Oracle9*i* Forms Developer

Reference Guide

Release 9.0.2

March 2002

Part No. A97289-01

ORACLE®

Oracle9i Forms Developer Reference Guide, Release 9.0.2

Part No. A97289-01

Contributing Authors: Orlando Cordero, Cathy Godwin, and Dean Ho

Table of Contents

Oracle9*i* Forms Developer Reference Guide

1. Builder and Developer Options

- About Developer Components
- About the Form Compiler Window
- About the PL/SQL Compilation Messages Window
- Application Server URL
- Build Before Running Option
- Forms Developer Options
- Builder Preferences
- Color Mode
- Color Palette
- Compiler Options
- Default Color Palette Colors
- Help (Forms Developer)
- Module_Type (Form Builder)
- Options
- Runtime Options
- Setting Forms Developer Preferences
- Starting Forms Developer Components from the Command Line
- Subclassing Path
- Suppress Hints
- Use System Editor
- User Preference File
- Welcome Dialog
- Welcome Pages

2. Forms Compiler Options

- Add_Triggers (Form Compiler)
- Batch (Form Compiler)
- Build (Form Compiler)
- Compile_All (Form Compiler)
- CRT_File (Form Compiler)
- Help (Form Compiler)
- Logon (Form Compiler)
- Module_Type (Form Compiler)
- Nofail (Form Compiler)
- Options_Screen (Form Compiler)
- Parse (Form Compiler)
- Script (Form Compiler)
- Strip_Source (Form Compiler)
- Upgrade (Forms Compiler)
- Upgrade_Roles (Form Compiler)
- Version (Form Compiler)
- Widen_Fields (Form Compiler)

3. Runtime Options

- Forms Runtime Options
- Array (Forms Runtime)
- Buffer_Records (Forms Runtime)
- Debug (Forms Runtime)
- Debug Messages (Forms Runtime)
- Query_Only (Forms Runtime)

- Quiet (Forms Runtime)
- USESDI (Web Forms Runtime)

4. Built-ins

- About Built-in Code Examples
- Restricted Built-in Subprograms
- Built-in Constants
- Built-in Packages
- Individual Built-in Descriptions
- Built-in syntax
- Built-in Named Parameters
- Built-in ObjectIDs
- Built-in Form Coordinate Units
- Built-in Uppercase Return Values

List of Built-in Packages

- Alert Built-ins
- Application Built-ins
- Block Built-ins
- Canvas Built-ins
- Chart Built-ins
- Form Builder Built-ins
- Item Built-ins
- Menu Built-ins
- Message Built-in
- Multiple Form Built-ins
- Parameter List Built-ins
- Query Built-ins

- Record Built-ins
- Relation Built-ins
- Report Built-ins
- Tab Page Built-ins
- Transaction Built-ins
- Web Built-ins
- Window Built-ins

List of Built-ins

- ABORT_QUERY Built-in
- ADD_GROUP_COLUMN Built-in
- ADD_GROUP_ROW Built-in
- ADD_LIST_ELEMENT Built-in
- ADD_PARAMETER Built-in
- ADD_TREE_DATA Built-in
- ADD_TREE_NODE Built-in
- ADJUST_TZ Built-in
- BELL Built-in
- CALL_FORM Built-in
- CALL_INPUT Built-in
- CANCEL_REPORT_OBJECT Built-in
- CHECKBOX_CHECKED Built-in
- CHECK_RECORD_UNIQUENESS Built-in
- CLEAR_BLOCK Built-in
- CLEAR_EOL Built-in
- CLEAR_FORM Built-in
- CLEAR_ITEM Built-in
- CLEAR_LIST Built-in
- CLEAR_MESSAGE Built-in
- CLEAR_RECORD Built-in

- CLOSE_FORM Built-in
- COMMIT_FORM Built-in
- CONVERT_OTHER_VALUE Built-in
- COPY Built-in
- COPY_REGION Built-in
- COPY_REPORT_OBJECT_OUTPUT Built-in
- COUNT_QUERY Built-in
- CREATE_GROUP Built-in
- CREATE_GROUP_FROM_QUERY Built-in
- CREATE_PARAMETER_LIST Built-in
- CREATE_QUERIED_RECORD Built-in
- CREATE_RECORD Built-in
- CREATE_TIMER Built-in
- CREATE_VAR Built-in
- CUT_REGION Built-in
- DBMS_ERROR_CODE Built-in
- DBMS_ERROR_TEXT Built-in
- DEFAULT_VALUE Built-in
- DELETE_GROUP Built-in
- DELETE_GROUP_ROW Built-in
- DELETE_LIST_ELEMENT Built-in
- DELETE_PARAMETER Built-in
- DELETE_RECORD Built-in
- DELETE_TIMER Built-in
- DELETE_TREE_NODE Built-in
- DESTROY_PARAMETER_LIST Built-in
- DESTROY_VARIANT Built-in
- DISPLAY_ERROR Built-in
- DISPLAY_ITEM Built-in
- DOWN Built-in

- DO_KEY Built-in
- DUMMY_REFERENCE Built-in
- DUPLICATE_ITEM Built-in
- DUPLICATE_RECORD Built-in
- EDIT_TEXTITEM Built-in
- ENFORCE_COLUMN_SECURITY Built-in
- ENTER Built-in
- ENTER_QUERY Built-in
- ERASE Built-in
- ERROR_CODE Built-in
- ERROR_TEXT Built-in
- ERROR_TYPE Built-in
- EXECUTE_QUERY Built-in
- EXIT_FORM Built-in
- FETCH_RECORDS Built-in
- FIND_ALERT Built-in
- FIND_BLOCK Built-in
- FIND_CANVAS Built-in
- FIND_COLUMN Built-in
- FIND_EDITOR Built-in
- FIND_FORM Built-in
- FIND_GROUP Built-in
- FIND_ITEM Built-in
- FIND_LOV Built-in
- FIND_MENU_ITEM Built-in
- FIND_RELATION Built-in
- FIND_REPORT_OBJECT Built-in
- FIND_TAB_PAGE Built-in
- FIND_TIMER Built-in
- FIND_TREE_NODE Built-in

- FIND_VA Built-in
- FIND_VIEW Built-in
- FIND_WINDOW Built-in
- FIRST_RECORD Built-in
- FORM_FAILURE Built-in
- FORM_FATAL Built-in
- FORM_SUCCESS Built-in
- FORMS_DDL Built-in
- GENERATE_SEQUENCE_NUMBER Built-in
- GENERATE_SEQUENCE_NUMBER Built-in
- GET_APPLICATION_PROPERTY Built-in
- GET_BLOCK_PROPERTY Built-in
- GET_CANVAS_PROPERTY Built-in
- GET_CUSTOM_PROPERTY Built-in
- GET_FILE_NAME Built-in
- GET_FORM_PROPERTY Built-in
- GET_GROUP_CHAR_CELL Built-in
- GET_GROUP_DATE_CELL Built-in
- GET_GROUP_NUMBER_CELL Built-in
- GET_GROUP_RECORD_NUMBER Built-in
- GET_GROUP_ROW_COUNT Built-in
- GET_GROUP_SELECTION Built-in
- GET_GROUP_SELECTION_COUNT Built-in
- GET_ITEM_INSTANCE_PROPERTY Built-in
- GET_ITEM_PROPERTY Built-in
- GET_LIST_ELEMENT_COUNT Built-in
- GET_LIST_ELEMENT_LABEL Built-in
- GET_LIST_ELEMENT_VALUE Built-in
- GET_LOV_PROPERTY Built-in
- GET_MENU_ITEM_PROPERTY Built-in

- GET_MESSAGE Built-in
- GET_PARAMETER_ATTR Built-in
- GET_PARAMETER_LIST Built-in
- GET_RADIO_BUTTON_PROPERTY Built-in
- GET_RECORD_PROPERTY Built-in
- GET_RELATION_PROPERTY Built-in
- GET_REPORT_OBJECT_PROPERTY Built-in
- GET_TAB_PAGE_PROPERTY Built-in
- GET_TREE_NODE_PARENT Built-in
- GET_TREE_NODE_PROPERTY Built-in
- GET_TREE_PROPERTY Built-in
- GET_TREE_SELECTION Built-in
- GET_VA_PROPERTY Built-in
- GET_VAR_BOUNDS Built-in
- GET_VAR_DIMS Built-in
- GET_VAR_TYPE Built-in
- GET_VIEW_PROPERTY Built-in
- GET_WINDOW_PROPERTY Built-in
- GO_BLOCK Built-in
- GO_FORM Built-in
- GO_ITEM Built-in
- GO_RECORD Built-in
- HELP Built-in
- HIDE_VIEW Built-in
- HIDE_WINDOW Built-in
- HOST Built-in
- ID_NULL Built-in
- IMAGE_SCROLL Built-in
- IMAGE_ZOOM Built-in
- INSERT_RECORD Built-in

- ISSUE_ROLLBACK Built-in
- ISSUE_SAVEPOINT Built-in
- LAST_RECORD Built-in
- LIST_VALUES Built-in
- LOCK_RECORD Built-in
- LOGON Built-in
- LOGON_SCREEN Built-in
- LOGOUT Built-in
- MESSAGE Built-in
- MESSAGE_CODE Built-in
- MESSAGE_TEXT Built-in
- MESSAGE_TYPE Built-in
- MOVE_WINDOW Built-in
- NAME_IN Built-in
- NEW_FORM Built-in
- NEXT_BLOCK Built-in
- NEXT_FORM Built-in
- NEXT_ITEM Built-in
- NEXT_KEY Built-in
- NEXT_RECORD Built-in
- NEXT_SET Built-in
- OPEN_FORM Built-in
- PASTE_REGION Built-in
- PAUSE Built-in
- POPULATE_GROUP Built-in
- POPULATE_GROUP_FROM_TREE Built-in
- POPULATE_GROUP_WITH_QUERY Built-in
- POPULATE_LIST Built-in
- POPULATE_TREE Built-in
- POST Built-in

- POST Built-in
- PREVIOUS_BLOCK Built-in
- PREVIOUS_FORM Built-in
- PREVIOUS_ITEM Built-in
- PREVIOUS_RECORD Built-in
- PRINT Built-in
- READ_IMAGE_FILE Built-in
- RECALCULATE Built-in
- REDISPLAY Built-in
- REPLACE_CONTENT_VIEW Built-in
- REPLACE_MENU Built-in
- REPORT_OBJECT_STATUS Built-in
- RESET_GROUP_SELECTION Built-in
- RESIZE_WINDOW Built-in
- RETRIEVE_LIST Built-in
- RUN_PRODUCT Built-in
- RUN_REPORT_OBJECT Built-in
- SCROLL_DOWN Built-in
- SCROLL_UP Built-in
- SCROLL_VIEW Built-in
- SELECT_ALL Built-in
- SELECT_RECORDS Built-in
- SET_ALERT_BUTTON_PROPERTY Built-in
- SET_APPLICATION_PROPERTY Built-in
- SET_BLOCK_PROPERTY Built-in
- SET_CANVAS_PROPERTY Built-in
- SET_CUSTOM_ITEM_PROPERTY Built-in
- SET_CUSTOM_PROPERTY Built-in
- SET_FORM_PROPERTY Built-in
- SET_GROUP_CHAR_CELL Built-in

- SET_GROUP_DATE_CELL Built-in
- SET_GROUP_NUMBER_CELL Built-in
- SET_GROUP_SELECTION Built-in
- SET_ITEM_INSTANCE_PROPERTY Built-in
- SET_ITEM_PROPERTY Built-in
- SET_LOV_COLUMN_PROPERTY Built-in
- SET_LOV_PROPERTY Built-in
- SET_MENU_ITEM_PROPERTY Built-in
- SET_PARAMETER_ATTR Built-in
- SET_RADIO_BUTTON_PROPERTY Built-in
- SET_RECORD_PROPERTY Built-in
- SET_RELATION_PROPERTY Built-in
- SET_REPORT_OBJECT_PROPERTY Built-in
- SET_TAB_PAGE_PROPERTY Built-in
- SET_TIMER Built-in
- SET_TREE_NODE_PROPERTY Built-in
- SET_TREE_PROPERTY Built-in
- SET_TREE_SELECTION Built-in
- SET_VA_PROPERTY Built-in
- SET_VIEW_PROPERTY Built-in
- SET_WINDOW_PROPERTY Built-in
- SHOW_ALERT Built-in
- SHOW_EDITOR Built-in
- SHOW_KEYS Built-in
- SHOW_LOV Built-in
- SHOW_VIEW Built-in
- SHOW_WINDOW Built-in
- SYNCHRONIZE Built-in
- UNSET_GROUP_SELECTION Built-in
- UP Built-in

- UP Built-in
- UPDATE_CHART Built-in
- UPDATE_RECORD Built-in
- USER_EXIT Built-in
- VALIDATE Built-in
- WEB.SHOW_DOCUMENT Built-in
- WRITE_IMAGE_FILE File Built-in

5. Properties

- About Properties
- Properties
- About Setting and Modifying Properties
- Reading Property Descriptions

List of Property Types

- Alert Properties
- Application Properties
- Automatic Skip Properties
- Block Properties
- Canvas Properties
- Chart Properties
- Communication Mode Properties
- Data Source Data Block properties
- Editor Properties
- Execution Mode Properties
- Forms Developer Properties
- Frame Properties
- Graphics Properties
- Hint properties

- Item Properties
- Label Properties
- LOV Properties
- Menu Properties
- Module Properties
- Name Visual Attribute Properties
- Parameter Properties
- Prompt Properties
- Record Group Properties
- Relation Properties
- Report Properties
- Tab Page Properties
- Trigger Properties
- Window Properties

List of Properties

- Access Key Property
- Alert Style Property
- Alias Property
- Allow Empty Branches Property
- Allow Expansion Property
- Allow Multi-Line Prompts Property
- Allow Start-Attached Prompts Property
- Allow Top-Attached Prompts Property
- Application Instance Property
- Arrow Style Property
- Automatic Column Width Property
- Automatic Display Property
- Automatic Position Property
- Automatic Query Property

- Automatic Refresh Property
- Automatic Select Property
- Automatic Skip (Item) Property
- Automatic Skip (LOV) Property
- Background Color Property
- Block Description Property
- Border Bevel Property
- Bottom Title (Editor) Property
- Bounding Box Scalable Property
- Builtin Date Format Property
- Button 1 Label, Button 2 Label, Button 3 Label Property
- Calculation Mode Property
- Calling Form Property
- Canvas Property
- Canvas Type Property
- Cap Style Property
- Case Insensitive Query Property
- Case Restriction Property
- Character Cell WD/HT Property
- Chart Subtype Property
- Check Box Mapping of Other Values Property
- Checked Property
- Clip Height Property
- Clip Width Property
- Clip X Position Property
- Clip Y Position Property
- Closed Property
- Column Mapping Properties Property
- Column Name Property
- Column Specifications Property

- Column Title (LOV) Property
- Column Value (Record Group) Property
- Command Text Property
- Command Type Property
- Comments Property
- Communication Mode (Chart) Property
- Communication Mode (Report) Property
- Compression Quality Property
- Conceal Data Property
- Console Window Property
- Coordinate System Property
- Coordination Property
- Copy Value from Item Property
- Current Form Name Property
- Current Form Property
- Current Record Property
- Current Record Visual Attribute Group Property
- Current Row Background Color PropertyProperty
- Current Row Fill Pattern Property
- Current Row Font Name Property
- Current Row Font Size Property
- Current Row Font Spacing Property
- Current Row Font Style Property
- Current Row Font Weight Property
- Current Row Foreground Color Property
- Cursor Mode Property
- Cursor Style Property
- Custom Spacing Property
- Dash Style Property
- Data Length Semantics Property

- Data Query Property
- Data Source Data Block (Chart) Property
- Data Source Data Block (Report) Property
- Data Source X Axis Property
- Data Source Y Axis Property
- Data Type (Record Group) Property
- Data Type Property
- Database Value Property
- Datetime Local TZ Property
- Datetime Server TZ Property
- Default Alert Button Property
- Default Button Property
- Default Font Scaling Property
- Defer Required Enforcement Property
- Deferred Property
- Delete Allowed Property
- Delete Procedure Arguments Property
- Delete Procedure Name Property
- Delete Procedure Result Set Columns Property
- Delete Record Behavior Property
- Detail Block Property
- Detail Reference Item Property
- Direction Property
- Display Hint Automatically Property
- Display in Keyboard Help Property
- Display Height Property
- Displayed Property
- Display Quality Property
- Display Width Property
- Display Width (LOV) Property

- Display without Privilege Property
- Distance Between Records Property
- Dither Property
- DML Array Size Property
- DML Data Target Name Property
- DML Data Target Type Property
- DML Returning Value Property
- Edge Background Color Property
- Edge Foreground Color Property
- Edge Pattern Property
- Editor Property
- Editor X Position, Editor Y Position
- Elements in List Property
- Enabled (Item) Property
- Enabled (Menu Item) Property
- Enabled (Tab Page) Property
- Enabled properties
- End Angle Property
- Enforce Column Security Property
- Enforce Primary Key (Block) Property
- Enterable Property
- Error Date/Datetime Format Property
- Execution Hierarchy Property
- Execution Mode (Chart) Property
- Execution Mode (Report) Property
- Filename Property
- Fill Pattern Property
- Fill Property
- Filter Before Display Property
- Fire in Enter-Query Mode Property

- First Detail Relation Property
- First_Block Property
- First Item Property
- First Master Relation Property
- First Navigation Block Property
- Fixed Bounding Box Property
- Flag User Value Too Long Property
- Font Name Property
- Font Size Property
- Font Spacing Property
- Font Style Property
- Font Weight Property
- Foreground Color Property
- Form Horizontal Toolbar Canvas Property
- Form Name Property
- Form Vertical Toolbar Canvas Property
- Format Mask Property
- Formula Property
- Frame Alignment Property
- Frame Title Alignment Property
- Frame Title Font Name Property
- Frame Title Font Size Property
- Frame Title Font Spacing Property
- Frame Title Font Style Property
- Frame Title Font Weight Property
- Frame Title Foreground Color Property
- Frame Title Offset Property
- Frame Title Property
- Frame Title Visual Attribute Group Property
- Graphics Type Property

- Group Name Property
- Hide on Exit Property
- Hint (Item) Property
- Horizontal Justification Property
- Horizontal Margin
- Horizontal Object Offset Property
- Horizontal Toolbar Canvas Property
- Icon Filename Property
- Icon in Menu Property
- Iconic Property
- Image Depth Property
- Image Format Property
- Implementation Class Property
- Include REF Item Property
- Inherit Menu Property
- Initial Value (Item) Property
- Insert Allowed (Block) Property
- Insert Allowed (Item) Property
- Insert Procedure Arguments Property
- Insert Procedure Name Property
- Isolation Mode Property
- Item Is Valid Property
- Item Roles Property
- Item Tab Page Property
- Join Condition Property
- Join Style Property
- Justification Property
- Keep Cursor Position Property
- Key Mode Property
- Keyboard Accelerator Property

- Keyboard Help Description Property
- Keyboard Navigable Property
- Label (Item) Property
- Label (Menu Item) Property
- Label (Tab Page) Property
- Last Block Property
- Last Item Property
- Last Query Property
- Layout Data Block Property
- Layout Style
- Layout Style Property
- Length (Record Group) Property
- Line Spacing Property
- Line Width Property
- List of Values Property
- List Style Property
- List X Position Property
- List Y Position Property
- Lock Procedure Arguments Property
- Lock Procedure Name Property
- Lock Procedure Result Set Columns Property
- Lock Record Property
- Locking Mode Property
- Magic Item Property
- Main Menu Property
- Mapping of Other Values Property
- Maximize Allowed Property
- Maximum Length (Form Parameter) Property
- Maximum Length Property
- Maximum Objects Per Line Property

- Maximum Query Time Property
- Maximum Records Fetched Property
- Menu Directory Property
- Menu Filename Property
- Menu Item Code Property
- Menu Item Radio Group Property
- Menu Item Type Property
- Menu Module Property
- Menu Style Property
- Message Property
- Minimize Allowed Property
- Minimized Title Property
- Modal Property
- Module Roles Property
- Module NLS Lang Property
- Mouse Navigate Property
- Mouse Navigation Limit Property
- Move Allowed Property
- Multi-Line Property
- Multi-Selection Property
- Name Property
- Navigation Style Property
- Next Block Property
- Next Detail Relation Property
- Next Item Property
- Next Master Relation Property
- Next Navigation Block Property
- Next Navigation Item Property
- Number of Items Displayed Property
- Number of Records Buffered Property

- Number of Records Displayed Property
- Onetime Where Property
- Operating System Property
- Optimizer Hint Property
- Order By Property
- Other Reports Parameters Property
- Output Date/Datetime Format Property
- Parameter Data Type Property
- Parameter Initial Value (Form Parameter) Property
- Password Property
- PL/SQL Date Format Property
- PL/SQL Library Location Property
- PL/SQL Library Source Property
- Popup Menu Property
- Precompute Summaries Property
- Prevent Masterless Operations Property
- Previous Block Property
- Previous Item Property
- Previous Navigation Block Property
- Previous Navigation Item Property
- Primary Canvas Property
- Primary Key (Item) Property
- Program Unit Text Property
- Prompt Alignment Offset Property
- Prompt Alignment Property
- Prompt Attachment Edge Property
- Prompt Attachment Offset Property
- Prompt Background Color Property
- Prompt Display Style Property
- Prompt Fill Pattern Property

- Prompt Font Name Property
- Prompt Font Size Property
- Prompt Font Spacing Property
- Prompt Font Style Property
- Prompt Font Weight Property
- Prompt Foreground Color Property
- Prompt Justification Property
- Prompt Property
- Prompt Reading Order Property
- Prompt Visual Attribute Group Property
- Property Class Property
- Query All Records Property
- Query Allowed (Block) Property
- Query Allowed (Item) Property
- Query Array Size Property
- Query Data Source Arguments Property
- Query Data Source Columns Property
- Query Data Source Name Property
- Query Data Source Type Property
- Query Hits Property
- Query Length Property
- Query Name Property
- Query Only Property
- Query Options Property
- Radio Button Value Property
- Raise on Entry Property
- Reading Order Property
- Real Unit Property
- Record Group Property
- Record Group Query Property

- Record Group Type Property
- Record Orientation Property
- Records To Fetch Property
- Relation Type Property
- Rendered Property
- Report Destination Format Property
- Report Destination Name Property
- Report Destination Type Property
- Report Server Property
- Required (Item) Property
- Resize Allowed Property
- Return Item (LOV) Property
- Rotation Angle Property
- Runtime Compatibility Mode Property
- Savepoint Mode Property
- Savepoint Name Property
- Scroll Bar Alignment Property
- Scroll Bar Height Property
- Scroll Bar Width Property
- Share Library with Form Property
- Show Horizontal Scroll Bar Property
- Show Lines Property
- Show Palette Property
- Show Scroll Bar Property
- Show Symbols Property
- Show Vertical Scroll Bar Property
- Shrinkwrap Property
- Single Object Alignment Property
- Single Record Property
- Size Property

- Sizing Style Property
- SSO USERID Property
- Start Angle Property
- Start Prompt Alignment Property
- Start Prompt Offset Property
- Startup Code Property
- Status (Block) Property
- Status (Record) Property
- Subclass Information Property
- Tab Page Property
- Tab Page X Offset Property
- Tab Page Y Offset Property
- Tab Style Property
- Title Property
- Tooltip Background Color Property
- Tooltip Fill Pattern Property
- Tooltip Font Name Property
- Tooltip Font Size Property
- Tooltip Font Spacing Property
- Tooltip Font Style Property
- Tooltip Font Weight Property
- Tooltip Foreground Color Property
- Tooltip Property
- Tooltip Visual Attribute Group Property
- Top Prompt Alignment Property
- Top Prompt Offset Property
- Top Record Property
- Top Title Property
- Topmost Tab Page Property
- Transactional Triggers Property

