator to run another application by selecting it from the Application Menu.

[Background Menu n] Selects Background Menu item n, where n is 1 through 10.

[Block Menu] Displays a list of all the blocks in the current form. From this list, you can select the block to which you would like to move.

[Cancel] Returns to normal operation without performing the query.

[Clear Block] Clears all records from the current block and creates a new record. [Clear Block] may prompt you to commit your changes.
[Clear Block] does not delete records from the database; it only removes records from the work space.
[Clear Form] clears all the blocks of the current form, deletes all data in all blocks of the form, and performs a rollback. [Clear Form] might prompt you to commit your changes.

[Clear Form] undoes all inserts, updates, and deletes posted to the database. [Clear Form] does not delete records from the database; it only removes records from the work space.

[Clear Item] clears the entire item of all characters.

[Clear Record] removes the current record from the current block, reversing any uncommitted changes made to that record. A cleared record is not deleted from the database.

[Close Window] allows you to close the current window. Functions in character mode only.

[Copy] copies an area of text after you select it with [Select] or highlight it with a mouse, and stores it in the paste buffer. Only the last selected or highlighted text remains in the paste buffer.

[Count Query Hits] clears the current block and displays on the message line the number of rows that a query would retrieve if executed. If used in Enter Query mode, [Count Query Hits] does not clear the current query.

**Note:** If you press [Count Query Hits] after pressing [Execute Query], Oracle Forms terminates the query, clears all the records from the work space, and counts all of the records in the table that can be retrieved by the block.

[Cut] cuts an area of text after you select it with [Select] or highlight it with a mouse, and stores it in the paste buffer. Only the last selected or highlighted text remains in the paste buffer.

[Debug Mode] invokes the Runform Debugger if the form is running in debug mode.

[Delete Backward] deletes the character to the left of the current cursor position.

[Delete Line] deletes the current line. Available in the pop–up editor and in a multi–line text item.

[Delete Record] deletes a retrieved record from the screen and from the database. Records are not permanently deleted until you commit your changes to the database.
[Display Error] Displays error information and/or advanced help information, if available, for the item where the last error occurred.

[Down] Moves the cursor to the same item in the next record.
When in a multi-line text editor, [Down] shifts the cursor down within the current item.
If the next record is a new record, [Down] moves the cursor to the first item of the new record.

[Duplicate Item] Copies the item value into the current item.

[Duplicate Record] Copies all item values from the previous record into a new record.

[Edit] Displays a pop-up window in which the operator can edit an item.
Press [Edit] twice to dismiss the window. Press [Cancel] to dismiss the window without accepting its contents.

[End of Line] Moves cursor to the right of the last character in the line. Available in the pop-up editor and in a multi-line text editor.

[Enter 1 Operating System Command] Allows you to temporarily exit Runform so that you can enter a single operating system command.

[Enter >>1 Operating System Command] Allows you to temporarily exit Runform so that you can enter operating system commands.

[Enter Query] Clears the current block, forcing you to commit your previous changes, and allows you to enter query criteria.
In [Enter Query] mode, the following keys have these functions:
[Enter Query] displays the query criteria last used.
[Execute Query] performs the query. If records are retrieved, returns to normal operation; however, if no records are found, remains in [Enter Query] mode.
[Exit] returns to normal operation without performing the query.
[Count Query Hits] displays the number of rows that satisfy the current query criteria.

[Execute Query] Clears the current block, forcing you to commit your previous changes and retrieves all the records from the database table referenced by the block. (Only those records that can fit on the screen are displayed.)
When used after [Enter Query], [Execute Query] executes a query with the criteria you have specified.

[Exit] Exits the current form and returns to the system command prompt or to a previous form. [Exit] also terminates query processing or interrupts the [List of Values] function.

In the editor, functions like [Undo] by undoing all changes in that session.

[Exit] also dismisses the editing window.

[First Line] Moves the cursor to the top of the text in the window. Available in the pop-up editor and in a multi-line text editor.

[Help] Displays a brief help message for the current field. Pressing [Help] twice may display advanced help information if available for that field.

[Insert Record] Inserts a new record after the current record. Functions in character mode only.

[Last Line] Moves the cursor to the bottom of the text in the window. Available in the pop-up editor and in a multi-line text editor.

[Left] Moves the cursor one character to the left (within an item or a line).

[List of Values] Activates a list of values, if there is one available for this item. Functions in character mode only. Following are the two types of lists of values:

1. If a pop-up window appears, the window will display an enterable item and a list of possible values for the current item. Press [Next Item] to move the cursor to the enterable item, then enter search criteria, and press [List of Values]. Oracle Forms executes the selection and returns the cursor to the list.

   Use the cursor keys or [Scroll Up] and [Scroll Down] to navigate through the list. Press [Select] to choose the selection and dissolve the list of values pop-up. Press [Cancel] to leave the list without making a selection.

2. If nothing pops up when you press [List of Values], that item will display possible values for the current item. Press [Next Item] to see subsequent values. Press [Exit] to select a value.

[Menu] Activates the main menu in Oracle Forms if it is available. Functions in character mode only.
<table>
<thead>
<tr>
<th>Key Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Move Window]</td>
<td>Allows you to move the current window. Functions in character mode only.</td>
</tr>
<tr>
<td>[Next Block]</td>
<td>Moves the cursor to the next block in the form that contains at least one enterable field. The order is established by the form designer.</td>
</tr>
<tr>
<td>[Next Item]</td>
<td>Moves the cursor to the next enterable item in the current record. (The order is established by the form designer.)</td>
</tr>
<tr>
<td>[Next Primary Key]</td>
<td>Moves the cursor to the next enterable item in the current record that has been designated as part of the “primary key”—those items that uniquely identify a particular row of a database table.</td>
</tr>
<tr>
<td>[Next Record]</td>
<td>Moves the cursor to the next record in the current block. If no more records are found, [Next Record] creates a new blank record (unless the current record is blank).</td>
</tr>
<tr>
<td>[Next Set of Records]</td>
<td>Retrieves the next set of records (a number specified by the designer) into the current block from records that satisfy an active query and places the cursor on the first record of the set.</td>
</tr>
<tr>
<td>[Paste]</td>
<td>Functions in character mode only. Pastes text in paste buffer at current cursor location.</td>
</tr>
<tr>
<td>[Previous Block]</td>
<td>Moves the cursor to the previous block in the form that contains at least one enterable item. (The order is established by the form designer.)</td>
</tr>
<tr>
<td>[Previous Item]</td>
<td>Moves the cursor to the previous enterable item in the current record. (The order is established by the form designer.)</td>
</tr>
<tr>
<td>[Previous Record]</td>
<td>Moves the cursor to the previous record in the current block.</td>
</tr>
<tr>
<td>[Print]</td>
<td>Writes the current screen to a file and asks if you want to print it. Functions in character mode only.</td>
</tr>
<tr>
<td>[Redefine User Password]</td>
<td>Changes your password to a new one that you select and enter.</td>
</tr>
<tr>
<td>[Refresh]</td>
<td>Redraws the screen image. Functions in character mode only.</td>
</tr>
<tr>
<td>[Replace Next Occurrence]</td>
<td>Functions in pop-up editor only. Replaces next occurrence of item specified for search and replace.</td>
</tr>
<tr>
<td>[Return]</td>
<td>Acknowledges the current message.</td>
</tr>
<tr>
<td>[Resize Window]</td>
<td>Allows you to resize the current window. Functions in character mode only.</td>
</tr>
</tbody>
</table>
[Right] Moves the cursor one character to the right (within a item or a line).

[Rotate Window] Allows you to rotate the current window. Functions in character mode only.

[Scroll Down] Functions in character mode only. Shifts the window of the current block or list down by approximately 80 percent, displaying records that are outside of the window.

When in a multi–line text editor, [Scroll Down] shifts the cursor down within the current item.

[Scroll Left] Functions in character mode only. Shifts the item’s window to the left by approximately 80 percent of the item’s entire display width, displaying the item contents that are outside of the window.

[Scroll Right] Functions in character mode only. Shifts the item’s display window to the right by approximately 80 percent of the item’s entire display width, displaying the item contents that are outside of the window.

[Scroll Up] Functions in character mode only. Shifts the window of the current block or list up by approximately 80 percent, displaying records that are outside of the window.

When in a multi–line text editor, [Scroll Up] shifts the cursor up within the current item.

[Search] Displays a dialog box for entering search and replace criteria. Searches forward or backward from the current cursor location. Available only in the pop–up editor.

[Search Next Occurrence] Functions in pop–up editor only. Searches for next occurrence of item defined in pop–up editor.

[Select] Selects a choice in a list or in a dialog box.

Marks a point on the screen that Oracle Forms uses for text cutting and copying.

[Show Background Menu] Displays the Background Menu if one is available for the current application.

[Show Keys] Displays the function key assignments currently in effect. You may return to the form or menu you were working on by pressing [Accept]. [Show Keys] is available in normal and enter query modes.
[Toggle Insert/Replace] Toggles between Insert mode and Replace mode. Functions in character mode only. [Toggle Insert/Replace] is not available on Microsoft Windows.

[Up] Moves the cursor to the same item in the previous record.
When in a multi-line text editor, [Up] shifts the cursor up within the current item.

[Where Display] [Where Display] displays the menu path, which is the list of menu names that lead to the current menu. [Where Display] is available in full-screen menu style only.
This chapter describes Oracle Forms default processing. It includes the following topics:

- Query Processing 7 – 2
- Validation 7 – 6
- Commit Processing 7 – 11
- Locking 7 – 16
Query Processing

Each base table block in a form has an associated query; that is, a specific SQL SELECT statement that is associated with the block. When the operator or the application executes a query in the block, it is this SELECT statement that Oracle Forms issues to the database.

At runtime, operators can place the form in Enter Query mode and enter query–by–example conditions, either directly in fields, or by invoking the built–in Query Where dialog and entering more complex conditions. For example, an operator might enter the value ‘10’ in a Department Number field to retrieve all employees in Department 10.

These conditions become the WHERE clause that Oracle Forms issues for the query. In the previous example, the block’s default SELECT statement would be modified to include the SQL clause WHERE DEPTNO=10. You can also modify the example record programmatically to further restrict a query by specifying additional conditions.

The SELECT statement, which specifies the selection criteria for fetching records into a block, has the following form:

```
SELECT select_list FROM <table_name>  
    [WHERE  
        [default_where_clause [AND]]  
        [column_condition [AND]] . . .  
        [query_where_condition]  
        [ORDER BY default_ordering | ORDER BY query_where_ordering]]
```

where

- `where` specifies the column names that correspond to the base table items in the block.
- `select_list` specifies the WHERE condition set by the block’s WHERE Clause property setting.
- `default_where_clause` specifies the value of an individual item as specified in the example record.
- `query_where_condition` specifies the WHERE condition set by the operator in the Query Where dialog.
- `default_ordering` specifies the ORDER BY condition set by the block’s ORDER BY Clause property setting.
- `query_where_ordering` specifies the ordering condition set by the operator in the Query Where dialog. Note that this parameter, if specified, takes precedence over default ordering.
The following table indicates the SELECT statement parameters that can be modified, the elements that correspond to the parameters (i.e., what the designer or operator must change in order to affect the SELECT statement), and who can modify the parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Element</th>
<th>Modifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_condition</td>
<td>example record</td>
<td>designer, operator</td>
</tr>
<tr>
<td>default_ordering</td>
<td>default ORDER BY clause</td>
<td>designer</td>
</tr>
<tr>
<td>default_where_clause</td>
<td>default WHERE clause</td>
<td>designer</td>
</tr>
<tr>
<td>query_where_condition</td>
<td>Query Where statement</td>
<td>operator</td>
</tr>
</tbody>
</table>

Default WHERE Clause

You can specify the default WHERE clause for a block’s SELECT statement by setting the WHERE Clause block property. Oracle Forms applies this WHERE Clause to every query issued from the block. Any additional query criteria specified by the operator in Enter Query mode or added programmatically augments, but does not replace, the block’s default WHERE clause.

You can programmatically change a block’s current WHERE clause by using SET_BLOCK_PROPERTY to set the Default_Where property.

Default ORDER BY Clause

You can specify the default ORDER BY clause for a block’s SELECT statement by setting the ORDER BY Clause block property. Oracle Forms applies the default ORDER BY clause to every query issued from the block. If the operator specifies an ORDER BY statement in the Query WHERE dialog, it overrides the block’s default Order By clause.

You can programmatically change a block’s ORDER BY clause by using SET_BLOCK_PROPERTY to set the ORDER_BY property.

Example Record

The columns included in the select list plus any additional conditions that the operator enters become the example record that is the basis for the SELECT statement that Oracle Forms issues to the database. Oracle Forms creates and initializes an example record whenever the following occurs:

- ENTER_QUERY, EXECUTE_QUERY, or COUNT_QUERY built-ins are invoked
- [Enter Query], [Execute Query], or [Count Query Hits] function keys are pressed
- Enter, Execute, or count Hits default query menu items are selected
The example record can affect the query SELECT statement in the following ways:

- The example record can contain (non–NULL) values for database and non–database items that are referenced in the WHERE clause.
- The example record can specify conditions for base table items. For example, it can specify that the query should only retrieve records where the value in the QUANTITY column is less than 100.
- Operators can enter item values directly into the example record while in Enter Query mode. Designers can enter item values by setting the Copy Value from Item item property and through programmatic assignment in a Pre–Query trigger. Note that changes made to the example record by the Pre–Query trigger override those values entered by an operator in Enter Query mode. This functionality allows you to screen illegal queries.

Operators can supply values for the Query Where statement by invoking the Query WHERE dialog in Enter Query mode and entering query conditions, using pattern matching and relational operators. The Query Where statement:

- uses as its WHERE clause any default WHERE clause that you specified in the default Where property for the block
- uses as its ORDER BY clause any default ORDER BY clause that you specified in the default Order By property for the block

The WHERE conditions specified by the Query Where statement supplement the conditions in the default WHERE clause and replace the conditions in the default ORDER BY clause. Only operators can specify the Query Where statement.
Enter Query Mode Restrictions

When a form is in Enter Query mode, Oracle Forms, by default, follows a modified set of processing rules. Specifically, Oracle Forms:

- does not perform any validation
- uses standard function key definitions (not those defined in key triggers)
- disables the following function keys:

  [Block Menu]   [Next Block]
  [Clear Form/Rollback]   [Next Record]
  [Commit]   [Next Set of Records]
  [Down]   [Previous Block]
  [Duplicate Item]   [Previous Record]
  [Duplicate Record]   [Scroll Down]
  [Insert Record]   [Scroll Up]
  [Lock Record]   [Up]

Fetch Processing Restrictions

When Oracle Forms populates a record with fetched values, it processes any Post–Change triggers defined as V2 triggers with the following restrictions:

- Oracle Forms ignores any attempt to change a database item in the record
- Oracle Forms does not execute any SELECT statement that only affects database items in the record
- Oracle Forms does not execute any SELECT statement that does not have an INTO clause
Validation

When Oracle Forms performs default validation on an item, it checks to make sure that the item conforms to all of the rules for that item. You can augment default validation by defining appropriate triggers. You can also control the extent and frequency of validation in a form by defining the Validation Unit form property. Oracle Forms has a validation process for each of the following objects:

- item
- record
- block
- form

Item validation consists of standard validation checks (refer to the section on “Standard Validation Checks,” later in this chapter) and the firing of any When–Validate–Item trigger. Item validation begins with a status check to determine whether the item is already valid. If the item is valid, the validation process stops. If the item is not already valid, the validation process continues.

Record validation consists of validating all items in the record followed by firing the When–Validate–Record trigger, if present. Like item validation, record validation begins with a status check to determine whether the record is already valid. If the record is already valid, the validation process stops. If the record is not already valid, the validation process continues.

Block validation consists of validating all records in the block. Form validation consists of validating all blocks in the form.

Validation Unit

The Validation Unit form property determines the scope of validation that applies to the form at runtime. The validation unit corresponds to a unit of data in Oracle Forms. This unit can be the item, record, block, or form. The validation unit is most often the item, which means that Oracle Forms validates data in an item when Oracle Forms navigates out of that item.

When Validation Occurs

Oracle Forms performs validation for the validation unit under the following conditions:

- when Oracle Forms navigates out of the validation unit, as a result of an operator pressing certain function keys or a trigger executing certain built–ins
- when the ENTER built–in is invoked or the operator presses [Enter]
When [Commit] is pressed or the COMMIT_FORM built-in is invoked, Oracle Forms performs validation for the form regardless of the validation unit. Validation does not occur in Enter Query mode. This behavior allows the operator to enter query criteria that Oracle Forms would otherwise consider invalid.

Status Checking
The first step in item and record validation is status checking. Status checking determines whether Oracle Forms must perform any additional validation, such as standard validation checks.

- If an item’s status is New or Changed, validation continues. If an item’s status is Valid, item validation stops.
- If a record’s status is Changed, validation continues. If a record’s status is New or Valid, record validation stops.

Item Validation States
Oracle Forms marks each item in a form as either New, Changed, or Valid.

When an Item is New
When a record is created, Oracle Forms marks every item in that record as New. This is true even if Oracle Forms populates an item with a default or copied value (via the Default Value or Copy item property).

When an Item is Changed
Oracle Forms marks an item as Changed under the following conditions:

- When an item is altered, Oracle Forms immediately marks it as Changed. This is true whether an operator types a value into an item or a trigger selects, copies, or defaults a value into an item.
- When any item in a New record is altered, Oracle Forms marks all of the items in the record as Changed.

   Note: If the current value in an item is re-entered into that item, GUI/character mode platforms will register a change and Oracle Forms will mark the item as Changed. However, block-mode platforms cannot register such a change. Block-mode platforms only register a change if the entered value is different than the current value.

When an Item is Valid
Oracle Forms marks an item as Valid under the following conditions:

- After Oracle Forms successfully validates an item, it marks the item as Valid.
- If validation changes the value of the item being validated (through a When-Validate-Item, When-Validate-Record, or Post-Change trigger), and validation succeeds, Oracle Forms marks the item as Valid and does not re-validate the item.
**Note:** This behavior is necessary to avoid validation loops. Be aware, however, that it creates the potential for an invalid value to be committed to the database.

- Oracle Forms marks as valid all items in records that are fetched from the database into the workspace. This is true even if a Post–Change trigger (fired during Fetch Records) changes item values.

- After Oracle Forms successfully commits data to the database, it sets all items in the form to Valid. Oracle Forms does not validate any changes caused by triggers during the commit transaction. Exceptions to this rule are explained in the section on “Changing Data During Commit Processing,” later in this chapter.

- Each item in a duplicated record (created by executing the DUPLICATE_RECORD built-in or pressing [Duplicate Record]) inherits the item state of its source. Therefore, any item that is a duplicate of a Valid item will also be Valid.
Oracle Forms marks each record in a form as either New, Changed, or Valid. Note that a record’s validation status is not equivalent to the record’s status as recorded by the SYSTEM.RECORD_STATUS system variable.

**When a Record is New**
Oracle Forms marks a record as New when the record is first created. This is true even if Oracle Forms populates the record via the Default Value or the Copy item property.

**When a Record is Changed**
Oracle Forms marks a record as Changed any time an item in the record is marked Changed.

**When a Record is Valid**
Oracle Forms marks a record as Valid under the following conditions:

- After Oracle Forms successfully validates all of the Changed items in a Changed record (changing their statuses to Valid), it marks the record as Valid. If all items in a Changed record do not pass validation, the record remains as Changed.
- After Oracle Forms successfully validates all of the New items in a New record (changing their statuses to Valid), it marks the record as Valid. If any item in a New record fails validation, Oracle Forms marks the record as Changed.
- Oracle Forms marks all records fetched from the database into the workspace as valid.
- After Oracle Forms successfully commits data to the database, it marks all records in the form as valid. Exceptions to this rule are discussed in the Section “Changing Data During Commit Processing” later in this chapter.
- A duplicate record (created by executing the DUPLICATE_RECORD built-in or by pressing [Duplicate Record]) inherits the status of its source. Therefore, any record that is a duplicate of a Valid record will also be Valid.
Validation of a New or Changed item includes standard validation checks. These checks assure that item data meets the requirements that you set via the following item properties:

- Format Mask
- Required
- Fixed Length
- Data Type
- Range (Low – High)
- Use LOV for validation

If you specify a format mask for a text item (via the Format Mask text item property), the standard validation checks involve checking the format mask status. Oracle Forms sets the format mask status whenever a value is entered into an item. The status can have one of two values:

**Valid**
This status indicates that the entered item value satisfies the specified format mask. As a result, item validation continues.

**Invalid**
This status indicates that the entered item value does not satisfy the specified format mask. As a result, item validation fails.
Commit Processing

Oracle Forms commit processing is the process that attempts to make the data in the database identical to the data in the form. This process involves posting data and then committing the data to the database.

Posting Data

Posting consists of writing deletes, inserts, and updates in the form to the database. Inserts, updates, and deletes are pending changes that involve any records which have been marked for insertion, update, or deletion since the last post during the current Oracle Forms session. Note that posting does not commit any changes to the database. Any data that you post to the database is committed to the database by the next [Commit] function key or COMMIT_FORM built-in that executes during the current Oracle Forms session. This data can also be rolled back via the [Clear Form] function key or CLEAR_FORM built-in. Posting always occurs during commit processing. However, posting can occur separately from a commit via the POST built-in. Note that there is no Oracle Forms function key that corresponds to this built-in.

When data is posted to the database, Oracle Forms processes inserts, updates, and deletes for all blocks in a form. Oracle Forms allows you to alter the default commit processing behavior of inserts, updates, and deletes through the use of commit time triggers.

Inserts

In Oracle Forms, an insert is the pending change of a record that has been marked for insertion as a new row in the database. Every block that has a record marked as an insert has a SQL INSERT statement associated with it that is executed when the insert is posted. The INSERT statement takes the following form:

```
INSERT INTO table [(column, column, . . .)]
    VALUES (value, value, . . .);
```

- `table` specifies the name of the base table for the current block that Oracle Forms is processing.
- `column` specifies the columns in the specified base table that corresponds to the database items in the record that Oracle Forms is processing. Note that if an item is marked as a derived column, it is not included in the column clause.
- `value` specifies the values that Oracle Forms will insert into the specified columns in the base table.

Updates

In Oracle Forms, an update is the pending change of a record that has been marked as an update to an existing row in the database. Every block that has a record marked as an update has a SQL UPDATE
statement associated with it that is executed when the update is posted. The UPDATE statement takes the following form:

```
UPDATE table
    SET (column=value, column=value, . . .)
    WHERE ROWID=rowid_value;
```

table specifies the name of the base table for the current block that Oracle Forms is processing.

column specifies all columns in the specified base table that correspond to the database items in the record that Oracle Forms is processing. Note that if an item is marked as a derived column, it is not included in the column clause.

value (s) specifies the values that Oracle Forms will insert into the specified columns in the base table.

rowid_value specifies the ROWID value for the row in the database that corresponds with the record that Oracle Forms is processing.

**Note:** The condition specified in the WHERE clause only specifies a ROWID value. This condition identifies the one correct row in the database that the database should update.

**Note:** If an operator does not have update privileges on a column and the Column Security block property is turned on, Oracle Forms does not include the column in the block’s UPDATE statement. Therefore, the block’s UPDATE statement can change from operator to operator. However, the statement remains the same for the duration of the operator’s Oracle Forms session.

**Note:** Oracle Forms uses the ROWID construct only when the Key Mode block property is set to Unique_Key (the default).
Deletes

A delete is the pending change of a record that has been marked for the deletion of its corresponding row in the database. Every block that has a record marked as a delete has a SQL DELETE statement associated with it that is executed when the delete is posted. The DELETE statement takes the following form:

```
DELETE FROM table WHERE ROWID=rowid_value;
```

table specifies the name of the base table for the current block that Oracle Forms is processing.

rowid_value specifies the ROWID value for the row in the database that corresponds with the record that Oracle Forms is processing except when running against a non-ORACLE database.

When Commit Processing Occurs

Commit processing occurs under the following conditions:

- when a trigger invokes the COMMIT_FORM built-in
- when an operator presses [Commit]
- when there are changes to database items in the current block since the last commit and a trigger invokes the CLEAR_BLOCK built-in (with the DO_COMMIT parameter)
- when there are changes to database items in any block in the current form since the last commit and a trigger invokes the CLEAR_FORM built-in (with the DO_COMMIT parameter)
- when an operator answers Yes to the alert

The alert appears under the following conditions:

- changes have been made to database items in the current block since the last commit and any of the following events occur:
  - CLEAR_BLOCK (with the ASK_COMMIT parameter or with no parameter)
  - COUNT_QUERY
  - ENTER_QUERY
  - EXECUTE_QUERY
  - NEW_FORM
- changes have been made to database items in any block in the current form since the last commit and any of the following events occur:
CLEAR_FORM (with the ASK_COMMIT parameter or with no parameter)

Leave the Form

**Note:** When a PL/SQL block issues a database commit from within Oracle Forms (via the SQL COMMIT statement), Oracle Forms commit processing occurs as if the COMMIT_FORM built-in had been invoked.

### Changing Data During Commit Processing

Commit processing performs validation and can potentially fire a number of triggers. As a result, a commit event can generate changes to database items. Depending on the type of trigger you use to change data during a commit, Oracle Forms might attempt to commit those changes to the database during the current commit event.

### Changing Data before Posting

Be cautious when using Pre-Delete, Pre-Insert, and Pre-Update triggers to make changes to database items. Database changes made in these triggers might be committed without being validated. For example, assume that a Pre-Insert trigger selects a value into a database item in the form:

- If the affected record has not already been processed for updates, the changes will be committed during the current commit event, but they will not be validated before being committed.
- If the affected record has already been processed for updates, the changes will not be committed during the current commit event and Oracle Forms will mark the records as having come from the database. If there is an attempt to update such a record before it is committed, an error will occur.

### Changing Data after Posting

Be cautious when using Post-Commit, Post-Delete, Post-Insert, and Post-Update triggers to make changes to database items. Database changes made in these triggers might not be committed during the current commit event. For example, assume that a Post-Insert trigger selects a value into a database item in the form:

- If the affected record is in the block that the current commit is currently processing or a block that the current commit already processed, Oracle Forms does not commit the changes during the current commit event.

However, Oracle Forms does mark the affected items and records as changed. If the current commit succeeds, Oracle Forms marks all items and records as Valid. This results in an
Replacing Default Commit Processing

You can replace the processing that Oracle Forms performs while posting data during a commit. The On-Insert, On-Update, and On-Delete triggers each replace standard Oracle Forms processing.

error during subsequent processing. Therefore, you should avoid making such changes.

- If the affected record is in a block that has not been processed yet by the current commit, Oracle Forms validates the changes and they are committed during the current commit event.
Locking

Locking tables and rows is an essential part of maintaining the consistency and integrity of your database. Locks prevent the inconsistencies that can arise when more than one operator attempts to change the same data at the same time. Oracle Forms uses locks to ensure that no changes are lost when multiple operators are updating the same data. When one operator updates a row, Oracle Forms locks that row, preventing other users from updating the same row until the initial operator commits or rolls back changes. ORACLE and Oracle Forms automatically provide the basic provisions of data locking. However, designers and operators can exercise control over when and how resources are locked. For a complete explanation of ORACLE locking methods and lock types, refer to the Oracle7 Server Concepts Manual.

Oracle Forms automatically attempts to obtain exclusive row locks in response to operator events and trigger and transaction processing. An exclusive row lock prevents other operators from updating or deleting a row and from acquiring any locks on that row. Note, however, that other users can still query a row that is exclusively locked.

Locks Caused by Operators

Oracle Forms attempts to place an exclusive lock on a row when an operator changes the value of a base table item in a record that corresponds to a row in the database, or as a result of the operator explicitly requesting a lock on such a record. The exclusive row lock statement takes the following form:

```
SELECT
FROM <all database items in the block>
WHERE ROWID = <table upon which current block is based>
FOR UPDATE OF <all database items in block>
```

Locks Caused by Triggers

When a trigger makes a change to a table through a Data Manipulation Language (DML) command, the database attempts to place an exclusive lock on the affected row. Note that locks caused by triggers are not managed by Oracle Forms. Oracle Forms attempts to place an exclusive lock on each row corresponding to a fetched record when a trigger uses the following built-ins:

- `DELETE_RECORD`
- `LOCK_RECORD`

As a general rule, any built-in that modifies a value in a queried record will cause Oracle Forms to perform the exclusive lock sequence.
### Methods for Locking Rows

Implicit locking and explicit locking are the two methods that Oracle Forms uses to guarantee that rows are locked during an update or delete. These methods of locking perform identical functions, but they differ as to when they perform these functions.

**Implicit Locking**

All Oracle Forms applications use implicit locking, which means that operators never have to manually lock rows. Through implicit locking, Oracle Forms tries to place an exclusive lock on a row as soon as the operator attempts to type a change into an item in the associated record. On block-mode terminals, Oracle Forms attempts to place the lock as soon as the operator presses a function key.

*Note:* When the Validation Unit is the item, Oracle Forms does not allow the change if it cannot acquire the lock. When the Validation Unit is the record, block or form, Oracle Forms allows the change and attempts to place an exclusive lock on the row during record validation.

**Explicit Locking**

Explicit locking means an operator or a trigger requests that Oracle Forms lock a particular row immediately. An operator can select Record→Lock from the default menu to explicitly lock the row associated with the current record. Oracle Forms also attempts to explicitly lock rows associated with records in a block when an application executes any of the following built-ins:

- `LOCK_RECORD`

**Failing to Obtain a Lock**

When there is resource contention on a multi-user system, Oracle Forms might not be able to obtain a requested lock on a table immediately. If this occurs, Oracle Forms attempts to obtain a lock a fixed number of times and then, if no lock is obtained, it requests (via the lock dialog) whether it should retry. The operator can respond Yes and wait for Oracle Forms to acquire the lock, or terminate the locking procedure by pressing CTRL–C or its equivalent. Failing to obtain a lock has serious implications. If one user has a table exclusively locked, other users can query records from the table, but cannot update or delete those rows. This condition will persist, without notice, until the resource becomes available. Therefore, is is recommended that you not use exclusive table locks in multi-user environments unless absolutely necessary.
When Locks are Released

Oracle Forms initiates the unlocking sequence under the following circumstances:

- the form commits successfully
- the form aborts
- Oracle Forms performs a full Oracle Forms rollback (not a rollback to a savepoint)
- logout

Oracle Forms maintains locks when it rolls back to a savepoint, as might happen after an unsuccessful commit. This behavior allows the operator to make any necessary changes and re-commit.

If a trigger explicitly locked a batch of rows via the FOR_UPDATE parameter, those rows need to be re-queried after a commit to re-lock all of the rows.

Note:

- Clearing a record or clearing a block does not release the locks on any corresponding rows.

- If you use a ROLLBACK or COMMIT statement in a trigger or a form-level procedure, Oracle Forms releases all locks. Oracle Forms interprets a ROLLBACK statement as a CLEAR_FORM built-in with no parameters. It interprets a COMMIT statement as a COMMIT_FORM built-in.

- Oracle Forms does not support a “raw” COMMIT (one which does not include the normal Oracle Forms commit processing). For example, Oracle Forms does not support a commit issued from an ORACLE Precompiler user exit or from a stored procedure.

Replacing Default Locking

In rare cases, you might want to replace or augment the default locking behavior of Oracle Forms. To do this, Oracle Forms includes the On-Lock trigger, which fires whenever Oracle Forms would attempt to obtain a lock. Whatever code you place in an On-Lock trigger completely replaces Oracle Forms default locking statements.
This chapter contains flowcharts that describe the fundamental Oracle Forms internal processes, and include the following elements:

- decision points
- trigger points
- conditions

The flowcharts in this chapter are provided to help you understand the Oracle Forms Runform default processing. Pay particular attention to the flowcharts in which Oracle Forms fires triggers. Those flowcharts illustrate the points at which you, as the designer, can affect the default processing, by way of triggers. Refer to Chapter 2, “Triggers,” in the Oracle Forms Reference Manual, Vol. 1, for information about specific triggers.
ABORT_QUERY

Start.

is there a query open in the current block?

yes

Close the query

Display message: FRM-40353: Query cancelled.

Stop with success.

no

Display message: FRM-40352: Last record of query retrieved.

Stop with failure.
Start.

Display the block menu

Accept operator input

Did the operator press [Exit/Cancel]?

yes

Navigate to the Form Level with validation

no

Stop with failure.

failure

Put Cursor At target block=selected block

success

Stop with failure.

success

failure
CALL_FORM

Start.

Is the form being called in QUERY ONLY mode?

yes (QUERY ONLY = on)

Are there changes in the current form?

no (POST ONLY = off)

Does the datasource support savepoints?

yes

Savepoint

no

Is the HIDE parameter specified?

yes

Hide the current form

no

(SWITCH_MENU = off)

Is the REPLACE_MENU parameter specified?

yes

(SWITCH_MENU = on)

Run the form with SWITCH MENU and QUERY MODE options and the parameter list passed to the CALL FORM built-in

failure

Stop with failure

success

Stop with success

Display message: FRM-40702: Cannot call form with changes to commit.
CALL_INPUT

Start.

Accept function key input from operator.

Process the Function Key

success or failure

no

Was the function key execution terminated by EXIT_FORM?

yes

Stop calling all triggers, if any, up to the one that contained the call to the last CALL_INPUT.

Stop with success.
Check Block for Query

Start.

Are queries allowed on this block?

yes

Display message: FRM-40360: Query not allowed here.

no

Stop with failure.

no

Is the block a control block?

yes

Display message: FRM-41003: This block does not correspond to a table. Function ignored.