- Trigger Style Property
- Trigger Text Property
- Trigger Type Property
- Update Allowed (Block) Property
- Update Allowed (Item) Property
- Update Changed Columns Only Property
- Update Column Property
- Update Commit Property
- Update Layout Property
- Update Only if NULL Property
- Update Permission Property
- Update Procedure Arguments Property
- Update Procedure Name Property
- Update Procedure Result Set Columns Property
- Update Query Property
- Use 3D Controls Property
- Use Security Property
- User Date/Datetime Format Property
- User Interface Property
- User NLS Date Format Property
- User NLS Lang Property
- Username Property
- Validate From List Property
- Validation Property
- Validation Unit Property
- Value When Checked Property
- Value When Unchecked Property
- Vertical Fill Property
- Vertical Justification Property
- Vertical Margin

- Vertical Margin Property
- Vertical Object Offset Property
- Vertical Origin Property
- Vertical Toolbar Canvas Property
- Viewport Height, Viewport Width Property
- Viewport X Position on Canvas, Viewport Y Position on Canvas Property
- Visible (Canvas) Property
- Visible (Tab Page) Property
- Visible In Horizontal/Vertical Menu Toolbar Property
- Visible In Menu Property
- Visible Property
- Visual Attribute Group Property
- Visual Attribute Property
- Visual Attribute Type Property
- WHERE Clause/ORDER BY Clause Properties
- Width/Height (WD, HT) Property
- Window Handle Property
- Window Property
- Window State Property
- Window Style Property
- Wrap Style Property
- Wrap Text Property
- X Corner Radius Property
- X Position, Y Position
- Y Corner Radius Property

6. System Variables

- About System Variables
- Date and Time System Default Values

List of System Variables

- $$DATE$$ System Variable
- $$DATETIME$$ System Variable
- $$DBDATE$$ System Variable
- $$DBDATETIME$$ System Variable
- $$DBTIME$$ System Variable
- $$TIME$$ System Variable
- SYSTEM.BLOCK_STATUS System Variable
- SYSTEM.COORDINATION_OPERATION System Variable
- SYSTEM.CURRENT_BLOCK System Variable
- SYSTEM.CURRENT_DATETIME System Variable
- SYSTEM.CURRENT_FORM System Variable
- SYSTEM.CURRENT_ITEM System Variable
- SYSTEM.CURRENT_VALUE System Variable
- SYSTEM.CURSOR_BLOCK System Variable
- SYSTEM.CURSOR_ITEM System Variable
- SYSTEM.CURSOR_RECORD System Variable
- SYSTEM.CURSOR_VALUE System Variable
- SYSTEM.DATE_THRESHOLD System Variable
- SYSTEM.EFFECTIVE_DATE System Variable
- SYSTEM.EVENT_WINDOW System Variable
- SYSTEM.FORM_STATUS System Variable
- SYSTEM.LAST_FORM System Variable
- SYSTEM.LAST_QUERY System Variable
- SYSTEM.LAST_RECORD System Variable
- SYSTEM.MASTER_BLOCK System Variable
- SYSTEM.MESSAGE_LEVEL System Variable
- SYSTEM.MODE System Variable
- SYSTEM.MOUSE_BUTTON_MODIFIERS System Variable

- SYSTEM.MOUSE_BUTTON_PRESSED System Variable
- SYSTEM.MOUSE_BUTTON_SHIFT_STATE System Variable
- SYSTEM.MOUSE_CANVAS System Variable
- SYSTEM.MOUSE_FORM System Variable
- SYSTEM.MOUSE_ITEM System Variable
- SYSTEM.MOUSE_RECORD System Variable
- SYSTEM.MOUSE_RECORD_OFFSET System Variable
- SYSTEM.MOUSE_X_POS System Variable
- SYSTEM.MOUSE_Y_POS System Variable
- SYSTEM.RECORD_STATUS System Variable
- SYSTEM.SUPPRESS_WORKING System Variable
- SYSTEM.TAB_NEW_PAGE System Variable
- SYSTEM.TAB_PREVIOUS_PAGE System Variable
- SYSTEM.TRIGGER_BLOCK System Variable
- SYSTEM.TRIGGER_ITEM System Variable
- SYSTEM.TRIGGER_MENUITEM System Variable
- SYSTEM.TRIGGER_NODE System Variable
- SYSTEM.TRIGGER_NODE_SELECTED System Variable
- SYSTEM.TRIGGER_RECORD System Variable

7. Triggers

- Triggers

List of Trigger Types

- Block Processing Triggers
- Forms Developer Triggers
- Interface Event Triggers
- Key Triggers
- Master-Detail Triggers

- Message-Handling Triggers
- Mouse Event Triggers
- Navigation Triggers
- On Triggers
- Pre Triggers
- Post Triggers
- Query-Time Triggers
- Stored Procedure Triggers
- Transactional Triggers
- Validation Triggers
- When Triggers

List of Triggers

- Delete-Procedure Trigger
- Function Key Triggers
- Insert-Procedure Trigger
- Key-Fn Trigger
- Key-Others Trigger
- Lock-Procedure Trigger
- On-Check-Delete-Master Trigger
- On-Check-Unique Trigger
- On-Clear-Details Trigger
- On-Close Trigger
- On-Column-Security Trigger
- On-Commit Trigger
- On-Count Trigger
- On-Delete Trigger
- On-Error Trigger
- On-Fetch Trigger
- On-Insert Trigger

- On-Lock Trigger
- On-Logon Trigger
- On-Logout Trigger
- On-Message Trigger
- On-Populate-Details Trigger
- On-Rollback Trigger
- On-Savepoint Trigger
- On-Select Trigger
- On-Sequence-Number Trigger
- On-Update Trigger
- Post-Block Trigger
- Post-Change Trigger
- Post-Database-Commit Trigger
- Post-Delete Trigger
- Post-Form Trigger
- Post-Forms-Commit Trigger
- Post-Insert Trigger
- Post-Logon Trigger
- Post-Logout Trigger
- Post-Query Trigger
- Post-Record Trigger
- Post-Select Trigger
- Post-Text-Item Trigger
- Post-Update Trigger
- Pre-Block Trigger
- Pre-Commit Trigger
- Pre-Delete Trigger
- Pre-Form Trigger
- Pre-Insert Trigger
- Pre-Logon Trigger

- Pre-Logout Trigger
- Pre-Popup-Menu Trigger
- Pre-Query Trigger
- Pre-Record Trigger
- Pre-Select Trigger
- Pre-Text-Item Trigger
- Pre-Update Trigger
- Query-Procedure Trigger
- Update-Procedure Trigger
- User-Named Trigger
- When-Button-Pressed Trigger
- When-Checkbox-Changed Trigger
- When-Clear-Block Trigger
- When-Create-Record Trigger
- When-Custom-Item-Event Trigger
- When-Database-Record Trigger
- When-Form-Navigate Trigger
- When-Image-Activated Trigger
- When-Image-Pressed Trigger
- When-List-Activated Trigger
- When-List-Changed Trigger
- When-Mouse-Click Trigger
- When-Mouse-DoubleClick Trigger
- When-Mouse-Down Trigger
- When-Mouse-Enter Trigger
- When-Mouse-Leave Trigger
- When-Mouse-Move Trigger
- When-Mouse-Up Trigger
- When-New-Block-Instance Trigger
- When-New-Form-Instance Trigger

- When-New-Item-Instance Trigger
- When-New-Record-Instance Trigger
- When-Radio-Changed Trigger
- When-Remove-Record Trigger
- When-Tab-Page-Changed
- When-Timer-Expired Trigger
- When-Tree-Node-Activated Trigger
- When-Tree-Node-Expanded Trigger
- When-Tree-Node-Selected Trigger
- When-Validate-Item Trigger
- When-Validate-Record Trigger
- When-Window-Activated Trigger
- When-Window-Closed Trigger
- When-Window-Deactivated Trigger
- When-Window-Resized Trigger

About Developer Components

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

Forms Developer

Forms Developer is the design component you use to create, compile, and run Forms Developer applications. Using Forms Developer, you can create three types of modules: forms, menus, and libraries.

Forms Runtime

Form operators use Forms Services to run the completed application. As an application designer, you can also use Forms Services to test and debug forms during the design stage. Forms Services reads the machine-readable file created by the Form Compiler, and executes the form.

Form Compiler

Most often, you use the Form Compiler to create a machine-readable file that Forms Services can execute.

Form Compiler also allows you to convert various representations of a form. Using Form Compiler, you can:

- Convert files between binary, text, and database module storage formats.
- Recompile application modules when porting to different platforms.
- Upgrade applications created with previous versions of Forms Developer, SQL*Forms, and SQL*Menu.

About the Form Compiler Window

The Form Compiler displays compilation error messages during batch compilation and allows you to halt and resume compilation as well as navigate to error source.

Status Line

Specifies the following information while program units are being compiled:

- name and type of the program unit currently being compiled
- [INTERRUPTED] when you click Interrupt
- percentage of program units that have been compiled

Compilation Errors list box

A scrollable, mouse-sensitive area that displays errors encountered during compilation of the selected program units.

Interrupt

Temporarily stops compilation—for example, when too many errors are being generated.

Resume

Resumes compilation after an interrupt.

OK

Closes the dialog after compilation has completed.

Cancel

Cancels the current operation and closes the dialog.

Goto Error

Presents the PL/SQL containing the program unit source at the location of the first error.

If you click this button while in an interrupted state of compilation (as a result of clicking Interrupt) an alert appears warning you that going to the source of an error will abandon the batch compilation.

If compilation ends with no errors detected, the message Compilation completed successfully is displayed on the status line. If compilation ends with errors detected, the message Compilation completed with errors is displayed. You can click on an error in the Compilation Errors pane to display the PL/SQL editor and the program unit where the error is located.

If you click an error in the Compilation Errors pane while in an interrupted state of compilation (as a result of clicking Interrupt) an alert appears warning you that going to the source of an error will abandon the batch compilation. You can click Goto Error to present the PL/SQL editor and the program unit that contains the first error in the Compilation Errors list box.

About the PL/SQL Compilation Messages Window

The PL/SQL Compilation Messages window displays error messages, if any, generated as a result of compilation. Clicking on an error message highlights it, scrolls the Source pane to the offending source statement, and positions the text cursor at the location of the error.

By default, the Compilation messages pane appears only when there are compilation error messages generated. However, you can display the Compilation messages pane at any time by moving the split bar at the bottom right corner of the editor.

Application Server URL

Description

The URL to the FormsServlet. This is used to run the Form in servlet mode. The Application Server URL can point to a Forms Servlet on any machine, not necessarily the local machine. The Application Server URL can contain parameters also which can be used while running the Form.

For example, if you wish to run the Form in Internet Explorer native mode, you can specify the Application Server URL as:

http://formsddr-sun.us.oracle.com:8888/forms90/f90servlet?config=ie50native

All other runtime parameters are appended to the Application Server URL. If this preference is blank, a default URL is generated when the user runs the Form. The default URL is of the form:

http://localmachine:port/forms90/f90servlet.

where localmachine is your local machine and port can be specified in the preferences file (cauprefs.ora) as forms.port = xxx. If this preference is not specified, the default port 80 is used.

Build Before Running Option

Description

Determines whether Forms Developer automatically compiles the active module before running a form. When Build Before Running is checked, Forms Developer does the following when you issue the **Program | Run Form** command to run a specified form:

- builds 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 box.

This option lets you avoid issuing separate Compile and Run commands each time you modify and then run a form. However, this option does not save the module. You must select **File | Save** to save the module, or check the Save Before Building preference.

Also, when the Build Before Running option is checked, Forms Developer does not automatically compile any menu or library modules attached to that form. You must compile menu and library modules separately before running a form that references them.

Default: Off

Forms Developer Options

Color Mode

Color Palette

Build Before Run

Help

Module_Type

Printer

Save Before Generate

Suppress Hints

Use System Editor

User Preferences

Builder Preferences

Options

A

Add_Triggers (Form Compiler)

Application Server URL

Array Processing

B

Batch (Form Compiler)

Buffer_Records in File

Build (Forms Compiler)

Build Before Running

C

Color Mode

Color Palette

Compile_All

CRT_File (Form Compiler)

D

Data Block Wizard Welcome Page

Debug (Forms Runtime)

Debug_Messages (Forms Runtime)

E

F

G

H

Help (Forms Developer)

Help (Form Compiler)

I

J

K

L

Layout Wizard Welcome Page

Logon (Form Compiler)

LOV Welcome Page

M

Module_Type (Forms Developer)

Module_Type (Form Compiler)

N

Nofail (Form Compiler)

O

P

Parse (Form Compiler)

Printer

Q

Query_Only (Forms Runtime)

Quiet (Forms Runtime)

R

S

Save Before Building

Script (Form Compiler)

Strip_Source (Form Compiler)

Subclassing Path

Suppress Hints

T

U

Upgrade (Form Compiler)

Upgrade_Roles (Form Compiler)

Use System Editor

V

Version (Form Compiler)

W

Welcome Dialog

Wizard Welcome Pages

X

Y

Z

Color Mode

Determines how a Forms Developer color palette will be loaded on your system. Each time you load, open, or create a form, Forms Developer loads the Forms Developer color palette into your current system color table. Because this system color table can handle only a limited number of colors at once, Forms Developer 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 Forms Developer color palette.

Color Mode options:

Editable Select Editable mode only when you want to change the Forms Developer 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 Forms Developer color palette:

1. Change Color Mode to Editable and save your options (**Edit | Preferences | General tab | Color Mode**).
2. Restart Forms Developer.
3. Use **Format | Layout Options | Color Palette** to make changes to the color palette (only when the Layout Editor is open).
4. Use **File | Export | Color Palette** to save the Forms Developer color palette to a file (only when the Layout Editor is open).
5. Change your options to use the new color file (**Edit | Preferences | General tab | Color Palette**).
6. Change **Color Mode** back to `Read Only - Shared` and save your options.
7. Restart Forms Developer.

Read Only-Shared

In Read Only - Shared mode, Forms Developer maps duplicate colors to the same entry in the current system color table before appending new entries from your Forms Developer color palette. Read Only - Shared will help you avoid the color flicker that can result when you switch between Forms Developer 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 Graphics, and is not relevant for Forms Developer. In Forms Developer, it maps to Read Only - Shared.

Default Read Only - Shared

Color Palette

Description

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

This dialog box contains:

Current Color	This text box contains the name of the currently selected color.
Edit	Click **Edit** to change the currently selected color to a different color.
Palette Name	The name of the palette imported into the current session. Blank if none was imported.
Rename	Click to change the name of a color after typing the new name into the Current Color text box.
Select Color to Edit	Displays all the colors currently available in the Layout Editor. Select a color by clicking on its square.

Compiler Options

Add_Triggers

Batch

Build

Compile_All

CRT_File

Help

Logon

Module_Type

Nofail

Options_Screen

Parse

Script

Strip_Source

Upgrade

Upgrade_Roles

Version

Widen_Fields

Default Color Palette Colors

The available default colors for the Color Palette include:

black	yellow
white	darkyellow
green	magenta
darkgreen	darkmagenta
gray96	gray48
gray92	gray44
gray88	gray40
gray84	gray36
r0g0b0	r0g75b0
r25g0b0	r25g75b0
r50g0b0	r50g75b0
r75g0b0	r75g75b0
r88g0b0	r88g75b0
r100g0b0	r100g75b0
r0g0b50	r0g75b50
r25g0b50	r25g75b50
r50g0b50	r50g75b50
r75g0b50	r75g75b50
r88g0b50	r88g75b50
r100g0b50	r100g75b50
r0g0b75	r0g75b75
r25g0b75	r25g75b75
r50g0b75	r50g75b75
r75g0b75	r75g75b75
r88g0b75	r88g75b75
r100g0b75	r100g75b75
r0g0b88	r0g75b88
r25g0b88	r25g75b88
r50g0b88	r50g75b88
r75g0b88	r75g75b88
r88g0b88	r88g75b88
r100g0b88	r100g75b88
r0g0b100	r0g75b100
r25g0b100	r25g75b100
r50g0b100	r50g75b100
r75g0b100	r75g75b100
r88g0b100	r88g75b100
r100g0b100	r100g75b100
gray	custom1
darkgray	custom2
cyan	custom3
darkcyan	custom4
gray80	gray32
gray76	gray28
gray72	gray24

gray68	gray20
r0g25b0	r0g88b0
r25g25b0	r25g88b0
r50g25b0	r50g88b0
r75g25b0	r75g88b0
r88g25b0	r88g88b0
r100g25b0	r100g88b0
r0g25b50	r0g88b50
r25g25b50	r25g88b50
r50g25b50	r50g88b50
r75g25b50	r75g88b50
r88g25b50	r88g88b50
r100g25b50	r100g88b50
r0g25b75	r0g88b75
r25g25b75	r25g88b75
r50g25b75	r50g88b75
r75g25b75	r75g88b75
r88g25b75	r88g88b75
r100g25b75	r100g88b75
r0g25b88	r0g88b88
r25g25b88	r25g88b88
r50g25b88	r50g88b88
r75g25b88	r75g88b88
r88g25b88	r88g88b88
r100g25b88	r100g88b88
r0g25b100	r0g88b100
r25g25b100	r25g88b100
r50g25b100	r50g88b100
r75g25b100	r75g88b100
r88g25b100	r88g88b100
r100g25b100	r100g88b100
red	custom5
darkred	custom6
blue	custom7
darkblue	custom8
gray64	gray16
gray60	gray12
gray56	gray8
gray52	gray4
r0g50b0	r0g100b0
r25g50b0	r25g100b0
r50g50b0	r50g100b0
r75g50b0	r75g100b0
r88g50b0	r88g100b0
r100g50b0	r100g100b0
r0g50b50	r0g100b50
r25g50b50	r25g100b50
r50g50b50	r50g100b50
r75g50b50	r75g100b50
r88g50b50	r88g100b50
r100g50b50	r100g100b50

r0g50b75	r0g100b75
r25g50b75	r25g100b75
r50g50b75	r50g100b75
r75g50b75	r75g100b75
r88g50b75	r88g100b75
r100g50b75	r100g100b75
r0g50b88	r0g100b88
r25g50b88	r25g100b88
r50g50b88	r50g100b88
r75g50b88	r75g100b88
r88g50b88	r88g100b88
r100g50b88	r100g100b88
r0g50b100	r0g100b100
r25g50b100	r25g100b100
r50g50b100	r50g100b100
r75g50b100	r75g100b100
r88g50b100	r88g100b100
r100g50b100	r100g100b100

Help (Forms Developer)

Description

Invokes the Forms Developer help screen.

Module: All

Default NO

Help (Form Compiler) Example

```
ifbld90 help=YES
```

Module Type (Forms Developer)

Description

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

Module: All

Default FORM

Module_Type (Forms Developer) Example

```
ifbld90 module=orders userid=scott/tiger module_type=menu
```

Options

Runtime options

Compiler options

Developer options

Runtime Options

Array

Buffer_Records

Debug

Debug_Messages

Query_Only

Quiet

Save Before Building

Determines whether Forms Developer saves the current module automatically before it is built either when you choose **Program | Compile Module** or when the Form is built before running when the Build Before Running preference is checked.

Default Off

Setting Forms Developer Preferences

Forms Developer preferences specify Forms Developer session default behavior. Choose **Edit | Preferences** in Forms Developer to invoke the **Preferences** dialog. To set options, click on the check boxes or fill in file names for the options you choose.

The **Preferences** dialog includes both Forms Developer and Forms Runtime preferences.

Forms Developer Preferences

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

- Save Before Building
- Build Before Running
- 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 Forms Developer:

- Buffer Records in File
- Debug Mode
- Array Processing
- 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 Forms Developer command line:

- Module_Type
- Module_Access
- Help

Setting Forms Developer Options Example

```
ifbld90 module=orders userid=scott/tiger module_type=menu
```

Starting Forms Developer Components from the Command Line

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

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

where:

component_name Specifies the Forms Developer component you want to use:

- Forms Developer - ifbld90
- Forms Runtime - ifrun90
- Form Compiler - ifcmp90

Starting Forms Developer Components Examples

ifrun90 Starts the Forms Runtime 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*.

module_name Specifies the module you want to load: a form, menu, or library name. If you omit the module name, Forms Developer 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...

ifrun90 custform scott/tiger

Note: The examples assume that you're running Forms Developer on Microsoft Windows, with no calls to the user exit interface, so the Forms Runtime component name is shown as "ifrun90." You should substitute the correct value of *component_name* for your platform and application.

Keyword Usage

There are three categories of parameters in Forms Developer:

- 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:

```
ifrun90 custform scott/tiger
```

Invalid Example:

```
ifrun90 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:

```
ifrun90 module=custform userid=scott/tiger
ifrun90 userid=scott/tiger
ifrun90
```

If you indicate only the module name, Forms Developer 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 Forms Runtime Options

Setting Form Compiler Options

Setting Forms Developer Options

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:

```
ifrun90 custform scott/tiger
ifrun90 custform scott/tiger debug=yes
```

Valid Examples:

```
ifrun90 custform scott/tiger
ifrun90 custform scott/tiger 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 Forms Runtime 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 Forms Developer.

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

Example

```
ifrun90 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

```
ifrun90 help=yes
```

Subclassing Path

Description

Specifies whether to save the path of an original object with the subclassed object.

Specify one of the following preferences for saving the path of original objects with subclassed objects:

Remove

> The path will be removed from the filename of the original object referenced in the subclassed object.

Keep

> The subclassed object will reference the original object according to the full path.

Ask

> Each time you subclass an object Forms Developer will display a dialog box prompting whether to remove or keep the path.

Default ASK

Notes

A subclassed object inherits property values from the original object and references the original object by the file name of the form in which it is saved. The full path name of the form may be saved with the subclassed object or only the filename. When the form containing the subclassed object is opened, Forms Developer looks for the file specified for the subclassed object. If the filename is specified without the path, Forms Developer looks in the current directory in which Forms Developer was started.

Suppress Hints

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

Default Off

Use System Editor

Determines which editor Forms Developer uses when you invoke an editor from a multi-line text item. When Use System Editor is unchecked, Forms Developer displays the default editor. When Use System Editor is checked, Forms Developer displays the default system editor defined on your system.

Note: If Use System Editor is checked 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 Forms Developer documentation for your operating system.

Default Off

User Preference File

Although the Preferences dialog box is the most convenient way to set preferences, you can also set them directly in the preference file (usually called `CAUPREFS.ORA`).

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

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

```
FORMS.QUIET=ON
```

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

If you start Forms Developer 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, Forms Developer ignores that line when it reads the file.

Syntax for Options

To preset a Forms Developer or Forms Runtime 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 `CAUPREFS.ORA` file.

Welcome Dialog

Description

Determines whether the welcome screen is displayed when Forms Developer is started.

When checked, the welcome screen will be displayed when Forms Developer is started. When unchecked, Forms Developer starts with a new, blank module called module1.

Default ON

Welcome Pages

Description

Determines whether the welcome page for a specific wizard is displayed when the wizard is invoked.

When checked, the welcome page will be displayed when the wizard is started. When unchecked, the wizard does not display the welcome page.

Applies to

- Data Block Wizard
- LOV Wizard
- Layout Wizard

Default ON

Add Triggers (Form Compiler)

Description

Indicates whether to add key-up and key-down triggers when upgrading from Forms 2.0 or 2.3 to 9.0 wherever KEY-PRVREC and KEY-NXTREC triggers existed.

Module: Form

Default NO

Add_Triggers (Form Compiler) Examples

```
ifcmp90 module=myform userid=scott/tiger upgrade=yes version=23 add_triggers=YES
```

Batch (Form Compiler)

Description

Suppresses interactive messages; use when performing a batch generation.

Module: Form

Default NO

Batch (Form Compiler) Examples

```
ifcmp90 module=myform userid=scott/tiger batch=YES
```

Build (Form Compiler)

Description

Use the Build option in conjunction with Upgrade. Forms Developer creates two files when you specify upgrade=YES and omit build, thus accepting the default of build=YES:

- an upgraded binary design module (.FMB or .MMB file)
- an upgraded Forms Runtime executable module (.FMX or .MMX file)

If you do *not* want to automatically create the Forms Runtime module, specify build=NO.

Module: Form, Menu

Default YES

Build (Form Compiler) Examples

```
ifcmp90 module=myform userid=scott/tiger upgrade=YES build=NO
```

Compile All (Form Compiler)

Description

Compiles the program units within the specified module.

Module: Form, Menu, Library

Default NO

Compile_All (Form Compiler) Example

```
ifcmp90 module=myform userid=scott/tiger compile_all=YES
```

CRT File (Form Compiler)

Description

Indicates CRT file to use when upgrading from SQL*Forms Version 2.0 or 2.3.

Module: Form

CRT_File (Form Compiler) Example

```
ifcmp90 module=myform userid=scott/tiger upgrade=yes version=20 crt_file=myfile.crt
```

Help (Form Compiler)

Description

Invokes the Forms Developer help screen.

Module: All

Default No

Help (Form Compiler) Example

```
ifcmp90 help=YES
```

Logon (Form Compiler)

Description

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

Module: Form

Default YES

Logon (Form Compiler) Examples

```
ifcmp90 module=myform userid=scott/tiger logon=NO
```

Module Type (Form Compiler)

Description

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

Module: All

Default FORM

Module_Type (Form Compiler) Examples

```
ifcmp90 module=orders userid=scott/tiger module_type=menu
```

Nofail (Form Compiler)

Description

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

Module: Form

Default NO

Nofail (Form Compiler) Examples

```
ifcmp90 module=myform userid=scott/tiger upgrade=yes version=20 nofail=YES
```

Options Screen (Form Compiler)

Description

Invokes the Options window.

This parameter applies to GUI displays only.

Module: All

Default NO

Options_Screen (Form Compiler) Examples

```
ifcmp90 module=myform userid=scott/tiger options_screen=YES
```

Parse (Form Compiler)

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 within Forms Developer by using the Convert command.

Module: All

Default NO

Parse (Form Compiler) Examples

```
ifcmp90 module=myform parse=YES
```

Script (Form Compiler)

Description

Converts a binary file format (.FMB, .MMB, or .PLL) to a text format (.FMT, .MMT, or .PLD).

This operation can also be done within Forms Developer by using the Convert command.

Module: All

Default NO

Script (Form Compiler) Examples

```
ifcmp90 module=myform script=YES
```

Strip Source (Form Compiler)

Description

Removes the source code from the library file and generates a library file that only contains pcode. The resulting file can be used for final deployment, but cannot be subsequently edited in Forms Developer.

Module: Library

Default NO

Strip_Source (Form Compiler) Examples

```
ifcmp90 module=old_lib.pll userid=scott/tiger strip_source=YES output_file=new_lib.pll
```

Upgrade (Form Compiler)

Description

Upgrades modules from SQL*Forms 2.0, 2.3, or 3.0 to Forms Developer 4.5, or from SQL*Menu 5.0 to an Forms Developer 9.0 menu module:

- To upgrade from SQL*Forms 3.0 or SQL*Menu 5.0 to Forms Developer 9.0, 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.

Module: Form, Menu

Default NO

Upgrade (Form Compiler) Example

```
ifcmp90 module=myform userid=scott/tiger upgrade=YES
```

Upgrade Roles (Form Compiler)

Description

Upgrades SQL*Menu 5.0 table privileges to Oracle9*i* 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.

Module: none

Default NO

Upgrade_Roles (Form Compiler) Example

```
ifcmp90 userid=system/manager upgrade_roles=YES
```

Version (Form Compiler)

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.

Module: Form

Default **version=30**

Version (Form Compiler) Examples

ifcmp90 module=myform userid=scott/tiger upgrade=yes version=23

Widen Fields (Form Compiler)

Description

Use the `Widen_Fields` option in conjunction with Upgrade. When upgrading to Version 9.0, 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.

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.

Module: Form

Default NO

Widen_Fields (Form Compiler) Examples

ifcmp90 module=myform userid=scott/tiger upgrade=yes widen_fields=YES

Forms Runtime Options

Forms Runtime options specify Forms Developer default behavior during a Forms Runtime session. You can set Forms Runtime options in several ways:

- in the URL
- in the HTML files
- in formsweb.cfg

In addition, you can set Forms Runtime options to specify the defaults for forms you run from Forms Developer in the **Preferences** dialog. To display the **Preferences** dialog, select **Edit | Preferences.**

Note: Forms Runtime preferences set in Forms Developer apply only to forms run from within Forms Developer.

Options may also be set for the Web Previewer in the serverargs parameter of a base HTML file. You can specify this HTML filename in the **Runtime** tab of the **Preferences** dialog, or on the command line. For details on creating a base HTML file see the section on "Configuring the Forms Servlet" in Deploying Forms on the Web.

The following chart lists the Forms Runtime options from the **Options** window and their corresponding keyword parameters.

Option Name	**Keyword Parameter**	**Default**
Array processing	Array	Yes
Buffer records to temporary file	Buffer_Records	No
Run in quiet mode	Quiet	No
Run in query only mode	Query_Only	No
Use SDI mode	USESDI	No

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

```
http://myserver/forms90/f90servlet?form=MODULE1.fmx&debug_messages=YES
```

Array (Forms Runtime)

Description

Use array processing during a Forms Runtime session.

When you suppress array processing, Forms requests that the database only returns a single row of query results at a time from server to client. Similarly, Forms requests that the database only send a single row at a time from the client to the server for an INSERT, UPDATE, or DELETE when array processing is suppressed.

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 Processing

Default YES

Array (Forms Runtime) Examples

http://myserver/forms90/f90servlet?form=MODULE1.fmx&array=YES

Buffer Records (Forms Runtime)

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, Forms Developer buffers these additional records to a temporary file on disk.

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

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

Option Name Buffer Records to Temporary File

Default NO

Buffer_Records (Forms Runtime) Example

http://myserver/forms90/f90servlet?form=MODULE1.fmx&buffer_records=yes

Debug (Forms Runtime)

Description

Invokes the debug mode for the Forms Runtime 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 Forms Developer menu.

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

```
ifdbg90 module=myform userid=scott/tiger debug=YES
```

Option Name Run in Debug Mode

Default No

Debug (Forms Runtime) Examples

```
ifdbg90 module=myform userid=scott/tiger debug=YES
```

Debug Messages (Forms Runtime)

Description

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

Default NO

Debug_Messages (Forms Runtime) Example

```
ifrun90 module=myform userid=scott/tiger debug_messages=YES
```

Query Only (Forms Runtime)

Description

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

Preference Name Query Only Mode

Default NO

Query_Only (Forms Runtime) Example

http://myserver/forms90/f90servlet?form=MODULE1.fmx&query_only=YES

Quiet (Forms Runtime)

Description

Invokes the quiet mode for the Forms Runtime 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

Quiet (Forms Runtime) Examples

http://myserver/forms90/f90servlet?form=MODULE1.fmx&quiet=YES

USESDI (Web Forms Runtime)

Description

Use single document interface (SDI) system of window management during a Web Forms Runtime session.

There is no multiple document interface (MDI) root window. MDI toolbars exist in parent windows and menus will be attached to each window.

Calls to the FORMS_MDI_WINDOW constant returns NULL as the MDI window handle when usesdi=YES.

Option Name None

Default YES

USESDI (Forms Runtime) Examples

http://myserver/forms90/f90servlet?form=MODULE1.fmx&usesdi=YES

About Built-in 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.

Related topic

Forms Developer Built-ins

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 Forms Developer 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 built-in descriptions include a heading, Built-In Type, that indicates if the built-in is restricted or unrestricted.

Built-in 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.

```
SET_FORM_PROPERTY('my_form', VALIDATION_UNIT, BLOCK_SCOPE);
```

Built-in Packages

Alert Built-ins

Application Built-ins

Block Built-ins

Canvas Built-ins

Chart Built-ins

Form Built-ins

Item Built-ins

Menu Built-ins

Message Built-ins

Multiple Form Built-ins

Parameter List Built-ins

Query Built-ins

Record Built-ins

Relation Built-ins

Report Built-ins

Tab Page Built-ins

Transactional Built-ins

Web Built-ins

Window Built-ins

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.

Individual built-in descriptions Restrictions

Indicates any restrictions.

Individual built-in descriptions Examples

Provides an actual example that can be used in conjunction with the syntax to develop a realistic call to the built-in.

Built-in syntax

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;
```

Built-in Named Parameters

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);
```

Related topic

About Built-in Code Examples

Built-in 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 Forms Developer. 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.

Built-in Form Coordinate Units

Many built-in subprograms allow you to specify size and position coordinates, using properties such as:

- HEIGHT
- WIDTH
- DISPLAY_POSITION
- VIEWPORT_X_POS
- VIEWPORT_Y_POS
- VIEW_SIZE
- VIEWPORT_X_POS_ON_CANVAS
- VIEWPORT_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 Forms Developer. Use the Coordinate System form module property 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 Forms Developer 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 Forms Developer 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.

Built-in 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.

Alert Built-ins

FIND_ALERT

ID_NULL

SET_ALERT_BUTTON_PROPERTY

SET_ALERT_PROPERTY

SHOW_ALERT

Application Built-ins

DO_KEY

GET_APPLICATION_PROPERTY

HOST

PAUSE

SET_APPLICATION_PROPERTY

USER_EXIT

Block Built-ins

CLEAR_BLOCK

FIND_BLOCK

GET_BLOCK_PROPERTY

GO_BLOCK

ID_NULL

PREVIOUS_BLOCK

SET_BLOCK_PROPERTY

Canvas Built-ins

FIND_CANVAS

FIND_VIEW

GET_CANVAS_PROPERTY

GET_VIEW_PROPERTY

HIDE_VIEW

ID_NULL

PRINT

SCROLL_VIEW

SET_CANVAS_PROPERTY

SET_VIEW_PROPERTY

SHOW_VIEW

Chart Built-ins

UPDATE_CHART

Forms Developer Built-ins

Built-in packages

A

ABORT_QUERY

ADD_GROUP_COLUMN

ADD_GROUP_ROW

ADD_LIST_ELEMENT

ADD_PARAMETER

B

BELL

C

CALL_FORM

CALL_INPUT

CANCEL_REPORT_OBJECT

CHECK_RECORD_UNIQUENESS

CHECKBOX_CHECKED

CLEAR_BLOCK

CLEAR_EOL

CLEAR_FORM

CLEAR_ITEM

CLEAR_LIST

CLEAR_MESSAGE

CLEAR_RECORD

CLOSE_FORM

COMMIT_FORM

CONVERT_OTHER_VALUE

COPY

COPY_REGION

COPY_REPORT_OUTPUT

COUNT_QUERY

CREATE_GROUP

CREATE_GROUP_FROM_QUERY

CREATE_PARAMETER_LIST

CREATE_QUERIED_RECORD

CREATE_RECORD

CREATE_TIMER

CREATE_VAR

CUT_REGION

D

DBMS_ERROR_CODE

DBMS_ERROR_TEXT

DEFAULT_VALUE

DELETE_GROUP

DELETE_GROUP_ROW

DELETE_LIST_ELEMENT

DELETE_PARAMETER

DELETE_RECORD

DELETE_TIMER

DESTROY_PARAMETER_LIST

DISPLAY_ERROR

DISPLAY_ITEM

DOWN

DO_KEY

DUMMY_REFERENCE

DUPLICATE_ITEM

DUPLICATE_RECORD

E

EDIT_TEXTITEM

ENFORCE_COLUMN_SECURITY

ENTER

ENTER_QUERY

ERASE

ERROR_CODE

ERROR_TEXT

ERROR_TYPE

EXECUTE_QUERY

EXECUTE_TRIGGER

EXIT_FORM

F

FETCH_RECORDS

FIND_ALERT

FIND_BLOCK

FIND_CANVAS

FIND_COLUMN

FIND_EDITOR

FIND_FORM

FIND_GROUP

FIND_ITEM

FIND_LOV

FIND_MENU_ITEM

FIND_RELATION

FIND_REPORT_OBJECT

FIND_TAB_PAGE

FIND_TIMER

FIND_VIEW

FIND_WINDOW

FIRST_RECORD

FORM_FAILURE

FORM_FATAL

FORM_SUCCESS

FORMS_DDL

G

GENERATE_SEQUENCE_NUMBER

GET_APPLICATION_PROPERTY

GET_BLOCK_PROPERTY

GET_CANVAS_PROPERTY

GET_FILE_NAME

GET_FORM_PROPERTY

GET_GROUP_CHAR_CELL

GET_GROUP_DATE_CELL

GET_GROUP_NUMBER_CELL

GET_GROUP_RECORD_NUMBER

GET_GROUP_ROW_COUNT

GET_GROUP_SELECTION

GET_GROUP_SELECTION_COUNT

GET_ITEM_INSTANCE_PROPERTY

GET_ITEM_PROPERTY

GET_LIST_ELEMENT_COUNT

GET_LIST_ELEMENT_LABEL

GET_LIST_ELEMENT_VALUE

GET_LOV_PROPERTY

GET_MENU_ITEM_PROPERTY

GET_MESSAGE

GET_PARAMETER_ATTR

GET_PARAMETER_LIST

GET_RADIO_BUTTON_PROPERTY

GET_RECORD_PROPERTY

GET_RELATION_PROPERTY

GET_REPORT_OBJECT_PROPERTY

GET_TAB_PAGE_PROPERTY

GET_VAR_BOUNDS

GET_VAR_DIMS

GET_VAR_TYPE

GET_VIEW_PROPERTY

GET_WINDOW_PROPERTY

GO_BLOCK

GO_FORM

GO_ITEM

GO_RECORD

H

HELP

HIDE_VIEW

HIDE_WINDOW

HOST

I

ID_NULL

IMAGE_SCROLL

IMAGE_ZOOM

INSERT_RECORD

ISSUE_ROLLBACK

ISSUE_SAVEPOINT

J

K

L

LAST_RECORD

LIST_VALUES

LOCK_RECORD

LOGON

LOGON_SCREEN

LOGOUT

M

MESSAGE

MESSAGE_CODE

MESSAGE_TEXT

MESSAGE_TYPE

MOVE_WINDOW

N

NAME_IN

NEW_FORM

NEXT_BLOCK

NEXT_ITEM

NEXT_FORM

NEXT_KEY

NEXT_RECORD

NEXT_SET

O

OPEN_FORM

P

PASTE_REGION

PAUSE

POPULATE_GROUP

POPULATE_GROUP_WITH_QUERY

POPULATE_LIST

POST

PREVIOUS_BLOCK

PREVIOUS_FORM

PREVIOUS_ITEM

PREVIOUS_RECORD

PRINT

Q

R

READ_IMAGE_FILE

RECALCULATE

REDISPLAY

REPLACE_CONTENT_VIEW

REPLACE_MENU

REPORT_OBJECT_STATUS

RESET_GROUP_SELECTION

RESIZE_WINDOW

RETRIEVE_LIST

RUN_PRODUCT

RUN_REPORT_OBJECT

S

SCROLL_DOWN

SCROLL_UP

SCROLL_VIEW

SELECT_ALL

SELECT_RECORDS

SET_ALERT_BUTTON_PROPERTY

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_ITEM_INSTANCE_PROPERTY

SET_ITEM_PROPERTY

SET_LOV_COLUMN_PROPERTY

SET_LOV_PROPERTY

SET_MENU_ITEM_PROPERTY

SET_PARAMETER_ATTR

SET_RADIO_BUTTON_PROPERTY

SET_RECORD_PROPERTY

SET_RELATION_PROPERTY

SET_REPORT_OBJECT_PROPERTY

SET_TAB_PAGE_PROPERTY

SET_TIMER

SET_VIEW_PROPERTY

SET_WINDOW_PROPERTY

SHOW_ALERT

SHOW_EDITOR

SHOW_KEYS

SHOW_LOV

SHOW_VIEW

SHOW_WINDOW

SYNCHRONIZE

T

U

UNSET_GROUP_SELECTION

UP

UPDATE_CHART

UPDATE_RECORD

USER_EXIT

V

VALIDATE

W

WEB.SHOW_DOCUMENT

WRITE_IMAGE_FILE

X

Y

Z

Item Built-ins

CHECKBOX_CHECKED

CLEAR_EOL

CLEAR_ITEM

CONVERT_OTHER_VALUE

COPY

COPY_REGION

CUT_REGION

DEFAULT_VALUE

DISPLAY_ITEM

DUMMY_REFERENCE

DUPLICATE_ITEM

EDIT_TEXTITEM

FIND_ITEM

GET_FILE_NAME

GET_ITEM_INSTANCE_PROPERTY

GET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

GO_ITEM

ID_NULL

IMAGE_SCROLL

IMAGE_ZOOM

NAME_IN

NEXT_ITEM

NEXT_KEY

PASTE_REGION

PREVIOUS_ITEM

READ_IMAGE_FILE

RECALCULATE

SELECT_ALL

SET_ITEM_INSTANCE_PROPERTY

SET_ITEM_PROPERTY

SET_RADIO_BUTTON_PROPERTY

WRITE_IMAGE_FILE

Menu Built-ins

FIND_MENU_ITEM

GET_MENU_ITEM_PROPERTY

REPLACE_MENU

SET_MENU_ITEM_PROPERTY

MESSAGE Built-in

Description

Displays specified text on the message line.

Syntax

PROCEDURE MESSAGE
(*message_string* VARCHAR2*,*
user_response NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

message_string

Specify a character string enclosed in single quotes or a variable of VARCHAR2 data type.

user_response

Specifies one of the following constants:

ACKNOWLEDGE Specifies that Forms Developer 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 Forms Developer 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.

MESSAGE Restrictions

The 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.

MESSAGE Examples

```
/*

** 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);
/*
** Long-running update statement goes here
*/

SYNCHRONIZE;
Message('Working... (30%)', NO_ACKNOWLEDGE);
/*
** Another long-running update statement goes here
*/
Message('Working... (80%)', NO_ACKNOWLEDGE);
/*
** Last long-running statement here
*/
Message('Done...', NO_ACKNOWLEDGE);
END;
```

Multiple Form Built-ins

CLOSE_FORM

GO_FORM

NEW_FORM

NEXT_FORM

OPEN_FORM

PREVIOUS_FORM

Parameter List Built-ins

ADD_PARAMETER

CREATE_PARAMETER_LIST

DELETE_PARAMETER

DESTROY_PARAMETER_LIST

GET_PARAMETER_ATTR

ID_NULL

RUN_PRODUCT

SET_PARAMETER_ATTR

Query Built-ins

ABORT_QUERY

COUNT_QUERY

ENTER_QUERY

EXECUTE_QUERY

Record Built-ins

CHECK_RECORD_UNIQUENESS

CLEAR_RECORD

CREATE_QUERIED_RECORD

CREATE_RECORD

DELETE_RECORD

DOWN

DUPLICATE_RECORD

FIRST_RECORD

GENERATE_SEQUENCE_NUMBER

GET_RECORD_PROPERTY

GO_RECORD

INSERT_RECORD

LAST_RECORD

LOCK_RECORD

NEXT_RECORD

NEXT_SET

PREVIOUS_RECORD

SCROLL_DOWN

SCROLL_UP

SELECT_RECORDS

SET_RECORD_PROPERTY

UP

UPDATE_RECORD

Relation Built-ins

FIND_RELATION

GET_RELATION_PROPERTY

ID_NULL

SET_RELATION_PROPERTY

Report Built-ins

CANCEL_REPORT_OBJECT

COPY_REPORT_OUTPUT

FIND_REPORT_OBJECT

GET_REPORT_OBJECT_PROPERTY

RUN_REPORT_OBJECT

SET_REPORT_OBJECT_PROPERTY

Tab Page Built-ins

FIND_TAB_PAGE

GET_TAB_PAGE_PROPERTY

SET_TAB_PAGE_PROPERTY

Transaction Built-ins

CHECK_RECORD_UNIQUENESS

DELETE_RECORD

ENFORCE_COLUMN_SECURITY

FETCH_RECORDS

FORMS_DDL

GENERATE_SEQUENCE_NUMBER

INSERT_RECORD

ISSUE_ROLLBACK

ISSUE_SAVEPOINT

LOGON

LOGON_SCREEN

LOGOUT

SELECT_RECORDS

UPDATE_RECORD

Web Built-ins

WEB.SHOW_DOCUMENT

Window Built-ins

FIND_WINDOW

GET_WINDOW_PROPERTY

HIDE_WINDOW

ID_NULL

MOVE_WINDOW

REPLACE_CONTENT_VIEW

RESIZE_WINDOW

SET_WINDOW_PROPERTY

SHOW_WINDOW

ABORT_QUERY Built-in

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.

Syntax

PROCEDURE ABORT_QUERY;

Built-in Type unrestricted procedure

Enter Query Mode yes

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.

ABORT_QUERY 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 Forms Developer 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 Built-in

Description

Adds a column of the specified type to the given record group.

Syntax

FUNCTION ADD_GROUP_COLUMN
(*recordgroup_id* RecordGroup,
groupcolumn_name VARCHAR2,
column_type NUMBER);

FUNCTION ADD_GROUP_COLUMN
(*recordgroup_name* VARCHAR2,
groupcolumn_name VARCHAR2,
column_type NUMBER);

FUNCTION ADD_GROUP_COLUMN
(*recordgroup_id,* RecordGroup
groupcolumn_name VARCHAR2,
column_type NUMBER,
column_width NUMBER);

FUNCTION ADD_GROUP_COLUMN
(*recordgroup_name* VARCHAR2,
groupcolumn_name VARCHAR2,
column_type NUMBER,
column_width NUMBER);

Built-in Type unrestricted function

Enter Query Mode yes

Returns GroupColumn

Parameters

recordgroup_id

The unique ID that Forms Developer 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 VARCHAR2.

groupcolumn_name

Specifies the name of the column. The data type of the column name is VARCHAR2.

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 VARCHAR2 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.

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.

ADD_GROUP_COLUMN Restrictions

- 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.

ADD_GROUP_COLUMN Examples

```
/* ** 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
*/
```

```
IF Id_Null(rg_id) THEN
rg_id := Create_Group( rg_name );
gc_id := Add_Group_Column(rg_id, 'NumCol',NUMBER_COLUMN);
gc_id := Add_Group_Column(rg_id, 'CharCol',CHAR_COLUMN,15);
END IF;
END;
```

ADD_GROUP_ROW Built-in

Description

Adds a row to the given record group.

Syntax

PROCEDURE ADD_GROUP_ROW
recordgroup_id RecordGroup,
row_number NUMBER);

PROCEDURE ADD_GROUP_ROW
recordgroup_name VARCHAR2,
row_number NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

recordgroup_id

The unique ID that Forms Developer 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 VARCHAR2.

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.

Error Conditions:

Forms Developer 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).

ADD_GROUP_ROW Restrictions

- 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.

ADD_GROUP_ROW Examples

```
/* **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 );
 END IF;
 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;
```

ADD_LIST_ELEMENT Built-in

Description

Adds a single element to a list item.

Syntax

PROCEDURE ADD_LIST_ELEMENT
(*list_name* VARCHAR2*,*
list_index, NUMBER
list_label VARCHAR2*,*
list_value NUMBER);

PROCEDURE ADD_LIST_ELEMENT
(*list_id* ITEM,
list_index VARCHAR2*,*
list_label VARCHAR2*,*
list_value NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

list_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

list_index

Specifies the list index value. The list index is 1 based.

list_label

Specifies the VARCHAR2 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.

ADD_LIST_ELEMENT Restrictions

For a base table list with the List Style property set to Poplist or T-list, Forms Developer does not allow you to add another values element when the block contains queried or changed records. Doing so causes an error. This situation can occur 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 Mapping of 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.

ADD_LIST_ELEMENT Examples

```
/* ** 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 Built-in

Description

Adds parameters to a parameter list. Each parameter consists of a key, its type, and an associated value.