Stop with failure.

Stop with success.
Check Block for Update

Start.

Is the current block a control block?

yes

no

Is the current block based on a non-updatable view?

yes

no

Have the records in this block been retrieved by a query FOR UPDATE statement and committed but not cleared?

yes

no

Is the form running as query-only?

yes

no

Display message: FRM-40602: Can't insert into or update data in a view.

Display message: FRM-40603: Records no longer reserved for update. Re-query to make changes.

Display message: FRM-40208: Form running in query-mode only. Can't change database fields.

Stop with success.

Stop with failure.
Check Item for Edit

Start.

Is the cursor in a prompt item? yes
no

Is the form in Enter Query mode? yes → Stop with success.
no

Does the item correspond to a database field? no → Stop with failure.
yes

Check the Record for Update

Is the Input Allowed item property turned on? no → Display message: FRM-40200: Item is protected against update.
yes

Does the current record correspond to a row in the database? no → Stop with success.
yes

Is the Update If Null item property turned on?
no → Stop with success.
yes

Was the fetched value of this item instance NULL? yes
no → Display a Message: message: FRM-40200: Item is protected against update.

Is the Update Allowed item property turned on? no → Stop with failure.
yes
Check Item for Edit (continued)

1

Is this a block mode environment?

yes

Lock the Row lock not required

no

Lock the Row lock required

failure

Stop with failure.

success

Stop with success.
Check Record Uniqueness

Start

Does an On-Check-Unique trigger exist for this block?

yes → Fire the On-Check-Unique trigger → Return with success

no → Construct and execute SQL statement to select row matching current row

Did the SQL statement identify a row in the database?

yes → Display message: FRM-40600: Row has already been inserted.

no → Return success (record is unique)

Return with failure
Check Record for Update

Start.

Check the Block for Update

failure

success

Is the record disabled due to a prior database rollback?

no

Has the record been updated by another user?

no

Stop with success.

yes

Display message: FRM:40656
Update can't be made due to prior rollback. Clear the record.

yes

Display message: FRM:40656
Record changed or deleted by another user.

Stop with failure.
CLEAR_BLOCK

Start.

Navigate to the Block Level with validation unless NO_VALIDATE is specified

Are there changes in the block?

- No
  - Post and Commit Transactions with NO_COMMIT, NO_VALIDATE or DO_COMMIT if specified

- Yes
  - Fire the When-Clear-Block trigger
  - Flush the current block

Put Cursor At target_block = current cursor block

Stop with success.  Stop with failure.
Start.

Check Item for Edit

success

Change all of the characters from the cursor position to the rightmost character of the item value to blank characters.

Mark Items and Records as Changed

stop with success.

failure

Stop with failure.
CLEAR_FORM

Start.

Navigate to the Form Level with validation unless NO_VALIDATE specified

Are there changes in the form?

Post and Commit Transactions with DO_COMMIT, NO_COMMIT, or NO_VALIDATE if specified current_operation = COMMIT

Is the current form a called form?

Is the current form running in post-only mode?

Is FULL_R RollBACK specified?

Display message: FRM-40739: CLEAR_FORM with FULL_RollBACK not allowed in post_only form.

Flush the form

Put Cursor At no target

Did the rollback result in failure?

Stop with success.

Stop with failure.
CLEAR_ITEM

Start.

Check Item for Edit

success

Change all of the characters in the item to blank characters.

Mark Items and Records as Changed

Stop with success.

Stop with failure.

failure
CLEAR_RECORD

Start.

Is the last record of the block (in the block's list of records) visible?

yes

no

Fetch Records
n = 1

success

Fire the When-Remove-Record trigger.

success or failure

Navigate to the Block Level without validation

failure → Stop with failure.

success

Flush the current record from the block, decrementing record numbers of all records after the cleared record.

Is there a record with a higher sequence number than the cleared record?

yes

no

Is there a record number with a lower sequence number than the cleared record?

yes

no

Put Cursor At
target_block = current cursor block

failure

success

Stop with failure.

Stop with success.

Make the subsequent record the target record

target_record = subsequent record

Make the previous record the target record

target_record = previous record

Put Cursor At
target_block = current cursor block (as described above)

failure

success

Stop with failure.

Stop with success.
Close the Query

Start

- Remove any records in the block’s fetch buffer
- Update count and vertical scrolling lamps on the console

- Is there an open query on the block?
  - Yes: Fire the On-Close trigger if one exists, ignore failure, mark the block as not having an open query.
  - No: Stop with success.

Stop with success.
COMMIT_FORM

Start.

Navigate to the Form Level without validation

failure

success

Validate the Form

failure

success

Post and Commit Transactions

failure
current_operation = COMMIT

success

Put Cursor At

failure
target_block=current cursor block
target_record=current cursor record
target_field=current cursor field

success

Stop with success.

Stop with failure.
Start.

1. Is the destination a global variable? If yes, does the variable exist? If yes, copy the source value into the global variable. If no, create the variable and copy the source value into it. If no, check the record for update. If failure, stop with failure. If success, copy the source value into the item. If the item too short to hold the entire value, causing the item to truncate the value? If no, display message: FRM-40819: System variable is not modifiable. If yes, stop with failure.

2. Is the destination an item? If yes, does the item correspond to a database field? If yes, check the record for update. If failure, stop with failure. If success, copy the source value into the item. If no, display message: FRM-40831: COPY value too long for destination item.

3. Is the destination a read-only system variable (i.e., all system variables except SYSTEM.MESSAGE.LEVEL)? If yes, display message: FRM-40819: System variable is not modifiable. If no, copy the source value into the system variable. If stop with success.
COUNT_QUERY

Start.

Check the Block for Query

success

CLEAR_BLOCK

success

Initialize the example record.

failure

Prepare the query

success

Is there an On-Count trigger?

yes

Fire the On-Count trigger

no

Identify the number of rows the query will retrieve.

Display message: FRM-40355: Query will retrieve <number> records.

Put Cursor At target_block = current cursor block

failure

Stop with failure

success

Stop with success.
CREATE_RECORD

Start.

Navigate to the Block Level with validation

success

failure

no

Are inserts allowed on this block?

yes

Create a new record in the block after the current cursor record. Increment by 1 the record numbers of all subsequent records.

Initialize the new record.

Fire the When-Create-Record trigger.

Remove created record

failure

success

Put Cursor At target_record = new record

failure

success

Stop with success.

Stop with failure
DEFAULT_VALUE

Start.

Is the destination a global variable?

yes → Does the variable exist?

no → Create the variable.

yes → Is the value of the destination NULL?

no → Copy the source value into the global variable.

yes → Is the value of the destination NULL?

no → Copy the source value into the global variable.

yes → Is the destination an item?

no → Does the item correspond to a database field?

yes → Check the Record for Update

failure → Stop with failure.

success → Copy the source value into the item.

no → Display message: FRM-40819: System variable is not modifiable.

yes → Is the destination a read-only system variable?

no → Copy the source value into the system variable.

yes → Stop with failure.

no → Stop with success.
DELETE_RECORD

Start.

Does the record correspond to a row already in the database? 

no -> Are deletes allowed in this block? 

no -> Display message: FRM-41049: Cannot delete records from block. 

yes -> CLEAR_RECORD 

success -> Stop with success 

failure -> Stop with failure. 

yes -> Check the Record for Update 

failure -> Stop with failure. 

success -> is the last record of the block (in the block's list of records) visible? 

no -> Fetch Records 

n = 1 

yes -> Fire the When-Remove-Record trigger. 

success or failure -> Navigate to the Block Level with validation 

failure -> Stop with failure. 

success -> Remove the current record from the block and add it to the list of the deleted records. Decrement record numbers of all records after records to be deleted.

1
DELETE_RECORD (continued)

1

Is there a record with a higher sequence number than the deleted record?

yes: Put Cursor At
target_record = subsequent record

失败

Stop with failure.

no: Put Cursor At
target_block = current cursor block

Success

Stop with success.

Is there a record with a lower sequence number than the deleted record?

yes: Put Cursor At
target_record = previous record

Success

Stop with success.

no: Stop with failure.
DO_KEY

Start.

Does the referenced built-in routine exist?  

yes  

is there a corresponding key trigger?  

yes  

Fire the key trigger  

Stop with success.

no  

Stop with failure.

no  

Execute the packaged procedure  

Stop with success.

Stop with failure.
Start.

Is the current cursor record a blank record?
  yes → Display message: FRM-40102: Record must be entered or deleted first.
  no → Is there a record in the block with a higher sequence number?
    no → Is a query open in the current block?
      no → Did the previous DOWN or NEXT_RECORD close the query?
        no → Stop with failure.
        yes → Display message: FRM-40352: Last row of query retrieved.
              Stop with success.
      yes → Fetch Records
        n = 1
          failure → Stop with failure.
          success → Navigate to Block Level with validation
                        failure → Stop with success.
                        success → Put Cursor At
                                      target_record = subsequent record
                                      failure → Stop with failure.
                                      success → Stop with success.
Start.

Check Item for Edit

success

Is there a record in the block's list of records with a lower sequence number?

no

Display
message: FRM-41803:
No previous record to copy value from.

yes

Give the current item the same value as the instance of this item in the previous record.

Mark Items and Records as Changed

success

Stop with success.

Stop with failure.
DUPLICATE_RECORD

Start.

Is the current record the first record in the block's list of records?
  no

Does the current record correspond to a row in the database?
  no

Are inserts allowed in this block?
  yes

  Display message: FRM-41803: No previous record to copy a value from.

  Display message: FRM-41802: Duplicate record function allowed on new records only.

  Display message: FRM-41051: Cannot insert records into block.

Copy the value of each item in the previous record to the corresponding items in the new record.

Copy the validation status (i.e. Changed, New, or Valid) of each item in the previous record to the corresponding items in the new record.

Mark the new record as an insert.

Fire the When-Database-Record trigger.

Stop with success.
Start.

What is the current validation unit?

form

Navigate to the Form Level without validation

Validate the Form

Put Cursor At target_item = current cursor item

Stop with success.

success

failure

success or failure

block

Navigate to the Block Level without validation

Validate the Block

Put Cursor At target_item = current cursor item

Stop with success.

Stop with failure.

record or item
ENTER (continued)

1. Navigate to the Record Level without validation
   - Success
   - Failure

   Validate the Record
   - Success
   - Failure

   Put Cursor At target_item = current cursor item
   - Success
   - Failure

   Stop with success.

2. Navigate to the Item Level without validation
   - Success
   - Failure

   Validate the Item
   - Success
   - Failure

   Put Cursor At target_item = current cursor item
   - Success
   - Failure

   Stop with success.
   Stop with failure.
Enter the Block

Start.

Make the block the navigation unit.

Does the block consist of exactly one New record?

no

block or form

What is the validation unit?

Item or record

Initialize a record

Set n equal to 1

Initialize a record

Fire the When-Create-Record trigger

Failure Fire the When-Create-Record trigger.

success

1

Set n equal to n+1

yes

Is n greater than the number of records the block can display?

no

What is the validation unit?

Item, record or block

Fire Pre-Block trigger

Failure Make the form the navigation unit.

success

Is the cursor currently in an item?

no

Exit the form.

yes

Is the cursor within the navigation unit?

Set the cursor to "undefined"

no

Stop with success.

Put the Cursor At target_item = current cursor item

Exit the form.
Enter the Form

Start.

Make the form the navigation unit

Fire the Pre-Form trigger

success

failure

Make the navigation unit "outside"

Is the cursor currently in an item?

yes

no

Exit the form.

Is the cursor within the navigation unit?

yes

no

Put Cursor At

*target_item = current cursor item*

success

Exit the form.

Stop with success.
Enter the Item

Start.

Make the item the navigation unit

What is the validation unit?

- Item

- Failure
  - Fire the Pre-Text-Item trigger
  - Make the containing record the navigation unit
  - Is the cursor currently in a item?
    - No → Exit the form.
    - Yes
      - Is the cursor within the navigation unit?
        - Yes → Set the cursor to "undefined"
        - No
          - Put Cursor At target_item = current cursor item
            - Success
        - Exit the form.

Stop with success.
ENTER_QUERY

Start.

Is the form already In Enter Query mode?

yes

Has a query for this block been executed during the current session?

no

Clear the example record.

no

success

Check the Block for Query

failure

Stop with failure.

no

success

CLEAR_BLOCK

Are there any queryable items in this block?

no

Display message: FRM-40302: Cannot enter a query. No items are queryable.

yes

Put the form in Enter Query mode

Stop with success.

Stop with failure.
Enter the Record

Start.

Make the record the navigation unit

What is the validation unit?

block or form

item or record

Fire the Pre-Record trigger

success

failure

Make the containing block the navigation unit

Is the cursor currently in a item?

no

Exit the form.

yes

Is the cursor within the navigation unit?

no

Put Cursor At
target_item = current cursor item

success

yes

Set the cursor to "undefined"

Exit the form.

Stop with success.
Enter the Value into an Item

Start.

Check Item for Edit

success

Put the new value into the item instance

failure

Is there a format mask specified for this item?

no

Does the item value satisfy the format mask?

no

Set the item's mask status to invalid

yes

Set the item's mask status to Valid

Is the current operation ENTER_QUERY or DUPLICATE_RECORD?

yes

Mark Items and Records as Changed

no

Stop with success.

Stop with failure.
EXECUTE_QUERY

Start.

Execute the Query

- Success: Stop with success.
- Failure: Stop with failure.
Execute the Query

Start.

Check the Block for Query

success

Navigate to the Block Level with validation

success

Are there changes to commit in the block?

yes

Post and Commit transactions

current_operation = COMMIT

failure

success

Open the Query

failure → Stop with failure.

success

Were any records fetched?

yes

Put Cursor At

target_record = first record in the block

success

Stop with success.

failure

no

Display message: FRM-40301: Query caused no records to be retrieved.

Put Cursor At
target_block = current cursor block

Stop with failure.
EXECUTE_TRIGGER

Start.

Is there an appropriate user-named trigger?

- yes
  - Fire the user-named trigger
  - success: Stop with success.
  - failure: Stop with failure.

- no
  - Display message: FRM-40504; ORACLE error - unable to execute a <trigger type> trigger.
  - Stop with fatal error.
EXIT_FORM

1. Start.

- Navigate to the Form Level with validation unless NO_VALIDATE specified
  - success
  - failure

2. Are there changes in the form?
   - no
   - yes

   - Post and Commit Transactions with DO_COMMIT, NO_COMMIT, or NO_VALIDATE if specified current_operation = COMMIT
     - success
     - failure

   - Is the rollback option TO_SAVEPOINT or FULL_ROLLBACK?
     - no
     - yes

     - ROLLBACK_FORM
     - Flush the form
     - Put Cursor At outside
     - Stop with success.
     - Stop with failure.

   - success
   - failure
Fetch Records

Start.

Is the current operation "DELETE_RECORD", "DOWN", "NEXT_RECORD", "NEXT_SET", or "SCROLL_DOWN"?
- Yes
- No

Are there rows on the waiting list that have not been placed on the block's list of records?
- Yes
- No

Is the current operation "CLEAR_RECORD"?
- Yes
- No

Close the query

Set the RECORDS_TO_FETCH block property to 1

Is there an On-Fetch trigger for this block?
- Yes
- No

Fire the On-Fetch trigger

Stop with failure

Look at the first such row on the waiting list

Create a record at the end of the block's list of records

Read values from the row into the record's database items

Look at the first row in the buffer

Fetch RECORDS_TO_FETCH records from the database to the block's waiting list
Fetch Records (continued)

1

Is there a non-NULL item in the record? no
yes

Look at the first non-NULL item in item sequence

Fire the Post-Change trigger (Note: See Query Processing Restrictions for details)

Is there another non-Null item in the record with a higher sequence number? yes no

Look at the item

Mark the record and all of the items as Valid

Fire the Post-Query trigger fail success

Flush the record from the block

Has the record been modified? yes no
Validate the Record

Stop with failure.

Is array processing on? yes no
Synchronize the screen

Is there another row in the buffer? yes no
Look at the next row

Is there an additional selected row in the database? yes no
Close the query

Stop with failure.
Fetch Records (continued)

1. **Is the current operation CLEAR_RECORD, DOWN, DELETE_RECORD, LAST_RECORD or SCROLL_DOWN?**
   - yes
   - no

2. **Is the block full?**
   - yes
   - no

   3. **Are there additional rows on the waiting list?**
   - yes
   - no

   4. **Are there additional selected rows in the database?**
   - yes
   - no

   5. **Look at the next row**

   6. **Synchronize the screen**

   7. **Are there additional selected rows on the waiting list or in the database?**
   - yes
   - no

   **Close the query**

Stop with success.
FIRST_RECORD

Start.

Navigate to the Block Level with validation

success
Stop with failure.

failure

Is the cursor in a blank record?

no
Remove the current record from the block.

yes

Are there any records in the block?

yes

Put Cursor At target_record = first record in block

failure
Stop with failure.

no

Put Cursor At target_record = current cursor block

failure
Stop with failure.

Stop with success.
Generate Sequence Number

Start

no

Is there an On-Sequence-Number trigger for this item?

yes

Select the sequence number from the database → Stop with failure

Fire the On-Sequence-Number trigger

Stop with success
GO_BLOCK

Start.

Does the referenced block exist?
  yes
  no Stop with fatal error.

Is the referenced block enterable?
  yes
  no

Display message: FRM-40106: No enterable items in destination block.

Navigate to the Form Level with validation

Put Cursor At target_block = referenced block

Stop with success.

Stop with failure.
GO_ITEM

Start.

Does the referenced item exist?  
yes

Is the referenced item a displayed item?  
no  
Display message: FRM-41090: Cannot navigate to display item.

no  
Stop with fatal error.

yes  
Is the referenced item the same as the current cursor item?  
no  
Stop with fatal error.

yes  
Is the referenced item in the current cursor block?

yes  
Navigate to the Record Level with validation  
success

Stop with failure.

no  
Navigate to the Form Level with validation  
success

Stop with failure.

no  
Put Cursor At target_item = referenced item  
success

Stop with success.
GO_RECORD

Start.

Is the specified record number less than zero?
  yes → Make the first record in the block the target record
  no

Is the specified record number less than the current record number?
  yes → Make the specified record the target record
  no

Fetch Records
  n = specified record number minus current record number

  success

  Make the specified record the target record

  failure → Make the last record in the block the target block

Navigate to the Block Level with validation

  failure

  success

  Put Cursor At target_record = target record identified above

    failure

    success

    Stop with success.

    Stop with failure.
HOST

Start.

Is NO_SCREEN specified?

yes

no

Clear the terminal screen and place the terminal in normal interactive mode

yes

Is the specified host command NULL?

no

Execute the operating system command

Pass control to the operating system in a newly created interactive process or "shell"

Execute operating system commands until the operator logs out

Is NO_SCREEN specified?

yes

no

Remove the terminal from interactive mode

Prompt and Answer prompt: Press Return to return to Oracle Forms, until: any function key.

Redraw the screen the next time the screen synchronizes

success

What was the return code from the last operating system command?

failure

Stop with success.

fatal error

Stop with fatal error.

Stop with failure.
LAST_RECORD

Start.

Is the query open in the current block?

no

yes

Fetch Records
n = number of selected rows left in the database for the query

success

Navigate to the Block Level with validation

failure

success

Put Cursor At
target_record = last record in block

failure

success

Stop with success.

Stop with failure.
Leave the Block

Start.

What is the validation unit?

- field, record, or block

Is the current operation

- CLEAR_RECORD, or
- DUPLICATE_RECORD?

- yes

  Fire the Post-Block trigger
  
  - success
  
  - failure

- no

  Validate the Block
  
  - failure
  
  - success

  Fire the Post-Block trigger
  
  - failure
  
  - success

- no

  Did the Post-Block trigger change any fields in the block?

  - yes

    Validate the Block
    
    - failure
    
    - success

    Make the form the navigation unit
    
    Stop with success.

  - no

    Leave Unit Error Processing
    
    Stop with failure.
Leave the Form

Start.

Fire the Post-Form trigger

success

Make the navigation unit "outside"

Stop with success.

Stop with failure.

Leave Unit Error Processing

failure
Leave the Item

Start.
- Terminate editing of the item
- Remove the help message if it is on the message line

What is the validation unit?
- Item
  - Is the current operation CLEAR_RECORD or DUPLICATE_RECORD?
    - no
      - Is navigation attempting to leave a new item via anything other than NEXT_ITEM or NEXT_KEY?
        - no
          - Fire the Post-Text-Item trigger
            - failure
            - success
        - yes
          - Is NO_VALIDATE specified?
            - no
              - Validate the Item
                - failure
                - success
            - yes
              - Fire the Post-Text-Item trigger
                - failure
                - success
  - yes
    - Fire the Post-Text-Item trigger
      - failure
      - success

Did the Post-Text-item trigger change the item?
- yes
  - Is NO_VALIDATE specified?
    - no
      - Validate the Item
        - failure
        - success
    - yes
      - Leave Unit Error Processing
        - Stop with success.
- no
  - Make the containing record the navigation unit
    - Stop with success.
Leave the Record

Start.

What is the validation unit?

item or record

Is the current operation CLEAR_RECORD, or DUPLICATE_RECORD?

yes

Fire the Post-Record trigger

failure

no

Is NO_VALIDATE specified?

no

Validate the Record

failure

success

Fire the Post-Record trigger

success

no

Did the Post-Record trigger change any items in the block?

yes

Is NO_VALIDATE specified?

no

Validate the Record

failure

success

no

Make the containing block the navigation unit

Stop with success.

Stop with failure.

Leave Unit Error Processing

failure

success
Leave Unit Error Processing

Start.

Is there an error location? yes → Make the error block, record and item the target block, record and item
no

Is the cursor defined? yes → Make the current cursor block, record and item the target record, block and item
no

Set the target block, record and field to NULL

Is the target outside of the current unit? yes → 1
no

Is a target block defined? yes → Give abort message: Form aborted: Failure prevents leaving the form
no

Give abort message: Form aborted: Can’t get back to starting place.

Abort the form.
Leave Unit Error Processing (continued)

1

Is navigation attempting to exit the form?

yes

Prompt and Answer
prompt: Do you want to quit?
until: any function key.

What was the operator's response?

yes

Abort the form.

no

anything other than yes

Put Cursor At

| target_block = current target block |
| target_records = current target record |
| target_item = current target item |

Stop with success.
LOCK_RECORD

Start.

yes
Is the LOCK_RECORD function being invoked from a Pre- or Post-Form trigger? (i.e., Is there no current block?)

no

Check Record for Update

failure

success

Lock the Row

failure

success

Stop with success.

Stop with failure.
Lock the Row

Start.

Is the block locking mode delayed? no → Stop with success.

Is the current record associated with a row in the database? no → no

yes → Does the operator already have an exclusive lock on the row or table?

no → Stop with success.

yes → Is there an On-Lock trigger? no → Can looking lock the row?

yes → Stop with success.

no → Stop with failure.

yes → Is a lock required? no → Stop with success.

yes → Attempt to lock the row

failure → Retry the lock

success → Lock the row.

1

Can looking lock the row?

no → Stop with failure.

yes → Is a lock required?

no → Stop with success.

yes → Attempt to lock the row

failure → Display a dialog requesting whether the user wants to continue

no → Stop with failure.

2
Lock the Row (continued)

1. Lock the row
   - Yes: Display message: FRM-40501: ORACLE error-unable to reserve record for update or delete.
   - No: Has the row been changed or deleted by another operator?

2. Has the row been changed or deleted by another operator?
   - Yes: Record changed or deleted by another user.
   - No: Did an ORACLE error prevent the row from being locked?

3. Did an ORACLE error prevent the row from being locked?
   - Yes: Display message: FRM-40501: ORACLE error-unable to reserve record for update or delete.
   - No: Stop with success.

Stop with failure.
LOGON

Start

Is Oracle Forms already logged on?

no

Fire the Pre-Logon trigger

success

Does a user connection already exist?

no

Does an On-Logon trigger exist?

no

Is there a username/password and is the logon_screen parameter off?

no

Display logon screen

yes

Fire the On-Logon trigger

Return with failure

no

Return failure after third attempt

success

Stop with failure

Display message: FRM-40029
Already logged on. Must logout before changing connections.

1
LOGON (continued)

1

Was an Oracle or SQL*Connect connection established?

yes

Reset savepoint counter

Set column security for all blocks in form

Open four shared cursors

Start SQL tracing if requested

Stop with failure

Fire Post-Logon trigger

Stop with failure

If connected to Oracle7, disable commits and rollbacks in stored procedures

Stop with success
LOGOUT

Start

Fire the Pre-Logout trigger

If Oracle Forms is using a user established connection, re-enable commits in stored procedures

Closed shared cursors

Is there an On-Logout trigger?

yes

Fire the On-Logout trigger

Stop with failure

no

Logout

Stop with failure

Fire Post-Logout Trigger

Print cursor statistics if requested

Return success
Mark Items and Records as Changed

Start.

Mark the item as Changed.

Is the record status New?

no

Mark every New field in the record as Changed

Yes

Does the record contain a database field that is marked as Changed?

no

Fire the When-Database-Record trigger success or failure

yes

Is the record marked as an insert or an update?

no

yes

Does the record correspond to a row in the database?

yes

Mark the record as an update

no

Mark the record as an insert

Stop with success.
Master-Detail Coordination

Start

Is Oracle Forms in Enter Query Mode?
  yes → Return with success
  no

Is the event coordination causing?
  yes →
  no

Is the event a mouse event?
  yes →
  no

Is the event a scroll event on the non-current block?
  yes →
  no

Is the target record the current record of the target block?
  yes →
  no

Make the target block the current block

Is the current block a master block?
  yes → Return with success
  no
Master-Detail Coordination (continued)

1. Set the coordination phase to clear
2. Set the coordination block and coordination operations

Is the event a CLEAR_RECORD?

- Yes: Insert an On-Populate -Details trigger on the runform stack after the current instruction
  - Put the On-Clear-Details trigger on the runform stack so it occurs before the instruction
  - Insert the On-Clear-Details trigger in runform stack after instruction

- No:
  - Is the event a DELETE_RECORD?
    - Yes: Put the On-Check-Delete-Master trigger on the runform stack
    - No: Process runform (event stack)

What is the coordination phase?

Clear

Is the current event the event that initiated the coordination?

- Yes: Transition to populate phase and recurse
- No:
  - Insert an On-Clear-Details trigger for this block after the current instruction
  - Insert an On-Populate -Details trigger on the runform stack after the current instruction

Populate
Navigate to the <Navigation Unit> Level

Start.

1. Is the target unit different from the navigation unit?
   - yes
   - no Stop with success.

2. Is the navigation unit larger than the target unit?
   - yes
   - no

What is the navigation unit?

- Item
  - Leave the Item with/without validation as specified by the current event
  - success

- record
  - Leave the Record with/without validation as specified by the current event
  - success

- block
  - Leave the Block with/without validation as specified by the current event
  - success

- form
  - Leave the Form with/without validation as specified by the current event
  - success

2. What is the next smaller sub-unit of the navigation unit that contains a target?

- Item
  - Enter the Item
  - success

- record
  - Enter the Record
  - success

- block
  - Enter the Block
  - success

- form
  - Enter the Form
  - success

1
NEW_FORM

Start.

Is the specified form name NULL?  

yes

no

Make the specified form the target form

Leave the Form

failure

success

Stop with success.  Stop with failure.
NEXT_BLOCK

Start.

Navigate to the Form Level with validation

failure → Stop with failure.

success

Is the next block specified?

yes → Make the specified block the target block

no

Is there an enterable block in the form with a sequence number higher than the current block?

yes → Make the enterable block with the next higher sequence number the target block

no

Is there an enterable block in the form with a sequence number lower than the current block?

yes → Make the enterable block with the lowest sequence number the target block

no

Make the current cursor block the target block

Put Cursor At target_block = target block identified above

failure → Stop with failure.

success → Stop with success.
NEXT_ITEM

Start.

Is the current item the last enterable item in the block?

Yes

Is the navigation style SAME_RECORD?

Yes

Is the navigation style CHANGE_BLOCK?

No

Navigation style is CHANGE_RECORD. Convert event to NEXT_RECORD and stop with success.

No

Navigate to the Record Level with validation

failure

Stop with failure.

success

Is there an enterable item in the block with a higher sequence number than the current item?

Yes

Make the enterable item with the next higher sequence number the target item

No

Is there an enterable item in the block with a lower sequence number than the current item?

Yes

Make the enterable item with the lowest sequence number the target item

No

Make the current cursor item the target block
NEXT_ITEM (continued)

1

Is the validation unit the item?

yes

Are there non-enterable items with sequence numbers "between" (wrapping from last item to first if necessary) the current cursor item and the target item?

no

Identify the first such non-enterable item "between" the current cursor item and the target item

Valdate the Item

success or failure

Is there another non-enterable item "between" the current cursor item and the target item?

yes

Identify the next such non-enterable item in sequence order (wrapping if necessary)

Put Cursor At target_item = current target_item

Stop with success.

failure

Stop with failure.
Start.

Navigate to the Record Level with validation

success

Is there an enterable, primary key item in the current record with a higher sequence number than the current item?

no

Is there an enterable, primary key item in the current record with a lower sequence number than the current item?

no

Make the current cursor item the target block

no

Is the validation until the item?

yes

Are there non-enterable items with sequence numbers "between" (wrapping from last item to first if necessary) the current cursor item and the target item?

yes

Identify the first such non-enterable item "between" the current cursor item and the target item

no

Make the enterable, primary key item with the next higher sequence number the target item

yes

Make the enterable, primary key item with the lowest sequence number the target item

Stop with failure.
NEXT_KEY (continued)

1

Validate the Item

success or failure

Is there another non-enterable item 'between' the current cursor item and the target item?

no

yes

Identify the next such non-enterable item in sequence order (wrapping if necessary)

2

Put Cursor At target_item = current target item

failure

Stop with success.

Stop with failure.
NEXT_RECORD

Start.

Is the current cursor record a blank record? yes
no

Is there a record in the block with a higher sequence number? yes
no

Is a query open in the current block? yes
no

Did the previous DOWN or NEXT_QUERY close the query? yes
no

Display message: FRM-40102: Record must be entered or deleted first.

Display message: FRM-40352: Last row of query retrieved.

Stop with success.

Fetch Records
n = 1

success

Navigate to the Block Level with validation

success

failure

Put Cursor At target_record = subsequent record

success

Stop with success.

Stop with failure.

Stop with success.
Open the Query

Start.

- Fire the Pre-Query trigger
  - success
  - failure

Prepare the Query and select for UPDATE
- If the query is an UPDATE query
  - failure
  - Flush any example record from the block

Is there an On-Select trigger?
- yes
  - Fire the On-Select trigger
  - failure
- no
  - Identify in the database the rows that the query SELECT statement will retrieve
  - Fire the Post-Select trigger

Fire the When-Clear-Block trigger
- success or failure

Flush any example record from the block

- Is the current operation ENTER_QUERY [ALL RECORDS] or EXECUTE_QUERY [ALL RECORDS]?
  - yes
    - Set <number of records to fetch> to the number of selected rows in the database
  - no
    - Is array processing on?
      - yes
        - Set <number of records to fetch> to 1
      - no
        - Is the array size larger than the number of selected rows in the database?
          - yes
            - Set <number of records to fetch> to the array size
          - no
            - Set <number of records to fetch> to the number of selected rows in the database

Fetch Records \( n = \) <number of records to fetch>

- success
  - Stop with success.
- failure
  - Stop with failure.
POST

Start.

Navigate to the Form Level without validation

success

Validate the Form

success

Post and Commit Transactions
current_operation = POST

failure

Put Cursor At

target_block = current cursor block
target_record = current cursor record
target_field = current cursor field

success

Stop with success.

Stop with failure.
Post and Commit Transactions

Start.

Is NO_COMMIT or NO_VALIDATE specified? yes \rightarrow Stop with success.

no

Is DO_COMMIT specified? yes

Is the form running in post-only mode? no \rightarrow Stop with success.

no

yes

Prompt and Answer prompt: Do you want to post the changes you have made? until: any function key

Prompt and Answer prompt: Do you want to commit changes? until: any function key

What was the operator's response? yes

no

Stop with success.

Stop with failure.

Is the form running in query-only mode? yes \rightarrow Stop with fatal error.

no

Are you trying to commit while running in post-only mode? yes \rightarrow Stop with fatal error.

no
Post and Commit Transactions (continued)

1. Validate the Form
   - failure: Note error location → Stop with failure.
   - success
     - Are there any changes in the form to post or commit?
       - no → 6
       - yes
         - Savepoint
           - error: Stop with failure.
           - no error
             - Fire the Pre-Commit trigger
               - failure: Stop with failure.
               - success
                 - Display a Message message: Working...

10. Look at the first block in sequence

Validate the Block
   - failure: Note error location. → Handling the Error
   - success
     - Does the block have a base table or is it a transactional control block?
       - no → 2
       - yes
         - Is the block based on a non-updatable view and are there no transactional triggers defined for the block?
           - yes → 7
           - no → 2
Post and Commit Transactions (continued)

2. Are there any changes (inserts, updates, or deletes) in the block?
   - Yes
   - No
     7. Have any records in this block been deleted from the workspace since the last POST or COMMIT FORM?
       - Yes
         Look at the record that's most recently been deleted from the workspace and that hasn't already been deleted from the database
         - Fire the Pre-Delete trigger
           - Success
           - Fire the On-Delete trigger for the block
             - Success
             - Are there any more deletes to process for this block?
               - Yes
               - Fire the Post-Delete trigger
                 - Success
                 - No
               - No
                 7. Have any records in this block been inserted or updated since the last POST or COMMIT FORM?
                   - Yes
                     Look at the first record in the block in sequence order
                     3
                   - No
                     Set the block as the error location
                     Handling the Error

   - No
     7. Have any records in this block been deleted from the workspace since the last POST or COMMIT FORM?
Post and Commit Transactions (continued)

1. Is the record disabled due to a prior rollback?
   - yes
   - no

2. Is the record an INSERT or an UPDATE?
   - INSERT
   - UPDATE
   - neither

3. Display message: FRM-40656: Update can't be made to database due to prior rollback. Clear the record.

4. Fire the Post-Forms-Commit trigger

5. Set the record as the error location

6. Is there a block in the form with a higher sequence number?
   - yes
   - no

7. Is there another record in the block with a higher sequence number?
   - yes
   - no

8. INSERT the Record
   - UPDATE the Record
   - Handling the Error

9. Look at the record

10. Look at the block
Post and Commit Transactions (continued)

4

Commit

Is there an On-Commit trigger?

no

Fire the On-Commit trigger

failure

Commit the transaction to the database. This action releases all the locks held by the operator

success

Fire the Post-Database-Commit trigger

failure

Handling the Error

6

Is the current operation COMMIT or POST?

POST

no

Handling the Error

success

CLOSE_AT_COMMIT?

yes

Invalidate all open cursors

Re-open all shared cursors

no

Remove all deleted records from the workspace

Mark all inserted records as database records

Clear all Reserved, Insert, and Update record flags

Mark any changed items and records as Valid
Post and Commit Transactions (continued)

If a query with the FOR_UPDATE parameter was open for the block, mark the block as non-updatable

Determine if database item instances in insert or update records are editable, based on Update Allowed and Update If Null properties

Is the current operation POST or COMMIT?

Was the last POST or COMMIT_FORM?

POST

Have any changes been committed since the last POST or COMMIT_FORM?

no

Display message: FRM-40406: Transaction complete.<number> records posted: all records committed.

yes


COMMIT_FORM

Have there been changes committed since the last COMMIT_FORM?

no

Display message: FRM-40400: Transaction complete. <number> records posted and committed.

yes

Display message: FRM-40401: No changes to commit.

Have any changes been posted since the last POST or COMMIT_FORM?

no

Display message: FRM-40404: Database posting completed.<number> records.

yes

Display message: FRM-40405: No changes to post.

Stop with success.
Post and Commit Transactions (continued)

Expanded from Step 3

Is the record an INSERT or an UPDATE?

- INSERT
  - INSERT the Record
    - Fire the Pre-Insert Trigger
      - success
        - Is there an On-Insert trigger for the block?
          - yes
            - Fire the On-insert trigger
              - success
                - Does the block have the Primary Key characteristic?
                  - yes
                    - Set the record as the error location
                      - Handling the Error
                  - no
                    - Check Record Uniqueness
                      - success
                        - Insert a row into the database
                          - success
                            - Fire the Post-Insert trigger
                              - success
                                - Is there another record in the block with a higher sequence number?
                              - failure
                          - failure
                        - failure
                      - failure
                    - failure
          - no
            - fails
    - failure
- neither

Handling the Error
Post and Commit Transactions (continued)

Expanding from Step 3

Is the record an INSERT or an UPDATE?

UPDATE

UPDATE the Record

Fire the Pre-Update Trigger

success

failure

Is there an On-Update trigger for the block?

no

yes

Fire the On-Update trigger

success

failure

Does the block have the Primary Key characteristic?

no

yes

Set the record as the error location

Handling the Error

Check Record Uniqueness

success

failure

Update the row in the database

success

failure

Fire the Post-Update trigger

success

failure

Is there another record in the block with a higher sequence number?
Post and Commit Transactions (continued)

Handling the Error

- Is an error location set?
  - Yes: Set the record and block in error as the error location
  - No: Display message: FRM-40402: Commit cancelled.

- Did the operator press CTRL-C (or the equivalent)?
  - Yes: Stop with failure.
  - No: Test the database to see if a deadlock occurred (A deadlock would roll back all database changes made by this commit process and release any locks held by the operator).

- Did deadlock occur?
  - Yes: Remove all records marked for delete from the form. Mark as disabled all records that were reserved (i.e., the corresponding rows were locked).
  - No: Rollback Form

- Stop with failure.
Prepare the Query

Start

Fire the Pre-query trigger

success

Is there a base table for this block?

yes

Construct the SQL statement

failure

Stop with failure

Fire the Pre-select trigger

success

Stop with success
PREVIOUS_BLOCK

Start.

Navigate to the Form Level with validation

failure → Stop with failure.

success

Is there a previous block specified?

yes → Make the specified block the target block

no

Is there an enterable block in the form with a sequence number lower than the current block?

no

Is there an enterable block in the form with a sequence number higher than the current block?

no

Make the current cursor block the target block

Put Cursor At target_block = current target block

failure → Stop with failure.

success

Stop with success.
Start.

Is the navigation style \textit{SAME\_RECORD}? no \rightarrow Is the navigation style \textit{CHANGE\_BLOCK}? yes \rightarrow Convert event to \textit{PREVIOUS\_BLOCK} no \rightarrow Navigation style is \textit{CHANGE\_RECORD}. Convert event to \textit{PREVIOUS\_RECORD} no \rightarrow Stop with success

Navigate to the Record Level with validation

Is there an enterable item In the form with a lower sequence number than the current item? yes \rightarrow Make the enterable item with the next lower sequence number the target item no \rightarrow Is there an enterable item In the form with a higher sequence number higher than the current item? yes \rightarrow Make the enterable item with the highest sequence number the target item no \rightarrow Make the current cursor item the target block

Put Cursor At target item = target item identified above

failure \rightarrow Stop with failure.

success

Stop with success.
Start.

Is the current record the first record in the block? yes
no

Navigate to the Block Level with validation

Display message: FRM-40100: At first record.

Is the current cursor record a blank record? (Note: This can only be the last record of the block.) yes
no

Fire the When-Remove-Record trigger success or failure

Put Cursor At target_record = record with the next lower sequence number than the current cursor record

Stop with success.

Stop with failure.
Process Expired Timer

Start

Does a When-Timer-Expired trigger exist

no

Is the timer a repeating timer?

no

Mark timers for later deletion

yes

Mark items for later rescheduling

Delete timer if non-repeating

Display message: FRM-40700: No such trigger.

Return with failure

Fire When-Timer-Expired trigger

Stop with failure. Unmark items that were marked for rescheduling

Stop with success
Process the Function Key

Start.

Is there a key trigger defined for the key that the operator pressed?

- **No**
  - Perform the appropriate action for the key that the operator pressed

- **Yes**
  - Fire the key trigger
  - success or failure

Stop with success.
Prompt and Answer

Start.

Display the prompt

Accept operator input

Did the operator press one of the specified keys?

no

yes

Remove the prompt

Stop with success.
Put Cursor At

Start.

Is the target "outside"?

- yes → Exit the form
- no → Stop with success.

1 → yes → is a target given?

- no → Navigate to the Form Level with validation
- success → Stop with failure.

Are there enterable blocks in the form?

- no → Exit the form
- yes → Stop with success.

Is the form starting execution with the -m switch?

- yes → Display the Block Menu and get a selection from the operator
- no → Make the first enterable block in the form the target block.

What did the operator choose?

- valid block numbers → Make the selected block the target block.
- [Exit/Cancel] or no block selected → Prompt and Answer prompt: Do you want to quit? until: any function key.

What did the operator answer?

- "yes" → Exit the form.
- anything other than "yes" → Make the first enterable block in the form the target block.

Stop with success.
Put Cursor At (continued)

1

Is a target block specified?

yes

Navigate to the Block Level with validation

failure

Stop with failure.

success

Are there records in the block?

no

block or form

Set n equal to 1.

yes

What is the validation unit?

item or record

Initialize a record

Fire the When-Create-Record trigger

success

Set n equal to n+1

failure

Is n the greater than the number of records the block can display?

no

yes

Initialize the block

Is a target record specified?

no

Make the current record in the block the target record

Navigate to the Record Level with validation

success

2
Put Cursor At (continued)

2

Is a target item specified?

yes

Is there an enterable field in this record?

no

no

Is the validation until the item?

yes

no

Look at the first field in the record before the first enterable item

Make the current cursor item the target item

Validate the Item

success or failure

Look at the Item.

yes

Is there another item not already validated before the first enterable item?

no

Make the first enterable item the target item

Navigate to the item Level with validation

failure → Stop with failure.

success

Define the cursor to be the target item

Adjust the scrolling window for the current block

Return for Input
REPLACE_MENU

Start.

Is the specified menu application called DEFAULT?
  yes
  no
  yes

Is the default menu currently displayed?
  no
  Display the default menu over all canvases

Is the specified menu application NULL?
  no
  yes

Is any menu currently displayed?
  no
  yes

Remove the current menu

Is the Oracle Forms Menu Component Initialized?
  no
  Initialize Menu component
  success
  failure

Can the Oracle Forms Menu component build the specified menu?
  no
  Stop with failure.
  yes

Display the menu over all canvases

Display the active canvas over the menu

Stop with success.
Return for Input

Start

Does the cursor block differ compared to the last cursor block?
  yes
  Set the last cursor block to the new cursor block
  yes
  Does a When-New-Block-Instance trigger exist?
    no
    no
    Does the cursor record differ compared to the last cursor record?
      no
      no
      Does a When-New-Record-Instance trigger exist?
        no
        no
        Does the cursor item differ compared to the last cursor item?
          no
          no
          Set the last cursor item to the new cursor item
  no
  Accept operator input

Fire the trigger and return for input
Rollback Form

Start

Is there an On-Rollback trigger?

no

Construct savepoint name (unless performing a full rollback)

yes

Fire the On-Logout trigger

Stop with failure

ROLLBACK_FORM

Stop with failure

Are we running with cursor mode = close at commit?

no

yes

Invalidate open cursors

Stop with failure

re-open shared cursors

Stop with success
Run the Form

Start.

Is Oracle Forms starting up for the first time?

- yes
  - Logon
    - failure
      - Stop with failure.
    - success
      - Locate the form on the user environment and load it into memory
        - failure
          - Stop with failure.
        - success
          - Set the current navigation unit equal to "outside the form" and the cursor position as "undefined"
            - Put Cursor At no target
              - failure
                - Stop with failure.
              - success
                - Fire the When-New-Form-Instance trigger
                  - Return for Input
                    - Release resources used by the form
                      - Stop with failure.
Savepoint

Start

Is savepoint mode on?

no → Stop with success

yes → Increment savepoint number

no → Is there an On-Savepoint trigger?

no → Construct form savepoint FM_(savepoint number)

issue savepoint

success → Return with success

failure → Return with failure

yes → Fire the On-Savepoint trigger
Start.

is the last record in the block's list of records displayed?

yes

no

Calculate the <scroll amount>, where <scroll amount> equals the truncated value of \((\text{<number of records the block can display> - 1}) \times 0.80 + 1\)

Set count equal to 1

Make the last record in the block's list of records the target record

Fetch Records

n = 1

success

failure

Did the last record in the block's list of records scroll into the block's display?

yes

no

Set count equal to count plus 1

Is count greater than <scroll amount>?

yes

no

Is the current cursor record in the new display?

yes

no

Make the first record in the new display the target record

Make the current cursor record the target record

Display message: FRM-40352: Last row of query retrieved.

1
SCROLL_DOWN (continued)

1

Is the target record different than the current cursor record?

no

yes

Navigate to the Block Level with validation

success

failure

Put Cursor At

target_record = the target record identified above
target_item = current cursor item

failure

success

Stop with success.

Stop with failure.
Start.

yes

Is the first record in the block's list of records displayed?

no

Calculate the <scroll amount>, where <scroll amount> equals the truncated value of (number of records the block can display) * .80 + 1

Make the first record in the block's list of records the target record

Set count equal to 1

Scroll the list of records down one line

Did the first record in the block's list of records scroll into the block's display?

yes

no

Set count equal to count plus 1

Is count greater than <scroll amount>?

yes

Is the current cursor record in the new display?

no

Make the last record in the new display the target record

yes

Make the current cursor record the target record
SCROLL_UP (continued)

1

no

Is the target record different from the current cursor record?

yes

Navigate to the Block Level with validation

success

failure

Put Cursor At

target_record = the target record identified above

target_item = current cursor item

success

failure

Stop with success.

Stop with failure.
SHOW_LOV

Start.

Is a list of item values available for the current item?  
\[\text{yes} \rightarrow \text{no} \]

Check Item for Edit

failure

success

What style is the list of values?  
\[V2 \rightarrow V3\]

Execute the list of values SQL text, but do not execute any INTO clause

Does the current item contain a value?  
\[\text{yes} \rightarrow \text{no} \]

Is the RESTRICT parameter specified?  
\[\text{yes} \rightarrow \text{no} \]

Narrow the number of values accordingly

Has the list been narrowed to only one value?  
\[\text{yes} \rightarrow \text{no} \]

Display the list of values in a pop-up window

Accept operator input

Did the operator press [Exit/Cancel]?  
\[\text{yes} \rightarrow \text{no} \]

Did the operator press [Select]?  
\[\text{yes} \rightarrow \text{no} \]

Make the list of values disappear

Stop with failure.
SHOW_LOV (continued)

1

- Identify the list of Item values and start at the first one
- Enter the Value into an Item
- Fire the Post-Change trigger
- Accept operator input
- Did the operator press [Exit/Cancel]?
  - yes: Make the list of values disappear → Stop with success,
  - no: Did the operator press [Next Field] or [Enter]?
    - no: Perform the appropriate operation
    - yes: Identify the next value in the list of values

2

- Does the list of values SQL text contain an INTO clause that refers to an item?
  - no
  - yes: Enter the Value into an Item
    - Is there another item referred to in the INTO clause?
      - yes
      - no: Stop with success.
Start.

Is the current record the first record in the block? no

Navigate to the Block Level with validation success

Is the current cursor record a blank record? (Note: Such a record could only be the last record of the block) no

Remove the record from the block

Put Cursor At target_record = record with the next lower sequence number than the current cursor record

target_item = current cursor item success

Stop with success.

Display message: FRM-40100: At first record.

Stop with failure.
Validate the Block

Start.

What is the validation unit?

- block or form
  Look at the first record in sequence order
  Validate the Record
  success → Stop with success.
  failure
    yes → Is there a record in the block with a higher sequence number?
    no → Look at the record
    success → Stop with success.
    failure → Stop with failure.

- field or record
  Look at the current record
  Validate the Record
  success → Stop with success.
  failure → Stop with failure.
Validate the Form

Start.

Look at the first block in sequence order

Validate the Block

- success
- failure

Look at the block yes

Is there a block in the form with a higher sequence number?

no

Are there any Changed items or records in the form?

yes

Mark all Changed items and records on the form as Valid, without validating them

Stop with success.

no

Stop with failure.
Validate the Item

Start.

Is the item marked as Valid?
  yes → Stop with success.
  no →

Is the item’s format mask Valid?
  yes →
  no →

Is the item NULL?
  yes →
  no →

Is the item enterable?
  yes →
  no →

Does the item have the Fixed Length property?
  yes →
  no →

Is the length of the item’s value equal to the item’s length?
  yes →
  no →

Is the item value of the proper data type?
  yes →
  no →

Does the item have a range defined?
  yes →
  no →

Is the item value within the range?
  yes →
  no →

Fire the Post-Change trigger

1

Display message: FRM-40209: Item must be of form <format mask>.

Display message: FRM-40203: Item must be entered completely.

Display message: <Valid data type description>.

Display message: FRM-40207: Must be in range <low value> to <high value>.

Stop with failure.

2
Validate the Item (continued)

1

Is the current operation PREVIOUS_ITEM?

yes

no

Does the item have the Required property?

yes

no

Does the item have the Input Allowed property?

yes

no

Fire the When-Validate-Item trigger

success

Mark the Item as Valid

Stop with success.

Display message: FRM-40202: Item must be entered.

no

failure

Stop with failure.
Validate the Record

Start.

What is the status of the record?  
- New or Valid  → Stop with success.
- Changed

Look at the first item in sequence order

Validate the Item

- failure  → Stop with failure.
- success

Look at the item

Is there an item in the record with a higher sequence number?

- yes
- no

Are there any Changed items in the record?  
[i.e. Did validating an item mark any previous items in the current record as Changed?]

- no
- yes

Mark these Changed items as Valid, but do not validate them

Is the record marked for update?

- yes

Is this a block mode environment?

- yes
- no

Lock the Row lock not required

- success
- failure  → Stop with failure.

- failure

Lock the Row lock required

- success
- failure  → Stop with failure.

Fire the When-Validate-Record trigger

- success
- failure

Mark the record as Valid

- success  → Stop with success.
- failure  → Stop with failure.

Stop with success.
Oracle Forms Version 4.5 is upwardly compatible with Versions 2.0, 2.3, 3.0, and 4.0.

This chapter covers:

- About Upgrading Forms and Menus to Version 4.5 A – 2
- Upgrading a Form A – 2
- Upgrading a Menu A – 11
- Output When Upgrading from Version 3.0 to Version 4.5 A – 14
- Enhanced Version 3.0 Functionality A – 18
- New Features in Version 4.5 A – 23
- Terminology Changes A – 24
- Triggers, Built-ins, and Properties A – 27
- Moving from Character Mode to GUI A – 39
About Upgrading Forms and Menus to Version 4.5

Oracle Forms Version 4.5 includes the functionality previously offered by the separate product SQL*Menu, so you use the Version 4.5 Generate component to convert and upgrade:

- Forms created with SQL*Forms Versions 2.3 and 3.0, as well as Oracle Forms 4.0
- Menus created with SQL*Menu Version 5.0

Note: Before converting your forms or menus, Oracle Corporation recommends that you first make backup copies of all .INP files and menu export files to insure form and menu protection.

Upgrading a Form

In earlier versions, Convert and Generate were separate Oracle Forms components. In Version 4.5, the Generate component performs both conversion and generation functions, using two keyword parameters:

- upgrade specifies that you want to use the conversion function.
- version specifies the version you’re starting from.

To upgrade your Version 2.3 or 3.0 form to Version 4.5, type this statement at the command line, substituting the correct form of the Generate command for your environment:

```
f45gen <module_name> <username>/<password> upgrade=yes
```

<table>
<thead>
<tr>
<th>To upgrade from</th>
<th>Use this keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 4.0</td>
<td>no conversion required; re-generate only</td>
</tr>
<tr>
<td>Version 3.0</td>
<td>default; no version keyword required</td>
</tr>
<tr>
<td>Version 2.3</td>
<td>version=23</td>
</tr>
<tr>
<td>Version 2.0</td>
<td>version=20</td>
</tr>
</tbody>
</table>

On MS Windows, type this “command line” in the File Manager’s File/Run dialog, or automate it by creating an icon with this command string.

Note: For bitmapped applications, consider using the widen_fields option to add one character to the width of each text item to compensate for the border bevel. Refer to “Widen Fields,” later in this chapter.
Upgrading from Version 4.0 to Version 4.5

Moving from Version 4.0 to Version 4.5 does not require conversion, but only re-generation of forms, menus, and libraries.

Once you install Version 4.5, you can either:

- Simply re-generate the .FMB file.
  **Result:** Version 4.5 executable (.FMX) file.

  If you are upgrading a large number of forms developed under Version 4.0, you do not need to individually open each form in the Designer in order to run your forms under Version 4.5. You can create a shell script to generate all the forms. (On MS Windows there are several batch utilities available to automate this task.) To suppress interactive messages during batch generation, use the command line option, `batch=yes`. Any error messages produced will be saved in a file called `module_name.err`.

- Open, save, and re-generate the file.
  **Result:** Version 4.5 binary (.FMB) file
  Version 4.5 executable (.FMX) file.

  You do not need to create your Version 4.5 .FMB file until the next time you want to make changes to a form. At that time, open the form in the Oracle Forms Version 4.5 Designer. When you save the form in Version 4.5, you will have a Version 4.5 .FMB file.

**Note:** Once a form has been generated using Version 4.5, you cannot open or run it in Version 4.0.
Upgrading from Version 3.0 to Version 4.5

The following example upgrades the form called ORDER_ENTRY from Version 3.0 to Version 4.5:

```
f45gen order_entry scott/tiger upgrade=yes widen_fields=yes
```

When you specify `upgrade=yes`, Oracle Forms creates both an .FMB and an .FMX file; for the .FMB file only, specify `generate_on_upgrade=no`.

The `widen_fields` option adds one character to the width of each text item to compensate for the border bevel.

**Note:** The specific command to start any Oracle Forms component (Designer, Generate, Runform) varies according to component and operating system. For example, `f45gen` starts the Generate component in an MS Windows environment. For more information about starting Oracle Forms components, refer to the *Oracle Forms Reference Manual, Vol. 1*, Ch. 1, “Components and Options.”
Upgrading from Version 2.3 to Version 4.5

The following example upgrades the form called ORDER_ENTRY from Version 2.3 to Version 4.5 and creates only an .FMB file:

```
f45gen order_entry scott/tiger upgrade=yes version=23
generate_on_upgrade=no
```

Running an Application After Upgrading

To run a form after upgrading, type the following statement at the command line, substituting the correct form of the Runform command for your environment:

```
f45run <form_name> <username>/<password>
```

If you do not supply a username and password, Oracle Forms displays a logon screen on which you can enter the username, password, and the database you intend to connect to.

GUI Platforms

On GUI platforms, where a command line is less commonly used, you can use the Generate dialog to specify module name, user name/password, upgrade, and version from which to upgrade.

To bring up the Generate dialog, double-click on the Generate icon. This will, for example, allow you to specify the name of SQL*Menu 5.0 files in the database that you want to convert to Oracle Forms 4.5.
Upgrading Forms Containing Referenced Objects

*Upgrade referenced forms first.* All applications that are used as the source for references must be upgraded first, followed by upgrading any applications that make references to them. Forms that are the source for referenced objects must be saved into Oracle Forms Version 4.5 tables.

For example, if Application A references objects from Application B, you must upgrade in this sequence:

- Application B
- Application A

Issue the upgrade commands in this sequence:

```
f45gen app_b scott/tiger upgrade=yes
f45gen app_b scott/tiger insert=yes
f45gen app_a scott/tiger upgrade=yes
```

**Note:** If you fail to upgrade applications in the correct sequence, you will have to delete the application that was upgraded out of order, and begin again to upgrade the applications in the correct sequence.

In Version 4.5, you have the option of referencing forms saved to either the file system or the database. Upgraded forms, however, will still expect to find referenced objects saved to the database. During the conversion process, Oracle Forms expects to find both Version 3.0 (or, for menus, Version 5.0) and Version 4.5 tables in the database.

<table>
<thead>
<tr>
<th>Database must contain:</th>
<th>On upgrade, Oracle Forms reads the referenced object from:</th>
<th>Subsequently, Oracle Forms reads from:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upgrading from SQL*Forms 3.0</td>
<td>3.0 tables</td>
<td>3.0 tables</td>
</tr>
<tr>
<td>SQL*Menu 5.0</td>
<td>5.0 tables and 4.5 tables</td>
<td>5.0 tables</td>
</tr>
<tr>
<td>Upgrading from Oracle Forms 4.0</td>
<td>4.0 tables and 4.5 tables</td>
<td>4.0 tables</td>
</tr>
</tbody>
</table>

When you upgrade a Version 3.0 form (or a Version 5.0 menu) containing referenced procedures, blocks, fields, or triggers, Oracle Forms Version 4.5 reads the reference information from the Version 3.0 tables (or the Version 5.0 tables). The references are noted in the .FMB or .MMB file that Version 4.5 produces.
When you subsequently open or generate the Version 4.5 application (including references), Oracle Forms expects the referenced modules to be saved in Version 4.5 tables. (For menu applications, Version 4.5 reads referenced information from the SQL*Menu 5.0 tables during upgrade, and then subsequently reads from the Oracle Forms Version 4.5 menu tables.)

For Oracle Forms 4.5, the supported method is to save shared procedures in PL/SQL libraries, rather than directly referencing PL/SQL program units. Version 3.0 or 4.0 applications that currently reference PL/SQL program units directly will upgrade correctly to Version 4.5, and will run as they always did. However, once the application is running under Version 4.5, new direct references to program units are not supported. If you want to eliminate referencing completely from your upgraded application, follow this process:

- From the source form, drag and drop the program units into a library.
- Attach the library to all forms that reference the program units.
- Delete the referenced program units from target forms.

Occasionally, Version 3.0 and 4.0 forms which served as the source for referenced program units did not have any blocks or pages. Upgrading such forms creates a special case. The Oracle Forms Version 4.5 upgrade process consists of two stages:

- Convert the document format.
- Attempt to generate a new executable module.

When you upgrade forms containing no blocks or pages, Oracle Forms Version 4.5 will complete the first stage, successfully producing a new .FMB file, but the second stage, the generate, will fail, producing an error listing.
Resolving Naming Conflicts in Form–Level Procedures

The Oracle Forms upgrade utility does not rename, rewrite, or otherwise manipulate any form–level procedures (now called user–named subprograms).

If your application uses a procedure (either in SQL* Forms Version 3.0 or SQL*Menu Version 5.0) that has the same name as a new Version 4.5 built–in, you must rename the procedure and modify all triggers that call it before you can create the new .FMX file. (The conversion will create the .FMB file, but until the naming conflicts are resolved, you can expect to encounter Generate errors.)

For example, Oracle Forms Version 4.5 provides a new DELETE_GROUP built–in which did not exist in Version 3.0. If you have an existing form–level procedure called DELETE_GROUP, you will need to rename the form–level procedure to eliminate the naming conflict.

To prepare for a successful conversion, check your list of procedures and functions against the following:

- Built–in names listed in the table of contents of this manual
- Oracle Forms reserved words found in Oracle Forms Reference Manual, Vol. 2, Appendix C, ”Reserved Words”
Upgrading File Formats

Version 4.5 files have different file formats from Version 3.0 files. As part of the upgrade process, the Version 3.0 files will be upgraded to Version 4.5 formats, with Version 4.5 file extensions.

The following table shows the file extensions for each type of module and storage format.

<table>
<thead>
<tr>
<th>Type</th>
<th>Binary (Design)</th>
<th>Text</th>
<th>Executable Runfile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>.FMB</td>
<td>.FMT</td>
<td>.FMX</td>
</tr>
<tr>
<td>Menu</td>
<td>.MMB</td>
<td>.MMT</td>
<td>.MMX</td>
</tr>
<tr>
<td>Library</td>
<td>.PLL</td>
<td>.PLD</td>
<td>.PLX</td>
</tr>
</tbody>
</table>

The default file extensions indicate the module type and storage format:

- .FMB: Form Module Binary
- .FMT: Form Module Text
- .FMX: Form Module Executable
- .MMB: Menu Module Binary
- .MMT: Menu Module Text
- .MMX: Menu Module Executable
- .PLL: PL/SQL Library Module Binary
- .PLD: PL/SQL Library Module Text
- .PLX: PL/SQL Library Module Executable (contains no source)

For more information on file formats, see Appendix, "Module Storage."
Version 3.0 .INP Files

Oracle Forms upgrades SQL*Forms Version 3.0 .INP files to compressed binary format (.FMB).

The .INP files were both human-readable and portable across platforms. To create Version 4.5 human-readable files that are portable across platforms, convert the .FMB files to .FMT (text) files.

Version 4.5 .FMT files differ from .INP files in sequence and organization. While .INP files used a hierarchical organization, printing each block followed by its related items, for performance purposes, Version 4.5 .FMT files group all the blocks together, followed by all the items. In addition, much of the information in the .FMT file, although in text format, actually represents binary data. Images, translatable text strings, boilerplate graphics, and boilerplate text are all represented in hexadecimal format.

Converting File Formats

After you upgrade to Version 4.5, you may want to convert binary files to text files, or vice versa.

To convert an .FMB file to an .FMT file in the Designer, select File->Administration->Convert and fill in the Convert dialog.

To convert a form module .FMB file to a .FMT file, use the keyword parameter \texttt{script}:

\texttt{f45gen order_entry scott/tiger logon=no script=yes}

To convert a form module .FMT file to a .FMB file, use the keyword parameter \texttt{parse}:

\texttt{f45gen order_entry scott/tiger logon=no parse=yes}

To convert a menu module, either .MMB to .MMT or .MMT to .MMB, use the appropriate syntax as given above, with the addition of the keyword parameter \texttt{module_type=menu}. 
Upgrading a Menu

To upgrade your SQL*Menu Version 5.0 menu to an Oracle Forms Version 4.5 menu module:

- Use the Oracle Forms Generate component
- Use the menu application name as the module name
- Use module_type=menu
- Use upgrade=yes

```
f45gen <module_name> <username>/<password> module_type=menu upgrade=yes
```

Upgrading from SQL*Menu 5.0

The following example upgrades the menu called MAIN_MENU from SQL*Menu Version 5.0 to Oracle Forms Version 4.5:

```
f45gen main_menu scott/tiger module_type=menu upgrade=yes
```

The keyword parameter version is not required for menu modules because you can only convert from SQL*Menu Version 5.0, not from earlier versions.

You can use a wild-card character as part of the name of the menu application. Oracle Forms will upgrade all menu applications from the SQL*Menu Version 5.0 tables that match the specified name. To upgrade all menu applications that begin with “ap_”:

```
f45gen ap_% scott/tiger module_type=menu upgrade=yes
```

To upgrade standalone menus from SQL*Menu 5.0:

- Create an empty form
- Attach the menu to the form
- Run the form rather than the menu
Creating an .MMB File Only

By default, when you specify `upgrade=yes`, Oracle forms creates both an .MMB file and an .MMX file. Use the keyword parameter `generate_on_upgrade` to create .MMB files only.

The following example upgrades all SQL*Menu Version 5.0 applications beginning with the word “test,” and creates only .MMB files:

```
f45gen test% scott/tiger module_type=menu upgrade=yes generate_on_upgrade=no
```

To avoid overwriting any existing applications, any application whose name matches an application that is already in the Version 4.5 menu tables will not be upgraded. If you upgrade an application to Version 4.5, then change it in SQL*Menu Version 5.0, you must first delete the application from Oracle Forms Version 4.5 and then re-upgrade it.

Unless wildcard characters are supplied in the application name, the Generate component used for upgrade will first upgrade, and, after upgrading, generate an .MMX file for the application. If, however, wildcard characters are supplied in the application name, no .MMX files will be generated.

For more information on menus, refer to the *Oracle Forms Developer’s Guide, Book 1*, Ch. 22, “Basic Menu Design.”
Upgrading Menu Security Roles

Oracle Forms now implements menu security using the currently enabled Oracle7 application security roles as defined by the SESSION_ROLES database view.

To upgrade menu security roles, a DBA or developer who has CREATE ROLE privileges must run the following command:

```
frm45gen userid=scott/tiger upgrade_roles=yes
```

To grant a user access to base tables, a DBA or developer who has access to the SYSTEM account must run the following SQL script with the username as the first argument:

```
frm45grt.sql SCOTT
```

To revoke a user’s access to base tables, a DBA or developer who has access to the SYSTEM account must run the following SQL script with the username as the first argument:

```
frm45rvk.sql SCOTT
```

**Note:** Menu roles are independent of any specific menu application (no module name is specified). For more information on menu roles, refer to the Oracle Forms Developer’s Guide, Ch. 23, “Customizing Menus.”
Output When Upgrading from Version 3.0 to Version 4.5

The Generate component performs the following tasks when you upgrade from Version 3.0 (or an earlier version) to Version 4.5:

- transcribes Version 3.0 triggers to Version 4.5 triggers
- converts fields to Version 4.5 text items
- converts pages to Version 4.5 windows and canvas–views
- converts LOVs to Version 4.5 LOVs and record group objects
- converts .INP (text) files to .FMB (binary) files

If you are upgrading from Version 4.0 to Version 4.5, you already encountered these changes when converting applications from Version 3.0 to Version 4.0.

Triggers

Oracle Forms converts some trigger names to Version 4.5 nomenclature, but it does not change the text of triggers in any way. For example, an On–Validate–Field trigger becomes a When–Validate–Item trigger, but the text of the trigger remains the same (key triggers are not changed to mouse–oriented triggers).

During conversion, Oracle Forms copies the text of your triggers verbatim:

- V2–style triggers remain V2–style triggers running under Version 4.5.
- V3–style PL/SQL triggers remain PL/SQL–style triggers.

Note: Upgrading a V2 form does not convert the triggers to PL/SQL. Rather, Oracle Forms fully supports the V2 trigger definition style.
Fields

Fields are now called text items. All fields are converted to single-line text items.

Pages

Because of the change from pages to canvas-views displayed in windows, Oracle Forms converts each full-screen page to a content canvas-view in a default document window with a default name, such as Window1. Pop-up pages are converted to content canvas-views in separate windows.

You may want to make former pop-up pages into stacked canvas-views. For more information on stacked canvas-views, see “Converting Pop-Up Pages,” later in this chapter.

LOVs

Oracle Forms converts LOV SQL text on a field in Version 3.0 into two objects:

- A query record group object having the same name as the LOV.
- An accompanying LOV object to display the data in the record group.
Master–Detail Block Relationships

Oracle Forms Version 4.5 does not modify Version 3.0 master–detail triggers when you upgrade a form. If you want to take advantage of the new Version 4.5 master–detail functionality, you must delete the existing master–detail triggers and then define a Version 4.5 relation object.

Properties

To provide greater navigation control, the Version 3.0 Input Allowed item property, also called Enterable, upgrades to two Version 4.5 properties: Navigable and Enabled.