Syntax

PROCEDURE ADD_PARAMETER
(*list* VARCHAR2,
key VARCHAR2,
paramtype VARCHAR2,
value VARCHAR2);

PROCEDURE ADD_PARAMETER
(*name* VARCHAR2,
key VARCHAR2,
paramtype VARCHAR2,
value VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

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 VARCHAR2 name of the parameter list.

key

The name of the parameter. The data type of the key is VARCHAR2.

paramtype

Specifies one of the following two types:

TEXT_PARAMETER A VARCHAR2 string literal.

DATA_PARAMETER A VARCHAR2 string specifying the name of a record group defined in the current form. When Forms Developer passes a data parameter to Reports or Graphics, the data in the specified record group can substitute for a query that Reports or 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 64K characters. Data type of the value is VARCHAR2.

ADD_PARAMETER Restrictions

- 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.
- You cannot add a parameter of type DATA_PARAMETER if the parameter list is being passed to another form.

ADD_PARAMETER Examples

```
/* ** 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;
```

ADD_TREE_DATA Built-in

Description

Adds a data set under the specified node of a hierarchical tree item.

Syntax

PROCEDURE ADD_TREE_DATA (*item_id* ITEM,
node FTREE.NODE,
offset_type NUMBER,
offset NUMBER,
data_source NUMBER,
data VARCHAR2);

PROCEDURE ADD_TREE_DATA (*item_name* VARCHAR2,
node FTREE.NODE,
offset_type NUMBER,
offset NUMBER,
data_source NUMBER,
data VARCHAR2);

PROCEDURE ADD_TREE_DATA (*item_name* VARCHAR2,
node FTREE.NODE,
offset_type NUMBER,
offset NUMBER,
data_source NUMBER,
data RECORDGROUP);

PROCEDURE ADD_TREE_DATA (*item_id* ITEM,
node FTREE.NODE,
offset_type NUMBER,
offset NUMBER,
data_source NUMBER,
data RECORDGROUP);

Built-in Type unrestricted procedure

Enter Query Mode no

Parameters

item_name	Specifies the name of the object created at design time. The data type of the name is VARCHAR2 string.
Item_id	Specifies the unique ID that Forms Developer 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.
node	Specifies a valid node. There is a special value FTREE.ROOT_NODE which can be used to define the root of the tree.
offset_type	Specifies the type of offset for the node. Possible values are: PARENT_OFFSET SIBLING_OFFSET If *offset_type* is PARENT_OFFSET, adds a data subset immediately under the specified node at the location among its children indicated by *offset*. If *offset_type* is SIBLING_OFFSET, adds the new data as a sibling to the specified node.
offset	Indicates the position of the new node. If *offset_type* is PARENT_OFFSET, then *offset* can be either 1-n or LAST_CHILD. If *offset_type* is SIBLING_OFFSET, then *offset* can be either NEXT_NODE or PREVIOUS_NODE.
data_source	Indicates the type of data source. Possible values are: RECORD_GROUP QUERY_TEXT
data	Specifies the data to be added. If data source is QUERY_TEXT, then data is the text of the query. If data source is RECORD_GROUP, then data is an item of type RECORDGROUP or the name of a record group.

ADD_TREE_DATA Examples

```
/*
** Built-in: ADD_TREE_DATA */

-- This code copies a set of values from a record group
-- and adds them as a top level node with any children
-- nodes specified by the structure of the record group.
-- The new top level node will be inserted as the last
-- top level node.

DECLARE
htree ITEM;
rg_data RECORDGROUP;
BEGIN
-- Find the tree itself.
htree := Find_Item('tree_block.htree3');

-- Find the record group.
rg_data := FIND_GROUP('new_data_rg');

-- Add the new node at the top level and children.
Ftree.Add_Tree_Data(htree,
Ftree.ROOT_NODE,
Ftree.PARENT_OFFSET,
Ftree.LAST_CHILD,
Ftree.RECORD_GROUP,
rg_data);
END;
```

ADD_TREE_NODE Built-in

Description

Adds a data element to a hierarchical tree item.

Syntax

FUNCTION ADD_TREE_NODE (*item_name* VARCHAR2,
node FTREE.NODE,
offset_type NUMBER,
offset NUMBER,
state NUMBER,
label VARCHAR2,
icon VARCHAR2,
value VARCHAR2);

FUNCTION ADD_TREE_NODE (*item_id* ITEM,
node FTREE.NODE,
offset_type NUMBER,
offset NUMBER,
state NUMBER,
label VARCHAR2,
icon VARCHAR2,
value VARCHAR2);

Built-in Type unrestricted procedure

Returns NODE

Enter Query Mode no

Parameters

item_name	Specifies the name of the object created at design time. The data type of the name is VARCHAR2 string.
Item_id	Specifies the unique ID that Forms Developer 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.

node	Specifies a valid node. There is a special value FTREE.ROOT_NODE which can be used to define the root of the tree.
offset_type	Specifies the type of offset for the node. Possible values are: PARENT_OFFSET SIBLING_OFFSET
offset	Indicates the position of the new node. If *offset_type* is PARENT_OFFSET, then *offset* can be either 1-n or LAST_CHILD. If *offset_type* is SIBLING_OFFSET, then *offset* can be either NEXT_NODE or PREVIOUS_NODE.
state	Specifies the state of the node. Possible vaues are: COLLAPSED_NODE EXPANDED_NODE LEAF_NODE
label	The displayed text for the node.
icon	The filename for the node’s icon.
value	Specifies the VARCHAR2 value of the node.

ADD_TREE_NODE Examples

```
/*
** Built-in: ADD_TREE_NODE
*/

-- This code copies a value from a Form item and
-- adds it to the tree as a top level node. The
-- value is set to be the same as the label.

DECLARE

htree ITEM;
top_node FTREE.NODE;
new_node FTREE.NODE;
item_value VARCHAR2(30);

BEGIN
```

```
-- Find the tree itself.

htree := Find_Item('tree_block.htree3');

-- Copy the item value to a local variable.

item_value := :wizard_block.new_node_data;

-- Add an expanded top level node to the tree
-- with no icon.

new_node := Ftree.Add_Tree_Node(htree,
Ftree.ROOT_NODE,
Ftree.PARENT_OFFSET,
Ftree.LAST_CHILD,
Ftree.EXPANDED_NODE,
item_value,
NULL,
item_value);

END;
```

ADJUST_TZ Built-in

Description

Adjusts an Oracle DATE value from one time zone region to another.

Syntax

```
PROCEDURE ADJUST_TZ
(date_var IN OUT DATE,
from_tz VARCHAR2,
to_tz VARCHAR2);

PROCEDURE ADJUST_TZ
(date_var IN OUT DATE,
from_tz VARCHAR2,
to_tz VARCHAR2,
is_daylight IN OUT BOOLEAN,
is_in_overlap OUT BOOLEAN,
timezone_label OUT VARCHAR2);
```

Parameters

`date_var` is adjusted from the `from_tz` time zone region to the `to_tz` time zone region. `date_var` can be a Forms `DATE` or `DATETIME` item, or a Forms `DATE` parameter, or a local PL/SQL `DATE` variable.

Three additional parameters provide extra feedback as to where the date falls in the `to_tz` time zone region.

`is_daylight` indicates whether the date is in daylight saving time. The `is_daylight` parameter is also used as an input parameter, to disambiguate dates that fall within the `from_tz` time zone region's overlap period. It's ignored if `date_var` falls outside the `from_tz` time zone region's overlap period. If omitted, `is_daylight` defaults to `FALSE`.

`is_in_overlap` indicates whether the date falls within the "overlap period" (the ambiguous 2-hour period that occurs once a year when switching from daylight saving time back to standard time).

`timezone_label` is a label for daylight versus standard that's appropriate for the `to_tz` time zone region. For example, when the `to_tz` time zone region is `US/Pacific`, `timezone_label` will be set to `PST` when `is_daylight` is set to `FALSE`, and to `PDT` when `is_daylight` is set to `TRUE`.

Related topics

Datetime_Server_TZ Property

Datetime_Local_TZ Property

BELL Built-in

Description

Sets the display's bell to ring the next time the 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.

Syntax

PROCEDURE BELL;

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

none

BELL Examples

The following example rings the bell three times:

```
FOR i in 1..3 LOOP
BELL;
SYNCHRONIZE;
END LOOP;
```

CALL_FORM Built-in

Description

Runs an indicated form while keeping the parent form active. Forms Developer runs the called form with the same Runform preferences as the parent form. When the called form is exited Forms Developer processing resumes in the calling form at the point from which you initiated the call to CALL_FORM.

Syntax

PROCEDURE CALL_FORM
(*formmodule_name* VARCHAR2);

PROCEDURE CALL_FORM
(*formmodule_name* VARCHAR2,
display NUMBER);

PROCEDURE CALL_FORM
(*formmodule_name* VARCHAR2,
display NUMBER,
switch_menu NUMBER);

PROCEDURE CALL_FORM
(*formmodule_name* VARCHAR2,
display NUMBER,
switch_menu NUMBER,
query_mode NUMBER);

PROCEDURE CALL_FORM
(*formmodule_name* VARCHAR2,
display NUMBER,
switch_menu NUMBER,
query_mode NUMBER,
data_mode NUMBER);

PROCEDURE CALL_FORM
(*formmodule_name* VARCHAR2,
display NUMBER,
switch_menu NUMBER,
query_mode NUMBER,

paramlist_id PARAMLIST);

PROCEDURE CALL_FORM
(*formmodule_name* VARCHAR2,
display NUMBER,
switch_menu NUMBER,
query_mode NUMBER,
paramlist_name VARCHAR2);

PROCEDURE CALL_FORM
(*formmodule_name* VARCHAR2,
display NUMBER,
switch_menu NUMBER,
query_mode NUMBER,
data_mode NUMBER,
paramlist_id PARAMLIST);

PROCEDURE CALL_FORM
(*formmodule_name* VARCHAR2,
display NUMBER,
switch_menu NUMBER,
query_mode NUMBER,
data_mode NUMBER,
paramlist_name VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

formmodule_name

The name of the called form (must be enclosed in single quotes). Datatype is VARCHAR2.

display

HIDE (The default.) Forms Developer will hide the calling form before drawing the called form.

NO_HIDE Forms Developer will display the called form without hiding the calling form.

switch_menu

NO_REPLACE (The default.) Forms Developer will keep the default menu module of the calling form active for the called form.

DO_REPLACE Forms Developer will replace the default menu module of the calling form with the default menu module of the called form.

query_mode

NO_QUERY_ONLY (The default.) Forms Developer will run the indicated form in normal mode, allowing the end user to perform inserts, updates, and deletes from within the called form.

QUERY_ONLY Forms Developer will run the indicated form in query-only mode, allowing the end user to query, but not to insert, update, or delete records.

data_mode

NO_SHARE_LIBRARY_DATA (The default.) At runtime, Forms Developer will not share data between forms that have identical libraries attached (at design time).

SHARE_LIBRARY_DATA At runtime, Forms Developer will share data between forms that have identical libraries attached (at design time).

paramlist_id

The unique ID Forms Developer assigns when it creates the parameter list. You can optionally include a parameter list as initial input to the called form. Datatype is PARAMLIST.

paramlist_name

The name you gave the parameter list object when you defined it. Datatype is VARCHAR2.

CALL_FORM Restrictions

- Forms Developer ignores the query_mode parameter when the calling form is running in QUERY_ONLY mode. Forms Developer 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.)
- If you execute CALL_FORM from Menu PL/SQL code, and subsequently replace the current menu in the called Form, the
code after the CALL_FORM in the menu program unit does not execute when the called Form is exited and control returns
to the calling Form. This behavior is because replacing the current menu destroys any code that is executing in the menu being replaced.

CALL_FORM Example

```
/* Example 1:   ** Call a form in query-only mode.
*/
BEGIN
  CALL_FORM('empbrowser', no_hide, no_replace, query_only);
END;

/* Example 2:
** Call a form, pass a parameter list (if it exists)
*/
DECLARE
  pl_id        PARAMLIST;
  theformname  VARCHAR2(20);
```

```
BEGIN
  theformname := 'addcust';

 /* 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 Built-in

Description

Accepts and processes function key input from the end user. When CALL_INPUT is terminated, Forms Developer resumes processing from the point at which the call to CALL_INPUT occurred.

Syntax

PROCEDURE CALL_INPUT;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

none

CALL_INPUT Restrictions

CALL_INPUT is included for compatibility with previous versions. You should not include this built-in in new applications.

CANCEL_REPORT_OBJECT Built-in

Description

Cancels a long-running, asynchronous report. You should verify the report is canceled by checking the status of the report using REPORT_OBJECT_STATUS .

Syntax

PROCEDURE CANCEL_REPORT_OBJECT
(*report_id* VARCHAR2
);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

report_id

> The VARCHAR2 value returned by the RUN_REPORT_OBJECT built-in. This value uniquely identifies the report that is currently running either locally or on a remote report server.

Usage Notes

- CANCEL_REPORT_OBJECT is useful only when a report is run asynchronously. You cannot cancel an report that is run synchronously.

CHECKBOX_CHECKED Built-in

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, Forms Developer returns the following error:

FRM-41038: Item <item_name> is not a check box.

Syntax

FUNCTION CHECKBOX_CHECKED
(*item_id* ITEM);

FUNCTION CHECKBOX_CHECKED
(*item_name* VARCHAR2);

Built-in Type unrestricted function

Returns BOOLEAN

Enter Query Mode yes

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.

Parameters

item_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

CHECKBOX_CHECKED 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 Mapping of Other Values property.

CHECKBOX_CHECKED Examples

```
/* ** 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 Built-in

Description

When called from an On-Check-Unique trigger, initiates the default Forms Developer 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.

Syntax

PROCEDURE CHECK_RECORD_UNIQUENESS;

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

none

CHECK_RECORD_UNIQUENESS Restrictions

Valid only in an On-Check-Unique trigger.

CHECK_RECORD_UNIQUENESS Examples

```
/*
** Built-in: CHECK_RECORD_UNIQUENESS
** Example: Perform Forms Developer 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 Built-in

Description

Causes Forms Developer to remove all records from, or "flush," the current block.

Syntax

PROCEDURE CLEAR_BLOCK;

PROCEDURE CLEAR_BLOCK
(*commit_mode* NUMBER);

Built-in Type restricted procedure

Enter Query Mode no

Parameters

If the end user has made changes to records in the current block that have not been posted or committed, Forms Developer 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
Forms Developer prompts the end user to commit the changes during CLEAR_BLOCK processing.

DO_COMMIT
Forms Developer validates the changes, performs a commit, and flushes the current block without prompting the end user.

NO_COMMIT
Forms Developer validates the changes and flushes the current block without performing a commit or prompting the end user.

NO_VALIDATE
Forms Developer flushes the current block without validating the changes, committing the changes, or prompting the end user.

CLEAR_BLOCK Examples

```
** 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;
END;
/*
** Trigger: Pre-Query
*/
BEGIN
Default_Value(NULL, 'Global.Employee_Id');
IF :Global.Employee_Id IS NOT NULL THEN
:Emp.Empno := :Global.Employee_Id;
END IF;
END;
```

Related topic

CLEAR_FORM built-in

CLEAR_EOL Built-in

Description

Clears the current text item's value from the current cursor position to the end of the line.

Note: If you call `CLEAR_EOL` from a Menu Item or Button, you may find the entire contents of the target field being cleared, rather
than just the contents at the insertion point. To prevent this behavior, set the `Keep Cursor Position` property for the item to `Yes`.

Syntax

PROCEDURE CLEAR_EOL;

Built-in Type restricted procedure

Enter Query Mode yes

CLEAR_EOL Example

```
** Built-in: CLEAR_EOL
** Example: Clears out the contents of any number field when
** the end user 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 Built-in

Description

Causes Forms Developer to remove all records from, or flush, the current form, and puts the input focus in the first item of the first block.

Syntax

PROCEDURE CLEAR_FORM;

PROCEDURE CLEAR_FORM
(*commit_mode* NUMBER);

PROCEDURE CLEAR_FORM
(*commit_mode* NUMBER*,*
rollback_mode NUMBER);

Built-in Type restricted procedure

Enter Query Mode no

Parameters

If the end user has made changes to records in the current form or any called form, and those records have not been posted or committed, Forms Developer processes the records, following the directions indicated by the argument supplied for the following parameter:

commit_mode

> **ASK_COMMIT** Forms Developer prompts the end user to commit the changes during CLEAR_FORM processing.
>
> **DO_COMMIT** Forms Developer validates the changes, performs a commit, and flushes the current form without prompting the end user.
>
> **NO_COMMIT** Forms Developer validates the changes and flushes the current form without performing a commit or prompting the end user.

NO_VALIDATE Forms Developer flushes the current form without validating the changes, committing the changes, or prompting the end user.

rollback_mode

TO_SAVEPOINT Forms Developer rolls back all uncommitted changes (including posted changes) to the current form's savepoint.

FULL_ROLLBACK Forms Developer 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, Forms Developer prevents any commit processing in the called form.)

CLEAR_FORM Restrictions

If you use a PL/SQL ROLLBACK statement in an anonymous block or a user-defined subprogram, Forms Developer interprets that statement as a CLEAR_FORM built-in subprogram with no parameters.

CLEAR_FORM 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 Built-in

Description

Clears the value from the current text item, regardless of the current cursor position, and changes the text item value to NULL.

Syntax

PROCEDURE CLEAR_ITEM;

Built-in Type restricted procedure

Enter Query Mode yes

CLEAR_ITEM 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;
```

Related topic

CLEAR_EOL Built-in

CLEAR_LIST Built-in

Description

Clears all elements from a list item. After Forms Developer clears the list, the list will contain only one element (the null element), regardless of the item's Required property.

Syntax

PROCEDURE CLEAR_LIST
(*list_id* ITEM);

PROCEDURE CLEAR_LIST
(*list_name* VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

list_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

Usage Notes

- Do not use the CLEAR_LIST built-in if the Mapping of Other Values property is defined and there are queried records in the block. Doing so may cause Forms Developer 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 Mapping of 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 Forms Developer 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.

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).

CLEAR_LIST Restrictions

- 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.

CLEAR_LIST Example

```
/*

** Built-in: CLEAR_LIST
** Example: See POPULATE_LIST
*/
```

CLEAR_MESSAGE Built-in

Description

Removes the current message from the screen message area.

Syntax

PROCEDURE CLEAR_MESSAGE;

Built-in Type restricted procedure

Enter Query Mode yes

CLEAR_MESSAGE Examples

```
/*

** 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 Built-in

Description

Causes Forms Developer to remove, or flush, the current record from the block, without performing validation. If a query is open in the block, Forms Developer 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.

Syntax

PROCEDURE CLEAR_RECORD;

Built-in Type restricted procedure

Enter Query Mode yes

CLEAR_RECORD Examples

```
/*

** 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;
```

Related topic

DELETE_RECORD built-in

CLOSE_FORM Built-in

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.

Syntax

PROCEDURE CLOSE_FORM
(form_name VARCHAR2);

PROCEDURE CLOSE_FORM
(*form_id* FORMMODULE);

Built-in Type restricted procedure

Enter Query Mode no

Parameters

form_name

Specifies the name of the form to close as a VARCHAR2.

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.

CLOSE_FORM Restrictions

- You cannot close a form that is currently disabled as a result of having issued CALL_FORM to invoke a modal called form.
- You cannot close a form that has called you. For example, if Form_A calls Form_B,

then Form_B cannot close Form_A.

COMMIT_FORM Built-in

Description

Causes Forms Developer to update data in the database to match data in the form. Forms Developer 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 end user 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, Forms Developer treats all records in all base-table blocks as if they are queried records from the database. Forms Developer does not recognize changes that occur in triggers that fire during commit processing.

Syntax

PROCEDURE COMMIT_FORM;

Built-in Type restricted procedure

Enter Query Mode no

COMMIT_FORM Restrictions

If you use a PL/SQL COMMIT statement in an anonymous block or a form-level procedure, Forms Developer interprets that statement as a call to the COMMIT_FORM built-in.

COMMIT_FORM Examples

Example 1

```
/*

** 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;
```

Example 2

```
/*

** Built-in: COMMIT_FORM
** Example: Perform Forms Developer 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');
/*
** Otherwise, do the right thing.
*/
ELSE
Commit_Form;
END IF;
END;
```

CONVERT_OTHER_VALUE Built-in

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.

Syntax

PROCEDURE CONVERT_OTHER_VALUE
(*item_id* ITEM);

PROCEDURE CONVERT_OTHER_VALUE
(*item_name* VARCHAR2);

Built-in Type restricted procedure

Enter Query Mode yes

Parameters

item_id

Specifies the unique ID that Forms Developer assigns to the item when it creates the item. The data type of the ID is ITEM.

item_name

Specifies the VARCHAR2 string you defined as the name of the item at design time.

CONVERT_OTHER_VALUE Restrictions

If the item is not a check box, radio group, or list item, Forms Developer 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.

CONVERT_OTHER_VALUE Examples

```
/*

** Built-in: CONVERT_OTHER_VALUE
** Example: Ensure that a particular checkbox's value
** represents either the checked or unchecked
** value before updating the record.
** Trigger: Pre-Update
*/
BEGIN
Convert_Other_Value('Emp.Marital_Status');
END;
```

COPY Built-in

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.

Syntax

PROCEDURE COPY
(*source* VARCHAR2*,*
destination VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

source The *source* is a literal value.

destination The *destination* can be either a text item or another global variable.

Usage Notes

- To use a text item as the source reference, you can use the following code:

  COPY(NAME_IN(*source*), *destination*);

COPY 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.

COPY Examples

Example 1

```
/*

** 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 Developer
** will use it as a query criterion.
*/
Copy( cur_val, 'Emp.Empno' );
END;
```

Example 2

```
/*

** 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 Built-in

Description

Copies the selected region of a text item from the screen and stores it in the paste buffer until you cut or copy another selected region.

Syntax

PROCEDURE COPY_REGION;

Built-in Type restricted procedure

Enter Query Mode yes

Parameters

None.

Usage Notes

Use COPY_REGION, as well as the other editing functions, on text items only. The cut and copy functions transfer the selected region into the system clipboard until you indicate the paste target. At that time, the cut or copied content is pasted onto the target location.

Related topics

CUT_REGION built-in

PASTE_REGION built-in

COPY_REPORT_OBJECT_OUTPUT Built-in

Description

Copies the output of a report to a file.

Syntax

PROCEDURE COPY_REPORT_OBJECT_OUTPUT
(*report_id* VARCHAR2(20),
output_file VARCHAR2
);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

report_id

The VARCHAR2 value returned by the RUN_REPORT_OBJECT built-in. This value uniquely identifies the report that is currently running either locally or on a remote report server.

output_file

The name of the file where the report output will be copied.

Usage Notes

- Use the Report Destination Type property to specify the format of the output file.
- To copy the output of a report from a remote machine, you must set the Report Destination Type property to Cache.

COPY_REPORT_OBJECT_OUTPUT Examples

```
DECLARE
 repid REPORT_OBJECT;
 v_rep VARCHAR2(100);
 rep_status varchar2(20);
BEGIN
 repid := find_report_object('report4');
 v_rep := RUN_REPORT_OBJECT(repid);
 rep_status := report_object_status(v_rep);

 if rep_status = 'FINISHED' then
  message('Report Completed');
  copy_report_object_output(v_rep,'d:/temp/local.pdf');
  host('netscape d:/temp/local.pdf');
 else
  message('Error when running report.');
 end if;
END;
```

COUNT_QUERY Built-in

Description

In an On-Count trigger, performs the default Forms Developer 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, Forms Developer prompts the end user to commit them during COUNT_QUERY processing. Forms Developer 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.

Syntax

PROCEDURE COUNT_QUERY;

Built-in Type restricted procedure

Enter Query Mode yes

Parameters

none

COUNT_QUERY Restrictions

Valid only in triggers that allow restricted built-ins.

COUNT_QUERY Examples

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 Forms Developer 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');
/*
** Deposit the number of query hits in the appropriate
** block property so Forms Developer can display its normal
** status message.
*/
Set_Block_Property(:System.Trigger_Block,QUERY_HITS,
:control.hits);
/*
** Otherwise, do the right thing.
*/
ELSE
Count_Query;
END IF;
END;
```

CREATE_GROUP Built-in

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.