<table>
<thead>
<tr>
<th>Version 3.0 Input Allowed</th>
<th>Version 4.5 Navigable</th>
<th>Version 4.5 Insert Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

For more information on properties, refer to the Oracle Forms Developer’s Guide, Ch. 9, “Defining Items.”
Packaged Procedures and Functions

All Oracle Forms Version 2.3 trigger macros and 3.0 packaged procedures and functions are supported in Version 4.5. In Version 4.5, packaged procedures and functions, now called *built-in subprograms*, are treated in one of three ways:

- Some built-ins, like ENTER, are the same in Version 4.5 as they were in earlier versions.
- Some built-ins have been replaced for better efficiency or more consistent naming. While they are not recommended for use in new applications, they are still supported.
- Some built-ins are new for Version 4.5. For all new applications, Oracle Corporation recommends that you use the Version 4.5 built-in subprograms.

No existing trigger code in your applications is changed during conversion.

Refer to "Oracle Forms Version 3.0 Packaged Procedures and Functions" later in this chapter for a list of packaged procedures that have been revised or renamed in Version 4.5.

Character-mode Applications

When you run a character-mode application with pop-up pages, you will notice these changes:

- Titles and scroll bars have a more streamlined appearance.
- Alerts behave differently—use [Space] to push a button.
Enhanced Version 3.0 Functionality

Many of the features available in SQL*Forms Version 3.0 have been significantly enhanced in Oracle Forms Version 4.5. This section explains some of the most important changes, and describes how some fundamental concepts have changed.

Integrated Forms and Menus

All the previous functionality of SQL*Menu is now an integral part of Oracle Forms. Instead of creating menu applications in a separate executable, you can now create menu modules directly from the Oracle Forms Designer. In addition, Version 4.5 menus can be stored in the database or in a file.

Another important change is in the way forms and menus call each other. In previous versions, designers often built stand-alone, front-end menu applications from which operators invoked individual forms. In Version 4.5, Oracle Corporation recommends that you always attach a menu to a form, such that the form is loaded before the menu at startup. You can achieve the same result as a stand-alone menu application by creating a “dummy” form with a main menu, and then calling form modules from the menu.

To make this functionality possible, Version 4.5 now supports the ability to create a form that contains no blocks or items (fields).

For more information, refer to the Oracle Forms Developer’s Guide, Ch. 22, “Basic Menu Design.”
New Block Window

The New Block window has been enhanced to provide more powerful defaulting capabilities. You can now specify the following when you create a new block:

- control blocks as well as base table blocks
- default layout style (form or tabular)
- record orientation (horizontal or vertical records)
- amount of space between items in the layout
- type of item that Oracle Forms should create for each column (text item, check box, radio group, and so on)
- whether the block should have a scroll bar
- whether Oracle Forms should create an additional button block with buttons for executing standard navigation and query functions

Master–Detail Block Relationship

The default master–detail facility now provides more control over the functionality of master–detail block relations. In Version 4.5, there is a new relation object. The properties of a relation define the coordination between a master block and its detail block. By defining relation objects, you can create complex relations, including a master with dependent details and a master with independent details, without having to edit the default master–detail trigger code.

As in earlier versions, you can define a relation when you create the detail block in the New Block facility. However, you can now also create relations as separate objects at any time during the form design process. Depending on how you set the properties of the relation object, Oracle Forms creates some or all of the following triggers to enforce block coordination at runtime:

- On–Clear–Details
- On–Populate–Details
- On–Check–Delete–Master
List of Values (LOV)

A list of values (LOV) is now a named object, separate from any particular text item (field). As such, an LOV can now be associated with more than one text item, and can be shown and hidden programmatically, independently of the current cursor location.

In Version 4.5, the values that are actually displayed in the LOV are derived from a separate object called a record group. Record groups are internal data structures that have a column/row framework similar to database tables. The rows in a record group can come from a SELECT statement that executes at runtime or from static values that you specify at design time.

Pages

There are no pages in Version 4.5. Taking the place of pages are two separate objects called windows and canvases. A window is an empty frame that can have scroll bars, title bars, status and message lines, and platform-specific window elements such as minimize buttons. A canvas is the background object on which you place interface items and boilerplate text and graphics. Each canvas is assigned to a window at design time and is displayed in that window at runtime. Like other objects, windows and canvases have user-defined names, rather than assigned numbers.

The canvas object has an attribute called a viewport. The viewport is a rectangular boundary that defines the canvas-view, that is, the portion of the canvas that is actually visible in the window at runtime. More than one canvas-view can be assigned to the same window, and canvas-views can overlap each other in a window when desired.

There are two types of canvas-views that can be displayed in a window: content views and stacked views. You can display and edit stacked canvas-views in the same Layout editor window with the content view.

There is no longer a Page 0 in Version 4.5. Instead, interface items (fields) that you want to use as non-displayed variables are simply not assigned to any canvas object, and so are never displayed. These NULL-canvas items can be used in the same way as Page 0 fields. Note, however, that there is no actual canvas named NULL, as there was a Page 0; NULL-canvas items cannot be displayed in the Designer or at runtime.

For more information, refer to the Oracle Forms Developer's Guide, Ch. 11, “Working with Windows and Canvas-views.”
In addition to form and menu modules, Oracle Forms provides library modules. A library module is a collection of user–named PL/SQL procedures, functions, and packages that can be attached to form and menu modules. Triggers, procedures, and menu commands can then reference program units in the attached library.

In Version 4.5, PL/SQL libraries support dynamic loading. A library’s program units are loaded into an application only when needed. This can dramatically reduce the runtime memory requirements of an application.

Version 4.5 libraries no longer produce .PLA or .LIB files. Libraries now have just two file formats:

- .PLL, which contains:
  - source code required at design time
  - compiled version of code required at runtime

- .PLD, the text format file that can be used for source–controlling library files

Previously, when you edited the library module, you had to detach the library, make changes, then re–attach the library. In Version 4.5, Oracle Forms automatically checks and updates references in place: you no longer have to detach and re–attach.

For more information, refer to the Oracle Forms Developer’s Guide, Ch. 20, “Working with Libraries.”
User Exits

Two statement types for user exits have been renamed in Version 4.5, and three new statement types have been added. The Version 3.0 names will continue to be supported, but you are encouraged to use the Version 4.5 statement types when creating new user exits.

The following table lists the new user exit statement names.

<table>
<thead>
<tr>
<th>Version 3.0 Statement</th>
<th>Version 4.5 Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXEC IAF GET</td>
<td>EXEC TOOLS GET</td>
</tr>
<tr>
<td>EXEC IAF PUT</td>
<td>EXEC TOOLS SET</td>
</tr>
<tr>
<td>—</td>
<td>EXEC TOOLS MESSAGE</td>
</tr>
<tr>
<td>—</td>
<td>EXEC TOOLS GET CONTEXT</td>
</tr>
<tr>
<td>—</td>
<td>EXEC TOOLS SET CONTEXT</td>
</tr>
</tbody>
</table>

For more information on Version 4.5 user exits, see Oracle Forms Advanced Techniques, Ch. 3, “User Exit Interface to Foreign Functions.”

Calling Oracle Forms from a C Program

The IAPCAL command, which allowed you to call a SQL*Forms application from any C language program, still works for Version 3.0 command lines. However, for Version 4.5 command lines, use the IFZCAL command to call an Oracle Forms application from a C language program. For more information, see the Oracle Forms Advanced Techniques Manual, Ch. 8, “Integrating with Other Oracle Tools.”
New Features in Version 4.5

Oracle Forms Version 4.5 includes an extensive number of new features providing improved productivity and performance.

Highlights of these features include:

- new developer productivity features, including:
  - a Designer interface including an Object Navigator showing all application objects in an outline presentation
  - a WYSIWYG Menu Editor
  - a PL/SQL debugger
  - incremental compilation

- extended MS Windows openness, including:
  - OLE2
  - VBX custom controls

- code reusability, including:
  - property classes
  - object groups
  - multi–level inheritance

- enhanced GUI control, including:
  - mouse awareness
  - combo boxes
  - tool bars

For more information, see Oracle Forms Getting Started, Ch. 1, "Oracle Forms 4.5 New Features."
Terminology Changes

Terminology within Oracle Forms has been changed in order to clarify new concepts and support the new GUI interface.

The following table lists general Version 3.0 terms and their corresponding Version 4.5 terms.

<table>
<thead>
<tr>
<th>Version 3.0 Term</th>
<th>Version 4.5 Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>application menu</td>
<td>n/a</td>
</tr>
<tr>
<td>built-in routine</td>
<td>built-in subprogram</td>
</tr>
<tr>
<td>characteristic</td>
<td>property</td>
</tr>
<tr>
<td>prompts</td>
<td>boilerplate text</td>
</tr>
<tr>
<td>Debug Mode Options menu</td>
<td>Debugger/Debugger window</td>
</tr>
<tr>
<td>Enterable field characteristic=TRUE</td>
<td>Navigable=TRUE</td>
</tr>
<tr>
<td></td>
<td>Insert_Allowed=TRUE</td>
</tr>
<tr>
<td>Enterable field characteristic=FALSE</td>
<td>Navigable=FALSE</td>
</tr>
<tr>
<td></td>
<td>Insert_Allowed=FALSE</td>
</tr>
<tr>
<td>field</td>
<td>text item</td>
</tr>
<tr>
<td>form</td>
<td>form module</td>
</tr>
<tr>
<td>form-level procedure</td>
<td>user-named routine</td>
</tr>
<tr>
<td>form/spread table</td>
<td>Properties window</td>
</tr>
<tr>
<td>Input Allowed field characteristic=TRUE</td>
<td>Enabled=TRUE</td>
</tr>
<tr>
<td>Input Allowed field characteristic=FALSE</td>
<td>Enabled=FALSE</td>
</tr>
<tr>
<td>menu application</td>
<td>menu module</td>
</tr>
<tr>
<td>packaged function or procedure</td>
<td>built-in subprogram</td>
</tr>
<tr>
<td>page</td>
<td>window, canvas-view</td>
</tr>
<tr>
<td>page 0 field</td>
<td>NULL-canvas item</td>
</tr>
<tr>
<td>pop-up page</td>
<td>window, canvas, canvas-view</td>
</tr>
<tr>
<td>screen painter</td>
<td>Layout editor</td>
</tr>
<tr>
<td>switches</td>
<td>preferences, keyword parameters</td>
</tr>
<tr>
<td>user-named procedure</td>
<td>user-named subprogram</td>
</tr>
</tbody>
</table>
Terminology Related to Properties

The following table lists new terminology relating to properties.

<table>
<thead>
<tr>
<th>Version 3.0 Term</th>
<th>Version 4.5 Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array Size block characteristic</td>
<td>Records Fetched block property</td>
</tr>
<tr>
<td>Echo Input field characteristic</td>
<td>Secure text item property</td>
</tr>
<tr>
<td>Enforce Key item characteristic</td>
<td>Copy item property</td>
</tr>
<tr>
<td>Field Length characteristic</td>
<td>Maximum Length property</td>
</tr>
<tr>
<td>Input Allowed characteristic</td>
<td>Enabled/Navigable properties</td>
</tr>
<tr>
<td>Lines per Record characteristic</td>
<td>Space Between Records property</td>
</tr>
<tr>
<td></td>
<td>(set in the Designer Layout editor by mouse gesture)</td>
</tr>
<tr>
<td>Page(Number) characteristic</td>
<td>Canvas(Name) property</td>
</tr>
</tbody>
</table>

Terminology Related to System Variables

The following table lists new terminology relating to system variables.

<table>
<thead>
<tr>
<th>Version 3.0 Term</th>
<th>Version 4.5 Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM.CURRENT_FIELD</td>
<td>SYSTEM.CURRENT_ITEM</td>
</tr>
<tr>
<td>SYSTEM.CURSOR_FIELD</td>
<td>SYSTEM.CURSOR_ITEM</td>
</tr>
<tr>
<td>SYSTEM.TRIGGER_FIELD</td>
<td>SYSTEM.TRIGGER_ITEM</td>
</tr>
<tr>
<td></td>
<td>SYSTEM.COORDINATION_OPERATION</td>
</tr>
<tr>
<td></td>
<td>SYSTEM.CURRENT_DATETIME</td>
</tr>
<tr>
<td></td>
<td>SYSTEM.EFFECTIVE_DATE</td>
</tr>
<tr>
<td></td>
<td>SYSTEM.EVENT_WINDOW</td>
</tr>
<tr>
<td></td>
<td>SYSTEM.MASTER_BLOCK</td>
</tr>
<tr>
<td></td>
<td>SYSTEM.MODE</td>
</tr>
<tr>
<td></td>
<td>SYSTEM.SUPPRESS_WORKING</td>
</tr>
<tr>
<td></td>
<td>$$DBDATE$$</td>
</tr>
<tr>
<td></td>
<td>$$DBDATETIME$$</td>
</tr>
<tr>
<td></td>
<td>$$DBTIME$$</td>
</tr>
</tbody>
</table>
Command Line Options

In Version 3.0, you set switches on the command line. In Version 4.5, you set preferences using two different methods:

- You can set Designer and Runtime preferences in the Tools->Options dialog. (The Runtime preferences apply only to running from the Designer).
- You can still set options on the command line. However, you now use keyword parameters instead of switches.

All Version 3.0 Runform command line switches have been replaced with corresponding keyword parameters. The Version 3.0 Generator and Designer command line switches are now obsolete. Instead, you can now choose from a larger selection of preferences in Version 4.5.

Oracle Forms 4.5 includes several new options, including:
- Debug (Runform) and Debug (Generator)
- Performance Event Collection Services (PECS) (Runform)

For details about all Version 4.5 preferences, refer to the Oracle Forms Reference Manual, Vol. 1, Ch. 1, "Components and Options."

Runform Keyword Parameters

The following table lists Version 3.0 Runform command line switches and their corresponding Version 4.5 keyword parameters and values.

<table>
<thead>
<tr>
<th>Version 3.0 Runform Command Line Switch</th>
<th>Version 4.5 Parameter/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td>array=NO</td>
</tr>
<tr>
<td>-b</td>
<td>buffer_records=YES</td>
</tr>
<tr>
<td>-c</td>
<td>term=&lt;filename&gt;</td>
</tr>
<tr>
<td>-d</td>
<td>debug=YES</td>
</tr>
<tr>
<td>-e</td>
<td>keyout=&lt;filename&gt;</td>
</tr>
<tr>
<td>-i</td>
<td>interactive=NO</td>
</tr>
<tr>
<td>-l</td>
<td>logon_screen=YES</td>
</tr>
<tr>
<td>-m</td>
<td>block_menu=YES</td>
</tr>
<tr>
<td>-o</td>
<td>optimizesql=YES</td>
</tr>
<tr>
<td>-q</td>
<td>quiet=YES</td>
</tr>
<tr>
<td>-r</td>
<td>keyin=&lt;filename&gt;</td>
</tr>
<tr>
<td>-s</td>
<td>statistics=YES</td>
</tr>
<tr>
<td>-t</td>
<td>optimizetp=YES</td>
</tr>
<tr>
<td>-w</td>
<td>output_file=&lt;filename&gt;</td>
</tr>
</tbody>
</table>
Triggers, Built–ins, and Properties

Oracle Forms 4.5 includes new and changed triggers, built–ins, and properties.

Triggers

New triggers have been added to Oracle Forms to enhance its capabilities, particularly in performing transactions between Oracle Forms and non–ORACLE data sources. In addition, several triggers have been added to provide new capabilities in the areas of master–detail coordination, interface object manipulation, access to custom events and better mouse event tracking.

New Triggers

<table>
<thead>
<tr>
<th>New Triggers</th>
<th>When–Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>On–Check–Unique</td>
<td>Button–Pressed</td>
</tr>
<tr>
<td>On–Clear–Details</td>
<td>Checkbox–Changed</td>
</tr>
<tr>
<td>On–Close</td>
<td>Custom–Item–Event</td>
</tr>
<tr>
<td>On–Column–Security</td>
<td>Image–Activated</td>
</tr>
<tr>
<td>On–Commit</td>
<td>Image–Pressed</td>
</tr>
<tr>
<td>On–Count</td>
<td>List–Activated</td>
</tr>
<tr>
<td>On–Check–Delete–Master</td>
<td>Mouse–Click</td>
</tr>
<tr>
<td>On–Fetch</td>
<td>Mouse–Doubleclick</td>
</tr>
<tr>
<td>On–Logon</td>
<td>Mouse–Down</td>
</tr>
<tr>
<td>On–Logout</td>
<td>Mouse–Enter</td>
</tr>
<tr>
<td>On–Populate–Details</td>
<td>Mouse–Leave</td>
</tr>
<tr>
<td>On–Rollback</td>
<td>Mouse–Move</td>
</tr>
<tr>
<td>On–Savepoint</td>
<td>Mouse–Up</td>
</tr>
<tr>
<td>On–Select</td>
<td>New–Block–Instance</td>
</tr>
<tr>
<td>On–Sequence–Number</td>
<td>New–Form–Instance</td>
</tr>
<tr>
<td>Post–Database–Commit</td>
<td>New–Record–Instance</td>
</tr>
<tr>
<td>Post–Logon</td>
<td>Radio–Changed</td>
</tr>
<tr>
<td>Post–Logout</td>
<td>Timer–Expired</td>
</tr>
<tr>
<td>Post–Select</td>
<td>Window–Activated</td>
</tr>
<tr>
<td>Pre–Logon</td>
<td>Window–Closed</td>
</tr>
<tr>
<td>Pre–Logout</td>
<td>Window–Deactivated</td>
</tr>
<tr>
<td>Pre–Select</td>
<td>Window–Resized</td>
</tr>
</tbody>
</table>
For more information on triggers and their typical usage, see the *Oracle Forms Developer’s Guide*, Ch. 7, “Writing Event Triggers.”

For information on a specific trigger, see the *Oracle Forms Reference Manual, Vol. 1*, Ch. 2, “Triggers.”

**Note:** If you are creating GUI applications, you may want to avoid key triggers. Key triggers fire when the operator presses a specific key or key–sequence, but in GUI applications, operators may execute commands by clicking a button, selecting a menu command, or navigating with the mouse, thus bypassing function keys completely.

### Renamed Triggers

The following triggers have been renamed.

<table>
<thead>
<tr>
<th>Version 3.0 Triggers</th>
<th>Version 4.5 Triggers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key–DUPFLD</td>
<td>Key–DUP–ITEM</td>
</tr>
<tr>
<td>Key–NXTFLD</td>
<td>Key–NEXT–ITEM</td>
</tr>
<tr>
<td>Key–PRVFLD</td>
<td>Key–PREV–ITEM</td>
</tr>
<tr>
<td>On–Clear–Block</td>
<td>On–Clear–Block</td>
</tr>
<tr>
<td>On–Database–Record</td>
<td>When–Database–Record</td>
</tr>
<tr>
<td>On–New–Field–Instance</td>
<td>When–New–Item–Instance</td>
</tr>
<tr>
<td>On–New–Record</td>
<td>When–Create–Record</td>
</tr>
<tr>
<td>On–Remove–Record</td>
<td>When–Remove–Record</td>
</tr>
<tr>
<td>On–Validate–Field</td>
<td>When–Validate–Field</td>
</tr>
<tr>
<td>On–Validate–Record</td>
<td>When–Validate–Record</td>
</tr>
<tr>
<td>Post–Commit</td>
<td>When–Validate–Record</td>
</tr>
<tr>
<td>Post–Commit</td>
<td>Post–Forms–Commit</td>
</tr>
<tr>
<td>Post–Field</td>
<td>Post–Text–Item</td>
</tr>
<tr>
<td>Pre–Field</td>
<td>Pre–Text–Item</td>
</tr>
</tbody>
</table>
Built–In Subprograms

New built–in subprograms have been added to Oracle Forms to provide powerful capabilities in the areas of GUI environments, record groups, non–ORACLE data sources, master–detail coordination, and image manipulation.

The following table lists new built–in subprograms.

<table>
<thead>
<tr>
<th>Built–in</th>
<th>Affects</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_GROUP_COLUMN</td>
<td>Record Groups</td>
</tr>
<tr>
<td>ADD_GROUP_ROW</td>
<td>Record Groups</td>
</tr>
<tr>
<td>ADD_LIST_ELEMENT</td>
<td>Lists</td>
</tr>
<tr>
<td>ADD_PARAMETER</td>
<td>Form Parameters</td>
</tr>
<tr>
<td>CALL_FORM</td>
<td>Multi–Form Applications</td>
</tr>
<tr>
<td>CHANGE_ALERT_MESSAGE</td>
<td>Alerts</td>
</tr>
<tr>
<td>CHECK_RECORD_UNIQUENESS</td>
<td>Transaction Processing</td>
</tr>
<tr>
<td>CHECKBOX_CHECKED</td>
<td>Check Boxes</td>
</tr>
<tr>
<td>CLEAR_MESSAGE</td>
<td>Messages</td>
</tr>
<tr>
<td>CLOSE_ALL_FORMS</td>
<td>Multi–Form Applications</td>
</tr>
<tr>
<td>CLOSE_FORM</td>
<td>Multi–Form Applications</td>
</tr>
<tr>
<td>COMMIT_FORM</td>
<td>Transaction Processing</td>
</tr>
<tr>
<td>CONVERT_OTHER_VALUE</td>
<td>Check Boxes, Radio Groups</td>
</tr>
<tr>
<td>CREATE_GROUP</td>
<td>Record Groups</td>
</tr>
<tr>
<td>CREATE_GROUP_FROM_QUERY</td>
<td>Record Groups</td>
</tr>
<tr>
<td>CREATE_PARAMETER_LIST</td>
<td>Form Parameters</td>
</tr>
<tr>
<td>CREATE_QUERIED_RECORD</td>
<td>Records</td>
</tr>
<tr>
<td>CREATE_TIMER</td>
<td>Timers</td>
</tr>
<tr>
<td>DBMS_ERROR_CODE</td>
<td>Messages</td>
</tr>
<tr>
<td>DBMS_ERROR_TEXT</td>
<td>Messages</td>
</tr>
<tr>
<td>DELETE_GROUP</td>
<td>Record Groups</td>
</tr>
<tr>
<td>DELETE_GROUP_ROW</td>
<td>Record Groups</td>
</tr>
<tr>
<td>DELETE_LIST_ELEMENT</td>
<td>List Items</td>
</tr>
<tr>
<td>DELETE_PARAMETER</td>
<td>Form Parameters</td>
</tr>
<tr>
<td>DELETE_TIMER</td>
<td>Timers</td>
</tr>
<tr>
<td>DESTROY_PARAMETER_LIST</td>
<td>Form Parameters</td>
</tr>
<tr>
<td>FETCH_RECORDS</td>
<td>Transaction Processing</td>
</tr>
<tr>
<td><strong>Built-in</strong></td>
<td><strong>Affects</strong></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>FIND_ALERT</td>
<td>Alerts</td>
</tr>
<tr>
<td>FIND_BLOCK</td>
<td>Blocks</td>
</tr>
<tr>
<td>FIND_CANVAS</td>
<td>Canvas–Views</td>
</tr>
<tr>
<td>FIND_COLUMN</td>
<td>Record Groups</td>
</tr>
<tr>
<td>FIND_EDITOR</td>
<td>Editors</td>
</tr>
<tr>
<td>FIND_FORM</td>
<td>Form Modules</td>
</tr>
<tr>
<td>FIND_GROUP</td>
<td>Record Modules</td>
</tr>
<tr>
<td>FIND_ITEM</td>
<td>Items</td>
</tr>
<tr>
<td>FIND_LOV</td>
<td>LOVs</td>
</tr>
<tr>
<td>FIND_MENU_ITEM</td>
<td>Menus</td>
</tr>
<tr>
<td>FIND_RELATION</td>
<td>Master–Detail Relations</td>
</tr>
<tr>
<td>FIND_TIMER</td>
<td>Timers</td>
</tr>
<tr>
<td>FIND_VIEW</td>
<td>Canvas–Views</td>
</tr>
<tr>
<td>FIND_WINDOW</td>
<td>Windows</td>
</tr>
<tr>
<td>FORMS_DDL</td>
<td>Form Modules</td>
</tr>
<tr>
<td>FORMS_OLE.CLOSE_ACTIVE</td>
<td>OLE Items</td>
</tr>
<tr>
<td>FORMS_OLE.EXEC_VERB</td>
<td>OLE Items</td>
</tr>
<tr>
<td>FORMS_OLE.FIND_OLE_VERB</td>
<td>OLE Items</td>
</tr>
<tr>
<td>FORMS_OLE.GET_VERB_COUNT</td>
<td>OLE Items</td>
</tr>
<tr>
<td>FORMS_OLE.GET_VERB_NAME</td>
<td>OLE Items</td>
</tr>
<tr>
<td>FORMS_OLE.GET_SERVER_ACTIVE</td>
<td>OLE Items</td>
</tr>
<tr>
<td>GENERATE_SEQUENCE_NUMBER</td>
<td>Transaction Processing</td>
</tr>
<tr>
<td>GET_APPLICATION_PROPERTY</td>
<td>Form Modules</td>
</tr>
<tr>
<td>GET_BLOCK_PROPERTY</td>
<td>Blocks</td>
</tr>
<tr>
<td>GET_CANVAS_PROPERTY</td>
<td>Canvas–Views</td>
</tr>
<tr>
<td>GET_FORMPROPERTY</td>
<td>Form Modules</td>
</tr>
<tr>
<td>GET_GROUP_CHAR_CELL</td>
<td>Record Groups</td>
</tr>
<tr>
<td>GET_GROUP_DATE_CELL</td>
<td>Record Groups</td>
</tr>
<tr>
<td>GET_GROUP_NUMBER_CELL</td>
<td>Record Groups</td>
</tr>
<tr>
<td>GET_GROUP_ROW_COUNT</td>
<td>Record Groups</td>
</tr>
<tr>
<td>GET_GROUP_SELECTION</td>
<td>Record Groups</td>
</tr>
<tr>
<td>GET_GROUP_SELECTION_COUNT</td>
<td>Record Groups</td>
</tr>
<tr>
<td>GET_ITEM_PROPERTY</td>
<td>Items</td>
</tr>
<tr>
<td>GET_LIST_ELEMENT_COUNT</td>
<td>List Items</td>
</tr>
<tr>
<td>Built-in</td>
<td>Affects</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>GET_LIST_ELEMENT_LABEL</td>
<td>List Items</td>
</tr>
<tr>
<td>GET_LIST_ELEMENT_VALUE</td>
<td>List Items</td>
</tr>
<tr>
<td>GET_LOV_PROPERTY</td>
<td>LOVs</td>
</tr>
<tr>
<td>GET_MENU_ITEM_PROPERTY</td>
<td>Menu Items</td>
</tr>
<tr>
<td>GET_MESSAGE</td>
<td>Form Parameters</td>
</tr>
<tr>
<td>GET_PARAMETER_ATTR</td>
<td>Messages</td>
</tr>
<tr>
<td>GET_PARAMETER_LIST</td>
<td>Form Parameters</td>
</tr>
<tr>
<td>GET_RADIO_BUTTON_PROPERTY</td>
<td>Radio Groups</td>
</tr>
<tr>
<td>GET_RECORD_PROPERTY</td>
<td>Records</td>
</tr>
<tr>
<td>GET_RELATION_PROPERTY</td>
<td>Master–Detail Relations</td>
</tr>
<tr>
<td>GET_VIEWPROPERTY</td>
<td>Canvas–Views</td>
</tr>
<tr>
<td>GET_WINDOWPROPERTY</td>
<td>Canvas–Views</td>
</tr>
<tr>
<td>GO_FORM</td>
<td>Multi–Form Applications</td>
</tr>
<tr>
<td>HIDE_VIEW</td>
<td>Canvas–Views</td>
</tr>
<tr>
<td>HIDE_WINDOW</td>
<td>Windows</td>
</tr>
<tr>
<td>ID_NULL</td>
<td>All Objects</td>
</tr>
<tr>
<td>IMAGE_ZOOM</td>
<td>Image Items</td>
</tr>
<tr>
<td>INSERT_RECORD</td>
<td>Transaction Processing</td>
</tr>
<tr>
<td>ISSUE_ROLLBACK</td>
<td>Transaction Processing</td>
</tr>
<tr>
<td>ISSUE_SAVEPOINT</td>
<td>Transaction Processing</td>
</tr>
<tr>
<td>LOCK_RECORD</td>
<td>Transaction Processing</td>
</tr>
<tr>
<td>LOGON</td>
<td>Transaction Processing</td>
</tr>
<tr>
<td>LOGON_SCREEN</td>
<td>Transaction Processing</td>
</tr>
<tr>
<td>LOGOUT</td>
<td>Transaction Processing</td>
</tr>
<tr>
<td>MOVE_WINDOW</td>
<td>Windows</td>
</tr>
<tr>
<td>NEXT_FORM</td>
<td>Multi–Form Applications</td>
</tr>
<tr>
<td>POPULATE_GROUP</td>
<td>Record Groups</td>
</tr>
<tr>
<td>POPULATE_GROUP_WITH_QUERY</td>
<td>Record Groups</td>
</tr>
<tr>
<td>POPULATE_LIST</td>
<td>Lists</td>
</tr>
<tr>
<td>PREVIOUS_FORM</td>
<td>Multi–Form Applications</td>
</tr>
<tr>
<td>READ_IMAGE_FILE</td>
<td>Image Items</td>
</tr>
<tr>
<td>REPLACE_CONTENT_VIEW</td>
<td>Windows</td>
</tr>
<tr>
<td>RESET_GROUP_SELECTION</td>
<td>Record Groups</td>
</tr>
<tr>
<td>RESIZE_WINDOW</td>
<td>Windows</td>
</tr>
<tr>
<td>Built-in</td>
<td>Affects</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>RETRIEVE_LIST</td>
<td>Lists</td>
</tr>
<tr>
<td>RUN_PRODUCT</td>
<td>Product Integration</td>
</tr>
<tr>
<td>SCROLL_VIEW</td>
<td>Canvas–Views</td>
</tr>
<tr>
<td>SELECT_ALL</td>
<td>Transaction Processing</td>
</tr>
<tr>
<td>SELECT_RECORDS</td>
<td>Records</td>
</tr>
<tr>
<td>SET_ALERT_PROPERTY</td>
<td>Alerts</td>
</tr>
<tr>
<td>SET_BLOCK_PROPERTY</td>
<td>Blocks</td>
</tr>
<tr>
<td>SET_CANVAS_PROPERTY</td>
<td>Canvas–Views</td>
</tr>
<tr>
<td>SET_FORM_PROPERTY</td>
<td>Forms</td>
</tr>
<tr>
<td>SET_GROUP_CHAR_CELL</td>
<td>Record Groups</td>
</tr>
<tr>
<td>SET_GROUP_DATE_CELL</td>
<td>Record Groups</td>
</tr>
<tr>
<td>SET_GROUP_NUMBER_CELL</td>
<td>Record Groups</td>
</tr>
<tr>
<td>SET_GROUP_SELECTION</td>
<td>Record Groups</td>
</tr>
<tr>
<td>SET_ITEM_PROPERTY</td>
<td>Items</td>
</tr>
<tr>
<td>SET_LOV_PROPERTY</td>
<td>LOVs</td>
</tr>
<tr>
<td>SET_MENU_ITEM_PROPERTY</td>
<td>Menu Items</td>
</tr>
<tr>
<td>SET_PARAMETER_ATTR</td>
<td>Form Parameters</td>
</tr>
<tr>
<td>SET_RADIO_BUTTON_PROPERTY</td>
<td>Radio Buttons</td>
</tr>
<tr>
<td>SET_RECORD_PROPERTY</td>
<td>Records</td>
</tr>
<tr>
<td>SET_RELATIONPROPERTY</td>
<td>Master–Details Relations</td>
</tr>
<tr>
<td>SET_TIMER</td>
<td>Timers</td>
</tr>
<tr>
<td>SET_VIEW_PROPERTY</td>
<td>Canvas–Views</td>
</tr>
<tr>
<td>SET_WINDOW_PROPERTY</td>
<td>Windows</td>
</tr>
<tr>
<td>SHOW_ALERT</td>
<td>Alerts</td>
</tr>
<tr>
<td>SHOW_EDITOR</td>
<td>Editors</td>
</tr>
<tr>
<td>SHOW_LOV</td>
<td>LOVs</td>
</tr>
<tr>
<td>SHOW_VIEW</td>
<td>Canvas–Views</td>
</tr>
<tr>
<td>SHOW_WINDOW</td>
<td>Windows</td>
</tr>
<tr>
<td>UPDATE_RECORD</td>
<td>Transaction Processing</td>
</tr>
<tr>
<td>VALIDATE</td>
<td>All Objects</td>
</tr>
<tr>
<td>VBX.GET_VALUE_PROPERTY</td>
<td>VBX Items</td>
</tr>
<tr>
<td>VBX_SETPROPERTY</td>
<td>VBX Items</td>
</tr>
<tr>
<td>VBX_SET_VALUEPROPERTY</td>
<td>VBX Items</td>
</tr>
<tr>
<td>WRITE_IMAGE_FILE</td>
<td>Image Items</td>
</tr>
</tbody>
</table>
Properties

The following table lists new Version 4.5 properties. For more information, refer to the *Oracle Forms Reference Manual, Vol. 2, Ch. 5, “Properties.”*

**New Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Key</td>
<td>OLE Popup Menu Items</td>
</tr>
<tr>
<td>Application Instance</td>
<td>OLE Resize Style</td>
</tr>
<tr>
<td>Canvas Type</td>
<td>OLE Tenant Aspects</td>
</tr>
<tr>
<td>Column Mapping</td>
<td>OLE Tenant Types</td>
</tr>
<tr>
<td>Column Specification</td>
<td>Parameter Menus</td>
</tr>
<tr>
<td>Console Window</td>
<td>POPUPMENU_CONTENT_ITEM</td>
</tr>
<tr>
<td>Coordinate Information</td>
<td>Previous Navigation Item</td>
</tr>
<tr>
<td>Cursor Style</td>
<td>Record Group Query</td>
</tr>
<tr>
<td>Custom Item Type</td>
<td>Record Group Type</td>
</tr>
<tr>
<td>Display X Position</td>
<td>Record Orientation</td>
</tr>
<tr>
<td>Display Y Position</td>
<td></td>
</tr>
<tr>
<td>Execution Style</td>
<td>Reference Information</td>
</tr>
<tr>
<td>Icon Name (Menu Item)</td>
<td>Rendered</td>
</tr>
<tr>
<td>Item Type</td>
<td>Scroll Bar (Block)</td>
</tr>
<tr>
<td>Items Displayed</td>
<td>Show OLE Popup Menu</td>
</tr>
<tr>
<td>Label (Menu Item)</td>
<td>SHOW_POPUPMENU</td>
</tr>
<tr>
<td>Label (Menu Parameter)</td>
<td>Space Between Records</td>
</tr>
<tr>
<td>List Style</td>
<td>Trigger Style</td>
</tr>
<tr>
<td>LOV Type</td>
<td>Trigger Type</td>
</tr>
<tr>
<td>Menu Item Roles</td>
<td>Update Changed Columns</td>
</tr>
<tr>
<td>Menu Item Type</td>
<td>VBX Control Class</td>
</tr>
<tr>
<td>Mirror Item</td>
<td>VBX Control File Name</td>
</tr>
<tr>
<td>Next Navigation Item</td>
<td>VBX Control Value Property</td>
</tr>
<tr>
<td>OLE Activation Style</td>
<td>WINDOW_HANDLE</td>
</tr>
<tr>
<td>OLE Classes</td>
<td>WINDOW_STATE</td>
</tr>
<tr>
<td>OLE In-place Activation</td>
<td>Window Style</td>
</tr>
<tr>
<td></td>
<td>X Position on Canvas,</td>
</tr>
<tr>
<td></td>
<td>Y Position on Canvas</td>
</tr>
</tbody>
</table>
## Renamed Properties

The following table lists renamed Version 4.5 properties.