Syntax

FUNCTION CREATE_GROUP
(*recordgroup_name* VARCHAR2*,*
scope NUMBER,
array_fetch_size NUMBER);

Built-in Type unrestricted function

Returns RecordGroup

Enter Query Mode yes

Parameters

recordgroup_name

The string you defined as the name of the record group at design time. When Forms Developer 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.

scope

Specifies whether tlhe record group can be used only within the current form or within every form in a multi-form application. Takes the following constants as arguments:

FORM_SCOPE
Indicates that the record group can by used only within the current form. This is the default value.

GLOBAL_SCOPE
Indicates that the record group is global, and that it can be used within all forms in the application. Once created, a global record group persists for the remainder of the runtime session.

array_fetch_size

Specifies the array fetch size. The default array size is 20.

CREATE_GROUP Examples

```
/*

** 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 Built-in

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, Forms Developer 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.

Syntax

FUNCTION CREATE_GROUP_FROM_QUERY
(*recordgroup_name* VARCHAR2*,*
query VARCHAR2,
scope NUMBER,
array_fetch_size NUMBER);

Built-in Type unrestricted function

Returns RecordGroup

Enter Query Mode yes

Parameters

recordgroup_name

The name of the record group. When Forms Developer creates the record group object it also assigns the object a unique ID of type RecordGroup.

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 Forms Developer creates only those columns in the record group

scope

Specifies whether tlhe record group can be used only within the current form or within every form in a multi-form application. Takes the following constants as arguments:

FORM_SCOPE
Indicates that the record group can by used only within the current form. This is the default value.

GLOBAL_SCOPE
Indicates that the record group is global, and that it can be used within all forms in the application. Once created, a global record group persists for the remainder of the runtime session.

array_fetch_size

Specifies the array fetch size. The default array size is 20.

CREATE_GROUP_FROM_QUERY Restrictions

- If a global record group is created from (or populated with) a query while executing form A, and the query string contains bind variable references which are local to A (:block.item or :PARAMETER.param), when form A terminates execution, the global query record group is converted to a global non-query record group (it retains the data, but a subsequent call to POPULATE_GROUP is considered an error).

CREATE_GROUP_FROM_QUERY Examples

```
/*

** 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 Built-in

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. A parameter list can be passed as an argument to the CALL_FORM, NEW_FORM, OPEN_FORM, RUN_REPORT_OBJECT, and RUN_PRODUCT built-in subprograms.

Syntax

FUNCTION CREATE_PARAMETER_LIST
(*name* VARCHAR2);

Built-in Type unrestricted function

Returns ParamList

Enter Query Mode yes

Parameters

name

Specifies the VARCHAR2 name of the parameter list object.

When Forms Developer 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.

CREATE_PARAMETER_LIST Restrictions

- You cannot create a parameter list named DEFAULT. DEFAULT is reserved for the parameter list that Forms Developer 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.

CREATE_PARAMETER_LIST Example

```
/*

** 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
pl_id := Get_Parameter_List(pl_name);
IF Id_Null(pl_id) THEN
pl_id := Create_Parameter_List(pl_name);
IF Id_Null(pl_id) THEN
Message('Error creating parameter list '||pl_name);
RAISE Form_Trigger_Failure;
END IF;
ELSE
Message('Parameter list '||pl_name||' already exists!');
RAISE Form_Trigger_Failure;
END IF;
END;
```

CREATE_QUERIED_RECORD Built-in

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.

Syntax

PROCEDURE CREATE_QUERIED_RECORD;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

none

CREATE_QUERIED_RECORD Restrictions

- In blocks with a large number of records, this procedure can have side effects on disk I/O, memory allocation, or both.

CREATE_QUERIED_RECORD Examples

```
/*

** 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 Forms Developer 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 Built-in

Description

Creates a new record in the current block after the current record. Forms Developer then navigates to the new record.

Syntax

PROCEDURE CREATE_RECORD;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

None.

CREATE_RECORD Examples

```
/*

** 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;
```

Related topic

INSERT_RECORD built-in

CREATE_TIMER Built-in

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, Forms Developer fires the When-Timer-Expired trigger.

Syntax

FUNCTION CREATE_TIMER
(*timer_name* VARCHAR2,
milliseconds NUMBER,
iterate NUMBER);

Built-in Type unrestricted function

Returns Timer

Enter Query Mode yes

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 VARCHAR2.

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, and 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.

CREATE_TIMER Restrictions

- Values greater than 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, Forms Developer 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.

CREATE_TIMER Examples

The following example creates a timer called EMP_TIMER, and sets it to 60 seconds and an iterate value of NO_REPEAT:

```
DECLARE
timer_id Timer;
one_minute NUMBER(5) := 60000;
BEGIN
timer_id := CREATE_TIMER('emp_timer', one_minute, NO_REPEAT);
END;
```

CREATE_VAR Built-in

Description

Creates an empty, unnamed variant. There are two versions of the function, one for scalars and the other for arrays.

Syntax

FUNCTION CREATE_VAR
(persistence BOOLEAN)
RETURN newvar OLEVAR;

FUNCTION CREATE_VAR
(bounds OLE_SAFEARRAYBOUNDS,
vtype VT_TYPE,
persistence BOOLEAN)
RETURN newvar OLEVAR;

Built-in Type unrestricted function

Returns the created OLE variant.

Parameters

persistence

Controls the persistence of the variant after its creation. A boolean value of TRUE establishes the variant as persistent; a value of FALSE establishes the variant as non-persistent.This is an optional parameter. If not specified, the default value is non-persistent.

bounds

A PL/SQL table that specifies the dimensions to be given to the created array.For more information about the contents and layout of this parameter and the type

OLE_SAFEARRAYBOUNDS, see ARRAY TYPES FOR OLE SUPPORT.

vtype

The OLE variant type (VT_TYPE) of the elements in the created array. If the array will contain mixed element types, specify VT_VARIANT.

Usage Notes

- The created variant is untyped, unless it is an array -- in which case its elements have the type you specify.
- The created variant is also without a value. Use the SET_VAR function to assign an initial value and type to the variant.
- A persistent variant exists across trigger invocations. A non-persistent variant exists only as long as the trigger that spawned the call runs. See also DESTROY_VARIANT

CUT_REGION Built-in

Description

Removes a selected region of a text item from the screen and stores it in the paste buffer until you cut or copy another selected region.

Syntax

PROCEDURE CUT_REGION;

Built-in Type restricted procedure

Enter Query Mode yes

Parameters

none

Usage Notes

Use CUT_REGION, as well as the other editing functions, on text items only. The cut and copy functions transfer the selected region into the system clipboard until you indicate the paste target. At that time, the cut or copied content is pasted onto the target location.

Related topics

COPY_REGION built-in

PASTE_REGION built-in

DBMS_ERROR_CODE Built-in

Description

Returns the error number of the last database error that was detected.

Syntax

FUNCTION DBMS_ERROR_CODE;

Built-in Type unrestricted function

Enter Query Mode yes

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.

DBMS_ERROR_CODE Examples

```
/*

** Built-in: DBMS_ERROR_CODE,DBMS_ERROR_TEXT
** Example: Reword certain Forms Developer 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 Developer 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.');
ELSE
/*
** Printout a generic message with the database
** error string in it.
*/
Message('Insert failed because of '||dbmserrtext);
END IF;
END IF;
END;
```

Related topic

DBMS_ERROR_TEXT built-in

DBMS_ERROR_TEXT Built-in

Description

Returns the message number (such as ORA-01438) and message text of the database error.

Syntax

FUNCTION DBMS_ERROR_TEXT;

Built-in Type unrestricted function

Enter Query Mode yes

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.

DBMS_ERROR_TEXT Examples

```
/*

** Built-in: DBMS_ERROR_CODE,DBMS_ERROR_TEXT
** Example: Reword certain Forms Developer 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 Developer 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.');
ELSE
/*
** Printout a generic message with the database
** error string in it.
*/
Message('Insert failed because of '||dbmserrtext);
END IF;
END IF;
END;
```

Related topic

DBMS_ERROR_CODE built-in

DEFAULT_VALUE Built-in

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, Forms Developer creates the variable.

Syntax

PROCEDURE DEFAULT_VALUE
(*value_string* VARCHAR2*,*
variable_name VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

value_string

A valid VARCHAR2 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 VARCHAR2. Any object passed as an argument to this built-in must be enclosed in single quotes.

DEFAULT_VALUE Restrictions

The DEFAULT_VALUE built-in is not related to the Initial Value item property.

DEFAULT_VALUE Examples

```
/*

** 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 Built-in

Description

Deletes a programmatically created record group.

Syntax

PROCEDURE DELETE_GROUP
(*recordgroup_id* RecordGroup);

PROCEDURE DELETE_GROUP
(*recordgroup_name* VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

recordgroup_id

> The unique ID that Forms Developer 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 VARCHAR2.

DELETE_GROUP Restrictions

This built-in cannot be used to delete a record group that was created at design time.

DELETE_GROUP Examples

```
/*

** Built-in: DELETE_GROUP
** Example: Delete a programmatically created record group
*/
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 Built-in

Description

Deletes the indicated row or all rows of the given record group. Forms Developer 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 Forms Developer.

If you choose to delete all rows of the group by supplying the ALL_ROWS constant, Forms Developer 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.

Syntax

PROCEDURE DELETE_GROUP_ROW
(*recordgroup_id* RecordGroup,
row_number NUMBER);

PROCEDURE DELETE_GROUP_ROW
(*recordgroup_name* VARCHAR2,
row_number NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

recordgroup_id

The unique ID that Forms Developer 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 VARCHAR2.

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 Forms Developer is to delete all rows without deleting the record group. ALL_ROWS is a constant.

DELETE_GROUP_ROW Restrictions

This built-in cannot be used to delete rows from a static record group.

DELETE_GROUP_ROW Examples

```
/*

** 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 Built-in

Description

Deletes a specific list element from a list item.

Syntax

PROCEDURE DELETE_LIST_ELEMENT
(*list_name* VARCHAR2*,*
list_index NUMBER);

PROCEDURE DELETE_LIST_ELEMENT
(*list_id*, ITEM
list_index NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

list_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

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 Mapping of Other Values property is defined and there are queried records in the block. Doing so may cause Forms Developer 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 Mapping of 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 Forms Developer 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 programmatically deleted from the list at runtime.

DELETE_LIST_ELEMENT Restrictions

For a Poplist or T-list-style list item, Forms Developer 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 Initial Value property setting.

For a Combobox list item, you can delete the default value element only if the Initial Value property was set to an actual value, rather than an element label.

For a base table Poplist or T-list list item, Forms Developer 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).

DELETE_LIST_ELEMENT Examples

```
/*

** Built-in: DELETE_LIST_ELEMENT
** Example: See ADD_LIST_ELEMENT
*/
```

DELETE_PARAMETER Built-in

Description

Deletes the parameter with the given key from the parameter list.

Syntax

PROCEDURE DELETE_PARAMETER
(*list* VARCHAR2,
key VARCHAR2);

PROCEDURE DELETE_PARAMETER
(*name* VARCHAR2,
key VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

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 VARCHAR2 name of the parameter list.

key

> The name of the parameter. The data type of the key is VARCHAR2.

DELETE_PARAMETER Restrictions

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.

DELETE_PARAMETER Examples

```
/*

** 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 Built-in

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, Forms Developer locks the record before removing it and marking it as a delete.

If a query is open in the block, Forms Developer 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 Forms Developer processing for deleting a record during the Post and Commit Transaction process, as shown in Example 2 below.

Syntax

PROCEDURE DELETE_RECORD;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

none

DELETE_RECORD Examples

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 Forms Developer 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
/*
** Check the global flag we set during form startup
*/
IF :Global.Using_Transactional_Triggers = 'TRUE' THEN
User_Exit('my_delrec block=EMP');
/*
** Otherwise, do the right thing.
*/
ELSE
Delete_Record;
END IF;
END;
```

Related topic

CLEAR_RECORD built-in

DELETE_TIMER Built-in

Description

Deletes the given timer from the form.

Syntax

PROCEDURE DELETE_TIMER
(*timer_id* Timer);

PROCEDURE DELETE_TIMER
(*timer_name* VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

timer_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

DELETE_TIMER Restrictions

- 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))...

DELETE_TIMER Examples

```
/*

** 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;
```

DELETE_TREE_NODE Built-in

Description

Removes the data element from the tree.

Syntax

PROCEDURE DELETE_TREE_NODE
(*item_name* VARCHAR2,
node NODE);

PROCEDURE DELETE_TREE_NODE
(*item_id* ITEM,
node NODE);

Built-in Type unrestricted procedure

Enter Query Mode no

Parameters

item_name

Specifies the name of the object created at design time. The data type of the name is VARCHAR2 string.

Item_id

Specifies the unique ID that Forms Developer 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.

node

Specifies a valid node

Usage Notes

Removing a branch node also removes all child nodes.

DELETE_TREE_NODE Examples

This code finds a node with the label "Zetie" and deletes it and all of its children.

```
/*
** Built-in: DELETE_TREE_NODE
*/

DECLARE

htree ITEM;
delete_node FTREE.NODE;

BEGIN


/* Find the tree itself.
*/
htree := FIND_ITEM('tree_block.htree3');

/* Find the node with a label of "Zetie".
** Start searching from the root of the tree.
*/

delete_node := FTREE.FIND_TREE_NODE(htree,
'Zetie',
FTREE.FIND_NEXT,
FTREE.NODE_LABEL,
FTREE.ROOT_NODE,
FTREE.ROOT_NODE);

/* Delete the node and all of its children.
*/
IF NOT FTREE.ID_NULL(delete_node)
THEN FTREE.DELETE_TREE_NODE(htree, delete_node);
END IF;
END;
```

DESTROY_PARAMETER_LIST Built-in

Description

Deletes a dynamically created parameter list and all parameters it contains.

Syntax

PROCEDURE DESTROY_PARAMETER_LIST
(*list* VARCHAR2*);*

PROCEDURE DESTROY_PARAMETER_LIST
(*name* VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

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 VARCHAR2 name of the parameter list.

Usage Notes

When a parameter list is destroyed using DESTROY_PARAMETER_LIST the parameter list handle is NOT set to NULL.

Use the GET_PARAMETER_LIST built-in to return the ID to a variable of the type PARAMLIST.

DESTROY_PARAMETER_LIST Examples

```
/*
```

```
** 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;
```

Related topic

CREATE_PARAMETER_LIST built-in

DESTROY_VARIANT Built-in

Description

Destroys a variant that was created by the CREATE_VAR Built-in.

Syntax

PROCEDURE DESTROY_VARIANT (variant OLEVAR);

Built-in Type unrestricted procedure

Parameters

variant

The OLE variant to be destroyed.

DISPLAY_ERROR Built-in

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, Forms Developer redisplays the form. If there is no error to display when you call this built-in, Forms Developer ignores the call and does not display the DISPLAY ERROR screen.

Syntax

PROCEDURE DISPLAY_ERROR;

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

None.

DISPLAY_ITEM Built-in

Description

Maintained for backward compatibility only. For new applications, you should use the SET_ITEM_INSTANCE_PROPERTY built-in. DISPLAY_ITEM modifies an item's appearance by assigning a specified display attribute to the item. DISPLAY_ITEM has the side-effect of also changing the appearance of any items that mirror the changed instance. SET_ITEM_INSTANCE_PROPERTY does not change mirror items.

You can reference any item in the current form.

Any change made by a DISPLAY_ITEM built-in is effective until:

- the same item instance is referenced by another DISPLAY_ITEM built-in, or
- the same item instance is referenced by the SET_ITEM_INSTANCE_PROPERTY built-in (with VISUAL_ATTRIBUTE property), or
- the instance of the item is removed (e.g., through a CLEAR_RECORD or a query), or
- you modify a record (whose status is NEW), navigate out of the record, then re-enter the record, or
- the current form is exited

Syntax

PROCEDURE DISPLAY_ITEM
(*item_id* ITEM,
attribute VARCHAR2);

PROCEDURE DISPLAY_ITEM
(*item_name* VARCHAR2,
attribute VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

item_id

Specifies the unique ID that Forms Developer assigns to the item when it creates the item. The data type of the ID is ITEM.

item_name

Specifies the VARCHAR2 string you gave to the item when you created it.

attribute

Specifies a named visual attribute that should exist. Forms Developer will search for named visual attribute first.

Note: You can specify Normal as a method for applying the default attributes to an item, but only if your form does not contain a visual attribute or logical (or otherwise) called Normal. You can also specify NULL as a method for returning an item to its initial visual attributes (default, custom, or named).

DISPLAY_ITEM Examples

```
/*

** 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;
```

Related topic

SET_ITEM_PROPERTY Built-in

SET_ITEM_INSTANCE_PROPERTY Built-in

DOWN Built-in

Description

Navigates to the instance of the current item in the record with the next higher sequence number. If necessary, Forms Developer fetches a record. If Forms Developer has to create a record, DOWN navigates to the first navigable item in the record.

Syntax

PROCEDURE DOWN;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

None.

Related topics

UP Built-in

NEXT_RECORD Built-in

PREVIOUS_RECORD Built-in

DO_KEY Built-in

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.

Syntax

PROCEDURE DO_KEY
(*built_in_subprogram_name* VARCHAR2);

Built-in Type restricted procedure

Enter Query Mode yes

Parameters

built_in_subprogram_name

Specifies the name of a valid built-in subprogram.

Built-in	**Key Trigger**	**Associated Function Key**
CLEAR_BLOCK	Key-CLRBLK	[Clear Block]
CLEAR_FORM	Key-CLRFRM	[Clear Form]
CLEAR_RECORD	Key-CLRREC	[Clear Record]
COMMIT_FORM	Key-COMMIT	[Commit]
COUNT_QUERY	Key-CQUERY	[Count Query Hits]
CREATE_RECORD	Key-CREREC	[Insert Record]

DELETE_RECORD	Key-DELREC	[Delete Record]
DOWN	Key-DOWN	[Down]
DUPLICATE_ITEM	Key-DUP-ITEM	[Duplicate Item]
DUPLICATE_RECORD	Key-DUPREC	[Duplicate Record]
EDIT_TEXTITEM	Key-EDIT	[Edit]
ENTER	Key-ENTER	[Enter]
ENTER_QUERY	Key-ENTQRY	[Enter Query]
EXECUTE_QUERY	Key-EXEQRY	[Execute Query]
EXIT_FORM	Key-EXIT	[Exit/Cancel]
HELP	Key-HELP	[Help]
LIST_VALUES	Key-LISTVAL	[List]
LOCK_RECORD	Key-UPDREC	[Lock Record]
NEXT_BLOCK	Key-NXTBLK	[Next Block]
NEXT_ITEM	Key-NEXT-ITEM	[Next Item]
NEXT_KEY	Key-NXTKEY	[Next Primary Key Fld]
NEXT_RECORD	Key-NXTREC	[Next Record]
NEXT_SET	Key-NXTSET	[Next Set of Records]
PREVIOUS_BLOCK	Key-PRVBLK	[Previous Block]
PREVIOUS_ITEM	Key-PREV-ITEM	[Previous Item]

PREVIOUS_RECORD	Key-PRVREC	[Previous Record]
PRINT	Key-PRINT	[Print]
SCROLL_DOWN	Key-SCRDOWN	[Scroll Down]
SCROLL_UP	Key-SCRUP	[Scroll Up]
UP	Key-UP	[Up]

DO_KEY Restrictions

- DO_KEY accepts built-in names only, not key names: DO_KEY(ENTER_QUERY).

  To accept a specific key name, use the EXECUTE_TRIGGER built-in: EXECUTE_TRIGGER('KEY_F11').

DO_KEY Example

```
/*

** Built-in: DO_KEY
** Example: Simulate pressing the [Execute Query] key.
*/
BEGIN
   DO_KEY('Execute_Query');
END;
```

DUMMY_REFERENCE Built-in

Description

Provides a mechanism for coding an explicit reference to a bind variable that otherwise would be referred to only indirectly in a formula (or in a function or procedure called by the formula). Use DUMMY_REFERENCE to ensure that a formula calculated item (that contains indirect references to bind variables) will be marked for recalculation by Forms Developer.

The expression can be an arbitrary expression of type Char, Number, or Date. Typically the expression will consist of a single reference to a bind variable.

Note: DUMMY_REFERENCE need not be executed for the referenced bind variable to be recognized by Forms Developer (thereby causing the related formula calculated item to be marked for recalcuation).

Syntax

PROCEDURE DUMMY_REFERENCE(expression);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

None.

DUMMY_REFERENCE Restrictions

None.

DUPLICATE_ITEM Built-in

Description

Assigns the current item the same value as the instance of this item in the previous record.

Syntax

PROCEDURE DUPLICATE_ITEM;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

None.

DUPLICATE_ITEM Restrictions

A previous record must exist in your current session, or Forms Developer returns error FRM-41803: No previous record to copy value from.

Related topic

DUPLICATE_RECORD built-in

DUPLICATE_RECORD Built-in

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.

Note: The duplicate record does not inherit the record status of the source record; instead, its record status is INSERT.

Syntax

PROCEDURE DUPLICATE_RECORD;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

None.

DUPLICATE_RECORD Restrictions

A previous record must exist in your current session.

DUPLICATE_RECORD Examples

```
/*

** 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 Built-in

Description

Invokes the Runform item editor for the current text item and puts the form in Edit mode.

Syntax

PROCEDURE EDIT_TEXTITEM;

PROCEDURE EDIT_TEXTITEM
(*x* NUMBER,
y NUMBER);
PROCEDURE EDIT_TEXTITEM
(*x* NUMBER,
y NUMBER,
width, NUMBER
height NUMBER);

Built-in Type restricted procedure

Enter Query Mode yes

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, Forms Developer 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, Forms Developer invokes the item editor with its default location and dimensions.

EDIT_TEXTITEM Restrictions

The Width must be at least wide enough to display the buttons at the bottom of the editor window.

EDIT_TEXTITEM Examples

```
/*

** 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
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 Built-in

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 Forms Developer transaction processing.

Default Check Column Security processing refers to the sequence of events that occurs when Forms Developer enforces column-level security for each block that has the Enforce Column Security block property set Yes. To enforce column security, Forms Developer 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, Forms Developer makes the corresponding base table items in the form non-updateable by setting the Update Allowed item property to No dynamically. By default, Forms Developer performs this operation at form startup, processing each block in sequence.

Syntax

PROCEDURE ENFORCE_COLUMN_SECURITY

Built-in Type unrestricted procedure

Enter Query Mode yes

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 Triggers in this help system.

ENFORCE_COLUMN_SECURITY Restrictions

Valid only in an On-Column-Security trigger.

Related topic

On-Column-Security Trigger

ENTER Built-in

Description

Validates data in the current validation unit. (The default validation unit is Item.)