<table>
<thead>
<tr>
<th>Previous Name</th>
<th>New Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Table (Block)</td>
<td>Base Table</td>
</tr>
<tr>
<td>Other Values (Check Box)</td>
<td>Check Box Other Values</td>
</tr>
<tr>
<td>Checked (Menu Item)</td>
<td>Checked</td>
</tr>
<tr>
<td>Copy</td>
<td>Copy Value from Item</td>
</tr>
<tr>
<td>Default (Alert)</td>
<td>Default Alert Button</td>
</tr>
<tr>
<td>Default (Form Parameter)</td>
<td>Default Value (Form Parameter)</td>
</tr>
<tr>
<td>Default (Item)</td>
<td>Default Value (Item)</td>
</tr>
<tr>
<td>Default (Menu Substitution Parameter)</td>
<td>Default Value (Menu Substitution Parameter)</td>
</tr>
<tr>
<td>Detail</td>
<td>Detail Block</td>
</tr>
<tr>
<td>Editor Name</td>
<td>Editor</td>
</tr>
<tr>
<td>Editor Position</td>
<td>Editor X Position, Editor Y Position</td>
</tr>
<tr>
<td>List Element</td>
<td>List Elements</td>
</tr>
<tr>
<td>Other Values (List Item, Radio Group)</td>
<td>List Item Other Values, Radio Group Other Values</td>
</tr>
<tr>
<td>Length (Form Parameter)</td>
<td>Maximum Length (Form Parameter)</td>
</tr>
<tr>
<td>Length (Menu Substitution Parameter)</td>
<td>Maximum Length (Menu Substitution Parameter)</td>
</tr>
<tr>
<td>LOV Name</td>
<td>LOV</td>
</tr>
<tr>
<td>Magic</td>
<td>Magic Item</td>
</tr>
<tr>
<td>Main</td>
<td>Main Menu</td>
</tr>
<tr>
<td>Max Length</td>
<td>Maximum Length</td>
</tr>
<tr>
<td>Radio Group</td>
<td>Menu Item Radio Group</td>
</tr>
<tr>
<td>Position (X,Y)</td>
<td>X Position, Y Position</td>
</tr>
<tr>
<td>Prompt</td>
<td>Label (Menu Parameter)</td>
</tr>
<tr>
<td>Radio Group</td>
<td>Radio Group Other Values</td>
</tr>
<tr>
<td>Range</td>
<td>Range High Value, Range Low Value</td>
</tr>
<tr>
<td>Show Keys</td>
<td>Show Keys Description</td>
</tr>
<tr>
<td>Style (Alert)</td>
<td>Alert Style</td>
</tr>
</tbody>
</table>
### Changed Properties

The following object ownership properties are no longer named properties that you must set explicitly.

- Canvas Sequence ID
- Item Sequence ID
- Master Block (relation property)
- Scope (trigger property)

For example, navigation sequence is now implicit from the sequence of objects in the Object Navigator. To change navigation sequence, re-arrange the sequence of objects in the Object Navigator.
Format Masks

In Version 4.5, the Format Mask property supports only ORACLE format masks that are used for both input and output. Input-only format masks, such as WW, are not supported.

The Case Restriction property serves as both an input and output format mask enforced by the user interface, so Case Restriction governs the display of all strings, whether they are entered by an operator or assigned programmatically.

If you programmatically assign string values that conflict with the setting for Case Restriction, you will not see the effect in the text item because its display will be forced to conform to the current setting of Case Restriction. For example, if Case Restriction is set to UPPER for a text item, and a trigger copies a lower-case value into that item, the item will be displayed as upper case. This also means that if data that violates the Case Restriction setting is queried into or programmatically assigned to an item, then the internal value of that text item may differ from what the operator sees on the screen.

3.0 Packaged Procedures and Functions

The following Oracle Forms Version 3.0 packaged procedures and functions are supported in Oracle Forms Version 4.5. In many cases, newer versions of these procedures have been implemented and you are encouraged to use them when creating new applications.

Note: During upgrade, these names will not be changed. The Version 3.0 names will still be supported for conversion purposes. However, you are encouraged to upgrade your existing Version 3.0 applications with the new names when converting from Version 3.0 to Version 4.5.

<table>
<thead>
<tr>
<th>Packaged Procedure/Function</th>
<th>Replacement Version 4.5 Built-in</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANCHOR_VIEW</td>
<td>SET_VIEWPROPERTY</td>
</tr>
<tr>
<td>APPLICATION_CHARACTERISTIC</td>
<td>GET_APPLICATION_PROPERTY</td>
</tr>
<tr>
<td>BLOCK_CHARACTERISTIC</td>
<td>GET_BLOCK_PROPERTY</td>
</tr>
<tr>
<td>CALL</td>
<td>CALL_FORM</td>
</tr>
<tr>
<td>CALL_QUERY</td>
<td>CALL_FORM (...,QUERY_ONLY,...)</td>
</tr>
<tr>
<td>CHANGE_ALERT_MESSAGE</td>
<td>SET_ALERT_PROPERTY</td>
</tr>
<tr>
<td>CLEAR_FIELD</td>
<td>CLEAR_ITEM</td>
</tr>
<tr>
<td>Packaged Procedure/Function</td>
<td>Replacement Version 4.5 Built-in</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>COMMIT</td>
<td>COMMIT_FORM</td>
</tr>
<tr>
<td>DISPLAY_FIELD</td>
<td>SET_ITEMPROPERTY</td>
</tr>
<tr>
<td>DISPLAY_PAGE</td>
<td>SET_CANVASPROPERTY</td>
</tr>
<tr>
<td>DUPLICATE_FIELD</td>
<td>DUPLICATE_ITEM</td>
</tr>
<tr>
<td>EDIT_FIELD</td>
<td>EDIT_TEXTITEM</td>
</tr>
<tr>
<td>FIELD_CHARACTERISTIC</td>
<td>GET_ITEMPROPERTY</td>
</tr>
<tr>
<td>FORM_CHARACTERISTIC</td>
<td>GET_FORMPROPERTY</td>
</tr>
<tr>
<td>GO_FIELD</td>
<td>GO_ITEM</td>
</tr>
<tr>
<td>HIDE_PAGE</td>
<td>HIDE_VIEW</td>
</tr>
<tr>
<td>HIDE_WINDOW</td>
<td></td>
</tr>
<tr>
<td>MOVE_VIEW</td>
<td>SCROLL_VIEW</td>
</tr>
<tr>
<td>NEXT_FIELD</td>
<td>NEXT_ITEM</td>
</tr>
<tr>
<td>OHOST</td>
<td>HOST</td>
</tr>
<tr>
<td>PREVIOUS_FIELD</td>
<td>PREVIOUS_ITEM</td>
</tr>
<tr>
<td>RESIZE_VIEW</td>
<td>for content view: RESIZE_WINDOW</td>
</tr>
<tr>
<td></td>
<td>for stacked view: SET_VIEWPROPERTY</td>
</tr>
<tr>
<td>SET_FIELD</td>
<td>SET_ITEMPROPERTY</td>
</tr>
<tr>
<td>SHOW_PAGE</td>
<td>SHOW_VIEW</td>
</tr>
<tr>
<td></td>
<td>SHOW_WINDOW</td>
</tr>
</tbody>
</table>

For more information, refer to the *Oracle Forms Reference Manual, Vol. 1, Ch. 3, “Built-ins.”*
## SQL*Menu Version 5.0 Packaged Procedures and Functions

This table lists the SQL*Menu Version 5.0 packaged procedures and functions and their status in Oracle Forms Version 4.5. Those marked N/A are no longer supported, but they will still work in an upgraded application. For more information, refer to the Oracle Forms Developer's Guide, Ch. 22, "Basic Menu Design."

<table>
<thead>
<tr>
<th>SQL*Menu Version 5.0 Packaged Procedure/Function</th>
<th>Oracle Forms Version 4.5 Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION_MENU</td>
<td>N/A</td>
</tr>
<tr>
<td>DISABLE_ITEM</td>
<td>SET_MENU_ITEM_PROPERTY</td>
</tr>
<tr>
<td>ENABLE_ITEM</td>
<td>SET_MENU_ITEM_PROPERTY</td>
</tr>
<tr>
<td>EXIT_MENU</td>
<td>N/A</td>
</tr>
<tr>
<td>MAIN_MENU</td>
<td>N/A</td>
</tr>
<tr>
<td>MENU_CLEAR_FIELD</td>
<td>N/A</td>
</tr>
<tr>
<td>MENU_HELP</td>
<td>N/A</td>
</tr>
<tr>
<td>MENU_MESSAGE</td>
<td>N/A</td>
</tr>
<tr>
<td>MENU_NEXT_FIELD</td>
<td>N/A</td>
</tr>
<tr>
<td>MENU_PARAMETER</td>
<td>N/A</td>
</tr>
<tr>
<td>MENU_PREVIOUS_FIELD</td>
<td>N/A</td>
</tr>
<tr>
<td>MENU_REDISPLAY</td>
<td>N/A</td>
</tr>
<tr>
<td>MENU_SHOW_KEYS</td>
<td>N/A</td>
</tr>
<tr>
<td>NEW_APPLICATION</td>
<td>N/A</td>
</tr>
<tr>
<td>NEW_USER</td>
<td>N/A</td>
</tr>
<tr>
<td>NEXT_MENU_ITEM</td>
<td>N/A</td>
</tr>
<tr>
<td>OS_COMMAND</td>
<td>HOST</td>
</tr>
<tr>
<td>OS_COMMAND1</td>
<td>HOST</td>
</tr>
<tr>
<td>PREVIOUS_MENU</td>
<td>N/A</td>
</tr>
<tr>
<td>PREVIOUS_MENU_ITEM</td>
<td>N/A</td>
</tr>
<tr>
<td>TERMINATE</td>
<td>N/A</td>
</tr>
<tr>
<td>WHERE_DISPLAY</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Moving from Character Mode to GUI

Making the transition from a character-mode environment to a GUI environment can be a challenge, both because of the inherent differences between the two environments, and because of the expectations that users bring to GUI applications.

To obtain the advantages of a GUI environment requires detailed planning and execution. Once the planning phase is complete, Oracle Forms 4.5 provides the tools you’ll need to implement your migration plan. Your choice of a migration strategy depends on factors unique to your organization, and defines how much of the potential of Oracle Forms 4.5 your organization can currently employ.

Oracle Forms developers can choose between two basic migration strategies:

- Minimal upgrade
- Complete conversion to GUI
The following table summarizes some differences you may want to consider when choosing your strategy. (Both approaches use Oracle Forms Version 4.5: the difference lies in the degree of change you want to implement at conversion time.)

**Note:** Your choices may be affected by platform–specific capabilities. For example, a minimal upgrade on a UNIX or VMS platform produces an application with same look and feel as a Version 3.0 character mode application, while a minimal upgrade on a DOS platform must be run as a Windows application since a character–mode runtime is not available.

<table>
<thead>
<tr>
<th>Minimal Upgrade</th>
<th>GUI Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong></td>
<td>Deploy a GUI application on GUI platform.</td>
</tr>
<tr>
<td><strong>Advantage</strong></td>
<td>Upgrade as quickly and inexpensively as possible.</td>
</tr>
<tr>
<td><strong>Effort Required</strong></td>
<td>Invest the up–front planning and redesign effort to reap the rewards of increased productivity while meeting user expectations of a GUI application.</td>
</tr>
<tr>
<td><strong>Result</strong></td>
<td>Deploy a 4.5 application as similar to a 3.0 character–mode application as possible.</td>
</tr>
<tr>
<td></td>
<td>Upgrade current screens “as is.”</td>
</tr>
<tr>
<td></td>
<td>Retain key triggers.</td>
</tr>
<tr>
<td></td>
<td>1. Redesign for GUI:</td>
</tr>
<tr>
<td></td>
<td>2. Use Oracle Forms 4.5 to implement the new design.</td>
</tr>
<tr>
<td></td>
<td>Character–mode application running on GUI platform.</td>
</tr>
<tr>
<td></td>
<td>Restrict mouse usage to menu choices only: navigation within the form by keys only.</td>
</tr>
<tr>
<td></td>
<td>GUI interface with buttons, check–boxes, and so on.</td>
</tr>
<tr>
<td></td>
<td>Full mouse navigation.</td>
</tr>
</tbody>
</table>
User Expectations

Oracle Forms customers who have completed the upgrade process report that their clients have specific expectations of GUI applications:

- Visually pleasing window appearance, specifically color
- Less crowded screens (which means designing more windows)
- GUI “widgets,” including scrollbars, which take up room on the canvas
- Proportional fonts (which may not be portable)
- Mouse navigation
- More types of help than in character–mode applications

There are advantages and disadvantages to each approach, and every organization will make a choice based on specific and local needs. Still, it may be useful to review the strategies chosen by some Oracle Forms customers who have completed the upgrade process.

Migration Strategies

It may be useful to review the strategies chosen by some Oracle Forms customers who have completed the migration process.

**Minimal Upgrade:** At Company A, the goal was to move to Forms 4.5 as quickly as possible. They wanted to get something familiar running on Windows while they evaluated how the company could best take advantage of GUI features in the future. So they upgraded all their current character–mode forms to run on Oracle Forms 4.5, deployed the application on Windows, and restricted mouse usage to menu choices only. Operators use the forms exactly the way they used Version 3.0 forms.
Full GUI Conversion: At Company B, the goal was to utilize the increased productivity of a true GUI application. Company B used a two-stage process:

- They re-designed all forms for a GUI interface (including re-designing both the appearance of all forms, and the navigation methods).
- They implemented the re-design using Oracle Forms 4.5.

At Company B, they knew conversion would be a big project, but they wanted the ease-of-use provided by a true GUI application. To make the redesign easier, they adopted standards for the applications’ appearance. They were able to keep all of their previous validation and transactional logic in place.

A Mixed Approach: At Company C, the goal was two-fold:

- Select the simplest applications, quickly upgrade them, disable the mouse and run them with character mode behavior.
- Select the major applications that would benefit most from GUI conversion and invest the time to redesign them.

They focused on two main areas in their redesign:

- They adopted standards for GUI appearance, and redesigned their forms to use the GUI items, such as check boxes and radio buttons.
- They also redesigned the process by which the operator moves through the form. They redesigned some poorly-written key triggers to make use of mouse navigation.

They did, however, keep almost all of their logic: they re-used the PL/SQL code, moving it from key triggers to mouse-oriented triggers.
Conversion Sequence

Whichever migration strategy you choose, your conversion will include three main stages: pre-conversion planning (including creating UI standards), upgrading, and post-conversion fine tuning.

The following table summarizes some differences in the effort required at each stage.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Minimal Upgrade</th>
<th>GUI Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stage 1. Planning</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plan UI changes</td>
<td>1. Review documentation about upgrading (this appendix and release notes).</td>
<td>Define standards for GUI applications. See Advanced Techniques, Ch. 9, “Designing for Portability.”</td>
</tr>
<tr>
<td>Plan navigation changes</td>
<td>Force the use of keys by limiting scope of mouse usage, using the Form property, Mouse Navigation.</td>
<td>Convert key triggers.</td>
</tr>
<tr>
<td>Resolve naming conflicts</td>
<td>Re-name any user-named subprograms that conflict with V.4.5 built-in names, modify any triggers that call these procedures to use new names.</td>
<td>Same</td>
</tr>
<tr>
<td>Plan upgrade sequence</td>
<td>Plan upgrade sequence so all modules used as the source for referenced objects are upgraded before any applications that include references to those objects.</td>
<td>Same</td>
</tr>
<tr>
<td>Widen fields</td>
<td>Plan to use the widen_fields upgrade option, if appropriate.</td>
<td>Same.</td>
</tr>
<tr>
<td><strong>Stage 2. Upgrade</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Set up the database environment for the conversion. 2. Use the Generate component with Convert option and other options as needed.</td>
<td>Same.</td>
<td></td>
</tr>
<tr>
<td><strong>Stage 3. Post-Conversion Fine Tuning</strong></td>
<td>Test to verify that all previous functionality works as it did before upgrading.</td>
<td>Check appearance of all canvas-views and function of all triggers, as well as mouse navigation. Convert character-mode “GUI-like” items to true GUI functionality. Convert pop-up pages to stacked canvases.</td>
</tr>
</tbody>
</table>
If you choose an intermediate migration strategy, you might start with the sequence for the minimal strategy, and add to the planning stage:

- A review of your application for those forms that would benefit most from minimal GUI changes.
- A review of your data for fields you might want to represent using new GUI functionality: buttons, checkboxes, radio groups, and alerts.
- A plan for adding, for example, a tool bar to certain forms and changing the triggers to allow using the mouse on those forms only.

Adding GUI Functionality

When you plan your migration, you may want to consider adding GUI features to represent specific data values:

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buttons</td>
<td>Use buttons to allow a user to start a PL/SQL routine or fire a trigger. May replace frequently-used menu selections.</td>
</tr>
<tr>
<td>Tool bar</td>
<td>Use a tool bar to present a group of buttons that allow a user to execute commands that are common throughout a form.</td>
</tr>
<tr>
<td>Check boxes</td>
<td>Use check boxes to replace a Boolean item, such as a one-character item that can only have a value of On or Off.</td>
</tr>
<tr>
<td>Radio group</td>
<td>Use a radio group to represent a group of possible values that are mutually exclusive, such as Credit Rating: Excellent, Good, Poor. Radio buttons allow the operator to see available values for an item and select one without manually de-selecting the others.</td>
</tr>
<tr>
<td>Alerts</td>
<td>Use alerts (modal pop-up windows) to notify and prompt operators before operations that may cause undesired outcomes, such as quitting before saving changes. The type and content of alerts can be controlled through built-ins.</td>
</tr>
<tr>
<td>Menu</td>
<td>Use a pull-down menu instead of full-screen menus.</td>
</tr>
</tbody>
</table>
Converting Key Triggers

In a GUI environment, the operator navigates primarily using the mouse, rather than keys. When the operator uses mouse navigation, any code based on firing key triggers may not be executed, particularly because with key triggers you can predict a navigation sequence, while with mouse navigation the operator can enter any field at any time. So in the conversion process, the goal is to move functionality that was previously This creates a conversion requirement, because most applications written for character-mode environments make extensive use of key triggers.

Here is one possible approach for converting key triggers:

1. Analyze the application:
   - Print out all triggers (using File→Print).
   - Print out all screens in the application.
   - Create a diagram representing the form by drawing in the triggers on the screen layouts.

2. Analyze the triggers, dividing them into two categories:
   - Navigation–sequence triggers, such as KEY–NXTBLK, that make assumptions about the target for input focus, should be replaced with When–New–Instance triggers. In this way, the functionality will occur when input focus arrives in the target item, rather than when it leaves the previous item.
   - Keyboard–function triggers, such as KEY–HELP, whose function cannot be invoked by simple use of the mouse.

3. Plan how to deal with each category of triggers:
   - Navigation–sequence triggers: The application must determine which navigation event has been performed, where the input focus originated and where it now lies, and which procedure to perform.
   - Keyboard–function triggers: Place this functionality in either the menu or a button, as well as in the key trigger. The menu item or button can fire the relevant DO_KEY operation.
When the operator uses the mouse to navigate from item to item, this freedom of movement can cause a number of problems:

- The operator may navigate into an item that would not normally be available.
- Pre- and post- triggers may fire when they would not be expected.
- Unpredictable routing: no way to be sure how the operator reached this item.

These problems can be lessened by enabling or displaying only items that are currently available for use, using the SET_ITEM_PROPERTY built-in and setting the Enabled and Displayed properties.

**Widening Fields**

The GUI “look and feel” includes field beveling to produce a three-dimensional look. However, the field bevel can take up to 1/2 character per field on some platforms, with the result that some fields may no longer display their full values.

To resolve this problem, use the `widen_fields` option when you generate your applications in Version 4.5.

The effects of the `widen_fields` option will depend on your interface design, and should be tested carefully. Effects can include:

- Text may be cut short.
- If two fields are currently flush against each other, the `widen_fields` option will cause the fields to overlap.
- If two fields currently have one space between them, the `widen_fields` option will cause them to have no space between them.

For more information about the `widen_fields` option, see the *Oracle Forms Reference Manual, Vol. 1, Ch. 1, “Components and Options.”*
Creating a Checkbox from a Version 3.0 Checkbox Field

If you have Version 3.0 applications which simulate checkboxes with single character fields, you will need to convert them to Version 4.5 checkbox items, using the process described below.

<table>
<thead>
<tr>
<th>Version 3.0</th>
<th>Version 4.5 after conversion</th>
<th>Further change required</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-character field</td>
<td>One-character text item</td>
<td>Change item type from text to checkbox</td>
</tr>
<tr>
<td>On-New-Field-Instance trigger calls the do_checkbox procedure</td>
<td>When-New-Item-Instance trigger (performs same function)</td>
<td>Delete trigger</td>
</tr>
<tr>
<td>Do_Checkbox procedure changes the underlying value of the item or changes its appearance from checked to unchecked</td>
<td>Do_Checkbox procedure (performs same function)</td>
<td>Put procedure call into a new When-Checkbox-Changed trigger</td>
</tr>
</tbody>
</table>
Converting Pop-Up Pages

The upgrading process converts pages to content canvases:

- A full-screen page becomes a content canvas-view in a default document window.
- A pop-up page also becomes a content canvas-view in a separate window.

To preserve the character-mode functionality, you may want to render the pop-up pages as stacked canvas-views, instead.

To convert pop-up pages to stacked canvas-views:

1. Convert the form to Version 4.5 and open it in the Designer.
2. In the Navigator, select the window you want to render as a stacked canvas-view.
3. In the Properties window, set the Canvas-view Type property to Stacked.
4. Set the Window property to the name of the window where you want the canvas-view to be displayed.
5. In the Navigator, delete the window the stacked canvas-view was assigned to previously.
6. If you want the stacked canvas-view to disappear when the operator navigates out of it, call the HIDE_VIEW built-in from within a trigger, such as a Post-Block trigger.
This chapter discusses how Oracle Forms stores modules, and includes the following topics:

- Managing Modules  B – 2
- Form Storage Formats  B – 3
- Menu Storage Formats  B – 5
- Library Storage Formats  B – 8
- Portability Among Systems  B – 10
Managing Modules

Form, menu, and library modules that you create in the Designer can be saved to files or in the database. When you save modules to files, they are stored in binary format. When you save modules to the database, they are saved as text and numbers. Module definitions saved both in binary files and in the database are portable across platforms.

When you convert a binary file to a text file, you create a human-readable file that is also portable across platforms.

However, when you generate a design module, you create a platform-specific executable runfile that is not portable across platforms.

The following table shows the file extensions for each type of module and storage format.

<table>
<thead>
<tr>
<th>Type</th>
<th>Binary (Design)</th>
<th>Text</th>
<th>Executable Runfile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>.FMB</td>
<td>.FMT</td>
<td>.FMX</td>
</tr>
<tr>
<td>Menu</td>
<td>.MMB</td>
<td>.MMT</td>
<td>.MMX</td>
</tr>
<tr>
<td>Library</td>
<td>.PLL</td>
<td>.PLD</td>
<td>.PLL</td>
</tr>
</tbody>
</table>

The default file extensions indicate the module type and storage format:

.FMB: Form Module Binary
.FMT: Form Module Text
.FMX: Form Module Executable
.MMB: Menu Module Binary
.MMT: Menu Module Text
/MMX: Menu Module Executable
.PLL: PL/SQL Library Binary
.PLD: PL/SQL Library Text
Form Storage Formats

Oracle Forms stores forms in four formats:

- database format
- .FMB format
- .FMX format
- .FMT format

To store a form in the database or in .FMB format, save the file from the Designer after setting the Module Access option. You can convert a form to .FMX or .FMT format with the Oracle Forms Generate component.

Database Format

Used For: A form in database format consists of the form definition stored in several tables in the ORACLE database.

Oracle Forms uses the database representation of the form when you open the form from the Oracle Forms Designer using File–>Open, providing that the Module Access option is set to Database. (Select Tools–>Options to reach the Options dialog.) If you set the Module Access option to File/Database, you can choose Database or File System from a filter dialog each time you open a file.

Created: First, set the Module Access option to Database. Then, when you save the form definition by selecting File–>Save in the Designer, the form is automatically saved to the database.

Example: You can also load a module into the database from the command line:

```
f45gen custform scott/tiger insert=yes
```
.FMB Format

Used For: The .FMB file is a machine-readable, binary representation of a form that is portable between systems.

Created: Oracle Forms creates an .FMB file when you do the following:

- Select Tools->Options in the Designer, then set Module Access to File on the Options dialog.
- Save your form by selecting File->Save in the Designer.

You can also create an .FMB (binary) file from an .FMT (text) file:

- Issue the Generate command on the command line using the keyword parameter parse=yes.

This operation can also be done from the Designer using the Convert command. For more information, refer to the Menu Commands section in online Help.

Example: f45gen custform parse=yes

Portables: Portable across platforms.

.FMX Format

Used For: The .FMX format of a form is a platform-specific executable runfile that Oracle Forms uses during a Runform session.

Created: Oracle Forms creates the .FMX file when you generate the form in one of the following ways:

- Select File->Run in the Designer when the Generate Before Run option is On.
- Select File->Administration->Generate in the Designer.
- Issue the GENERATE command on the command line.

Example: f45gen custform scott/tiger

Portables: Not portable across platforms.
.FMT Format

**Used For:** The .FMT file is a form in a text format file. Much of the information in the .FMT file, although in text format, actually represents binary data. Images, translatable text strings, boilerplate graphics and boilerplate text are all represented in hexadecimal format.

**Note:** Oracle Corporation reserves the right to change the format of the .FMT file at any time and without notice.

**Created:** Oracle Forms creates the .FMT file from the .FMB file when you issue the Generate command on the command line using the keyword parameter script=yes.

This operation can also be done from the Designer using the Convert command. For more information, refer to the Menu Commands section in online Help.

**Example:** `f45gen custform script=yes`

**Portable:** Portable across platforms.

---

Menu Storage Formats

Oracle Forms stores menus in four formats:

- database format
- .MMB format
- .MMX format
- .MMT format

To store a menu in the database or in .MMB format, save the file from the Designer after setting the Module Access option. You can store a menu in .MMX or .MMT format by using the Generate component of Oracle Forms.
Database Format

Used For: A menu in database format consists of the menu definition stored in several tables in the ORACLE database.

Oracle Forms uses the database representation of the menu when you open the menu from the Oracle Forms Designer using File->Open, providing that the Module Access option is set to Database.  (Use Tools->Options to set the Module Access option on the Options dialog.)  If you set the Module Access option to File/Database, you can choose Database or File System from a filter dialog each time you open a file.

Created: Oracle Forms creates the database representation of a menu when you perform either of the following actions:

- Set Module Access to Database in the Options dialog in the Designer.  When you save the menu by selecting File->Save in the Designer, the menu is automatically saved to the database.

- Convert a menu application in the database from the command line.

Example:  

Example: f45gen custmenu scott/tiger upgrade=yes module_type=menu

Portable: Portable across platforms.

.MMB Format

Used For: The .MMB file is a machine-readable, binary representation of a menu that is portable between systems.

Created: Oracle Forms creates an .MMB file when you do the following:

- Select Tools->Options in the Designer, then set Module Access to File on the Options dialog.

- Save your menu by selecting File->Save in the Designer.

You can also create an .MMB (binary) file from an .MMT (text) file:

- Issue the GENERATE command on the command line using the keyword parameters MODULE_TYPE=MENU and PARSE=YES.

Example:  

Example: f45gen custmenu module_type=menu parse=yes

Portable: Portable across platforms.
.MMX Format

Used For: The .MMX format of a menu is a platform–specific executable runfile that Oracle Forms uses during a Runform session.

Created: Oracle Forms creates the .MMX file when you generate the menu in either of the following ways:
- Select File–>Run in the Designer when the Generate Before Run option is On.
- Select File–>Administration–>Generate in the Designer.
- Issue the GENERATE command on the command line using the keyword parameter MODULE_TYPE=MENU.

Example: `f45gen custmenu scott/tiger module_type=menu`

Portable: Not portable across platforms.

.MMT Format

Used For: The .MMT file is a menu in a text format file. Much of the information in the .MMT file, although in text format, actually represents binary data. Translatable text strings are represented in hexadecimal format.

Created: Oracle Forms creates the .MMT (text) file from the .MMB (binary) file when you generate the form in the following way:
- Issue the GENERATE command on the command line using the keyword parameters MODULE_TYPE=MENU and SCRIPT=YES.

Example: `f45gen custmenu module_type=menu script=yes`

Portable: Portable across platforms.
Library Storage Formats

Oracle Forms stores libraries in four formats:
- database format
- .PLL format
- .PLL format stripped of source code
- .PLD format

To store a library in the database or in .PLL format, save the file from the Designer after setting the Module Access option. You can store a library in .PLD format by using the Generate component of Oracle Forms.

Database Format

Used For: A library in database format consists of the library definition stored in several tables in the ORACLE database.

Oracle Forms uses the database representation of the form when you open the form from the Oracle Forms Designer using File->Open, providing that the Module Access option is set to Database. (Use Tools->Options to reach the Options dialog.) If you set the Module Access option to File/Database, you can choose Database or File System from a filter dialog each time you open a file.

Created: First, set the Module Access option to Database. Then, when you save the library definition by selecting File->Save in the Designer, the library is automatically saved to the database.

Portable: Portable across platforms.

.PLL Format

Used For: The .PLL file is a machine-readable, binary representation that is portable between systems.

Created: Oracle Forms creates a .PLL file when you do the following:
- Select Tools->Options in the Designer, set Module Access to File on the Options dialog.
- Save your library by selecting File->Save in the Designer.

Portable: Portable across platforms.
.PLL Format Stripped of Source Code

**Used For:** The .PLL file stripped of source code is a library file that only contains p-code. The pcde file can be used for final deployment.

**Created:** Oracle Forms creates a .PLL file stripped of source code when you generate with STRIP_SOURCE=YES.
When you use STRIP_SOURCE you must specify an output file by using the OUTPUT_FILE Runform parameter.

**Example:**

```
f45gen module=old_lib.pll userid=scott/tiger strip_source=YES output_file=new_lib.pll
```

**Portable:** Not portable across platforms.

.PLD Format

**Used For:** The .PLD file is the text format file for libraries.
For information about creating text versions of library modules for source control, refer to the Oracle Forms Developer’s Guide, Chapter 3, “Working with Modules.”

**Created:** Oracle Forms creates the .PLD (text) file from the .PLL (binary) file when you generate the library in the following way:

- Issue the GENERATE command on the command line using the keyword parameters MODULE_TYPE=LIBRARY and SCRIPT=YES.

**Example:**

```
f45gen custlib module_type=library script=yes
```

**Portable:** Portable across platforms.
Portability Among Systems

Database and binary file formats are the same on every type of computer system that supports Oracle Forms; executable file formats are not. Therefore, do not try to transport a form from one type of computer to another by transporting the executable (.FMX or .MMX) file. Instead, copy the binary file (.FMB, .MBM, or .PLL) and perform one of the following actions:

- If you want to modify the module, open the binary file in the Designer.
- If you want to run a form, generate the form and any attached menu and library modules to make new .FMX, .MMX and .PLL files on the target platform. If you are going to generate the form from the Designer, you will have to load the form first.