Syntax

PROCEDURE ENTER;

Built-in Type restricted procedure

Enter Query Mode yes

Parameters

None

ENTER 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;
```

Related topic

Validation Unit Property

VALIDATE Built-in

ENTER_QUERY Built-in

Description

The behavior of ENTER_QUERY varies depending on any parameters you supply.

Syntax

PROCEDURE ENTER_QUERY;

PROCEDURE ENTER_QUERY
(*keyword_one* VARCHAR2);

PROCEDURE ENTER_QUERY
(*keyword_two* VARCHAR2);

PROCEDURE ENTER_QUERY
(*keyword_one* VARCHAR2*,*
keyword_two VARCHAR2);

PROCEDURE ENTER_QUERY
(*keyword_one* VARCHAR2*,*
keyword_two VARCHAR2*,*
locking VARCHAR2);

Built-in Type restricted procedure

Enter Query Mode yes (to redisplay the example record from the last query executed in the block)

Parameters

no parameters

> ENTER_QUERY flushes the current block and puts the form in Enter Query mode. If there are changes to commit, Forms Developer prompts the operator to commit them during the ENTER_QUERY process.

keyword_one

ENTER_QUERY(ALL_RECORDS) performs the same actions as ENTER_QUERY except that when EXECUTE_QUERY is invoked, Forms Developer 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, Forms Developer 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, Forms Developer 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, Forms Developer displays a dialog to notify the operator if a record cannot be reserved for update immediately.

Without the NO_WAIT parameter, Forms Developer 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.

ENTER_QUERY 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.

ENTER_QUERY Examples

```
/*

** Built-in: ENTER_QUERY
** Example: Go Into Enter-Query mode, and exit the form if
** the user cancels out of enter-query mode.
*/
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.
*/

IF :System.Record_Status = 'NEW' THEN
Exit_Form(No_Validate);
END IF;
END;
```

ERASE Built-in

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.

Syntax

PROCEDURE ERASE
(*global_variable_name* VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

global_variable_name

Specifies the name of a valid global variable.

ERASE Example

```
ERASE('global.var');
```

ERROR_CODE Built-in

Description

Returns the error number of the Forms Developer error.

Syntax

PROCEDURE ERROR_CODE;

Built-in Type unrestricted function

Enter Query Mode yes

Parameters

None

ERROR_CODE Examples

```
/*

** 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 = 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;
```

Related topics

ERROR_TEXT Built-in

ERROR_TYPE Built-in

ERROR_TEXT Built-in

Description

Returns the message text of the Forms Developer error.

Syntax

FUNCTION ERROR_TEXT;

Built-in Type unrestricted function

Enter Query Mode yes

Description

Returns the message text of the Forms Developer error.

Parameters

none

Usage Notes

You can use this function to test error messages during exception handling subprograms.

ERROR_TEXT Examples

```
/*

** 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 = 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;
```

Related topics

ERROR_CODE Built-in

ERROR_TYPE Built-in

ERROR_TYPE Built-in

Description

Returns the error message type for the action most recently performed during the current Runform session.

Syntax

FUNCTION ERROR_TYPE;

Built-in Type unrestricted function

Returns ERROR_TYPE returns one of the following values for the error message type:

FRM Indicates an Forms Developer error.

ORA Indicates an ORACLE error.

Enter Query Mode yes

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.

ERROR_TYPE Examples

```
/*

** 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;
```

Related topics

ERROR_CODE Built-in

ERROR_TEXT Built-in

EXECUTE_QUERY Built-in

Description

Flushes the current block, opens a query, and fetches a number of selected records. If there are changes to commit, Forms Developer prompts the operator to commit them before continuing EXECUTE_QUERY processing.

Syntax

PROCEDURE EXECUTE_QUERY;

PROCEDURE EXECUTE_QUERY
(*keyword_one* VARCHAR2);

PROCEDURE EXECUTE_QUERY
(*keyword_two* VARCHAR2);

PROCEDURE EXECUTE_QUERY
(*keyword_one* VARCHAR2*,*
keyword_two VARCHAR2);

PROCEDURE EXECUTE_QUERY
(*keyword_one* VARCHAR2*,*
keyword_two VARCHAR2*,*
*lockin*g VARCHAR2);

Built-in Type restricted procedure

Enter Query Mode yes

Parameters

no parameters

EXECUTE_QUERY flushes the current block, opens a query, and fetches a number of selected records.

keyword_one

EXECUTE_QUERY(ALL_RECORDS) performs the same actions as EXECUTE_QUERY except that Forms Developer fetches all of the selected records.

keyword_two

EXECUTE_QUERY(FOR_UPDATE) performs the same actions as EXECUTE_QUERY except that Forms Developer attempts to lock all of the selected records immediately.

keyword_one/ keyword_two

EXECUTE_QUERY(ALL_RECORDS, FOR_UPDATE) performs the same actions as EXECUTE_QUERY except that Forms Developer 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, Forms Developer displays a dialog to notify the operator if a record cannot be reserved for update immediately.

Without the NO_WAIT parameter, Forms Developer 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.

EXECUTE_QUERY Restrictions

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.

EXECUTE_QUERY Examples

```
/*
```

```
** Built-in: EXECUTE_QUERY
** Example: Visit several blocks and query their contents,
** then go back to the block where original block.
*/
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;
```

EXIT_FORM Built-in

Description

Provides a means to exit a form, confirming commits and specifying rollback action.

- In most contexts, EXIT_FORM navigates "outside" the form. If there are changes in the current form that have not been posted or committed, Forms Developer 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.

Syntax

PROCEDURE EXIT_FORM;

PROCEDURE EXIT_FORM
(*commit_mode* NUMBER);

PROCEDURE EXIT_FORM
(*commit_mode* NUMBER,
rollback_mode NUMBER);

Built-in Type restricted procedure

Enter Query Mode yes

Parameters

commit_mode

ASK_COMMIT

Forms Developer prompts the operator to commit the changes during EXIT_FORM processing.

However, if RECORD_STATUS is INSERT but the record is not valid, Forms Developer instead asks the operator if the form should be closed. If the operator says yes, the changes are rolled back before the form is closed.

DO_COMMIT

Forms Developer validates the changes, performs a commit, and exits the current form without prompting the operator.

NO_COMMIT

Forms Developer validates the changes and exits the current form without performing a commit or prompting the operator.

NO_VALIDATE

Forms Developer exits the current form without validating the changes, committing the changes, or prompting the operator.

rollback_mode

TO_SAVEPOINT

Forms Developer rolls back all uncommitted changes (including posted changes) to the current form's savepoint.

FULL_ROLLBACK

Forms Developer 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, Forms Developer prevents any commit processing in the called form.)

NO_ROLLBACK

Forms Developer 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 Forms Developer from an external 3GL program. The locks remain in effect when Forms Developer returns control to the program.

Usage Notes

Because the default parameters of EXIT_FORM are ASK_COMMIT for commit_mode and TO_SAVEPOINT for rollback_mode, invoking EXIT_FORM without specifying any parameters in some contexts may produce undesired results. For example, if the form is in POST only mode and EXIT_FORM is invoked without parameters, the user will be prompted to commit the changes. However, regardless of the user's input at that prompt, the default rollback_mode of TO_SAVEPOINT rolls back the changes to the form despite a message confirming that changes have been made. To avoid conflicts explicitly specify parameters.

EXIT_FORM Examples

```
/*

** Built-in: EXIT_FORM and POST
** 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
** successfully posted.
*/
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 Built-in

Description

When called from an On-Fetch trigger, initiates the default Forms Developer processing for fetching records that have been identified by SELECT processing.

Syntax

PROCEDURE FETCH_RECORDS;

Built-in Type unrestricted procedure

Enter Query Mode no

This built-in is included primarily for applications that will run against a non-ORACLE data source.

Parameters

none

FETCH_RECORDS Examples

```
/*

** Built-in: FETCH_RECORDS
** Example: Perform Forms Developer 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 Forms Developer
** 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. Forms Developer 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;
/*
** Otherwise, do the right thing.
*/
ELSE
Fetch_Records;
```

```
END IF;
END;
```

Related topic

On-Fetch Trigger

FIND_ALERT Built-in

Description

Searches the list of valid alerts in Forms Developer. 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.

Syntax

FUNCTION FIND_ALERT
(*alert_name* VARCHAR2);

Built-in Type unrestricted function

Returns Alert

Enter Query Mode yes

Parameters

alert_name

Specifies the VARCHAR2 alert name.

FIND_ALERT Examples

```
/*

** Built-in: FIND_ALERT
** Example: Show a user-warning alert. If the user presses
** the OK button, then make REALLY sure they want
** to continue with another alert.
*/
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);
IF al_button = ALERT_BUTTON2 THEN
Erase_All_Employee_Records;
END IF;
END IF;
END IF;
END IF;
END;
```

FIND_BLOCK Built-in

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.

Syntax

FUNCTION FIND_BLOCK
(*block_name* VARCHAR2);

Built-in Type unrestricted function

Returns `Block`

Enter Query Mode yes

Parameters

block_name

Specifies the VARCHAR2 block name.

FIND_BLOCK Examples

```
/*

** 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 Built-in

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.

Syntax

FUNCTION FIND_CANVAS
(*canvas_name* VARCHAR2);

Built-in Type unrestricted function

Returns Canvas

Enter Query Mode yes

Parameters

canvas_name

Specifies the VARCHAR2 canvas name you gave the canvas when defining it.

FIND_CANVAS Examples

```
DECLARE
my_canvas Canvas;
BEGIN
my_canvas := Find_Canvas('my_canvas');
END;
```

FIND_COLUMN Built-in

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.

Syntax

FUNCTION FIND_COLUMN
(*recordgroup.groupcolumn_name* VARCHAR2);

Built-in Type unrestricted function

Returns `GroupColumn`

Enter Query Mode yes

Parameters

recordgroup. groupcolumn_name

Specifies the fully qualified VARCHAR2 record group column name.

FIND_COLUMN Examples

```
/*

** 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 Built-in

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.

Syntax

FUNCTION FIND_EDITOR
(*editor_name* VARCHAR2);

Built-in Type unrestricted function

Returns Editor

Enter Query Mode yes

Parameters

editor_name

Specifies a valid VARCHAR2 editor name.

FIND_EDITOR Examples

```
/*

** 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;
```

FIND_FORM Built-in

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.

Syntax

FUNCTION FIND_FORM
(*formmodule_name* VARCHAR2);

Built-in Type unrestricted function

Returns FormModule

Enter Query Mode yes

Parameters

formmodule_name

Specifies a valid VARCHAR2 form name.

FIND_FORM Examples

```
/*

** 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 Built-in

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.

Syntax

FUNCTION FIND_GROUP
(*recordgroup_name* VARCHAR2);

Built-in Type unrestricted function

Returns RecordGroup

Enter Query Mode yes

Parameters

recordgroup_name

Specifies the valid VARCHAR2 record group name.

FIND_GROUP Restrictions

Performance of this function depends upon the number of record groups.

FIND_GROUP Example

```
/*

** Built-in: FIND_GROUP
** Example: See CREATE_GROUP and DELETE_GROUP_ROW
*/
```

FIND_ITEM Built-in

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.

Syntax

FUNCTION FIND_ITEM
(*block.item_name* VARCHAR2);

Built-in Type unrestricted function

Returns Item

Enter Query Mode yes

Parameters

block_name.item_name

Specifies the fully qualified item name. The data type of the name is VARCHAR2.

FIND_ITEM Examples

```
/*

** Built-in: FIND_ITEM
** Example: Find an item's Id before setting several
** of its properties.
*/
PROCEDURE Hide_an_Item( item_name VARCHAR2, hide_it BOOLEAN) IS
it_id Item;
BEGIN
it_id := Find_Item(item_name);
IF Id_Null(it_id) THEN
```

```
Message('No such item: '||item_name);
RAISE Form_Trigger_Failure;
ELSE
IF hide_it THEN
Set_Item_Property(it_id,VISIBLE,PROPERTY_FALSE);
ELSE
Set_Item_Property(it_id,VISIBLE,PROPERTY_TRUE);
Set_Item_Property(it_id,ENABLED,PROPERTY_TRUE);
Set_Item_Property(it_id,NAVIGABLE,PROPERTY_TRUE);
END IF;
END IF;
END;
```

FIND_LOV Built-in

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.

Syntax

FUNCTION FIND_LOV
(*LOV_name* VARCHAR2);

Built-in Type unrestricted function

Returns LOV

Enter Query Mode yes

Parameters

LOV_name

Specifies the valid VARCHAR2 LOV name.

FIND_LOV 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 Built-in

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.

Syntax

FUNCTION FIND_MENU_ITEM
(*menu_name.menu_item_name* VARCHAR2);

Built-in Type unrestricted function

Returns MenuItem

Enter Query Mode yes

Parameters

menu_name.menu_item_name

Specifies a valid fully-qualified VARCHAR2 menu item name.

FIND_MENU_ITEM Examples

```
/*

** Built-in: FIND_MENU_ITEM
** Example: Find the id of a menu item before setting
** multiple properties
*/
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
Set_Menu_Item_Property(mi_id,CHECKED,PROPERTY_TRUE);
END IF;
END;
```

FIND_RELATION Built-in

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.

Syntax

FUNCTION FIND_RELATION
(*relation_name* VARCHAR2);

Built-in Type unrestricted function

Returns Relation

Enter Query Mode yes

Parameters

relation_name

Specifies a valid VARCHAR2 relation name.

FIND_RELATION Examples

```
/*

** Built-in: FIND_RELATION
** Example: Find the id of a relation before inquiring about
** multiple properties
*/
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;
```

FIND_REPORT_OBJECT Built-in

Description

Returns the report_id for a specified report. You can use this ID as a parameter for other built-ins, such as RUN_REPORT_OBJECT .

Syntax

FUNCTION FIND_REPORT_OBJECT
(*report_name* VARCHAR2*);*

Built-in Type unrestricted procedure

Returns `REPORT_OBJECT`

Enter Query Mode yes

Parameters

report_name

Specifies the unique name of the report to be found.

FIND_REPORT_OBJECT Example

```
DECLARE
 repid REPORT_OBJECT;
 v_rep VARCHAR2(100);
BEGIN
 repid := find_report_object('report4');
 v_rep := RUN_REPORT_OBJECT(repid);
 ....
END;
```

FIND_TAB_PAGE Built-in

Description

Searches the list of tab pages in a given tab canvas and returns a tab page ID when it finds a valid tab page with the given name. You must define a variable of type TAB_PAGE to accept the return value.

Syntax

FUNCTION FIND_TAB_PAGE
(tab_page_name VARCHAR2);

Built-in Type unrestricted function

Returns tab_page

Enter Query Mode yes

Parameters

tab_page_name

The unique tab page name. Datatype is VARCHAR2. (Note: if multiple tab canvases have tab pages with identical names, you must provide a fully-qualified name for the tab page (i.e., MY_TAB_CVS.TAB_PAGE_1).

FIND_TAB_PAGE Example

```
/* Use FIND_TAB_PAGE to find the ID of the top-most tab

** page on tab canvas TAB_PAGE_1, then use the ID to set
** properties of the tab page:
*/
DECLARE
tp_nm VARCHAR2(30);
tp_id TAB_PAGE;
```

```
BEGIN
tp_nm := GET_CANVAS_PROPERTY('tab_page_1', topmost_tab_page);
tp_id := FIND_TAB_PAGE(tp_nm);
SET_TAB_PAGE_PROPERTY(tp_id, visible, property_true);
SET_TAB_PAGE_PROPERTY(tp_id, label, 'Order Info');
END;
```

FIND_TIMER Built-in

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.

Syntax

FUNCTION FIND_TIMER
(*timer_name* VARCHAR2);

Built-in Type unrestricted function

Returns Timer

Enter Query Mode yes

Parameters

timer_name

Specifies a valid VARCHAR2 timer name.

FIND_TIMER 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_TREE_NODE Built-in

Description

Finds the next node in the tree whose label or value matches the search string.

Syntax

FUNCTION FIND_TREE_NODE
(*item_name* VARCHAR2,
search_string VARCHAR2,
search_type NUMBER,
search_by NUMBER,
search_root NODE,
start_point NODE);

FUNCTION FIND_TREE_NODE
(*item_id* ITEM,
search_string VARCHAR2,
search_type NUMBER,
search_by NUMBER,
search_root NODE,
start_point NODE);

Built-in Type unrestricted function

Returns NODE

Enter Query Mode no

Parameters

item_name

Specifies the name of the object created at design time. The data type of the name is VARCHAR2 string.

Item_id

Specifies the unique ID that Forms Developer 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.

search_string

Specifies the VARCHAR2 search string.

search_type

Specifies the NUMBER search type. Possible values are:

FIND_NEXT

FIND_NEXT_CHILD Searches only the children of the search_root node.

search_by

Specifies the NUMBER to search by. Possible values are:

NODE_LABEL

NODE_VALUE

search_root

Specifies the root of the search tree.

start_point

Specifies the starting point in the NODE search.

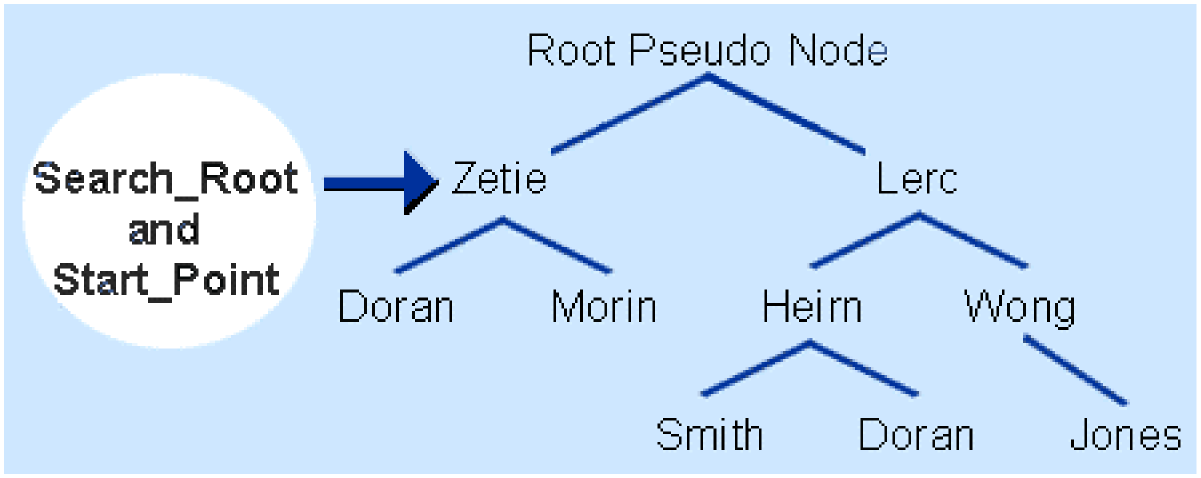
FIND_TREE_NODE Examples

```
/*

** Built-in: FIND_TREE_NODE
*/

-- This code finds a node with a label of "Doran"
-- within the subtree beginning with the node labeled
-- "Zetie".

DECLARE
htree ITEM;
find_node FTREE.NODE;
BEGIN
-- Find the tree itself.
htree := FIND_ITEM('tree_block.htree3');
-- Find the node with a label "Zetie".
find_node := FTREE.FIND_TREE_NODE(htree,
'Zetie',
FTREE.FIND_NEXT,
FTREE.NODE_LABEL,
FTREE.ROOT_NODE,
FTREE.ROOT_NODE);
```



```
-- Find the node with a label "Doran"
-- starting at the first occurance of "Zetie".
find_node := FTREE.FIND_TREE_NODE(htree,
'Doran',
FTREE.FIND_NEXT,
FTREE.NODE_LABEL,
find_node,
find_node);
IF NOT FTREE.ID_NULL(find_node)
then  ...
END IF;
END;
```

FIND_VA Built-in

Description

Searches the list of valid visual attributes in Forms Developer. When the given visual attribute is located, the subprogram
returns a visual attribute ID. You must return the ID to an appropriately typed variable. Define the variable with a type of
`VisualAttribute`.

Syntax

FUNCTION FIND_VA
(*va_name* PROPERTY);

Built-in Type unrestricted function

Returns `VisualAttribute`

Enter Query Mode yes

Parameters

va_name

The name you gave the visual attribute when you created it. The data type is `VARCHAR2`.

FIND_VIEW Built-in

Description

Searches the list of canvases and returns a view 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 ViewPort.

Syntax

FUNCTION FIND_VIEW
(*viewcanvas_name* VARCHAR2);

Built-in Type unrestricted function

Returns ViewPort

Enter Query Mode yes

Parameters

viewcanvas_name

Specifies the VARCHAR2 name of the canvas.

FIND_VIEW Examples

```
/*

** 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, VIEWPORT_X_POS, 30);
Set_View_Property(vw_id, VIEWPORT_Y_POS, 5);
Set_View_Property(vw_id, VISIBLE, PROPERTY_TRUE);
END;
```

FIND_WINDOW Built-in

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.

Syntax

FUNCTION FIND_WINDOW
(*window_name* VARCHAR2);

Built-in Type unrestricted function

Returns Window

Enter Query Mode yes

Parameters

window_name

Specifies the valid VARCHAR2 window name.

FIND_WINDOW Examples

```
/*

** 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 Built-in

Description

Navigates to the first record in the block's list of records.

Syntax

PROCEDURE FIRST_RECORD;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

None.

FIRST_RECORD Examples

```
/*

** 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 Built-in

Description

Returns a value that indicates the outcome of the action most recently performed during the current Runform session.

Outcome	Returned Value
success	FALSE
failure	TRUE
fatal error	FALSE

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.

Syntax

FUNCTION FORM_FAILURE;

Built-in Type unrestricted function

Returns BOOLEAN

Enter Query Mode yes

Parameters

None.

FORM_FAILURE 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;
```

Related topic

FORM_SUCCESS Built-in

FORM_FATAL Built-in

FORM_FATAL Built-in

Description

Returns the outcome of the action most recently performed during the current Runform session.

Outcome	Returned Value
success	FALSE
failure	FALSE
fatal error	TRUE

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.

Syntax

FUNCTION FORM_FATAL;

Built-in Type unrestricted function

Return Type:

BOOLEAN

Enter Query Mode yes

Parameters

None.

FORM_FATAL Examples

```
/*

** 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;
```

Related topic

FORM_SUCCESS Built-in

FORM_FAILURE Built-in

FORM_SUCCESS Built-in

Description

Returns the outcome of the action most recently performed during the current Runform session.

Outcome	Returned Value
success	TRUE
failure	FALSE
fatal error	FALSE

Syntax

FUNCTION FORM_SUCCESS;

Built-in Type unrestricted function

Return Type:

BOOLEAN

Enter Query Mode yes

Parameters

None.

Usage Notes

- 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.

- 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.
- On Microsoft Windows NT, when using HOST to execute a 16-bit application, the FORM_SUCCESS built-in will return TRUE whether the application succeeds or fails. This is a Microsoft a Win32 issue. 32-bit applications and OS commands will correctly return TRUE if executed sucessfully and FALSE if failed. Invalid commands will return FALSE.
- On Windows 95 platforms the FORM_SUCCESS built-in will always return TRUE for HOST commands which fail. This includes calls to command.com or OS functions, any 16-bit DOS or GUI application, or an invalid command. FORM_SUCCESS will return TRUE for 32-bit applications executed sucessfully and FALSE if failed.

FORM_SUCCESS Examples

```
/*

** 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;
```

Related topic

FORM_FAILURE Built-in

FORM_FATAL Built-in

FORMS_DDL Built-in

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 Forms Developer to process any pending changes.

Syntax

FUNCTION FORMS_DDL
(*statement* VARCHAR2);

Built-in Type unrestricted function

Enter Query Mode yes

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

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 Forms Developer to process any pending changes, as well as losing any locks Forms Developer 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.

FORMS_DDL 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 Oracle8'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 Forms Developer directly. (See Example 4.)

In addition, some DDL operations cannot be performed using FORMS_DDL, such as dropping a table or database link, if Forms Developer is holding a cursor open against the object being operated upon.

FORMS_DDL Examples

```
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 tmp(COL1 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:

```
/*

** 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:

```
/*

** 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;
```

GENERATE_SEQUENCE_NUMBER Built-in

Description

Initiates the default Forms Developer 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 Initial Value property to SEQUENCE.my_seq.NEXTVAL. By default, Forms Developer 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

Syntax

PROCEDURE GENERATE_SEQUENCE_NUMBER;

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

none

GENERATE_SEQUENCE_NUMBER Restrictions

Valid only in an On-Sequence-Number trigger.

GENERATE_SEQUENCE_NUMBER Examples

```
/*

** Built-in: GENERATE_SEQUENCE_NUMBER
** Example: Perform Forms Developer 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 Built-in

Description

Returns information about the current Forms Developer application. You must call the built-in once for each value you want to retrieve.

Syntax

FUNCTION GET_APPLICATION_PROPERTY
(*property* NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

Specify one of the following constants to return information about your application:

APPLICATION_INSTANCE Returns the pointer value of an instance handle. Only applies to the Microsoft Windows platform. For all other platforms, Forms Developer returns NULL.

BUILTIN_DATE_FORMAT Returns the current value of the Builtin date format mask (which is held in the Builtin_Date_Format property).

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, Forms Developer returns NULL.

CONNECT_STRING Returns the database connect string of the current operator. If the current operator does not have a connect string, Forms Developer 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 VARCHAR2 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 Forms Developer, not for connections established by On-Logon triggers.

DATETIME_LOCAL_TZ Specifies the local time zone region for `DATETIME` items.

DATETIME_SERVER_TZ Specifies the server time zone region for `DATETIME` items.

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.

ERROR_DATE/DATETIME_FORMAT Returns the current value of the error date or datetime format mask (which is established in the FORMSnn_Error_Date/Datetime_Format environment variable).

FLAG_USER_VALUE_TOO_LONG Returns the current value of this property, either 'TRUE' or 'FALSE', which controls truncation of user-entered values that exceed an item's Maximum Length property.

OPERATING_SYSTEM Returns the name of the operating system that is currently in use. Valid return values are MSWINDOWS, MSWINDOWS32, WIN32COMMON, UNIX, SunOS, MACINTOSH, VMS, and HP-UX.

OUTPUT_DATE/DATETIME_FORMAT Returns the current value of the output date or datetime format mask (which is established in the FORMSnn_Output_Date/Datetime_Format environment variable).

PASSWORD Returns the password of the current operator.

PLSQL_DATE_FORMAT Returns the current value of the PLSQL date format mask (which is held in the PLSQL_Date_Format property).

SAVEPOINT_NAME Returns the name of the last savepoint Forms Developer 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.

SSO_USERID Returns a string containing the Single Sign On user ID if the user has been authenticated via the Login Server; NULL
otherwise.

TIMER_NAME Returns the name of the most recently expired timer. Forms Developer returns NULL in response to this constant if there is no timer.

USER_DATE/DATETIME_FORMAT Returns the current value of the user date or datetime format mask (which is established in the FORMSnn_User_Date/Datetime_Format environment variable).

USER_INTERFACE Returns the name of the user interface that is currently in use. Valid return values are MOTIF, MACINTOSH, MSWINDOWS, MSWINDOWS32, WIN32COMMON, WEB, PM, BLOCKMODE, X, and UNKNOWN.

USER_NLS_CHARACTER_SET Returns the current value of the character set portion only of the USER_NLS_LANG environment variable defined for the form operator. If USER_NLS_LANG is not explicitly set, it defaults to the setting of NLS_LANG.

USER_NLS_DATE_FORMAT Obtains the current NLS date format mask. This is equal to the value of the NLS_DATE_FORMAT environment variable if it is set. If it is not set, a default value is derived based on the current NLS territory.

USER_NLS_LANG Returns the complete current value of the USER_NLS_LANG environment variable defined for the form operator, for national language support. If USER_NLS_LANG is not explicitly set, it defaults to the setting of NLS_LANG. 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 USER_NLS_LANG environment variable defined for the form operator. If USER_NLS_LANG is not explicitly set, it defaults to the setting of NLS_LANG.

USER_NLS_TERRITORY Returns the current value of the territory portion only of the USER_NLS_LANG environment variable defined for the form operator. If USER_NLS_LANG is not explicitly set, it defaults to the setting of NLS_LANG

USERNAME Returns the name of the current operator.

VERSION Returns the version number of the Forms Runtime that is running the application. This value is returned as a string e.g. 6.0.8.10.0
.

Usage Note

- To request a complete login, including an appended connect string, use the Username, Password, and Connect_String properties:

  ```
  http://myserver/forms90/f90servlet?form=scott/tiger@corpDB1
  ```

- Forms Developer returns the following string as the result of a call to GET_APPLICATION_PROPERTY(USERNAME):

  ```
  scott
  ```

- Forms Developer returns the following string as the result of a call to GET_APPLICATION_PROPERTY(PASSWORD):

  ```
  tiger
  ```

- Forms Developer returns the following string as the result of a call to GET_APPLICATION_PROPERTY(CONNECT_STRING):

  ```
  corpDB1
  ```

GET_APPLICATION_PROPERTY Restrictions

- To retrieve the timer name of the most recently executed timer, you must initiate a call to GET_APPLICATION_PROPERTY from within a When-Timer-Expired trigger. Otherwise, the results of the built-in are undefined.

GET_APPLICATION_PROPERTY Examples

Example 1

```
/*
```

```
** Built-in: GET_APPLICATION_PROPERTY
** Example: Determine the name of the timer that just
** expired, and based on the username perform a
** task.
** Trigger: When-Timer-Expired
*/
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_APPLICATION_PROPERTY
** Example: Capture the username and password of the
** currently logged-on user, for use in calling
** another Tool.
*/
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 Built-in

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.

Syntax

FUNCTION GET_BLOCK_PROPERTY
(*block_id* Block,
property NUMBER);

FUNCTION GET_BLOCK_PROPERTY
(*block_name* VARCHAR2*,*
property NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

block_id

> The unique ID Forms Developer assigned to the block when you created it. Datatype is BLOCK.

block_name

> The name you gave the block when you created it. Datatype is VARCHAR2.

property

Specify one of the following constants to return information about the given block:

ALL_RECORDS Specifies whether all the records matching the query criteria should be fetched into the data block when a query is executed.

BLOCKSCROLLBAR_X_POS Returns the x position of the block's scroll bar as a number specified in the form coordinate units indicated by the Coordinate System form property.

BLOCKSCROLLBAR_Y_POS Returns the y position of the block's scroll bar as a number specified in the form coordinate units indicated by the Coordinate System form property.

COLUMN_SECURITY Returns the VARCHAR2 value of TRUE if column security is set to Yes, and the VARCHAR2 string FALSE if it is set to No.

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 VARCHAR2 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 VARCHAR2 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 VARCHAR2 name of the named visual attribute of the given block.

CURRENT_ROW_BACKGROUND_COLOR The color of the object's background region.

CURRENT_ROW_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.

CURRENT_ROW_FONT_NAME The font family, or typeface, that should be used for text in the object. The list of fonts available is system-dependent.

CURRENT_ROW_FONT_SIZE The size of the font, specified in points.

CURRENT_ROW_FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

CURRENT_ROW_FONT_STYLE The style of the font.

CURRENT_ROW_FONT_WEIGHT The weight of the font.

CURRENT_ROW_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.

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 VARCHAR2 value TRUE if the Delete Allowed block property is Yes, FALSE if it is No. This property determines whether the operator or the application is allowed to delete records in the block.

DML_DATA_TARGET_NAME Returns the VARCHAR2 name of the block's DML data source.

DML_DATA_TARGET_TYPE Returns the VARCHAR2 value that indicates the current setting of the DML Data Target Type property. Return values for this property are NONE, TABLE, STORED PROCEDURE, or TRANSACTIONAL TRIGGER.

ENFORCE_PRIMARY_KEY Returns the VARCHAR2 value TRUE if the Enforce Primary Key property is set to Yes for the block. Otherwise, if the Enforce Primary Key property is set to No, this parameter returns the VARCHAR2 value FALSE.

ENTERABLE Returns the VARCHAR2 value TRUE if the block is enterable, that is, if any item in the block has its Enabled and Keyboard Navigable properties set to Yes. Returns the VARCHAR2 string FALSE if the block is not enterable.

FIRST_DETAIL_RELATION Returns the VARCHAR2 name of the first relation in which the given block is a detail. Returns NULL if one does not exist.

FIRST_ITEM Returns the VARCHAR2 name of the first item in the given block.

FIRST_MASTER_RELATION Returns the VARCHAR2 name of the first relation in which the given block is a master. Returns NULL if one does not exist.

INSERT_ALLOWED Returns the VARCHAR2 value TRUE if the Insert Allowed block property is Yes, FALSE if it is No. This property determines whether the operator or the application is allowed to insert records in the block.

KEY_MODE Returns the VARCHAR2 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.

LAST_QUERY Returns the SQL statement of the last query in the specified block.

LOCKING_MODE Returns the VARCHAR2 value IMMEDIATE if rows are to be locked immediately on a change to a base table item; otherwise, it returns the VARCHAR2 value DELAYED if row locks are to be attempted just prior to a commit.

MAX_QUERY_TIME Returns the VARCHAR2 value that indicates the current setting of the Maximum Query Time property. This property determines whether the operator can abort a query when the elapsed time of the query exceeds the value of this property.

MAX_RECORDS_FETCHED Returns a number representing the maximum number of records that can be fetched. This property is only useful when the Query All Records property is set to Yes.

NAVIGATION_STYLE Returns the VARCHAR2 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 VARCHAR2 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.

ONETIME_WHERE Returns the current one time where clause in effect for the specified block. It returns a NULL string if ONETIME_WHERE clause was not set before the query was executed.

OPTIMIZER_HINT Returns a hint in the form of a VARCHAR2 string that Forms Developer 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.

PRECOMPUTE_SUMMARIES Specifies that the value of any summarized item in a data block is computed before the normal query is issued on the block.

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 VARCHAR2 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.

QUERY_ALLOWED Returns the VARCHAR2 value TRUE if the Query Allowed block property is Yes, FALSE if it is No. This property determines whether the operator or the application is allowed to query records in the block.

QUERY_DATA_SOURCE_NAME Returns the VARCHAR2 name of the block's query data source.

QUERY_DATA_SOURCE_TYPE Returns the VARCHAR2 value that indicates the current setting of the Query Data Source Type property. Return values for this property are NONE, TABLE, STORED PROCEDURE, TRANSACTIONAL TRIGGER, or SUB-QUERY.

QUERY_HITS Returns the VARCHAR2 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 VARCHAR2 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 Forms Developer 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 Number of Records Displayed block property.

RECORDS_TO_FETCH Returns the number of records Forms Developer expects an On-Fetch trigger to fetch and create as queried records.

STATUS Returns the VARCHAR2 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 VARCHAR2 value TRUE if the Update Allowed block property is Yes, FALSE if it is No. 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 Only is set to No, 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.

GET_BLOCK_PROPERTY Examples

```
 /*

** 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
RETURN 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;
```

Related topic

SET_BLOCK_PROPERTY Built-in

GET_CANVAS_PROPERTY Built-in

Description

Returns the given canvas property for the given canvas. .

Syntax

FUNCTION GET_CANVAS_PROPERTY
(*canvas_id* Canvas*,*
property NUMBER);

FUNCTION GET_CANVAS_PROPERTY
(*canvas_name* VARCHAR2*,*
property NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

canvas_id

> The unique ID that Forms Developer assigns the canvas object when it creates it. Use the FIND_CANVAS built-in to return the ID to a variable with datatype of CANVAS.

canvas_name

> The name you gave the canvas object when you defined it.

property

> The property for which you want to get a value for the given canvas. You can enter the

following constants for return values:

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.

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_SPACING The width of the font, that is, the amount of space between characters (kerning).

FONT_STYLE The style of the font.

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.

HEIGHT Returns the height of the canvas, specified in the form coordinate units indicated by the Coordinate System form property.

TAB_PAGE_X_OFFSET Returns the distance between the left edge of the tab canvas and the left edge of the tab page. The value returned depends on the form coordinate system—pixel, centimeter, inch, or point.

TAB_PAGE_Y_OFFSET Returns the distance between the top edge of the tab canvas and the top edge of the tab page. The value returned depends on the form coordinate system—pixel, centimeter, inch, or point.

TOPMOST_TAB_PAGE Returns the name of the tab page currently top-most on the named tab canvas.

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.

GET_CANVAS_PROPERTY Examples

```
/*

** 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_CUSTOM_PROPERTY Built-in

Description

Gets the value of a user-defined property in a Java pluggable component.

Syntax

The built-in returns a VARCHAR2 value containing the string, numeric, or boolean data.

FUNCTION GET_CUSTOM_PROPERTY
(item_name VARCHAR2,
row_number NUMBER,
property VARCHAR2);

FUNCTION GET_CUSTOM_PROPERTY
(item_id ITEM,
row_number NUMBER,
property VARCHAR2);

Built-in Type unrestricted procedure

Returns VARCHAR2

Enter Query Mode yes

Parameters

*item*_id

> Specifies the unique ID that Forms Developer 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

> The name of the item associated with the target Java pluggable component. The name

can be in the form of either a VARCHAR2 literal or a variable set to the value of the name.

row_number

The row number of the instance of the item that you want to get. (Instance row numbers begin with 1.)

property

The particular property of the Java component that you want to get.

Usage Notes

- In the Java pluggable component, each custom property type must be represented by a single instance of the ID class, created by using `ID.registerProperty`.
- For each GET_CUSTOM_PROPERTY built-in executed in the form, the Java component's `getProperty` method is called.

GET_FILE_NAME Built-in

Description

Displays the standard open file dialog box where the user can select an existing file or specify a new file.

Syntax

```
FUNCTION GET_FILE_NAME
(directory_name VARCHAR2,
file_name VARCHAR2,
file_filter VARCHAR2,
message VARCHAR2,
dialog_type NUMBER,
select_file BOOLEAN;
```

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

directory_name

Specifies the name of the directory containing the file you want to open. The default value is NULL. If directory_name is NULL, subsequent invocations of the dialog may open the last directory visited.

file_name

Specifies the name of the file you want to open. The default value is NULL.

file_filter

Specifies that only particular files be shown. The default value is NULL. File filters take on different forms, and currently are ignored on the motif and character mode platforms.

message

Specifies the type of file that is being selected. The default value is NULL.

dialog_type

Specifies the intended dialog to OPEN_FILE or SAVE_FILE. The default value is OPEN_FILE.

select_file

Specifies whether the user is selecting files or directories. The default value is TRUE. If dialog_type is set to SAVE_FILE, select_file is internally set to TRUE.

GET_FORM_PROPERTY Built-in

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.

Syntax

FUNCTION GET_FORM_PROPERTY
(*formmodule_id* FormModule*,*
property NUMBER);

FUNCTION GET_FORM_PROPERTY
(*formmodule_name* VARCHAR2*,*
property NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

formmodule_id

Specifies the unique ID Forms Developer 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 VARCHAR2 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 VARCHAR2 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 VARCHAR2 name of the named visual attribute that should be used for the current row.

CURRENT_ROW_BACKGROUND_COLOR The color of the object's background region.

CURRENT_ROW_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.

CURRENT_ROW_FONT_NAME The font family, or typeface, that should be used for text in the object. The list of fonts available is system-dependent.

CURRENT_ROW_FONT_SIZE The size of the font, specified in points.

CURRENT_ROW_FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

CURRENT_ROW_FONT_STYLE The style of the font.

CURRENT_ROW_FONT_WEIGHT The weight of the font.

CURRENT_ROW_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.

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. Valid return values are TRUE, 4.5, and FALSE.

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.

FIRST_NAVIGATION_BLOCK Returns the name of the block into which Forms Developer 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.

FORM_NAME Returns the name of the form.

INTERACTION_MODE Returns the interaction mode for the form. Valid return values are BLOCKING or NONBLOCKING.

ISOLATION_MODE Returns the form’s isolation mode setting, either READ_COMMITTED or SERIALIZABLE.

LAST_BLOCK Returns the name of the block with the highest sequence number in the indicated form.

MAX_QUERY_TIME Returns the VARCHAR2 value that indicates the current setting of the Maximum Query Time property. This property determines whether the operator can abort a query when the elapsed time of the query exceeds the value of this property.

MAX_RECORDS_FETCHED Returns a number representing the maximum number of records that can be fetched. This property is only useful when the Query All Records property is set to Yes.

MODULE_NLS_CHARACTER_SET Returns the current value of the character set portion only of the DEVELOPER_NLS_LANG environment variable defined for the form. If DEVELOPER_NLS_LANG is not explicitly set, it defaults to the setting of NLS_LANG.

MODULE_NLS_LANG Returns the complete current value for national language support contained in the DEVELOPER_NLS_LANG environment variable defined for the form. If DEVELOPER_NLS_LANG is not explicitly set, it defaults to the setting of NLS_LANG. 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 DEVELOPER_NLS_LANG environment variable defined for the form. If DEVELOPER_NLS_LANG is not explicitly set, it defaults to the setting of NLS_LANG.

MODULE_NLS_TERRITORY Returns the current value of the territory portion only of the DEVELOPER_NLS_LANG environment variable defined for the form. If DEVELOPER_NLS_LANG is not explicitly set, it defaults to the setting of NLS_LANG.

SAVEPOINT_MODE Returns PROPERTY_ON or PROPERTY_OFF to indicate whether savepoints are supported in the data source.

VALIDATION Returns TRUE or FALSE to indicate whether default Forms Developer validation is enabled.

VALIDATION_UNIT Returns a VARCHAR2 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.

GET_FORM_PROPERTY Examples

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 Built-in

Description

Returns the VARCHAR2 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.

Syntax

FUNCTION GET_GROUP_CHAR_CELL
(*groupcolumn_id* GroupColumn,
row_number NUMBER);

FUNCTION GET_GROUP_CHAR_CELL
(*groupcolumn_name* VARCHAR2,
row_number NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

groupcolumn_id

Specifies the unique ID that Forms Developer 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 VARCHAR2 record group column name you gave the column when you defined it, preceded by the record group name and a dot, as in recordgroup_name.groupcolumn_name. 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.

GET_GROUP_CHAR_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.

GET_GROUP_CHAR_CELL Examples

```
/*

** 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;
gc_id GroupColumn;
col_val VARCHAR2(80);
Exit_Function Exception;
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.
*/
RAISE Exit_Function;
EXCEPTION
WHEN Exit_Function THEN
RETURN 0;
END;
```

GET_GROUP_DATE_CELL Built-in

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.

Syntax

FUNCTION GET_GROUP_DATE_CELL
(*groupcolumn_id* GroupColumn,
row_number NUMBER);

FUNCTION GET_GROUP_DATE_CELL
(*groupcolumn_name* VARCHAR2,
row_number NUMBER);

Built-in Type unrestricted function

Returns DATE

Enter Query Mode yes

Parameters

groupcolumn_id

Specifies the unique ID that Forms Developer 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 VARCHAR2 record group column name you gave the column when you defined it, preceded by the record group name and a dot, as in recordgroup_name.groupcolumn_name. 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.

GET_GROUP_DATE_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.

GET_GROUP_DATE_CELL Examples

```
/*

** 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 Built-in

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.

Syntax

FUNCTION GET_GROUP_NUMBER_CELL
(*groupcolumn_id* GroupColumn,
row_number NUMBER);

FUNCTION GET_GROUP_NUMBER_CELL
(*groupcolumn_name* VARCHAR2,
row_number NUMBER);

Built-in Type unrestricted function

Returns NUMBER

Enter Query Mode yes

Parameters

groupcolumn_id

Specifies the unique ID that Forms Developer 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 VARCHAR2 record group column name you gave the column when you defined it, preceded by the record group name and a dot, as in recordgroup_name.groupcolumn_name. 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.

GET_GROUP_NUMBER_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.

GET_GROUP_NUMBER_CELL Examples

```
/*

** 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 Built-in

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.

Syntax

FUNCTION GET_GROUP_RECORD_NUMBER
(groupcolumn_*id* GroupColumn*,*
cell_value NUMBER);

FUNCTION GET_GROUP_RECORD_NUMBER
(groupcolumn_*name* VARCHAR2*,*
cell_value NUMBER);

FUNCTION GET_GROUP_RECORD_NUMBER
(groupcolumn_*id* GroupColumn*,*
cell_value DATE);

FUNCTION GET_GROUP_RECORD_NUMBER
(groupcolumn_*name* VARCHAR2*,*
cell_value DATE);

FUNCTION GET_GROUP_RECORD_NUMBER
(groupcolumn_*id* GroupColumn*,*
cell_value VARCHAR2);

FUNCTION GET_GROUP_RECORD_NUMBER
(groupcolumn_*name* VARCHAR2*,*
cell_value VARCHAR2);

Built-in Type unrestricted function

Returns NUMBER

Enter Query Mode yes

Parameters

groupcolumn_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

cell_value

Specifies the value to find in the specified record group column. The data type of the name is VARCHAR2, NUMBER, or DATE.

GET_GROUP_RECORD_NUMBER Restriction

- The dataype of the cell_value parameter must match the datatype of the record group column. The comparison is case-sensitive for VARCHAR2 comparisons.

GET_GROUP_RECORD_NUMBER Example

```
/*

** Built-in: GET_GROUP_RECORD_NUMBER
** 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 Built-in

Description

Returns the number of rows in the record group.

Syntax

FUNCTION GET_GROUP_ROW_COUNT
(*recordgroup_id* RecordGroup);

FUNCTION GET_GROUP_ROW_COUNT
(*recordgroup_name* VARCHAR2);

Built-in Type unrestricted function

Returns NUMBER

Enter Query Mode yes

Parameters

recordgroup_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

GET_GROUP_ROW_COUNT Examples

```
/*
```

```
** 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 );
Message('The query retrieved '||to_CHAR(the_rowcount)||
' record(s)');
ELSE
Message('Error creating query record group.');
RAISE Form_Trigger_Failure;
END IF;
END;
```

GET_GROUP_SELECTION Built-in

Description

Retrieves the sequence number of the selected row for the given group.

Syntax

FUNCTION GET_GROUP_SELECTION
(*recordgroup_id* RecordGroup,
selection_number NUMBER);

FUNCTION GET_GROUP_SELECTION
(*recordgroup_name* VARCHAR2,
selection_number NUMBER);

Built-in Type unrestricted function

Returns NUMBER

Enter Query Mode yes

Parameters

recordgroup_id

> Specifies the unique ID that Forms Developer 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.