Reserved Words

This appendix lists the reserved words defined in Oracle Forms and in PL/SQL. PL/SQL reserved words appear with an asterisk (*).
CASE_RESTRICTION
CHANGE_STATUS
CHANGE_ALERT_MESSAGE
CHANGE_BLOCK
CHANGE_RECORD
CHARACTER_CELL_HEIGHT
CHARACTER_CELL_WIDTH
CHAR_BASE*
CHAR_COLUMN
CHECKBOX_CHECKED
CHECK*
CHECKED
CHECK_RECORD_UNIQUENESS
CLEAR_BLOCK
CLEAR_EOL
CLEAR_FIELD
CLEAR_FORM
CLEAR_ITEM
CLEAR_LIST
CLEAR_MESSAGE
CLEAR_RECORD
CLOSE*
CLOSE_AT_COMMIT
CLOSE_FORM
CLOSE_QUERY
CLUSTER*
CLUSTERS*
COLAUTH*
COLUMNS*
COLUMN_SECURITY
COUNT*
COUNT_QUERY
COUNT_QUERY
CRASH*
CREATE*
CREATE_GROUP
CREATE_GROUP_FROM_QUERY
CREATE_PARAMETER_LIST
CREATE_QUERIED_RECORD
CREATE_RECORD
CREATE_TIMER
CURRENT*
CURRENT_FORM
CURRENT_FORM_NAME
CURRENT_RECORD
CURRENT_RECORD_ATTRIBUTE
CURRVAL*
CURSOR*
CURSOR_MODE
CURSOR_STYLE
CUT_REGION
DATABASE*
DATABASE_VALUE
DATASOURCE
DATATYPE
DATA_PARAMETER
DATE*
DATE_COLUMN
DB
DBA*
DBMS_ERROR_CODE
DBMS_ERROR_TEXT
DEBUGOFF*
DEBUGON*
DEBUG_MODE
DECIMAL*
DECLARE*
DEFAULT*
DEFAULT_SCOPE
DEFAULT_VALUE
DEFAULT_WHERE
DEFERRED_COORDINATION
DEFER_ON_VISIBILITY
DEFER_REQUIRED_ENFORCEMENT
DEFINITION*
DELAY*
<table>
<thead>
<tr>
<th>Delayed</th>
<th>Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete*</td>
<td>Editor_Name</td>
</tr>
<tr>
<td>Delete_Allowed</td>
<td>Editor_X_Pos</td>
</tr>
<tr>
<td>Delete_Group</td>
<td>Editor_Y_Pos</td>
</tr>
<tr>
<td>Delete_Group_Row</td>
<td>Edit_Field</td>
</tr>
<tr>
<td>Delete_List_Element</td>
<td>Edit_TextItem</td>
</tr>
<tr>
<td>Delete_Parameter</td>
<td>ELSE*</td>
</tr>
<tr>
<td>Delete_Record</td>
<td>ELSIF*</td>
</tr>
<tr>
<td>Delete_Source</td>
<td>Enabled</td>
</tr>
<tr>
<td>Delete_Source_Type</td>
<td>Enable_Item</td>
</tr>
<tr>
<td>Delete_Timer</td>
<td>Enable_Verification</td>
</tr>
<tr>
<td>Delta*</td>
<td>END*</td>
</tr>
<tr>
<td>Derived_Column</td>
<td>End_of_Group</td>
</tr>
<tr>
<td>Desc*</td>
<td>Enforce_Column_Security</td>
</tr>
<tr>
<td>Destroy_Parameter_List</td>
<td>Enforce_Key</td>
</tr>
<tr>
<td>Detail_Name</td>
<td>Enter</td>
</tr>
<tr>
<td>Developer_Names</td>
<td>Enterable</td>
</tr>
<tr>
<td>Digits*</td>
<td>Enter_Query</td>
</tr>
<tr>
<td>Disabled</td>
<td>Entry*</td>
</tr>
<tr>
<td>Disable_Item</td>
<td>Erase</td>
</tr>
<tr>
<td>Disable_Verification</td>
<td>Error</td>
</tr>
<tr>
<td>Display</td>
<td>Error_Code</td>
</tr>
<tr>
<td>Displayed</td>
<td>Error_Notify</td>
</tr>
<tr>
<td>Display_Error</td>
<td>Error_Text</td>
</tr>
<tr>
<td>Display_Field</td>
<td>Error_Type</td>
</tr>
<tr>
<td>Display_Height</td>
<td>Exception*</td>
</tr>
<tr>
<td>Display_Item</td>
<td>Exception_Init*</td>
</tr>
<tr>
<td>Display_Length</td>
<td>Execute_Query</td>
</tr>
<tr>
<td>Display_Page</td>
<td>Execute_Trigger</td>
</tr>
<tr>
<td>Display_Position</td>
<td>Exists*</td>
</tr>
<tr>
<td>Display_Width</td>
<td>Exit*</td>
</tr>
<tr>
<td>Display_X_Pos</td>
<td>Exit_Form</td>
</tr>
<tr>
<td>Display_X_Pos</td>
<td>Exit_Menu</td>
</tr>
<tr>
<td>Dispose*</td>
<td>False*</td>
</tr>
<tr>
<td>Distinct*</td>
<td>Fetch*</td>
</tr>
<tr>
<td>Down</td>
<td>Fetch_Records</td>
</tr>
<tr>
<td>Do*</td>
<td>Field_Characteristic</td>
</tr>
<tr>
<td>Do_Commit</td>
<td>Field_Length</td>
</tr>
<tr>
<td>Do_Key</td>
<td>Filesystem</td>
</tr>
<tr>
<td>Do_Replace</td>
<td>FileName</td>
</tr>
<tr>
<td>Drop*</td>
<td>Find_Alert</td>
</tr>
<tr>
<td>Duplicate_Field</td>
<td>Find_Block</td>
</tr>
<tr>
<td>Duplicate_Item</td>
<td>Find_Canvas</td>
</tr>
<tr>
<td>Duplicate_Record</td>
<td>Find_Column</td>
</tr>
<tr>
<td>Echo</td>
<td>Find_Editor</td>
</tr>
</tbody>
</table>
OPEN*
OPEN_AT_COMMIT
OPEN_FORM
OPERATING_SYSTEM
OPTIMIZER_HINT
OPTION*
OR*
ORDER*
ORDER_BY
OS_COMMAND
OS_COMMAND1
OTHERS*
OUT *
PACKAGE*
PARAMLIST
PARAMTYPE_PARAMA
PARAM_ATTR
PARTITION*
PASSWORD
PASTE_REGION
PAUSE
PCTFREE*
PECS
POPULATE_GROUP
POPULATE_GROUP_WITH_QUERY
POPULATE_LIST
POPUPMENU_COPY_ITEM
POPUPMENU_CUT_ITEM
POPUPMENU_DELETE_ITEM
POPUPMENU_LINKS_ITEM
POPUPMENU_OBJECT_ITEM
POPUPMENU_PASTE_SPEC_ITEM
POPUPMENU_PASTE_ITEM
POSITION
POSITION_ON_CANVAS
POSITIVE*
POST
PRAGMA*
PREVENT_MASTERLESS_OPERATION
PREVIOUSBLOCK
PREVIOUS_FIELD
PREVIOUS_ITEM
PREVIOUS_BLOCK
PREVIOUS_FORM
PREVIOUS_ITEM
PREVIOUS_MENU
PREVIOUS_MENU_ITEM
PREVIOUS_NAVIGATION_BLOCK
PREVIOUS_NAVIGATION_ITEM
PREVIOUS_RECORD
PRIOR*
PRIVATE*
PROCEDURE*
PROCEDURE_SOURCE
PROPERTY_FALSE
PROPERTY_OFF
PROPERTY_ON
PROPERTY_TRUE
PUBLIC *
QUERYABLE
QUERY_ALLOWED
QUERY_HITS
QUERY_LENGTH
QUERY_ONLY
QUERY_OPTIONS
QUERY_SOURCE
QUERY_SOURCE_TYPE
QUERY_STATUS
RAISE*
RANGE*
RANGE_HIGH
RANGE_LOW
READ_IMAGE_FILE
REAL*
RECORD*
RECORDGROUP
RECORDS_DISPLAYED
RECORDS_TO_FETCH
RECORD_SCOPE
REDISPLAY
RELATION
RELEASE*
REM*
REMOVE_LIST
REMOVE_ON_EXIT
RENAME*
REPEAT
REPLACE_CONTENT_VIEW
REPLACE_MENU
REPORTS
REQUIRED
RESET_GROUP_SELECTION
RESIZE_VIEW
RESIZE_WINDOW
RESOURCE*
RESTRICT
RETRIEVE_LIST
RETURN*
REVERSE*
REVOKE*
ROLLBACK*
ROLLBACK_FORM
ROLLBACK_NR
ROLLBACK_SV
ROWID*
ROWLABEL*
ROWNUM*
ROWTYPE*
RUN*
RUNTIME
RUN_PRODUCT
SAME_RECORD
SAVEPOINT*
SAVEPOINT_MODE
SAVEPOINT_NAME
SCHEMA*
SCROLLBAR
SCROLL_DOWN
SCROLL_UP
SCROLL_VIEW
SECURE
SELECT*
SELECTION_RECTANGLE
SELECT_ALL
SELECT_RECORDS
SEPARATE*
SESSION
SET*
SET_ALERT_PROPERTY
SET_APPLICATION_PROPERTY
SET_BLOCK_PROPERTY
SET_CANVAS_PROPERTY
SET_FORM_PROPERTY
SET_GROUP_CHAR_CELL
SET_GROUP_DATE_CELL
SET_GROUP_NUMBER_CELL
SET_GROUP_SELECTION
SET_INPUT_FOCUS
SET_ITEM_PROPERTY
SET_LOV_PROPERTY
SET_MENU_ITEM_PROPERTY
SET_PARAMETER_ATTR
SET_RADIO_BUTTON_PROPERTY
SET_RECORD_PROPERTY
SET_RELATION_PROPERTY
SET_TIMER
SET_VIEW_PROPERTY
SET_WINDOW_PROPERTY
SHOWED
SHOW_ALERT
SHOW_BACKGROUND_MENU
SHOW_EDITOR
SHOW_KEYS
SHOW_LOV
SHOW_MENU
SHOW_PAGE
SHOW_POPUPMENU
SHOW_WINDOW
SIZE*
SMALLINT*
SPACE*
SPECIFIC_EXCEPTION
SQL*
SQLCODE*
SQLERROR*
SQLFORMS
SQLMENU
STACKED
START*
STATUS
This chapter discusses processing constraints for Oracle Forms:

- Object constraints
- Cursor constraints

Note: Many constraints, such as memory constraints, are machine-specific and are not discussed in this book. For information on platform-specific constraints, refer to the Oracle Forms documentation for your operating system.
Object Constraints

The following table presents constraints that apply to Oracle Forms objects and any other elements that are related to these objects.

<table>
<thead>
<tr>
<th>Element</th>
<th>Constraint/Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>blocks per form</td>
<td>no practical limit</td>
</tr>
<tr>
<td>canvases/windows</td>
<td>no practical limit</td>
</tr>
<tr>
<td>canvas width</td>
<td>no practical limit</td>
</tr>
<tr>
<td>column name length</td>
<td>32 characters</td>
</tr>
<tr>
<td>boilerplate text</td>
<td>no practical limit</td>
</tr>
<tr>
<td>database field references</td>
<td>no practical limit</td>
</tr>
<tr>
<td>detail blocks per master</td>
<td>no practical limit</td>
</tr>
<tr>
<td>global variable length</td>
<td>255 characters</td>
</tr>
<tr>
<td>item max length</td>
<td>2,048 characters, for CHAR values</td>
</tr>
<tr>
<td>items per block</td>
<td>no practical limit</td>
</tr>
<tr>
<td>join condition</td>
<td>255 characters</td>
</tr>
</tbody>
</table>
| levels of menu nesting           | 8 – 10, depending on your operating system  
(When using pulldown menus, it is a good practice to limit menu nesting to four levels.) |
| list of values                   | no practical limit                   |
| menu items                       | no practical limit                   |
| menus                            | no practical limit                   |
| named visual attributes          | 255 named visual attributes          |
| PL/SQL strings                   | no practical limit                   |
| WHERE Clause and ORDER BY Clause | total of 32,000 characters            |
| record group character columns   | 255 characters                       |
| record groups                    | 255 columns                          |
| substitution parameters          | no practical limit                   |
| text item range checks per block | no practical limit                   |
| triggers                         | no practical limit                   |
| user-defined local variable name length | 30 characters                       |
# Cursor Constraints

A database cursor is the memory that the ORACLE RDBMS assigns to a SQL command. This applies to the SQL commands that are implicit to posting and querying data and the SQL commands that are explicit in triggers. Oracle Forms controls cursor assignment for all implicit SQL commands and the explicit SQL commands in V2 triggers.

The following table presents constraints that apply to database cursors.

<table>
<thead>
<tr>
<th>Element</th>
<th>Constraint/Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>open cursors</td>
<td>(maximum) value of the OPEN_CURSORS parameter found in the INIT.ORA file</td>
</tr>
<tr>
<td></td>
<td>• 5 per ORACLE connection.</td>
</tr>
<tr>
<td></td>
<td>• 4 per runtime session when running against ORACLE or SQL*Connect.</td>
</tr>
<tr>
<td></td>
<td>• 5 per block (1 for SELECT, 1 for UPDATE, 1 for INSERT, 1 for DELETE, and 1 for QUERY).</td>
</tr>
<tr>
<td>open cursors</td>
<td>(minimum) The first 4 listed are opened as needed, but can be suppressed by running in OPTIMIZETP=YES mode. This restriction applies under normal circumstances, but it is possible to change the number of cursors by altering settings on the Set Options form or by using command line switches.</td>
</tr>
<tr>
<td></td>
<td>• 1 per record group created with query.</td>
</tr>
<tr>
<td>open cursors</td>
<td>(minimum) One cursor for query is always opened for each block as needed and is never shared even when running in OPTIMIZETP=NO mode.</td>
</tr>
</tbody>
</table>

Constraints D – 3
<table>
<thead>
<tr>
<th>Element</th>
<th>Constraint/Note</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• 1 cursor is used if database date or time default values ($$DBDATE$$) are referenced for any item.</td>
</tr>
<tr>
<td></td>
<td>• 1 for any SQL statement. These can be in triggers, procedures, or functions. This restriction applies under normal circumstances, but it is possible to change the number of cursors by altering settings on the Set Options form or by using command line switches. V2 style trigger cursor consumption can be tailored using the OPTIMIZESQL command line parameter.</td>
</tr>
</tbody>
</table>
Index

A
ABORT_QUERY, 3 – 17, 8 – 2
Accelerator, 5 – 21
Access keys, definition, 5 – 21
ADD_GROUP_COLUMN, 3 – 18
ADD_GROUP_ROW, 3 – 20
ADD_LIST_ELEMENT, 3 – 22
ADD_PARAMETER, 3 – 23
Add Triggers option, 1 – 23
Adjust, 5 – 184
Adjust–to–fit, 5 – 160
Alert built–ins
FIND_ALERT, 3 – 89
ID_NULL, 3 – 180
SET_ALERT_BUTTON_PROPERTY, 3 – 243
SET_ALERT_PROPERTY, 3 – 244
SHOW_ALERT, 3 – 286
Alert Properties
Alert Style, 5 – 22
Button 1, Button 2, Button 3, 5 – 31
Message, 5 – 130
table of Alert Properties, 5 – 5
Alert Style, 5 – 22
Alignment, 5 – 23
ALPHA data type, 5 – 56
Application built–ins
DO_KEY, 3 – 72
GET_APPLICATION_PROPERTY, 3 – 119
HOST, 3 – 178
PAUSE, 3 – 215
SET_APPLICATION_PROPERTY, 3 – 245
USER_EXIT, 3 – 297
Application Instance, 5 – 24
Application menu (V.3 term), A – 24
Application Properties
Calling_Form, 5 – 32
Connect_String, 5 – 44
Current_Form, 5 – 50
Current_Form_Name, 5 – 51
Datasource, 5 – 61
Directory, 5 – 72
Display_Height, 5 – 75
Display_Width, 5 – 76
Operating_System, 5 – 152
Password, 5 – 155
Savepoint_Name, 5 – 177
table of Application Properties, 5 – 5
Timer_Name, 5 – 188
User_Interface, 5 – 202
User_NLS_Lang, 5 – 203
Username, 5 – 202
APPLICATION_PARAMETER, 3 – 24
Array Processing option, 1 – 11
Asynchronously, Run Modules, 1 – 37
Auto Hint, 5 – 24
Auto–Confirm, 5 – 25
Auto–Display, 5 – 25
Auto–Query, 5 – 25
Auto–Refresh, 5 – 26
Auto–Skip (item), 5 – 28
Auto–Skip (LOV), 5 – 28
B

Background color, 5 – 214
BACKGROUND_MENU, 3 – 25
Base Table (Block), 5 – 29
Base Table (Item), 5 – 29
Batch option, 1 – 23
BELL, 3 – 26
Bevel, 5 – 30
Beveling, A – 46
Bg color, 5 – 214
Binary files, converting to text files, A – 10
Block built–ins
   BLOCK_MENU, 3 – 26
   CLEAR_BLOCK, 3 – 34
   FIND_BLOCK, 3 – 90
   GET_BLOCK_PROPERTY, 3 – 123
   GO_BLOCK, 3 – 171
   ID_NULL, 3 – 180
   NEXT_BLOCK, 3 – 208
   PREVIOUS_BLOCK, 3 – 220
   SET_BLOCK_PROPERTY, 3 – 246
Block Properties
   Base Table (Block), 5 – 29
   Column Security property, 5 – 40
   Current Record, 5 – 52
   Delete Allowed, 5 – 66
   Enterable, 5 – 79
   First_Block, 5 – 83
   First_Detail_Relation, 5 – 84
   First_Master_Relation, 5 – 85
   In Menu/Block Description, 5 – 102
   Insert Allowed, 5 – 103
   Join Condition, 5 – 107
   Key Mode, 5 – 109
   Last_Item, 5 – 112
   Locking Mode, 5 – 114
   Navigation Style, 5 – 139
   Next Navigation Block, 5 – 140
   NextBlock, 5 – 141
   Optimizer_Hint, 5 – 152
   Order By Clause, 5 – 217
   Previous Navigation Block, 5 – 156
   PreviousBlock, 5 – 158
   Primary Key (Block), 5 – 156
   Query Allowed, 5 – 160
   Query_Hits, 5 – 163
   Query_Options, 5 – 164
   Record Orientation, 5 – 169
   Records Buffered, 5 – 169
   Records Displayed, 5 – 170
   Records Fetched, 5 – 171
   Records_to_Fetch, 5 – 171
   Scroll Bar, 5 – 178
   Status, 5 – 186
   table of Block Properties, 5 – 6
   Top_Record, 5 – 190
   Transactional Triggers, 5 – 190
   Update Allowed, 5 – 193
   Where Clause/Order By Clause, 5 – 217
   BLOCK_MENU, 3 – 26, 8 – 3
   Block_Menu (Runform option), 1 – 11
   Blocks, flushing programmatically, 3 – 36
   Bottom Title, 5 – 31
   BREAK, 3 – 27
   Buffer Records in File option, 1 – 12
   Built–in subprograms, 3 – 1
      ABORT_QUERY, 3 – 17
      ADD_GROUP_COLUMN, 3 – 18
      ADD_GROUP_ROW, 3 – 20
      ADD_LIST_ELEMENT, 3 – 22
      ADD_PARAMETER, 3 – 23
      APPLICATION_PARAMETER, 3 – 24
      BACKGROUND_MENU, 3 – 25
      BELL, 3 – 26
      BLOCK_MENU, 3 – 26
      BREAK, 3 – 27
      CALL_FORM, 3 – 28
      CALL_INPUT, 3 – 31
      CHECK_RECORD_UNIQUENESS, 3 – 33
      CHECKBOX_CHECKED, 3 – 31
      COMMIT_FORM, 3 – 41
      COMMIT_INPUT, 3 – 41
      COMMIT_RECORD, 3 – 41
      COPY, 3 – 44
      COPY_REGION, 3 – 45
      COUNT_QUERY, 3 – 46

Index – 2   Forms Reference Manual
HIDE_VIEW, 3 – 176
HIDE_WINDOW, 3 – 176
HOST, 3 – 178
ID_NULL, 3 – 180
IMAGE_ZOOM, 3 – 181
INSERT_RECORD, 3 – 183
ISSUE_ROLLBACK, 3 – 184
ISSUE_SAVEPOINT, 3 – 185
ITEM_ENABLED, 3 – 186
LAST_RECORD, 3 – 186
LIST_VALUES, 3 – 187
LOCK_RECORD, 3 – 188
LOGON, 3 – 189
LOGON_SCREEN, 3 – 190
LOGOUT, 3 – 192
MAIN_MENU, 3 – 193
MENU_CLEAR_FIELD, 3 – 193
MENU_NEXT_FIELD, 3 – 194
MENU_PARAMETER, 3 – 194
MENU_PREVIOUS_FIELD, 3 – 195
MENU_REDISPLAY, 3 – 195
MENU_SHOW_KEYS, 3 – 195
MESSAGE, 3 – 196
MESSAGE_CODE, 3 – 197
MESSAGE_TEXT, 3 – 198
MESSAGE_TYPE, 3 – 199
MOVE_WINDOW, 3 – 200
NAME_IN, 3 – 202
new for Version 4.5, A – 29
NEW_FORM, 3 – 205
NEXT_BLOCK, 3 – 208
NEXT_FORM, 3 – 209
NEXT_ITEM, 3 – 210
NEXT_KEY, 3 – 211
NEXT_MENU_ITEM, 3 – 211
NEXT_RECORD, 3 – 212
NEXT_SET, 3 – 213
OPEN_FORM, 3 – 213
PASTE_REGION, 3 – 215
PAUSE, 3 – 215
POPULATE_GROUP, 3 – 216
POPULATE_GROUP_WITH_QUERY, 3 – 217
POPULATE_LIST, 3 – 218
POST, 3 – 220
PREVIOUS_BLOCK, 3 – 220
PREVIOUS_FORM, 3 – 221
PREVIOUS_ITEM, 3 – 222
PREVIOUS_MENU, 3 – 223
PREVIOUS_MENU_ITEM, 3 – 223
PREVIOUS_RECORD, 3 – 224
PRINT, 3 – 224
QUERY_PARAMETER, 3 – 225
READ_IMAGE_FILE, 3 – 227
REDISPLAY, 3 – 228
REPLACE_CONTENT_VIEW, 3 – 229
REPLACE_MENU, 3 – 230
RESET_GROUP_SELECTION, 3 – 232
RESIZE_WINDOW, 3 – 233
RETRIEVE_LIST, 3 – 234
RUN_PRODUCT, 3 – 235
SCROLL_DOWN, 3 – 238
SCROLL_UP, 3 – 238
SCROLL_VIEW, 3 – 239
SELECT_ALL, 3 – 241
SELECT_RECORDS, 3 – 242
SET_ALERT_BUTTON_PROPERTY, 3 – 243
SET_ALERT_PROPERTY, 3 – 244
SET_APPLICATION_PROPERTY, 3 – 245
SET_BLOCK_PROPERTY, 3 – 246
SET_CANVAS_PROPERTY, 3 – 250
SET_FORM_PROPERTY, 3 – 251
SET_GROUP_CHAR_CELL, 3 – 255
SET_GROUP_DATE_CELL, 3 – 256
SET_GROUP_NUMBER_CELL, 3 – 258
SET_GROUP_SELECTION, 3 – 259
SET_INPUT_FOCUS, 3 – 260
SET_ITEM_PROPERTY, 3 – 260
SET_LOV_COLUMN_PROPERTY, 3 – 270
SET_LOV_PROPERTY, 3 – 271
SET_MENU_ITEM_PROPERTY, 3 – 272
SET_PARAMETER_ATTR, 3 – 274
SET_RADIO_BUTTON_PROPERTY, 3 – 274
SET_RECORD_PROPERTY, 3 – 276
SET_RELATION_PROPERTY, 3 – 278
SET_TIMER, 3 – 280
SET_VIEW_PROPERTY, 3 – 282
SET_WINDOW_PROPERTY, 3 – 284
SHOW_ALERT, 3 – 286
SHOW_BACKGROUND_MENU, 3 – 287
SHOW_EDITOR, 3 – 288
SHOW_KEYS, 3 – 290
SHOW_LOV, 3 – 290
SHOW_MENU, 3 – 291
SHOW_VIEW, 3 – 292
SHOW_WINDOW, 3 – 293
SYNCHRONIZE, 3 – 294
TERMINATE, 3 – 295
UNSET_GROUP_SELECTION, 3 – 295
UP, 3 – 296
UPDATE_RECORD, 3 – 296
USER_EXIT, 3 – 297
VALIDATE, 3 – 299
VBX.FIRE_EVENT, 3 – 300
VBX.GET_PROPERTY, 3 – 301
VBX.GET_VALUE_PROPERTY, 3 – 302
VBX.INVOKE_METHOD, 3 – 303
VBX.SET_PROPERTY, 3 – 304
VBX.SET_VALUE_PROPERTY, 3 – 305
WHERE_DISPLAY, 3 – 306
WRITE_IMAGE_FILE, 3 – 306
Built-in Subprograms Tables, 3 – 7
Button 1, Button 2, Button 3, 5 – 31

C
CALL_FORM, 3 – 28, 8 – 4
CALL_INPUT, 3 – 31, 8 – 5
Calling a form
issuing savepoints for a called form, 3 – 29
passing parameters with a parameter list, 3 – 51
Calling_Form, 5 – 32
Canvas
Canvas property, 5 – 32
SET_CANVAS_PROPERTY, 3 – 250
setting VISUAL_ATTRIBUTE property, 3 – 266
Canvas and view built-ins
FIND_CANVAS, 3 – 91
FIND_VIEW, 3 – 99
GET_CANVAS_PROPERTY, 3 – 128
GET_VIEW_PROPERTY, 3 – 167
HIDE_VIEW, 3 – 176
ID_NULL, 3 – 180
PRINT, 3 – 224
SCROLL_VIEW, 3 – 239
SET_CANVAS_PROPERTY, 3 – 250
SET_VIEW_PROPERTY, 3 – 282
SHOW_VIEW, 3 – 292
Canvas–View Properties
Display X Position, 5 – 73
Display Y Position, 5 – 73
Displayed, 5 – 74
Horizontal Scroll Bar, 5 – 97
table of Canvas–view Properties, 5 – 7
Canvas–view Properties
Raise on Entry, 5 – 164
Size, 5 – 182
View Height, 5 – 210
View Horizontal Scroll Bar, 5 – 210
View Vertical Scroll Bar, 5 – 211
View Width, 5 – 210
Window, 5 – 218
X Position on Canvas, 5 – 224
Y Position on Canvas, 5 – 224
Canvas–view Type property, 5 – 33
Case Insensitive Query, 5 – 34
Case Restriction, 5 – 35
CHANGE_ALERT_MESSAGE, (now
SET_ALERT_PROPERTY), 3 – 244
CHAR data type, 5 – 56
Character Cell WD/HT, 5 – 36
Character mode logical attributes, 5 – 214
Characteristic (V.3 term), A – 24
Check Block for Query, 8 – 6
Check Block for Update, 8 – 7
Check Box Other Values, 5 – 36
Check Item for Edit, 8 – 8
Check Record for Update, 8 – 11
Check Record Uniqueness, 8 – 10
CHECK_RECORD_UNIQUENESS, 3 – 33
CHECKBOX_CHECKED, 3 – 31
Checked, 5 – 37
Checked Value, 5 – 37
Checked value (Check Box), 5 – 36
Class, 5 – 38
CLEAR_BLOCK, 3 – 34
CLEAR_EOL, 3 – 35, 8 – 13
CLEAR_FORM, 3 – 36, 8 – 14
CLEAR_ITEM, 8 – 15
CLEAR_ITEM (V.3 “CLEAR_FIELD”), 3 – 37
CLEAR_LIST, 3 – 38
CLEAR_MESSAGE, 3 – 39
CLEAR_RECORD, 3 – 40, 8 – 16
Close the Query, 8 – 17
CLOSE_FORM, 3 – 41
Closeable, 5 – 38
Color Mode option, 1 – 33
Color Palette option, 1 – 34
Column Mapping, 5 – 39
Column name length, constraints, D – 2
Column Security, 5 – 40
Column Specification, 5 – 41
Command line switches, A – 26
Command syntax, iv, iv
Command Text, 5 – 42
Command Type, 5 – 43
Comment, 5 – 44
Commit Processing, 7 – 11
  changing data during commit processing, 7 – 14
  posting data, 7 – 11
  processing inserts, updates, and deletes, 7 – 11
Commit processing, 3 – 34
COMMIT_FORM, 3 – 41, 8 – 18
Compression, 5 – 44
Configuration
  Forms components, 1 – 4
  User Preference File, 1 – 39
Connect_String, 5 – 44
Console Window, 5 – 45
Constraints
  canvases/windows, D – 2
  column name length, D – 2
  cursors, D – 3
  global variable length, D – 2
  item max length, D – 2
  join condition, D – 2
  menu items, D – 2
  menu nesting levels, D – 2
  menus, D – 2
  named visual attributes, D – 2
  objects, D – 2
  properties, D – 2
  record group character columns, D – 2
  record groups, D – 2
  substitution parameters, D – 2
  triggers, D – 2
  variable name length, D – 2
Convert command, .FMT format, B – 5
CONVERT_OTHER_VALUE, 3 – 43
Converting
  from character-mode applications, A – 39
  from Version 2.3, A – 5
  from Version 3.0, A – 4
  from Version 4.0, A – 3
  menus, A – 11
  planning and strategy, A – 43
  pop-up pages, A – 48
  running after conversion, A – 5
  standalone menus, A – 18
Coordinate Information, 5 – 45
Coordinate System, 5 – 45
Coordinates
  editor position, 5 – 77
  LOV position, 5 – 118
Coordination, 5 – 47
Coordination_Status, 5 – 48
COPY, 8 – 19
COPY (NAME_IN), 3 – 202
COPY built-in routine, 3 – 44
Copy Value from Item property, 5 – 49
COPY_REGION, 3 – 45
COUNT_QUERY, 3 – 46, 8 – 20
CREATE_GROUP, 3 – 47
CREATE_GROUP_FROM_QUERY, 3 – 49
CREATE_PARAMETER_LIST, 3 – 51
CREATE_QUERIED_RECORD, 3 – 52
CREATE_RECORD, 3 – 54, 8 – 21
CREATE_TIMER, 3 – 55
Crop, 5 – 184
CRT_File option, 1 – 23
Currency masks, 5 – 88
Current Record Attribute, 5 – 49
Current_Form, 5 – 50
Current_Form_Name, 5 – 51
Current_Record, 5 – 52
Cursor_Mode, 5 – 52
Cursor_Style, 5 – 54
Cursors
  constraints, D – 3
  database, D – 3
keeping open across commits, 5 – 52
Optimize SQL Processing option, 1 – 15
Optimize Transaction Mode Processing, 1 – 16
Statistics (Runform option), 1 – 20
CUT_REGION, 3 – 56

D
Data Type, 5 – 55
Data types
ALPHA, 5 – 56
CHAR, 5 – 56
DATE, 5 – 56
DATETIME, 5 – 56
INT, 5 – 57
LONG, 5 – 58
NUMBER, 5 – 58
Database
cursors, D – 3
login, 1 – 8
Database storage
libraries, B – 8
menus, B – 6
Database_Value, 5 – 60
Datasource, 5 – 61
Date and time system default values, 4 – 3
Date and Time System Variables
$$DATE$$, 4 – 5
$$DATETIME$$, 4 – 6
$$DBDATE$$, 4 – 7
$$DBDATETIME$$, 4 – 8
$$DBTIME$$, 4 – 9
$$TIME$$, 4 – 9
DATE data type, 5 – 56
$$DATE$$, 4 – 5
Dates, formatting, 5 – 88
DATETIME data type, 5 – 56
$$DATETIME$$, 4 – 6
$$DBDATE$$, 4 – 7
$$DBDATETIME$$, 4 – 8
DBMS_ERROR_CODE, 3 – 57
DBMS_ERROR_TEXT, 3 – 58
$$DBTIME$$, 4 – 9
Debug Messages option, 1 – 12
Debug Mode (Generate option), 1 – 24
Debug Mode (Runtime option), 1 – 12
DEBUG_MODE, 3 – 59
Default Alert Button, 5 – 61
Default Block facility, Item Type, 5 – 105
Default Button, 5 – 62
Default Directory, 5 – 72
Default file extensions, B – 2
Default Font Scaling, 5 – 62
Default Order By Clause, 5 – 217
Default value, 5 – 63, 5 – 65
Default Value (Form Parameter), 5 – 63
Default Value (Item), 5 – 63
Default Value (Menu Substitution Parameter), 5 – 65
DEFAULT_VALUE, 3 – 60, 8 – 22
Defer_Required_Enforcement, 5 – 65
Deferred, 5 – 65
Deferred (Coordination property), 5 – 47
Delete Allowed, 5 – 66
Delete option, 1 – 24
DELETE_GROUP, 3 – 61
DELETE_GROUP_ROW, 3 – 62
DELETE_LIST_ELEMENT, 3 – 64
DELETE_PARAMETER, 3 – 65
DELETE_RECORD, 3 – 66, 8 – 23
DELETE_TIMER, 3 – 67
delimiters, 5 – 88
Designer component
list of options, 1 – 31
starting, 1 – 4
Designer options
Color Mode, 1 – 33
Color Palette, 1 – 34
Generate Before Run, 1 – 34
Help, 1 – 35
Module Access, 1 – 35
Module_Type, 1 – 36
Printer, 1 – 36
Run Asynchronously, 1 – 37
Save Before Generate, 1 – 37
Suppress Hints, 1 – 37
Term, 1 – 38
Use System Editor, 1 – 38
DESTROY_PARAMETER_LIST, 3 – 69
Detail Block, 5 – 66
Direction, 5 – 67
Directory, 5 – 72
DISABLE_ITEM, A – 38
Disabled
  menu items, 5 – 73, 5 – 78
  text items, 5 – 104
Display Block Menu option, 1 – 11
Display Screen to Specify Logon option, 1 – 14
Display w/o Privilege, 5 – 73
Display X Position, 5 – 73
Display Y Position, 5 – 73
DISPLAY_ERROR, 3 – 70
Display_Height, 5 – 75
DISPLAY_ITEM, 3 – 70
Display_Width, 5 – 76
Displayed (Canvas–view), 5 – 74
Displayed (Item), 5 – 74
Displayed (Menu Item), 5 – 75
DO_KEY, 3 – 72, 8 – 25
Double–byte characters, 5 – 123
DOWN, 3 – 71, 8 – 26
DUPLICATE_ITEM, 3 – 73, 8 – 27
DUPLICATE_RECORD, 3 – 74, 8 – 28

E
EDIT_TEXTITEM, 3 – 75
Editor, 5 – 76
Editor Properties
  Bottom Title, 5 – 31
  Size, 5 – 182
  table of Editor Properties, 5 – 8
  Top Title, 5 – 189
  Vertical Scroll Bar, 5 – 208
  Wrap Style, 5 – 222
  X Position, 5 – 222
  Y Position, 5 – 222
  Editor X Position, 5 – 77
  Editor Y Position, 5 – 77
  ENABLE_ITEM, A – 38
  Enabled (Item), 5 – 77
  Enabled (Menu Item), 5 – 78
  END_OF_GROUP constant, 3 – 20
  ENFORCE_COLUMN_SECURITY, 3 – 76
  ENTER, 3 – 77, 8 – 29
  Enter query mode, 5 – 82
  Enter the Block, 8 – 31
  Enter the Form, 8 – 32
  Enter the Item, 8 – 33
  Enter the Record, 8 – 35
  Enter the Value into an Item, 8 – 36
  ENTER_QUERY, 3 – 77, 8 – 34
  Enterable, 5 – 79
  Enterable field characteristic (V.3 term), A – 24
  Environments, portability among, B – 10
  ERASE, 3 – 79
  ERROR_CODE, 3 – 79
  ERROR_TEXT, 3 – 80
  ERROR_TYPE, 3 – 81
  EXEC IAF GET/PUT statements (V.3), A – 22
  EXEC TOOLS statements, A – 22
  EXECUTE_QUERY, 3 – 82, 8 – 37
  EXECUTE_TRIGGER, 3 – 84, 8 – 39
  Exemacro, 1 – 27
  EXIT_FORM, 3 – 85, 8 – 40
  Extensions, file formats, A – 9
  Extract option, 1 – 24

F
Fetch Records, 8 – 41
FETCH_RECORDS, 3 – 87
Field (V.3 term), A – 24
File (Menu property), 5 – 80
File formats, A – 9, A – 21
File_Name, 5 – 81
Files
- changes in file formats from V.3, A – 9
- conversion between binary and text files, A – 10
  - Directory property, 5 – 72
  - file extensions, B – 2

FIND_ALERT, 3 – 89
FIND_BLOCK, 3 – 90
FIND_CANVAS, 3 – 91
FIND_COLUMN, 3 – 91
FIND_EDITOR, 3 – 92
FIND_FORM, 3 – 93
FIND_GROUP, 3 – 94
FIND_ITEM, 3 – 94
FIND_LOV, 3 – 95
FIND_MENU_ITEM, 3 – 96
FIND_RELATION, 3 – 97
FIND_TIMER, 3 – 98
FIND_VIEW, 3 – 99
FIND_WINDOW, 3 – 100
Fire in Enter Query Mode, 5 – 82
First Navigation Block, 5 – 83
First Block, 5 – 83
First_Detail_Relation, 5 – 84
First_Item, 5 – 84
First_Master_Relation, 5 – 85
FIRST_RECORD, 3 – 101, 8 – 44
Fixed Length (Item), 5 – 86
Fixed Length (Menu Substitution Param), 5 – 86
Fixed Size, 5 – 87

Flowcharts
- ABORT_QUERY, 8 – 2
- BLOCK_MENU, 8 – 3
- CALL_FORM, 8 – 4
- CALL_INPUT, 8 – 5
- Check Block for Query, 8 – 6
- Check Block for Update, 8 – 7
- Check Item for Edit, 8 – 8
- Check Record for Update, 8 – 11
- Check Record Uniqueness, 8 – 10
- CLEAR_EOL, 8 – 13
- CLEAR_FORM, 8 – 14
- CLEAR_ITEM, 8 – 15
- CLEAR_RECORD, 8 – 16
- Close the Query, 8 – 17
- COMMIT_FORM, 8 – 18
- COPY, 8 – 19
- COUNT_QUERY, 8 – 20
- CREATE_RECORD, 8 – 21
- DEFAULT_VALUE, 8 – 22
- DELETE_RECORD, 8 – 23
- DO_KEY, 8 – 25
- DOWN, 8 – 26
- DUPLICATE_ITEM, 8 – 27
- DUPLICATE_RECORD, 8 – 28
- ENTER, 8 – 29
- Enter the Block, 8 – 31
- Enter the Form, 8 – 32
- Enter the Item, 8 – 33
- Enter the Record, 8 – 35
- Enter the Value into an Item, 8 – 36
- ENTER_QUERY, 8 – 34
- EXECUTE_QUERY, 8 – 37
- EXECUTE_TRIGGER, 8 – 39
- EXIT_FORM, 8 – 40
- Fetch Records, 8 – 41
- FIRST_RECORD, 8 – 44
- Generate Sequence Number, 8 – 45
- GO_BLOCK, 8 – 46
- GO_ITEM, 8 – 47
- GO_RECORD, 8 – 48
- HOST, 8 – 49
- LAST_RECORD, 8 – 50
- Leave the Block, 8 – 51
- Leave the Form, 8 – 52
- Leave the Item, 8 – 53
- Leave the Record, 8 – 54
- Leave Unit Error Processing, 8 – 55
- Lock the Row, 8 – 58
- LOCK_RECORD, 8 – 57
- LOGON, 8 – 60
- LOGOUT, 8 – 62
- Mark Items and Records as Changed, 8 – 63
- Master–Detail Coordination, 8 – 64
- Navigate to the <Navigation Unit> Level, 8 – 66
- NEW_FORM, 8 – 67
- NEXT_BLOCK, 8 – 68
- NEXT_ITEM, 8 – 69
- NEXT_KEY, 8 – 71
- NEXT_RECORD, 8 – 73
- NEXT_SET, 8 – 74
FORM_SUCCESS, 3 – 104
Format Mask, 5 – 88
FORMS_DDL, 3 – 105
FORMS_MDI_WINDOW constant, 3 – 286
FORMS_OLE.ACTIVATE_SERVER, 3 – 109
FORMS_OLE.CLOSE_SERVER, 3 – 110
FORMS_OLE.EXEC_VERB, 3 – 111
FORMS_OLE.FIND_OLE_VERB, 3 – 112
FORMS_OLE.GET_VERB_COUNT, 3 – 114
FORMS_OLE.GET_VERB_NAME, 3 – 115
FORMS_OLE.INITIALIZE_CONTAINER, 3 – 116
FORMS_OLE.INTERFACE_POINTER, 3 – 113
FORMS_OLE.SERVER_ACTIVE, 3 – 117
Full-screen (Menu Style property), 5 – 130
Function keys
  Runform keys, 6 – 16
  static function keys, 2 – 7
typographic conventions, iii, iii
Functions and packaged procedures, A – 17

G
Generate Before Run option, 1 – 34
Generate component, configuring, 1 – 4
Generate on Upgrade option, A – 5
Generate options
  Add_Triggers, 1 – 23
  Batch, 1 – 23
  CRT_File, 1 – 23
  Debug, 1 – 24
  Delete, 1 – 24
  Extract, 1 – 24
  Generate_on_Upgrade, 1 – 25
  Help, 1 – 25
  Insert, 1 – 25
  Logon, 1 – 26
  Module_Access, 1 – 26
  Module_Type, 1 – 26
  Nofail, 1 – 27
  Options_Screen, 1 – 27
  Output_File, 1 – 27
  Parse, 1 – 28
  Script, 1 – 28
  Statistics, 1 – 29
  Upgrade, 1 – 29
  Upgrade_Roles, 1 – 30
  Version, 1 – 30
  Widen_Fields, 1 – 31
Generate Sequence Number, 8 – 45
Generate_on_Upgrade option, 1 – 25
GENERATE_SEQUENCE_NUMBER, 3 – 118
GET_APPLICATION_PROPERTY, 3 – 119
GET_BLOCK_PROPERTY, 3 – 123
GET_CANVAS_PROPERTY, 3 – 128
GET_FORM_PROPERTY, 3 – 129
GET_GROUP_CHAR_CELL, 3 – 133
GET_GROUP_DATE_CELL, 3 – 135
GET_GROUP_NUMBER_CELL, 3 – 136
GET_GROUP_RECORD_NUMBER, 3 – 138
GET_GROUP_ROW_COUNT, 3 – 139
GET_GROUP_SELECTION, 3 – 140
GET_GROUP_SELECTION_COUNT, 3 – 142
GET_ITEM_PROPERTY, 3 – 143
GET_LIST_ELEMENT_COUNT, 3 – 151
GET_LIST_ELEMENT_LABEL, 3 – 153
GET_LIST_ELEMENT_VALUE, 3 – 154
GET_LOV_PROPERTY, 3 – 155
GET_MENU_ITEM_PROPERTY, 3 – 156
GET_MESSAGE, 3 – 158
GET_PARAMETER_ATTR, 3 – 159
GET_PARAMETER_LIST, 3 – 160
GET_RADIO_BUTTON_PROPERTY, 3 – 160
GET_RECORD_PROPERTY, 3 – 163
GET_RELATION_PROPERTY, 3 – 165
GET_VIEW_PROPERTY, 3 – 167
GET_WINDOW_PROPERTY, 3 – 170
Global variables, constraints, D – 2
GO_BLOCK, 3 – 171, 8 – 46
GO_FORM, 3 – 172
GO_ITEM, 3 – 173, 8 – 47
GO_RECORD, 3 – 174, 8 – 48
Grayed-out items, Disabled, 5 – 73
Group_Name, 5 – 94
GUI
conversion, A – 39
functionality, A – 44
Generate dialog, A – 5
GUI hint
Closeable, 5 – 38
Fixed Size, 5 – 87
Iconifiable, 5 – 101
Inherit Menu, 5 – 102
Moveable, 5 – 135

H
Handles (Object IDs), 3 – 4
HELP, HELP built–in routine, 3 – 175
Help
Designer option, 1 – 35
Generate option, 1 – 25
Help property (help text for item), 5 – 95
Runform option, 1 – 13
HIDE_MENU, 3 – 175
HIDE_VIEW, 3 – 176
HIDE_WINDOW, 3 – 176
Hint (Item), 5 – 95
Hint (Menu Item), 5 – 96
Hint (Menu Substitution Parameter), 5 – 96
Horizontal MDI Toolbar, 5 – 97
Horizontal Scroll Bar, 5 – 97
HorizontalToolbar property, 5 – 98
HOST, 3 – 178, 8 – 49

I
IAPCAL command, A – 22
Icon Name, 5 – 99
Icon Title, 5 – 100
Iconic, 5 – 100
Iconifiable, 5 – 101
ID_NULL, 3 – 180
Identification, 5 – 101
IFZCAL command, A – 22
IMAGE_ZOOM, 3 – 181
In Menu/Block Description property, 5 – 102
Inherit Menu, 5 – 102
Initial Keyboard State, 5 – 103
INP format, A – 10
Insert Allowed (Block), 5 – 103
Insert Allowed (Item), 5 – 104
Insert option, 1 – 25
INSERT_RECORD, 3 – 183
INT data type, 5 – 57
Interactive option, 1 – 13
ISSUE_ROLLBACK, 3 – 184
ISSUE_SAVEPOINT, 3 – 185
Item built–ins
CHECKBOX_CHECKED, 3 – 31
CLEAR_EOL, 3 – 35
CLEAR_ITEM, 3 – 37
CONVERT_OTHER_VALUE, 3 – 43
COPY, 3 – 44
CUT_REGION, 3 – 56
DEFAULT_VALUE, 3 – 60
DISPLAY_ITEM, 3 – 70
DUPLICATE_ITEM, 3 – 73
EDIT_TEXTITEM, 3 – 75
FIND_ITEM, 3 – 94
GET_ITEMPROPERTY, 3 – 143
GET_RADIO_BUTTON_PROPERTY, 3 – 160
GO_ITEM, 3 – 173
ID_NULL, 3 – 180
IMAGE_ZOOM, 3 – 181
NAME_IN, 3 – 202
NEXT_ITEM, 3 – 210
PASTE_REGION, 3 – 215
PREVIOUS_ITEM, 3 – 222
READ_IMAGE_FILE, 3 – 227
SELECT_ALL, 3 – 241
SET_ITEMPROPERTY, 3 – 260
SET_RADIO_BUTTON_PROPERTY, 3 – 274
WRITE_IMAGE_FILE, 3 – 306
Item Properties
Alignment, 5 – 23
Auto Hint, 5 – 24
Auto–Skip, 5 – 28
Base Table (Item), 5 – 29
Bevel, 5 – 30
Case Insensitive Query, 5 – 34
Case Restriction, 5 – 35
Check Box Other Values, 5 – 36
Checked Value, 5 – 37
Compression, 5 – 44
Copy Value from Item, 5 – 49
Data Type, 5 – 55
Database_Value, 5 – 60
Default Button, 5 – 62
Default Value, 5 – 63
Direction, 5 – 67
Displayed, 5 – 74
Editor, 5 – 76
Editor X Position, 5 – 77
Editor Y Position, 5 – 77
Enabled (Item), 5 – 77
Fixed Length, 5 – 86
Format Mask, 5 – 88
Hint, 5 – 95
Horizontal Scroll Bar, 5 – 97
Icon Name, 5 – 99
Iconic, 5 – 100
Initial Keyboard State, 5 – 103
Insert Allowed, 5 – 104
Item Type, 5 – 105
Item_Is_Valid, 5 – 106
Items Displayed, 5 – 106
Keep Position, 5 – 108
Label, 5 – 110
List Element, 5 – 112
List Style, 5 – 113
Lock Record, 5 – 113
LOV for Validation, 5 – 116
LOV Position, 5 – 118
Maximum Length, 5 – 123
Mirror Item, 5 – 131
Mouse Navigate, 5 – 134
Multi–line, 5 – 136
Navigable, 5 – 138
NextItem, 5 – 142
Other Values, 5 – 154
Previous Navigation Item, 5 – 157
PreviousItem, 5 – 158
Primary Key (Item), 5 – 159
Quality, 5 – 160
Query Allowed (Item), 5 – 161
Query Length, 5 – 162
Query Only, 5 – 162
Range High Value/Range Low Value, 5 – 165
Reading Order, 5 – 166
Required (Item), 5 – 174
Secure, 5 – 179
Size, 5 – 182
Sizing Style, 5 – 184
table of Item Properties, 5 – 9
Unchecked Value, 5 – 192
Update Allowed, 5 – 194
Update Only if NULL, 5 – 197
Update_Column, 5 – 196
Update_Permission, 5 – 198
Value, 5 – 205
Vertical Scroll Bar, 5 – 208
Visual Attribute Name, 5 – 213
Wrap Style, 5 – 222
X Position, 5 – 222
Y Position, 5 – 222
Item properties, validation of changes with
SET_ITEM, 3 – 267
Item Type, 5 – 105
ITEM_ENABLED, 3 – 186
Item_Is_Valid, 5 – 106
Items, 3 – 260
dynamic changes (SET_ITEM), 3 – 260
validate settings (GET_ITEM), 3 – 143
Items Displayed, 5 – 106

J
Join Condition, 5 – 107
Join Condition, constraints, D – 2

K
Keep Position, 5 – 108
Key fields, displaying system–generated, 5 – 104
Key Mode, 5 – 109
Key triggers, A – 45
Key–F0 through Key–F9, 2 – 8
Key–Fn trigger, 2 – 8
Key–Others trigger, 2 – 9
Key–Startup trigger, 2 – 80
Keyin option, 1 – 13
Keyout option, 1 – 14
L
Label (Item), 5 – 110
Label (Menu Item), 5 – 110
Label (Menu Parameter), 5 – 111
Last.Block, 5 – 111
Last.Item, 5 – 112
LAST.RECORD, 3 – 186, 8 – 50
Leave the Block, 8 – 51
Leave the Form, 8 – 52
Leave the Item, 8 – 53
Leave the Record, 8 – 54
Leave Unit Error Processing, 8 – 55
LIB format, A – 21
Library storage formats
  .PLD format, B – 9
  .PLL format, B – 8
  .PLL format stripped of source code, B – 9
  database format, B – 8
List Element, 5 – 112
List Style, 5 – 113
LIST.VALUES, 3 – 187
Lock Record, 5 – 113
Lock the Row, 8 – 58
LOCK.RECORD, 3 – 188, 8 – 57
Locking, 7 – 16
  When locks are released, 7 – 18
Locking Mode, 5 – 114
Logical attributes, 5 – 214
Login to database, 1 – 8
LOGON, 3 – 189, 8 – 60
Logon option, 1 – 26
LOGON_SCREEN, 3 – 190
Logon Screen option, 1 – 14
LOGOUT, 3 – 192, 8 – 62
LONG data type, 5 – 58
Long List, 5 – 115
LOV, 5 – 116
LOV for Validation, 5 – 116
LOV Position, 5 – 118
LOV Properties
  Auto–Confirm, 5 – 25
  Auto–Display, 5 – 25
  Auto–Refresh, 5 – 26
  Auto–Skip, 5 – 28
  Group_Name, 5 – 94
  Long List, 5 – 115
  LOV, 5 – 116
  LOV Type, 5 – 119
  Record Group, 5 – 167
  Size, 5 – 182
  table of LOV Properties, 5 – 12
  X Position, 5 – 222
  Y Position, 5 – 222
  LOV Type, 5 – 119

M
Macros, A – 17
Magic Item, 5 – 120
Main Menu, 5 – 121
MAIN_MENU, 3 – 193
Mark Items and Records as Changed, 8 – 63
Master Deletes, 5 – 122
Master–detail (Join Condition), 5 – 107
Master–Detail Coordination, 8 – 64
Maximum Length, 5 – 123
Maximum Length (Form Parameter), 5 – 124
Maximum Length (Menu Substitution Parameter), 5 – 124
Maximum length constraint, D – 2
MDI, 5 – 45
  FORMS_MDI_WINDOW constant, 3 – 286
  Widow Style property, 5 – 221
Menu built–ins
  APPLICATION_PARAMETER, 3 – 24
  BACKGROUND_MENU, 3 – 25
  FIND_MENU_ITEM, 3 – 96
  GET_MENU_ITEM_PROPERTY, 3 – 156
  HIDE_MENU, 3 – 175
  ITEM_ENABLED, 3 – 186
  MAIN_MENU, 3 – 193
  MENU_CLEAR_FIELD, 3 – 193
  MENU_NEXT_FIELD, 3 – 194
  MENU_PARAMETER, 3 – 194
  MENU_PREVIOUS_FIELD, 3 – 195
Menu Item Radio Group, 5 – 125
Menu Item Roles, 5 – 125
Menu Item Type, 5 – 126
Menu Items
  check, 5 – 126
  constraints, D – 2
  inaccessible, 5 – 73
  magic, 5 – 126
  plain, 5 – 126
  radio, 5 – 126
  separator, 5 – 127
Menu Module, 5 – 128
Menu Module Roles, 5 – 129
Menu Nesting Levels, Constraints, D – 2
Menu Properties
  Accelerator property, 5 – 21
  Bottom Title, 5 – 31
  Checked, 5 – 37
  Command Text, 5 – 42
  Command Type, 5 – 43
  Default Value, 5 – 65
  Display w/o Privilege, 5 – 73
  Displayed, 5 – 75
  Enabled, 5 – 78
  File, 5 – 80
  File_Name, 5 – 81
  Fixed Length, 5 – 86
  Help, 5 – 95
  Hint (Menu Item), 5 – 96
  Hint (Menu Substitution Parameter), 5 – 96
  Icon Name, 5 – 99
  Identification, 5 – 101
  Label (Menu Item), 5 – 110
  Label (Menu Parameter), 5 – 111
  Magic Item, 5 – 120
  Main Menu, 5 – 121
  Maximum Length (Menu Substitution Parameter), 5 – 124
  Menu Item Radio Group, 5 – 125
  Menu Item Type, 5 – 126
  Menu Module, 5 – 128
  Menu Role, 5 – 129
  Menu Style, 5 – 130
  Parameter Menus, 5 – 154
  Required (Menu Substitution Parameter), 5 – 175
  Secure, 5 – 179
  Startup Code, 5 – 186
  Subtitle, 5 – 187
  Table of Menu Properties, 5 – 13
  Tear-off, 5 – 188
  Use Security, 5 – 201
Menu Role, 5 – 129
Menu Scroll Region (Command Text), 5 – 42
Menu Security Group (Menu Role), 5 – 129
Menu Storage Formats
  .MMB Format, B – 6
  .MMT Format, B – 7
  .MMX Format, B – 7
  Database Format, B – 6
Menu Style, 5 – 130
MENU_CLEAR_FIELD, 3 – 193
MENU_NEXT_FIELD, 3 – 194
MENU_PARAMETER, 3 – 194
MENU_PREVIOUS_FIELD, 3 – 195
MENU_REDISPLAY, 3 – 195
MENU_SHOW_KEYS, 3 – 195
NEXT_MENU_ITEM, 3 – 211
PREVIOUS_MENU, 3 – 223
QUERY_PARAMETER, 3 – 225
SET_INPUT_FOCUS, 3 – 260
SET_MENU_ITEM_PROPERTY, 3 – 272
SHOW_BACKGROUND_MENU, 3 – 287
TERMINATE, 3 – 295
WHERE_DISPLAY, 3 – 306
Message Built-ins
  CLEAR_MESSAGE, 3 – 39
  DISPLAY_ERROR, 3 – 70
  ERROR_CODE, 3 – 79
  ERROR_TEXT, 3 – 80
  GET_MESSAGE, 3 – 158
  MESSAGE_CODE, 3 – 197
  MESSAGE_TEXT, 3 – 198
Message Built-ins
  CLEAR_MESSAGE, 3 – 39
  DISPLAY_ERROR, 3 – 70
  ERROR_CODE, 3 – 79
  ERROR_TEXT, 3 – 80
  GET_MESSAGE, 3 – 158
  MESSAGE_CODE, 3 – 197
  MESSAGE_TEXT, 3 – 198
MESSAGE_TYPE, 3 – 199
Message property, 5 – 130
MESSAGE_CODE, 3 – 197
MESSAGE_TEXT, 3 – 198
MESSAGE_TYPE, 3 – 199
Messages, typographic conventions, iii, iii
Migration strategies, A – 39
Mirror Item, 5 – 131
Miscellaneous object built-ins
CREATE_TIMER, 3 – 55
DELETE_TIMER, 3 – 67
FIND_EDITOR, 3 – 92
FIND_LOV, 3 – 95
FIND_TIMER, 3 – 98
ID_NULL, 3 – 180
LIST_VALUES, 3 – 187
SET_LOV_COLUMN_PROPERTY, 3 – 270
SET_LOV_PROPERTY, 3 – 271
SET_TIMER, 3 – 280
SHOW_EDITOR, 3 – 288
SHOW_LOV, 3 – 290
VALIDATE, 3 – 299
.MMB format[MMB], B – 6
MMB, MMT, MMX file formats, A – 9
.MMT format[MMT], B – 7
.MMX format[MMX], B – 7
Modal, 5 – 132
Module Access, Designer option, 1 – 35
Module_Access, Generate option, 1 – 26
Module_NLS_Lang, 5 – 133
Module_Type
Designer option, 1 – 36
Generate option, 1 – 26
Mouse Navigate, 5 – 134
Mouse Navigation Limit, 5 – 135
MOVE_WINDOW, 3 – 200
Moveable, 5 – 135
Multi-byte characters, 5 – 123
Multi-line, 5 – 136

NAME_IN, 3 – 202
Named Visual Attribute Properties
  Background Color, 5 – 214
  Character Mode Logical Attribute, 5 – 214
  Font, 5 – 213
  Foreground Color, 5 – 214
  Pattern, 5 – 214
  Size, 5 – 213
  Style, 5 – 214
  Weight, 5 – 214
  White on Black, 5 – 214
  Width, 5 – 214
Naming conflicts, A – 8
Navigable, 5 – 138
Navigate to the <Navigation Unit> Level, 8 – 66
Navigation (Mouse Navigation Limit), 5 – 135
Navigation Style, 5 – 139
NEW_FORM, 3 – 205, 8 – 67
Next Navigation Block, 5 – 140
Next Navigation Item, 5 – 140
NEXT_BLOCK, 3 – 208, 8 – 68
Next_Detail_Relation, 5 – 142
NEXT_FORM, 3 – 209
NEXT_ITEM, 3 – 210, 8 – 69
NEXT_KEY, 3 – 211, 8 – 71
Next_Master_Relation, 5 – 142
NEXT_MENU_ITEM, 3 – 212
NEXT_RECORD, 3 – 212, 8 – 73
NEXT_SET, 3 – 213, 8 – 74
NextBlock, 5 – 141
NextItem, 5 – 142
Nofail option, 1 – 27
Non–Isolated, 5 – 122
Non–ORACLE data sources, On–Check–Unique trigger, 2 – 11
NUMBER data type, 5 – 58
Numbers, formatting, 5 – 88

N
Name, 5 – 136

O
Object ID, 3 – 4
Objects, constraints on, D – 2
OLE Activation Style, 5 – 143
OLE Class, 5 – 144
OLE Do In Out, 5 – 145
OLE In–place Activation, 5 – 146
OLE Popup Menu Items, 5 – 147
OLE Resize Style, 5 – 150
OLE Tenant Aspect, 5 – 150
OLE Tenant Types, 5 – 151
On–Check–Delete–Master trigger, 2 – 10
On–Check–Unique trigger, 2 – 11
On–Clear–Block trigger, 2 – 65
On–Clear–Details trigger, 2 – 12
On–Close trigger, 2 – 13
On–Column–Security trigger, 2 – 14
On–Commit trigger, 2 – 15
On–Count trigger, 2 – 16
On–Database–Record trigger, 2 – 68
On–Delete trigger, 2 – 17
On–Error trigger, 2 – 18
On–Fetch trigger, 2 – 19
On–Insert trigger, 2 – 21
On–Lock trigger, 2 – 22
On–Logon trigger, 2 – 23
On–Logout trigger, 2 – 24
On–Message trigger, 2 – 25
On–New–Field–Instance trigger, 2 – 81
On–New–Record trigger, 2 – 66
On–Populate–Details trigger, 2 – 26
On–Remove–Record trigger, 2 – 84
On–Rollback trigger, 2 – 27
On–Savepoint trigger, 2 – 28
On–Select trigger, 2 – 29
On–Sequence–Number trigger, 2 – 30
On–Update trigger, 2 – 31
On–Validate–Field trigger, 2 – 87
On–Validate–Record trigger, 2 – 89
Open the Query, 8 – 75
OPEN_FORM, 3 – 213
Operating_System, 5 – 152
Optimizer_Hint, 5 – 152
OptimizeSQL option, 1 – 15
OptimizeTP option, 1 – 16
Options
    Designer options, 1 – 31
    Generate options, 1 – 21
    Runform options, 1 – 9
    User Preference File, 1 – 39
Options_Screen
    Generate option, 1 – 27
    Runform option, 1 – 17
Oracle Terminal Resource File option, 1 – 20
Order By Clause property, 5 – 217
OS_COMMAND, A – 38
OS_COMMAND1, A – 38
Other Values, 5 – 154
Output file extensions, A – 9
Output_File
    Generate option, 1 – 27
    Runform option, 1 – 17
P
Packaged procedures and functions
    Oracle Forms, A – 36
    SQL*Menu V.5, A – 38
    V.4.5, A – 17
Page 0 field (V.3 term), A – 24
Pages, A – 20, A – 24
Palette option, 1 – 34
Parameter list built–ins
    ADD_PARAMETER, 3 – 23
    CREATE_PARAMETER_LIST, 3 – 51
    DELETE_PARAMETER, 3 – 65
    DESTROY_PARAMETER_LIST, 3 – 69
    GET_PARAMETER_ATTR, 3 – 159
    GET_PARAMETER_LIST, 3 – 160
    ID_NULL, 3 – 180
    RUN_PRODUCT, 3 – 235
    SET_PARAMETER_ATTR, 3 – 274
Parameter list, creating programmatically, 3 – 51
Parse option, 1 – 28, A – 10
Passing parameters with a parameter list
    CALL_FORM, 3 – 28
CREATE_PARAMETER_LIST, 3 – 51
NEW_FORM, 3 – 205
RUN_PRODUCT, 3 – 235
Password, 1 – 5, 1 – 8, 5 – 155
PASTE_REGION, 3 – 215
Pattern, 5 – 214
PAUSE, 3 – 215
PECS option, 1 – 18
PL/SQL libraries, A – 21
PLA format, A – 21
.PLD format[PLD], B – 9
.PLL format stripped of source code, B – 9
.PLL format[PLL], B – 8
PLL, PLD file formats, A – 9, A – 21
Pop-up pages
- converting, A – 48
- V.3 term, A – 24
POPULATE_GROUP, 3 – 216
POPULATE_GROUP_WITH_QUERY, 3 – 217
POPULATE_LIST, 3 – 218
Portability among systems, B – 10
POST, 3 – 220, 8 – 76
Post and Commit Transactions, 8 – 77
Post–Block trigger, 2 – 32
Post–Change trigger, 2 – 33
Post–Commit trigger, 2 – 35, 2 – 38
Post–Database–Commit trigger, 2 – 35
Post–Delete trigger, 2 – 36
Post–Field trigger (Post–Text–Item trigger), 2 – 46
Post–Form trigger, 2 – 37
Post–Forms–Commit trigger, 2 – 35, 2 – 38
Post–Insert trigger, 2 – 39
Post–Logon trigger, 2 – 40
Post–Logout trigger, 2 – 41
Post–Query trigger, 2 – 42
Post–Record trigger, 2 – 44
Post–Select trigger, 2 – 45
Post–Text–Item trigger, 2 – 46
Post–Update trigger, 2 – 47
Pre–Block trigger, 2 – 48
Pre–Commit trigger, 2 – 49
Pre–Delete trigger, 2 – 50
Pre–Field trigger (Pre–Text–Item trigger), 2 – 59
Pre–Form trigger, 2 – 51
Pre–Insert trigger, 2 – 52
Pre–Logon trigger, 2 – 53
Pre–Logout trigger, 2 – 54
Pre–Query trigger, 2 – 55
Pre–Record trigger, 2 – 57
Pre–Select trigger, 2 – 58
Pre–Text–Item trigger, 2 – 59
Pre–Update trigger, 2 – 60
Preferences
- Designer options, 1 – 31
- Generate options, 1 – 21
- Runform options, 1 – 9
- User Preference File, 1 – 39
Prepare the Query, 8 – 86
Prevent Masterless Operations, 5 – 155
Previous Navigation Block, 5 – 156
Previous Navigation Item, 5 – 157
PREVIOUS_BLOCK, 3 – 220, 8 – 87
PREVIOUS_FORM, 3 – 221
PREVIOUS_ITEM, 3 – 222, 8 – 88
PREVIOUS_MENU, 3 – 223
PREVIOUS_MENU_ITEM, 3 – 223
PREVIOUS_RECORD, 3 – 224, 8 – 89
PreviousBlock, 5 – 158
PreviousItem, 5 – 158
Primary Key (Block), 5 – 159
Primary Key (Item), 5 – 159
Primary key, checking programmatically, 2 – 11
PRINT, 3 – 224
Printer option, 1 – 36
Privileges, display menu items without, 5 – 73
Process Expired Timer, 8 – 90
Process the Function Key, 8 – 91
Processing
- Locking, 7 – 16
- Query processing, 7 – 2
- validation, 7 – 6
Prompt and Answer, 8 – 92
Prompts (V.3 term), A – 24
Properties
  Accelerator, 5 – 21
  Access Key, 5 – 21
  Alert Style, 5 – 22
  Alignment, 5 – 23
  Application Instance, 5 – 24
  Auto Hint, 5 – 24
  Auto–Confirm, 5 – 25
  Auto–Display, 5 – 25
  Auto–Query, 5 – 25
  Auto–Refresh, 5 – 26
  Auto–Skip (item), 5 – 28
  Auto–Skip (LOV), 5 – 28
  Base Table (Block), 5 – 29
  Base Table (Item), 5 – 29
  Bevel, 5 – 30
  Bottom Title, 5 – 31
  Button 1, Button2, Button 3, 5 – 31
  Calling_Form, 5 – 32
  Canvas, 5 – 32
  Canvas–view Type, 5 – 33
  Case Insensitive Query, 5 – 34
  Case Restriction, 5 – 35
  changing item properties, 3 – 268
  Character Cell WD/HT, 5 – 36
  Check Box Other Values, 5 – 36
  Checked, 5 – 37
  Checked Value, 5 – 37
  Class, 5 – 38
  Closeable, 5 – 38
  Column Mapping, 5 – 39
  Column Security, 5 – 40
  Column Specification, 5 – 41
  Command Text, 5 – 42
  Command Type, 5 – 43
  Compression, 5 – 44
  Connect_String, 5 – 44
  Console Window, 5 – 45
  Coordinate Information, 5 – 45
  Coordinate System, 5 – 45
  Coordination, 5 – 47
  Coordination_Status, 5 – 48
  Copy Value from Item, 5 – 49
  Current Record Attribute, 5 – 49
  Current_Form, 5 – 50
  Current_Form_Name, 5 – 51
  Current_Record, 5 – 52
  Cursor Mode, 5 – 52
  Cursor Style, 5 – 54
  Data Type, 5 – 55
  Database_Value, 5 – 60
  Datasource, 5 – 61
  Default Alert Button, 5 – 61
  Default Button, 5 – 62
  Default Font Scaling, 5 – 62
  Default Value (Form Parameter), 5 – 63
  Default Value (Item), 5 – 63
  Default Value (Menu Substitution Parameter), 5 – 65
  Defer_Required_Enforcement, 5 – 65
  Deferred, 5 – 65
  Delete Allowed, 5 – 66
  Displayed (Canvas–view), 5 – 66
  Displayed (Item), 5 – 66
  Displayed (Menu Item), 5 – 66
  Direction, 5 – 67
  Directory, 5 – 72
  Display w/o Privilege, 5 – 73
  Display X Position, 5 – 73
  Display Y Position, 5 – 73
  Displayed (Canvas–view), 5 – 74
  Displayed (Item), 5 – 74
  Displayed (Menu Item), 5 – 75
  Editor, 5 – 76
  Editor X Position, 5 – 77
  Editor Y Position, 5 – 77
  Enabled (Item), 5 – 77
  Enabled (Menu Item), 5 – 78
  Enterable, 5 – 79
  File, 5 – 80
  File_Name, 5 – 81
  First Navigation Block, 5 – 82
  Fire in Enter Query Mode, 5 – 82
  First_Block, 5 – 83
  First_Detail_Relation, 5 – 84
  First_Item, 5 – 84
  Fixed Length (Item), 5 – 86
  Fixed Length (Menu Substitution Parameter), 5 – 86
  Fixed Size, 5 – 87
  Form_Name, 5 – 94
  Format Mask, 5 – 88
  Group_Name, 5 – 94
  Help, 5 – 95
Hint (Item), 5 – 95
Hint (Menu Item), 5 – 96
Hint (Menu Substitution Parameter), 5 – 96
Horizontal MDI Toolbar, 5 – 97
Horizontal Scroll Bar, 5 – 97
Horizontal Toolbar, 5 – 98
Icon Name, 5 – 99
Icon Title, 5 – 100
Iconic, 5 – 100
Iconifiable, 5 – 101
Identification, 5 – 101
In Menu/Block Description, 5 – 102
Inherit Menu, 5 – 102
Initial Keyboard State, 5 – 103
Insert Allowed (Block), 5 – 103
Insert Allowed (Item), 5 – 104
Item Type, 5 – 105
Item_Is_Valid, 5 – 106
Items Displayed, 5 – 106
Join Condition, 5 – 107
Keep Position, 5 – 108
Key Mode, 5 – 109
Label (Item), 5 – 110
Label (Menu Item), 5 – 110
Label (Menu Parameter), 5 – 111
Last_Block, 5 – 111
Last_Item, 5 – 112
length constraint, D – 2
List Element, 5 – 112
List Style, 5 – 113
Lock Record, 5 – 113
Locking Mode, 5 – 114
Long List, 5 – 115
LOV, 5 – 116
LOV for Validation, 5 – 116
LOV Position, 5 – 118
LOV Type, 5 – 119
Magic Item, 5 – 120
Main Menu, 5 – 121
Master Deletes, 5 – 122
Maximum Length, 5 – 123
Maximum Length (Form Parameter), 5 – 124
Maximum Length (Menu Substitution Parameter), 5 – 124
Menu Item Radio Group, 5 – 125
Menu Item Roles, 5 – 125
Menu Item Type, 5 – 126
Menu Module, 5 – 128
Menu Module Roles, 5 – 129
Menu Role, 5 – 129
Menu Style, 5 – 130
Message, 5 – 130
Mirror Item, 5 – 131
Modal, 5 – 132
Module-NLS_Lang, 5 – 133
Mouse Navigate, 5 – 134
Mouse Navigation Limit, 5 – 135
Moveable, 5 – 135
Multi-line, 5 – 136
Navigable, 5 – 138
Navigation Style, 5 – 139
Next Navigation Block, 5 – 140
Next Navigation Item, 5 – 140
Next_Detail_Relation, 5 – 142
Next_Master_Relation, 5 – 142
NextBlock, 5 – 141
NextItem, 5 – 142
OLE Activation Style, 5 – 143
OLE Class, 5 – 144
OLE Do In Out, 5 – 145
OLE In-place Activation, 5 – 146
OLE Popup Menu Items, 5 – 147
OLE Resize Style, 5 – 150
OLE Tenant Aspect, 5 – 150
OLE Tenant Types, 5 – 151
Operating_System, 5 – 152
Optimizer_Hint, 5 – 152
Order By Clause, 5 – 217
Other Values, 5 – 154
Overview, 5 – 2
Parameter Menus, 5 – 154
Password, 5 – 155
Prevent Masterless Operations, 5 – 155
Previous Navigation Block, 5 – 156
Previous Navigation Item, 5 – 157
PreviousBlock, 5 – 158
PreviousItem, 5 – 158
Primary Key (Block), 5 – 159
Primary Key (Item), 5 – 159
Quality, 5 – 160
Query Allowed (Block), 5 – 160
Query Allowed (Item), 5 – 161
Query Length, 5 – 162
Query Only, 5 – 162
Query_Hits, 5 – 163
Query_Options, 5 – 164
Raise on Entry, 5 – 164
Range High Value/Range Low Value, 5 – 165
SELECT RECORDS, 3 – 242
SET RECORD_PROPERTY, 3 – 276
UP, 3 – 296
UPDATE RECORD, 3 – 296
Record Group, 5 – 167
Record group built–ins
ADD_GROUP_COLUMN, 3 – 18
ADD_GROUP_ROW, 3 – 20
CREATE_GROUP, 3 – 47
CREATE_GROUP_FROM_QUERY, 3 – 49
DELETE_GROUP, 3 – 61
DELETE_GROUP_ROW, 3 – 62
ENFORCE_COLUMN_SECURITY, 3 – 76
FIND_COLUMN, 3 – 91
FIND_GROUP, 3 – 94
GET_GROUP_CHAR_CELL, 3 – 133
GET_GROUP_DATE_CELL, 3 – 135
GET_GROUP_NUMBER_CELL, 3 – 136
GET_GROUP_RECORD_NUMBER, 3 – 138
GET_GROUP_ROW_COUNT, 3 – 139
GET_GROUP_SELECTION, 3 – 140
GET_GROUP_SELECTION_COUNT, 3 – 142
ID_NULL, 3 – 180
POPULATE_GROUP, 3 – 216
POPULATE_GROUP_WITH_QUERY, 3 – 217
RESET_GROUP_SELECTION, 3 – 232
SET_GROUP_CHAR_CELL, 3 – 255
SET_GROUP_DATE_CELL, 3 – 256
SET_GROUP_NUMBER_CELL, 3 – 258
SET_GROUP_SELECTION, 3 – 259
UNSET_GROUP_SELECTION, 3 – 295
Record Group Properties
Other Values, 5 – 154
Record Group Type, 5 – 168

table of Record Group Properties, 5 – 14
Record Group Query, 5 – 168
Record Group Type, 5 – 168
Record groups, constraints, D – 2
Record Orientation, 5 – 169
Record status (Status property), 5 – 187
Records, Records Fetched, 5 – 171
Records Buffered, 5 – 169
Records Displayed, 5 – 170

NEXT_KEY, 3 – 211
NEXT_SET, 3 – 213
Query Length, 5 – 162
Query Only, 5 – 162
Query Only Mode option, 1 – 19
Query_Hits, 5 – 163
Query_Options, 5 – 164
QUERY_PARAMETER, 3 – 225
Queryable (Query Allowed property), 5 – 161
Quiet Mode option, 1 – 19

R
Radio button
Label property, 5 – 110
Position property, 5 – 222
Value property, 5 – 205
Visual_Attribute property, 5 – 216
Radio Group
Label property, 5 – 110
Menu Item Radio Group property, 5 – 125
Raise on Entry, 5 – 164
Range High Value/Range Low Value, 5 – 165
Read Input Keystrokes from File option, 1 – 13
READ_IMAGE_FILE, 3 – 227
Reading Order, 5 – 166
Real Unit, 5 – 167
Record built–ins
CHECK_RECORD_UNIQUENESS, 3 – 33
CLEAR_RECORD, 3 – 40
CREATE_QUERIED_RECORD, 3 – 52
CREATE_RECORD, 3 – 54
DELETE_RECORD, 3 – 66
DOWN, 3 – 71
DUPLICATE_RECORD, 3 – 74
FIRST_RECORD, 3 – 101
GENERATE_SEQUENCE_NUMBER, 3 – 118
GET_RECORD_PROPERTY, 3 – 163
GO_RECORD, 3 – 174
INSERT_RECORD, 3 – 183
LAST_RECORD, 3 – 186
LOCK_RECORD, 3 – 188
NEXT_RECORD, 3 – 212
NEXT_SET, 3 – 213
PREVIOUS_RECORD, 3 – 224
SCROLL_DOWN, 3 – 238
SCROLL_UP, 3 – 238
SET_RECORDS, 3 – 242
UP, 3 – 296
UPDATE_RECORD, 3 – 296
Records Fetched, 5 – 171
Records, checking for uniqueness, 3 – 33
Records_to_Fetch, 5 – 171
REDISPLAY, 3 – 228
Reference Information, 5 – 173
Referenced objects, A – 6
Relation built–ins
   FIND_RELATION, 3 – 97
   GET_RELATION_PROPERTY, 3 – 165
   ID_NULL, 3 – 180
   SET_RELATION_PROPERTY, 3 – 278
Relation Properties
   Auto–Query, 5 – 25
   Coordination, 5 – 47
   Coordination_Status, 5 – 48
   Deferred property, 5 – 65
   Detail Block, 5 – 66
   Master Deletes, 5 – 122
   Next_Detail_Relation, 5 – 142
   Next_Master_Relation, 5 – 142
   Prevent Masterless Operations, 5 – 155
   table of Relation Properties, 5 – 14
Remove on Exit, 5 – 173
Rendered, 5 – 174
REPLACE_CONTENT_VIEW, 3 – 229
REPLACE_MENU, 3 – 230, 8 – 96
Required (Item), 5 – 174
Required (Menu Substitution Param), 5 – 175
RESET_GROUP_SELECTION, 3 – 232
RESIZE_WINDOW, 3 – 233
Restricted built–in subprograms, 3 – 5
   BLOCK_MENU, 3 – 26
   CALL_INPUT, 3 – 31
   CLEAR_BLOCK, 3 – 34
   CLEAR_EOL, 3 – 35
   CLEAR_FORM, 3 – 36
   CLEAR_ITEM, 3 – 37
   CLEAR_MESSAGE, 3 – 39
   CLEAR_RECORD, 3 – 40
   COMMIT_FORM, 3 – 41
   CONVERT_OTHER_VALUE, 3 – 43
   COPY_REGION, 3 – 45
   COUNT_QUERY, 3 – 46
   CREATE_RECORD, 3 – 54
   CUT_REGION, 3 – 56
   DELETE_RECORD, 3 – 66
   DO_KEY, 3 – 72
   DOWN, 3 – 71
   DUPLICATE_ITEM, 3 – 73
   DUPLICATE_RECORD, 3 – 74
   EDIT_TEXTITEM, 3 – 75
   ENFORCE_COLUMN_SECURITY, 3 – 76
   ENTER, 3 – 77
   ENTER_QUERY, 3 – 77
   EXECUTE_QUERY, 3 – 82
   EXECUTE_TRIGGER, 3 – 84
   EXIT_FORM, 3 – 85
   FIRST_RECORD, 3 – 101
   GO_BLOCK, 3 – 171
   GO_ITEM, 3 – 173
   GO_RECORD, 3 – 174
   HELP, 3 – 175
   INSERT_RECORD, 3 – 183
   LAST_RECORD, 3 – 186
   NEW_FORM, 3 – 205
   NEXT_BLOCK, 3 – 208
   NEXT_ITEM, 3 – 210
   NEXT_KEY, 3 – 211
   NEXT_RECORD, 3 – 212
   NEXT_SET, 3 – 213
   PASTE_REGION, 3 – 215
   POST, 3 – 220
   PREVIOUS_BLOCK, 3 – 220
   PREVIOUS_ITEM, 3 – 222
   SCROLL_DOWN, 3 – 238
   SCROLL_UP, 3 – 238
   SHOW_LOV, 3 – 290
   UP, 3 – 296
   RETRIEVE_LIST, 3 – 234
   Return for Input, 8 – 97
   Rollback Form, 8 – 98
   Rollbacks, 3 – 37
   ROWID, 5 – 109
   Run Asynchronously option, 1 – 37
   Run the Form, 8 – 99
   RUN_PRODUCT, 3 – 235
Runform
   Optimize Transaction Mode Processing, 1 – 16
   starting, 1 – 4
Runform Interface
choosing items and objects, 6 – 3
getting help, 6 – 2
interacting with form items, 6 – 4
modifying the database, 6 – 10
navigating around your form, 6 – 3
querying the database, 6 – 6
starting/exiting Runform, 6 – 2
viewing the status line, 6 – 2
Runform options
Array, 1 – 11
Block_Menu, 1 – 11
Buffer_Records, 1 – 12
Help, 1 – 13
Interactive, 1 – 13
Keyin, 1 – 13
Keyout, 1 – 14
Logon_Screen, 1 – 14
OptimizeSQL, 1 – 15
OptimizeTP, 1 – 16
Options_Screen, 1 – 17
Output_File, 1 – 17
PECS, 1 – 18
Query_Only, 1 – 19
Quiet, 1 – 19
Session, 1 – 19
Statistics, 1 – 20
Term, 1 – 20
Window_State, 1 – 21

S
Save Before Generate option, 1 – 37
Savepoint, 8 – 100
Savepoint Mode, 5 – 176
Savepoint_Name, 5 – 177
Screen painter (V.3 term), A – 24
Screen painter (V.3 term), A – 24
Scroll Bar, 5 – 178
SCROLL_DOWN, 3 – 238, 8 – 101
SCROLL_UP, 3 – 238, 8 – 101
SCROLL_VIEW, 3 – 239
Secure (Item), 5 – 179
Secure (Menu Substitution Parameter), 5 – 179
Security group (Menu Role property), 5 – 129
Security Roles, Display w/out Privilege property, 5 – 73
SELECT_ALL, 3 – 241
SELECT_RECORDS, 3 – 242
Separators
delimeters (decimal, thousands), 5 – 89
menu item type, 5 – 127
NLS, 5 – 93
Session option, 1 – 19
SET_ALERT_BUTTON_PROPERTY, 3 – 243
SET_ALERTPROPERTY, 3 – 244
SET_APPLICATIONPROPERTY, 3 – 245
SET_BLOCKPROPERTY, 3 – 246
SET_CANVASPROPERTY, 3 – 250
SET_FORMPROPERTY, 3 – 251
SET_GROUP_CHAR_CELL, 3 – 255
SET_GROUP_DATE_CELL, 3 – 256
SET_GROUP_NUMBER_CELL, 3 – 258
SET_GROUP_SELECTION, 3 – 259
SET_INPUT_FOCUS, 3 – 260
SET_ITEMPROPERTY, 3 – 260
SET_LOV_COLUMN_PROPERTY, 3 – 270
SET_LOV_PROPERTY, 3 – 271
SET_MENU_ITEM_PROPERTY, 3 – 272
SET_PARAMETER_ATTR, 3 – 274
SET_RADIO_BUTTON_PROPERTY, 3 – 274
SET_RECORD_PROPERTY, 3 – 276
SET_RELATION_PROPERTY, 3 – 278
SET_TIMER, 3 – 280
SET_VIEW_PROPERTY, 3 – 282
SHOW_ALERT, 3 – 286
SHOW_BACKGROUND_MENU, 3 – 287
SHOW_EDITOR, 3 – 288
SHOW_KEYS, 3 – 290
SHOW_LOV, 3 – 290, 8 – 105
SHOW_MENU, 3 – 291
SHOW_VIEW, 3 – 292
SHOW_WINDOW, 3 – 293
Single-user system, 2 – 22
Size, 5 – 182, 5 – 213
Sizing Style, 5 – 184
Space Between Records, 5 – 184
Spread tables (V.3 term), A – 24
Standalone menus, A – 11, A – 18
Starting Menu, 5 – 185
Startup of Forms components, 1 – 4
User Preference File, 1 – 39
Startup Code, 5 – 186
Static function keys, 2 – 7
Statistics
- Generate option, 1 – 29
- Runform option, 1 – 20
Status (Block), 5 – 186
Status (Record), 5 – 187
Storage formats
- Form storage formats, B – 5
- Library storage formats, B – 8
Substitution parameters, constraints, D – 2
Subtitle, 5 – 187
Suppress Hints option, 1 – 37
Switches, A – 26
SYNCHRONIZE, 3 – 294
Synchronously, Run Modules, 1 – 37
System editor (Editor property), 5 – 76
System variables, 4 – 2
SYSTEM.BLOCK_STATUS, 4 – 10
SYSTEM.COORDINATION_OPER., 4 – 11
SYSTEM.CURRENT_BLOCK, 4 – 13
SYSTEM.CURRENT_DATETIME, 4 – 14
SYSTEM.CURRENT_FORM, 4 – 15
SYSTEM.CURRENT_ITEM, 4 – 15
SYSTEM.CURRENT_VALUE, 4 – 16
SYSTEM.CURSOR_BLOCK, 4 – 16
SYSTEM.CURSOR_ITEM, 4 – 17
SYSTEM.CURSOR_RECORD, 4 – 18
SYSTEM.CURSOR_VALUE, 4 – 19
SYSTEM.CUSTOM_ITEM_EVENT, 4 – 20
SYSTEM.CUSTOM_ITEM_EVENT_PARAMETERS, 4 – 20
SYSTEM.DATE_THRESHOLD, 4 – 21
SYSTEM.EFFECTIVE_DATE, 4 – 22
SYSTEM.EVENT_WINDOW, 4 – 23
SYSTEM.FORM_STATUS, 4 – 24
SYSTEM.LAST_QUERY, 4 – 25
SYSTEM.LAST_RECORD, 4 – 27
SYSTEM.MASTER_BLOCK, 4 – 27
SYSTEM.MESSAGE_LEVEL, 4 – 28
SYSTEM.MODE, 4 – 28
SYSTEM.MOUSE_BUTTON_PRESSED, 4 – 29
SYSTEM.MOUSE_BUTTON_SHIFT_STATE, 4 – 29
SYSTEM.MOUSE_CANVAS, 4 – 30
SYSTEM.MOUSE_FORM, 4 – 31
SYSTEM.MOUSE_ITEM, 4 – 32
SYSTEM.MOUSE_RECORD, 4 – 33
SYSTEM.MOUSE_RECORD_OFFSET, 4 – 33
SYSTEM.MOUSE_X_POS, 4 – 34
SYSTEM.MOUSE_Y_POS, 4 – 34
SYSTEM.RECORD_STATUS, 4 – 35
SYSTEM.SUPPRESS_WORKING, 4 – 36
SYSTEM.TRIGGER_BLOCK, 4 – 36
SYSTEM.TRIGGER_ITEM, 4 – 37
SYSTEM.TRIGGER_RECORD, 4 – 38

T

Tables
- Built-in subprograms, 3 – 7
- properties, 5 – 5
- triggers, 2 – 3
Tear-off, 5 – 188
Term
- Designer option, 1 – 38
- Runform option, 1 – 20
TERMINATE, 3 – 295
Text files, converting to binary, A – 10
Time, formatting, 5 – 88
$\$TIME\$, 4 – 9
Timer_Name, 5 – 188
Timers, most recently expired, 3 – 120
Title property, 5 – 188
Top Title, 5 – 189
Top_Record, 5 – 190

Transactional built-ins

- CHECK_RECORD_UNIQUENESS, 3 – 33
- DELETE_RECORD, 3 – 66
- ENFORCE_COLUMN_SECURITY, 3 – 76
- FETCH_RECORDS, 3 – 87
- FORMS_DDL, 3 – 105
- GENERATE_SEQUENCE_NUMBER, 3 – 118
- INSERT_RECORD, 3 – 183
- ISSUE_Rollback, 3 – 184
- ISSUE_Savepoint, 3 – 185
- LOGON, 3 – 189
- LOGOUT, 3 – 192
- SELECT_RECORDS, 3 – 242
- UPDATE_RECORD, 3 – 296

Transactional options

- Key_Mode, 5 – 109
- Locking Mode, 5 – 114

Transactional Triggers, 5 – 190

Trigger macros, A – 17

Trigger Properties

- Show Keys/Show Keys Description, 5 – 180
- table of Trigger Properties, 5 – 15
- Trigger Style, 5 – 191
- Trigger Text, 5 – 191
- Trigger Type, 5 – 192

Trigger properties, Fire in Enter Query Mode, 5 – 82

Trigger Style, 5 – 191

Trigger Tables, 2 – 3

Trigger Text, 5 – 191

Trigger Type, 5 – 192

Triggers

- Add_Triggers Generate option, 1 – 23
- constraints, D – 2
- Fire in Enter Query Mode property, 5 – 82
- new for V.4.5, A – 27
- On–Check–Delete–Master, 2 – 10
- On–Check–Unique, 2 – 11
- On–Clear–Details, 2 – 12
- On–Close, 2 – 13
- On–Column–Security, 2 – 14
- On–Commit, 2 – 15
- On–Count, 2 – 16
- On–Delete, 2 – 17
- On–Error, 2 – 18
- On–Fetch, 2 – 19
- On–Insert, 2 – 21
- On–Lock, 2 – 22
- On–Logon, 2 – 23
- On–Logout, 2 – 24
- On–Message, 2 – 25
- On–Populate–Details, 2 – 26
- On–Rollback, 2 – 27
- On–Savepoint, 2 – 28
- On–Select, 2 – 29
- On–Sequence–Number, 2 – 30
- On–Update, 2 – 31
- Optimizing V2–style triggers, 1 – 15
- Post–Block, 2 – 32
- Post–Change, 2 – 33
- Post–Database–Commit, 2 – 35
- Post–Delete, 2 – 36
- Post–Form, 2 – 37
- Post–Forms–Commit, 2 – 38
- Post–Insert, 2 – 39
- Post–Logon, 2 – 40
- Post–Logout, 2 – 41
- Post–Query, 2 – 42
- Post–Record, 2 – 44
- Post–Select, 2 – 45
- Post–Text–Item, 2 – 46
- Post–Update, 2 – 47
- Pre–Block, 2 – 48
- Pre–Commit, 2 – 49
- Pre–Delete, 2 – 50
- Pre–Form, 2 – 51
- Pre–Insert, 2 – 52
- Pre–Logon, 2 – 53
- Pre–Logout, 2 – 54
- Pre–Query, 2 – 55
- Pre–Record, 2 – 57
- Pre–Select, 2 – 58
- Pre–Text–Item, 2 – 59
- Pre–Update, 2 – 60
- renamed for V.4.5, A – 27
- Triggers and Processes, 2 – 2
- User–Defined, 2 – 62
- When–Button–Pressed, 2 – 63
- When–Checkbox–Changed, 2 – 64
- When–Clear–Block, 2 – 65
- When–Create–Record, 2 – 66
- When–Custom–Item–Event trigger, 2 – 67
Index – 27

When–Database–Record, 2 – 68
When–Form–Navigate trigger, 2 – 69
When–Image–Activated, 2 – 69
When–Image–Pressed, 2 – 70
When–List–Activated, 2 – 70
When–List–Changed, 2 – 71
When–Mouse–Click, 2 – 71
When–Mouse–DoubleClick, 2 – 72
When–Mouse–Down, 2 – 74
When–Mouse–Enter, 2 – 75
When–Mouse–Leave, 2 – 76
When–Mouse–Move, 2 – 77
When–Mouse–Up, 2 – 78
When–New–Block–Instance, 2 – 79
When–New–Form–Instance, 2 – 80
When–New–Item–Instance, 2 – 81
When–New–Record–Instance, 2 – 82
When–Radio–Changed, 2 – 83
When–Remove–Record, 2 – 84
When–Timer–Expired, 2 – 85
When–Validate–Item, 2 – 87
When–Validate–Record, 2 – 89
When–Window–Activated, 2 – 91
When–Window–Closed, 2 – 92
When–Window–Deactivated, 2 – 92
When–Window–Resized, 2 – 93

Tuning applications
Array preference, 1 – 11
OptimizeSQL option, 1 – 15
OptimizeTP option, 1 – 16

Typographic conventions, iii, iii

U
Unchecked Value, 5 – 192
Unchecked Value (Check Box), 5 – 36
Unrestricted built-in subprograms, 3 – 6
UNSET_GROUP_SELECTION, 3 – 295
UP, 3 – 296, 8 – 107
Update Allowed (Block), 5 – 193
Update Allowed (Item), 5 – 194
Update Changed Columns, 5 – 195
Update Column, 5 – 196
Update Only if NULL, 5 – 197
Update_Permission, 5 – 198
UPDATE_RECORD, 3 – 296
Upgrade option, 1 – 29, A – 2
Upgrade_Roles option, 1 – 30
Upgrading
beveling, A – 46
character-mode applications, A – 17
command line options, A – 43
correction sequence, A – 43
enhanced functionality, A – 18
fields, A – 15
form–level procedures, A – 8
forms containing referenced objects, A – 6
from Version 2.3, A – 5
from Version 3.0, A – 4
from Version 4.0, A – 3
GUI conversion, A – 39
key triggers, A – 45
LOVs, A – 15, A – 20
master–detail relationships, A – 16
menu security roles, A – 13
menus, A – 11
migration strategies, A – 39
on a GUI platform, A – 5
pages, A – 15
Parse option, A – 10
PL/SQL libraries, A – 21
PL/SQL program units, A – 7
properties, A – 16
running after conversion, A – 5
Script option, A – 10
terminology changes, A – 24
triggers, A – 14
user exits, A – 22
using a command line on MS Windows, A – 2
using GUI functionality, A – 44
using the batch option, A – 3
V2–style triggers, A – 14
Use 3D Controls, 5 – 201, 5 – 215
Use File, 5 – 199
Use Security, 5 – 201
Use System Editor option, 1 – 38
User exits, new statements, A – 22
User interface, 3 – 120
User preferences, 1 – 39
User–defined trigger, 2 – 62
User–Interface, 5 – 202
USER_EXIT, 3 – 297
User_NLS_Lang, 5 – 203
Userid, for database, 1 – 8
Username, 5 – 202

V
V2–style trigger (Type property), 5 – 192
V2–style triggers, A – 14
VALIDATE, 3 – 299
Validate the Block, 8 – 108
Validate the Form, 8 – 109
Validate the Item, 8 – 110
Validate the Record, 8 – 112
Validation, 5 – 203
  augmenting default validation, 7 – 6
  Item validation states, 7 – 7
  Record validation states, 7 – 9
  Standard validation checks, 7 – 10
  Status checking, 7 – 7
  Validation objects, 7 – 6
  Validation unit, 7 – 6
  When validation occurs, 7 – 6
Validation of changed item properties, 3 – 267
Validation Unit, 5 – 204
Value, 5 – 205
Variable names, constraints on, D – 2
Variables (System variables), 4 – 1
VBX Control File, 5 – 205
VBX Control Name, 5 – 206
VBX Control Value Property, 5 – 206
VBX.FIRE_EVENT, 3 – 300
VBX.GET_PROPERTY, 3 – 301
VBX.GET_VALUE_PROPERTY, 3 – 302
VBX.INVOKE_METHOD, 3 – 303
VBX.SET_PROPERTY, 3 – 304
VBX.SET_VALUE_PROPERTY, 3 – 305
Version 4.5
  concepts and functionality, A – 23
  converting forms and menus, A – 2
  file formats, A – 9
  new built–in subprograms, A – 29
  new features, A – 23
  new properties, A – 33
  new terminology, A – 24
  new triggers, A – 27
Version option, 1 – 30, A – 2
Vertical MDI Toolbar, 5 – 207
Vertical Scroll Bar, 5 – 208
Vertical Toolbar property, 5 – 208
View, 5 – 209
View Height, 5 – 210
View Horizontal Scroll Bar, 5 – 210
View Vertical Scroll Bar, 5 – 211
View Width, 5 – 210
Visible, 5 – 211
Visual Attribute Name property, 5 – 213
Visual attributes
  constraints, D – 2
  Oracle Terminal Resource File option, 1 – 20
Visual_Attribute, 5 – 216

W
Weight, 5 – 214
When–Button–Pressed trigger, 2 – 63
When–Checkbox–Changed trigger, 2 – 64
When–Clear–Block trigger, 2 – 65
When–Create–Record trigger, 2 – 66
When–Custom–Item–Event trigger, 2 – 67
When–Database–Record trigger, 2 – 68
When–Form–Navigate trigger, 2 – 69
When–Image–Activated trigger, 2 – 69
When–Image–Pressed trigger, 2 – 70
When–List–Activated trigger, 2 – 70
When–List–Changed trigger, 2 – 71
When–Mouse–Click trigger, 2 – 71
When–Mouse–DoubleClick trigger, 2 – 72
When–Mouse–Down trigger, 2 – 74
When–Mouse–Enter trigger, 2 – 75
When–Mouse–Leave trigger, 2 – 76
When–Mouse–Move trigger, 2 – 77
When–Mouse–Up trigger, 2 – 78
When–New–Block–Instance trigger, 2 – 79
When–New–Form–Instance trigger, 2 – 80
When–New–Item–Instance trigger, 2 – 81
When–New–Record–Instance trigger, 2 – 82
When–Radio–Changed trigger, 2 – 83
When–Remove–Record trigger, 2 – 84
When–Timer–Expired trigger, 2 – 85
When–Validate–Item trigger, 2 – 87
When–Validate–Record trigger, 2 – 89
When–Window–Activated trigger, 2 – 91
When–Window–Closed trigger, 2 – 92
When–Window–Deactivated trigger, 2 – 92
When–Window–Resized trigger, 2 – 93
Where clause, 5 – 217
Where Clause/Order By Clause, 5 – 217
WHERE_DISPLAY, 3 – 306
Widen Fields option, A – 46
Widen_Fields option, 1 – 31
Widow Style, 5 – 221
Window, 5 – 218
Window built-ins
FIND_WINDOW, 3 – 100
GET_WINDOW_PROPERTY, 3 – 170
HIDE_WINDOW, 3 – 176
ID_NULL, 3 – 180
MOVE_WINDOW, 3 – 200
REPLACE_CONTENT_VIEW, 3 – 229
RESIZE_WINDOW, 3 – 233
SET_WINDOW_PROPERTY, 3 – 284
SHOW_WINDOW, 3 – 293
Window Properties
Closeable, 5 – 38
Fixed Size, 5 – 87
Horizontal Scroll Bar, 5 – 97
Horizontal Toolbar, 5 – 98
Icon Name, 5 – 99
Icon Title, 5 – 100
Iconifiable, 5 – 101
Inherit Menu, 5 – 102
Modal, 5 – 132
Moveable, 5 – 135
Remove on Exit, 5 – 173
Size, 5 – 182
table of Window Properties, 5 – 16
Vertical Scroll Bar, 5 – 208
Vertical Toolbar, 5 – 208
View, 5 – 209
Visible, 5 – 211
Window Style, 5 – 221
Window_Handle, 5 – 219
Window_State, 5 – 220
Zoomable, 5 – 224
Window_Handle, 5 – 219
Window_State, 5 – 220
Window_State option, 1 – 21
Windows
activating, 2 – 91
closing, 2 – 92
deactivating, 2 – 92
firing triggers for, 2 – 93
firing triggers when window activated, 2 – 91
firing triggers when window closed, 2 – 92
firing triggers when window deactivated, 2 – 92
FORMS_MDI_WINDOW constant, 3 – 286
MDI, 5 – 221
resizing, 2 – 93
ROOT_WINDOW, 5 – 218
Wrap Style, 5 – 222
Write Input Keystrokes to File option, 1 – 14
Write Output to Display option, 1 – 13
Write Output to File option, 1 – 17
WRITE_IMAGE_FILE, 3 – 306

X
X Position, 5 – 222
X Position on Canvas, 5 – 224

Y
Y Position, 5 – 222
Y Position on Canvas, 5 – 224
Z

Zoomable, 5 – 224
Reader’s Comment Form

Part No. A32510–2

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

• Did you find any errors?
• Is the information clearly presented?
• Do you need more information? If so, where?
• Are the examples correct? Do you need more examples?
• What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the topic, chapter, and page number below:

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

Please send your comments to:

Forms Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood City, CA  94065  U.S.A.
Fax: (415) 506–7200

If you would like a reply, please give your name, address, and telephone number below:

________________________________________________________________________
________________________________________________________________________

Thank you for helping us improve our documentation.
Forms Reference Manual

Release 4.5