GET_GROUP_SELECTION Examples

```
/*

** 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 (VARCHAR2) 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 Built-in

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.

Syntax

FUNCTION GET_GROUP_SELECTION_COUNT
(*recordgroup_id* RecordGroup);

FUNCTION GET_GROUP_SELECTION_COUNT
(*recordgroup_name* VARCHAR2);

Built-in Type unrestricted function

Returns NUMBER

Enter Query Mode yes

Parameters

recordgroup_id

> Specifies the unique ID that Forms Developer 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.

GET_GROUP_SELECTION_COUNT Examples

```
/*
```

```
** Built-in: GET_GROUP_SELECTION_COUNT
** Example: See GET_GROUP_SELECTION
*/
```

GET_ITEM_INSTANCE_PROPERTY Built-in

Description

Returns property values for the specified item instance. GET_ITEM_INSTANCE_PROPERTY returns the *initial value* or the value *last specified* by SET_ITEM_INSTANCE_PROPERTY for the specified item instance. It does not return the *effective value* of a property (i.e. the value derived from combining properties specified at the item instance, item, and block levels). See SET_ITEM_INSTANCE_PROPERTY for information about effective property values.

Syntax

FUNCTION GET_ITEM_INSTANCE_PROPERTY
(*item_id* ITEM,
record_number NUMBER,
property NUMBER);

FUNCTION GET_ITEM_INSTANCE_PROPERTY
(*item_name* VARCHAR2,
record_number NUMBER,
property NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

item_id

The unique ID that Forms Developer assigns to the object when it creates it. Use the FIND_ITEM built-in to return the ID to a variable of datatype ITEM.

item_name

The name you gave the object when you created it.

record_number

A record number or the constant CURRENT_RECORD to indicate the record that currently has focus within the block.

property

The property the value of which you want to get for the given item. Valid properties are:

BORDER_BEVEL Returns RAISED, LOWERED, or PLAIN if the BORDER_BEVEL property is set to RAISED, LOWERED, or PLAIN, respectively at the item instance level. If BORDER_BEVEL is not specfied at the item instance level, this property returns " ".

INSERT_ALLOWED Returns the VARCHAR2 string TRUE if the item instance INSERT_ALLOWED property is set to true. Returns the string FALSE if the property is set to false.

NAVIGABLE Returns the VARCHAR2 string TRUE if the item instance NAVIGABLE property is set to true. Returns the string FALSE if the property is set to false.

REQUIRED Returns the VARCHAR2 string TRUE if the item instance REQUIRED property is set to true. Returns the string FALSE if the property is set to false.

SELECTED_RADIO_BUTTON Returns the label of the selected radio button within the radio group in the specified record. Returns NULL if the radio group for the specified record does not have a selected radio button or if the specified record has been scrolled out of view.

UPDATE_ALLOWED Returns the VARCHAR2 string TRUE if the item instance UPDATE_ALLOWED property is set to true. Returns the string FALSE if the property is set to false.

VISUAL_ATTRIBUTE Returns the name of the visual attribute currently in force. If no named visual attribute is assigned to the item instance, returns DEFAULT for a default visual attribute. Returns '' if VISUAL_ATTRIBUTE is not specified at the item instance level.

GET_ITEM_PROPERTY Built-in

Description

Returns property values for the specified item. Note that in some cases you may be able to *get*—but not *set*—certain object properties.

Syntax

FUNCTION GET_ITEM_PROPERTY
(*item_id*, ITEM
property NUMBER);

FUNCTION GET_ITEM_PROPERTY
(*item_name* VARCHAR2,
property NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

item_id

The unique ID that Forms Developer assigns to the object when it creates it. Use the FIND_ITEM built-in to return the ID to a variable of datatype ITEM.

item_name

The name you gave the object when you created it.

property

The property the value of which you want to get for the given item. Valid properties are:

AUTO_HINT Returns the VARCHAR2 string TRUE if the Automatic Hint property is set to Yes, and the VARCHAR2 string FALSE if it is set to No.

AUTO_SKIP Returns the VARCHAR2 string TRUE if Automatic Skip is set to Yes for the item, and the string FALSE if it is set to No for the item.

BACKGROUND_COLOR The color of the object's background region.

BLOCK_NAME Returns the VARCHAR2 block name for the item.

BORDER_BEVEL Returns RAISED, LOWERED, or PLAIN if the BORDER_BEVEL property is set to RAISED, LOWERED, or PLAIN, respectively at the item level.

CASE_INSENSITIVE_QUERY Returns the VARCHAR2 string TRUE if this property is set to Yes for the item, and the string FALSE if the property is set to No.

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.

COLUMN_NAME Returns the name of the column in the database to which the datablock item is associated.

CONCEAL_DATA Returns the VARCHAR2 string TRUE if the text an operator types into the text item is to be hidden, and the VARCHAR2 string FALSE if the text an operator types into the text item is to be displayed.

CURRENT_RECORD_ATTRIBUTE Returns the VARCHAR2 name of the named visual attribute of the given item.

CURRENT_ROW_BACKGROUND_COLOR The color of the object's background region.

CURRENT_ROW_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.

CURRENT_ROW_FONT_NAME The font family, or typeface, that should be used for text in the object. The list of fonts available is system-dependent.

CURRENT_ROW_FONT_SIZE The size of the font, specified in points.

CURRENT_ROW_FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

CURRENT_ROW_FONT_STYLE The style of the font.

CURRENT_ROW_FONT_WEIGHT The weight of the font.

CURRENT_ROW_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.

DATA_LENGTH_SEMANTICS Determines the semantics of the `MAX_LENGTH` and `QUERY_LENGTH` properties, when it is used to enforce the maximum size of the internal value of a character 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 VARCHAR2 string TRUE or FALSE.

ECHO Returns the VARCHAR2 string TRUE if the Conceal Data property is set to No for the item, and the VARCHAR2 string FALSE if the Conceal Data property is set to Yes 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 Yes, FALSE if set to No.

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.

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 hundredths of a point (i.e., an item with a font size of 8 points will return 800).

FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

FONT_STYLE The style of the font.

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.

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 VARCHAR2 value TRUE if the button is defined as iconic, and the VARCHAR2 value FALSE if it is not an iconic button.

IMAGE_DEPTH Returns the color depth of the specified image item.

IMAGE_FORMAT Returns the format of the specified image item.

INSERT_ALLOWED Returns the VARCHAR2 string TRUE if the INSERT_ALLOWED property is set to true at the item level. Returns the string FALSE if the property is set to false.

ITEM_CANVAS Returns the name of the canvas to which the item is assigned.

ITEM_IS_VALID Returns the VARCHAR2 string TRUE if the current item is valid, and the VARCHAR2 string FALSE if the current item is not valid.

ITEM_NAME Returns the name of the item.

ITEM_TAB_PAGE Returns the name of the tab page to which the item is assigned. Note that the item must be assigned to a tab canvas in order for Forms Developer to return the name of the item's tab page.

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, RADIO GROUP if the item is a radio group, TEXT ITEM if the item is a text item, and USER AREA if the item is a user area.

JUSTIFICATION Returns the text alignment for text items and display items only. Valid return values are START, END, LEFT, CENTER, RIGHT.

KEEP_POSITION Returns the VARCHAR2 string TRUE if the cursor is to re-enter at the identical location it was in when it left the item, and the VARCHAR2 string FALSE if the cursor is to re-enter the item at its default position.

LABEL Returns the VARCHAR2 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 VARCHAR2 string TRUE if the item is a text item to which a list of values (LOV) is attached; otherwise returns the VARCHAR2 string FALSE.

LOCK_RECORD_ON_CHANGE Returns the VARCHAR2 string TRUE if Forms Developer should attempt to lock a row based on a potential change to this item, and returns the VARCHAR2 string FALSE if no lock should be attempted.

LOV_NAME Returns the VARCHAR2 name of the LOV associated with the given item. If the LOV name does not exist, you will get an error message.

LOV_X_POS Returns the x coordinate of the LOV associated with the text item. (Corresponds to the List X Position property.)

LOV_Y_POS Returns the y coordinate of the LOV associated with the text item. (Corresponds to the List Y Position property.)

MAX_LENGTH Returns the maximum length setting for the item. The value is returned as a whole NUMBER.

MERGE_CURRENT_ROW_VA Merges the contents of the specified visual attribute with the current row's visual attribute (rather than replacing it).

MERGE_TOOLTIP_ATTRIBUTE Merges the contents of the specified visual attribute with the tooltip's current visual attribute (rather than replacing it).

MERGE_VISUAL_ATTRIBUTE Merges the contents of the specified visual attribute with the object's current visual attribute (rather than replacing it).

MOUSE_NAVIGATE Returns the VARCHAR2 string TRUE if Mouse Navigate is set to Yes for the item, and the VARCHAR2 string FALSE if it is set to No for the item.

MULTI_LINE Returns the VARCHAR2 value TRUE if the item is a multi-line text item, and the VARCHAR2 string FALSE if it is a single-line text item.

NAVIGABLE Returns the VARCHAR2 string TRUE if the NAVIGABLE property is set to true at the item level. Returns the string FALSE if the property is set to false.

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.

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 VARCHAR2 value TRUE if the item is a primary key, and the VARCHAR2 string FALSE if it is not.

PROMPT_ALIGNMENT_OFFSET Returns the distance between the item and its prompt as a VARCHAR2 value.

PROMPT_BACKGROUND_COLOR The color of the object's background region.

PROMPT_DISPLAY_STYLE Returns the prompt’s display style, either FIRST_RECORD, HIDDEN, or ALL_RECORDS.

PROMPT_EDGE Returns a value that indicates which edge the item’s prompt is attached to, either START, END, TOP, or BOTTOM.

PROMPT_EDGE_ALIGNMENT Returns a value that indicates which edge the item’s prompt is aligned to, either START, END, or CENTER.

PROMPT_EDGE_OFFSET Returns the distance between the item and its prompt as a VARCHAR2 value.

PROMPT_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.

PROMPT_FONT_NAME The font family, or typeface, that should be used for text in the object. The list of fonts available is system-dependent.

PROMPT_FONT_SIZE The size of the font, specified in points.

PROMPT_FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

PROMPT_FONT_STYLE The style of the font.

PROMPT_FONT_WEIGHT The weight of the font.

PROMPT_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.

PROMPT_TEXT Returns the text label that displays for an item.

PROMPT_TEXT_ALIGNMENT Returns a value that indicates how the prompt is justified, either START, LEFT, RIGHT, CENTER, or END.

PROMPT_VISUAL_ATTRIBUTE Returns a value that indicates the prompt's named visual attribute .

QUERYABLE Returns the VARCHAR2 string TRUE if the item can be included in a query, and the VARCHAR2 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 VARCHAR2 string TRUE if property is set to Yes for the item, and the VARCHAR2 string FALSE if the property is set to No 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 VARCHAR2 string TRUE if the REQUIRED property is set to true at the item level. Returns the string FALSE if the property is set to false.

SCROLLBAR Returns the VARCHAR2 string TRUE if the Show Scroll Bar property is Yes, and the VARCHAR2 string FALSE if the Show Scroll Bar property is No.

TOOLTIP_BACKGROUND_COLOR The color of the object's background region.

TOOLTIP_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.

TOOLTIP_FONT_NAME The font family, or typeface, that should be used for text in the object. The list of fonts available is system-dependent.

TOOLTIP_FONT_SIZE The size of the font, specified in points.

TOOLTIP_FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

TOOLTIP_FONT_STYLE The style of the font.

TOOLTIP_FONT_WEIGHT The weight of the font.

TOOLTIP_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.

TOOLTIP_TEXT Returns the item’s tooltip text.

UPDATE_ALLOWED Returns the VARCHAR2 string TRUE if the UPDATE_ALLOWED property is set to true at the item level. Returns the string FALSE if the property is set to false.

UPDATE_COLUMN Returns the VARCHAR2 string TRUE if Forms Developer should treat the item as updated, and FALSE if it should not.

UPDATE_NULL Returns the VARCHAR2 string TRUE if the item should be updated only if it is NULL, and the VARCHAR2 string FALSE if it can always be updated. (Corresponds to the Update if NULL property.)

UPDATE_PERMISSION Returns the VARCHAR2 string TRUE if the UPDATE_PERMISSION property is set to ON, turning on the item's UPDATEABLE and UPDATE_NULL properties. The VARCHAR2 string FALSE indicates that UPDATEABLE and UPDATE_NULL are turned off.

VALIDATE_FROM_LIST Returns the VARCHAR2 string TRUE if Forms Developer should validate the value of the text item against the values in the attached LOV; otherwise returns the VARCHAR2 string FALSE.

VISIBLE Returns the VARCHAR2 string TRUE if the property is set to Yes for the item, and the VARCHAR2 string FALSE if the property is set to No for the item.

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. .

WIDTH Returns the width of the item.

WINDOW_HANDLE Returns the a unique internal VARCHAR2 constant that is used to refer to objects. Returns the VARCHAR2 value ‘0’ if the platform is not Microsoft Windows.

WRAP_STYLE Returns VARCHAR2 if the item has wrap style set to VARCHAR2, 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_ITEM_PROPERTY 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.

GET_ITEM_PROPERTY Examples

```
/*

** 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;
/*
** If we get here we wrapped all the way around the
** block's item names
*/
wrapped := TRUE;
RAISE Exit_Procedure;
EXCEPTION
WHEN Exit_Procedure THEN
/*
** If we found a required item and we didn't come back
** to the item we started in, then navigate there
*/
IF found AND NOT wrapped THEN
Go_Item(cur_itm);
END IF;
END;
```

GET_LIST_ELEMENT_COUNT Built-in

Description

Returns the total number of list item elements in a list, including elements with NULL values.

Syntax

FUNCTION GET_LIST_ELEMENT_COUNT
(*list_id* Item);

FUNCTION GET_LIST_ELEMENT_COUNT
(*list_name* VARCHAR2);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

list_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

GET_LIST_ELEMENT_COUNT Examples

```
/*
```

```
** 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 VARCHAR2(50);
list_element_value VARCHAR2(50);
list_element_to_add VARCHAR2(50);
list_value_to_add VARCHAR2(50);
element_match VARCHAR2(5) := 'TRUE';
value_match VARCHAR2(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 Built-in

Description

Returns information about the requested list element label.

Syntax

FUNCTION GET_LIST_ELEMENT_LABEL
(*list_id* ITEM,
list_name VARCHAR2,
list_index NUMBER);

FUNCTION GET_LIST_ELEMENT_LABEL
(*list_name* VARCHAR2,
list_index NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

list_id

> Specifies the unique ID that Forms Developer 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 VARCHAR2.

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.

GET_LIST_ELEMENT_LABEL Examples

```
/*

** Built-in: GET_LIST_ELEMENT_LABEL
** Example: See GET_LIST_ELEMENT_COUNT
*/
```

GET_LIST_ELEMENT_VALUE Built-in

Description

Returns the value associated with the specified list item element.

Syntax

FUNCTION GET_LIST_ELEMENT_VALUE
(*list_id* ITEM*,*
list_index NUMBER);

FUNCTION GET_LIST_ELEMENT_VALUE
(*list_name* VARCHAR2*,*
list_index NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

list_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

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.

GET_LIST_ELEMENT_VALUE Examples

```
/*

** Built-in: GET_LIST_ELEMENT_VALUE
** Example: See GET_LIST_ELEMENT_COUNT
*/
```

GET_LOV_PROPERTY Built-in

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.

Syntax

FUNCTION GET_LOV_PROPERTY
(*lov_id* LOV,
property NUMBER);

FUNCTION GET_LOV_PROPERTY
(*lov_name* VARCHAR2,
property NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

lov_id

Specifies the unique ID that Forms Developer 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 VARCHAR2 string TRUE if the property is set to Yes; that is, if Forms Developer re-executes the query each time the LOV is invoked. Returns the VARCHAR2 string FALSE if the property is set to No.

GROUP_NAME Returns the name of the record group currently associated with this LOV. The data type of the name is VARCHAR2.

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.

GET_LOV_PROPERTY Examples

```
/*

** Built-in: GET_LOV_PROPERTY
** 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 Built-in

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.

Syntax

FUNCTION GET_MENU_ITEM_PROPERTY
(*menuitem_id* MenuItem,
property NUMBER);

FUNCTION GET_MENU_ITEM_PROPERTY
(*menu_name.menuitem_name* VARCHAR2,
property NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

menuitem_id

The unique ID Forms Developer assigns to the menu item when you create it. Use the FIND_MENU_ITEM built-in to return the ID to an appropriately typed variable. Datatype is MenuItem.

menu_name.menuitem_name

The name you gave the menu item when you created it. If you specify the menu item by name, include the qualifying menu name, for example, menu_name.menuitem_name. Datatype is VARCHAR2.

property

Specify one of the following constants to retrieve information about the menu item:

CHECKED Returns the VARCHAR2 string TRUE if a check box menu item is checked, FALSE if it is unchecked. Returns the VARCHAR2 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.

ENABLED Returns the VARCHAR2 string TRUE if a menu item is enabled, FALSE if it is disabled (thus grayed out and unavailable).

ICON_NAME Returns the file name of the icon resource associated with a menu item having the Icon in Menu property set to TRUE.

LABEL Returns the VARCHAR2 string for the menu item label.

VISIBLE Returns the VARCHAR2 string TRUE if a menu item is visible, FALSE if it is hidden from view.

GET_MENU_ITEM_PROPERTY Examples

```
/*

** 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;
```

Related topic

SET_MENU_ITEM_PROPERTY built-in

GET_MESSAGE Built-in

Description

Returns the current message, regardless of type.

Syntax

FUNCTION GET_MESSAGE;

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

none

GET_MESSAGE Restrictions

GET_MESSAGE is only instantiated when a message is directed to the display device, either by Forms Developer 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 for more information.

GET_MESSAGE Examples

```
/*

** Built-in: GET_MESSAGE
** Example: Capture the contents of the Message Line in a
** local variable
*/
DECLARE
string_var VARCHAR2(200);
BEGIN
```

```
string_var := Get_Message;
END;
```

GET_PARAMETER_ATTR Built-in

Description

Returns the current value and type of an indicated parameter in an indicated parameter list.

Syntax

FUNCTION GET_PARAMETER_ATTR
(*list* VARCHAR2,
key VARCHAR2,
paramtype NUMBER,
value VARCHAR2);

FUNCTION GET_PARAMETER_ATTR
(*name* VARCHAR2,
key VARCHAR2,
paramtype NUMBER,
value VARCHAR2);

Built-in Type unrestricted procedure that returns two OUT parameters

Enter Query Mode yes

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 VARCHAR2 name of the parameter list.

key

The VARCHAR2 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 VARCHAR2. 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.

Related topic

SET_PARAMETER_ATTR built-in

GET_PARAMETER_LIST Built-in

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.

Syntax

FUNCTION GET_PARAMETER_LIST
(*name* VARCHAR2);

Built-in Type unrestricted function

Returns ParamList

Enter Query Mode yes

Parameters

name

Specifies a valid VARCHAR2 parameter list name.

GET_PARAMETER_LIST Examples

See CREATE_PARAMETER_LIST

GET_RADIO_BUTTON_PROPERTY Built-in

Description

Returns information about a specified radio button.

Syntax

FUNCTION GET_RADIO_BUTTON_PROPERTY
(*item_id* ITEM,
button_name VARCHAR2,
property NUMBER);

FUNCTION GET_RADIO_BUTTON_PROPERTY(
item_name VARCHAR2,
button_name VARCHAR2,
property NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

item_id

Specifies the radio group item ID. Forms Developer 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 VARCHAR2.

button_name

Specifies the name of the radio button whose property you want. The data type of the name is VARCHAR2.

property

Specifies the property for which you want the current state. The possible property constants you can indicate are as follows:

BACKGROUND_COLOR The color of the object's background region.

ENABLED Returns the VARCHAR2 string TRUE if property is set to Yes, and the VARCHAR2 string FALSE if property is set to No.

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.

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_SPACING The width of the font, that is, the amount of space between characters (kerning).

FONT_STYLE The style of the font.

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.

HEIGHT Returns the height of the radio button. The value is returned as a VARCHAR2 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.

PROMPT_BACKGROUND_COLOR The color of the object's background region.

PROMPT_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.

PROMPT_FONT_NAME The font family, or typeface, that should be used for text in the object. The list of fonts available is system-dependent.

PROMPT_FONT_SIZE The size of the font, specified in points.

PROMPT_FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

PROMPT_FONT_STYLE The style of the font.

PROMPT_FONT_WEIGHT The weight of the font.

PROMPT_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.

VISIBLE Returns the VARCHAR2 string TRUE if property is set to Yes, returns and the VARCHAR2 string FALSE if property is set to No.

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 VARCHAR2 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 VARCHAR2 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 VARCHAR2 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 VARCHAR2 and is expressed in the units defined by the form module Coordinate System property.

GET_RADIO_BUTTON_PROPERTY Examples

```
/*

** Built-in: GET_RADIO_BUTTON_PROPERTY
** 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', VISIBLE);
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 Built-in

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, Forms Developer issues an error. For example, you must pass a valid record number as the argument to the record_number parameter.

Syntax

FUNCTION GET_RECORD_PROPERTY
(*record_number* NUMBER,
block_name VARCHAR2,
property NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

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.

	Record Status	***Block Status***	***Form Status***
Created record with no modified fields	NEW	<N|Q|C>	<N|Q|C>
...and all records in current block are NEW	NEW	NEW	<N|Q|C>

...and all blocks in current form are NEW	NEW	NEW	NEW

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.

Type of Block/Type of Item Changed	***Record Status Before Change***	***Record Status After Change***	***Block Status***	***Form Status***
In a Base Table Block: Change a Base Table Item	NEW	INSERT	CHANGED	CHANGED
In a Base Table Block:Change a Base Table Item	QUERY	CHANGED	CHANGED	CHANGED
In a Base Table Block:Change a Control Item	QUERY	QUERY	<Q|C>	<Q|C>
...and no record in current block is changed		QUERY	QUERY	<Q|C>
...and no block in current form is changed		QUERY	QUERY	QUERY
In a Base Table Block: Change a Control Item	NEW	INSERT	<Q|C>	<Q|C>
In a Control Block: Change a Control Item	NEW	INSERT	<Q>	<Q|C>
...and no record in current block is changed		INSERT	QUERY	<Q|C>
...and no block in current form is changed		INSERT	QUERY	QUERY

Note:

In general, any assignment to a database item will change a record's status from QUERY to CHANGED (or from NEW to INSERT), even if the value being assigned is the same as the previous value. Passing an item to a procedure as OUT or IN OUT parameter counts as an assignment to it.

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, Forms Developer 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 Examples

```
/*

** 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_RELATION_PROPERTY Built-in

Description

Returns the state of the given relation property.

Syntax

FUNCTION GET_RELATION_PROPERTY
(*relation_id* Relation,
property NUMBER);

FUNCTION GET_RELATION_PROPERTY
(*relation_name* VARCHAR2,
property NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

relation_id

Specifies the unique ID Forms Developer 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 VARCHAR2 name you gave to the relation when you defined it, or the name that Forms Developer 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 VARCHAR2 value TRUE if the Automatic Query relation property is Yes, FALSE if it is No. When the Deferred relation property is set to Yes, this property determines whether Forms Developer automatically populates the detail block when a different record becomes the current record in the master block.

DEFERRED_COORDINATION Returns the VARCHAR2 value TRUE if the Deferred relation property is Yes, FALSE if it is No. 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 VARCHAR2 name of the detail block in the given master-detail relationship.

MASTER_DELETES Returns one of the following VARCHAR2 values to indicate the current setting of the block's Delete Record Behavior property: NON_ISOLATED, ISOLATED, or CASCADING.

MASTER_NAME Returns the VARCHAR2 name of the master block in the given master-detail relationship.

NEXT_DETAIL_RELATION Returns the VARCHAR2 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 VARCHAR2 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 VARCHAR2 value TRUE if this relation property is Yes, FALSE if it is No. When set to Yes, Forms Developer 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.

GET_RELATION_PROPERTY Examples

```
/*

** 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_Relation_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;
```

Related topic

SET_RELATION_PROPERTY built-in

GET_REPORT_OBJECT_PROPERTY Built-in

Description

Programmatically obtain a property of a report object.

Syntax

FUNCTION GET_REPORT_OBJECT_PROPERTY
(*report_id* REPORT_OBJECT,
property NUMBER);

FUNCTION GET_REPORT_OBJECT_PROPERTY
(*report_name* VARCHAR2*,*
property NUMBER);

Built-in Type unrestricted procedure

Returns VARCHAR2

Enter Query Mode yes

Parameters

report_id

Specifies the unique ID of the report. You can get the report ID for a particular report using FIND_REPORT_OBJECT .

report_name

Specifies the unique name of the report.

property

One of the following constants:

REPORT_EXECUTION_MODE: Returns a string value of the report execution mode, either BATCH or RUNTIME

REPORT_COMM_MODE: Returns a string value of the report communication mode, either SYNCHRONOUS or ASYNCHRONOUS

REPORT_DESTYPE: Returns a string value of the report destination type, either PREVIEW, FILE, PRINTER, MAIL, CACHE or SCREEN

REPORT_FILENAME: Returns a string value of the report filename

REPORT_SOURCE_BLOCK: Returns a string value of the report source block name

REPORT_QUERY_NAME: Returns a string value of the report query name

REPORT_DESNAME: Returns a string value of the report destination name

REPORT_DESFORMAT: Returns a string value of the report destination format

REPORT_SERVER: Returns a string value of the report server name

REPORT_OTHER: Returns a string value of the other user-specified report properties

Usage Note

- GET_REPORT_OBJECT_PROPERTY returns a string value for all properties. In contrast, SET_REPORT_OBJECT_PROPERTY sets properties using constant or string values. The value type depends on the particular property being set.

GET_REPORT_OBJECT_PROPERTY Example

```
DECLARE
 repid REPORT_OBJECT;
 report_prop VARCHAR2(20);
BEGIN
 repid := find_report_object('report4');
 report_prop := get_report_object_property(repid,
REPORT_EXECUTION_MODE);
 message('REPORT EXECUTION MODE PROPERTY IS ' || report_prop);
 report_prop := get_report_object_property(repid, REPORT_COMM_MODE);
 message('REPORT COMM_MODE PROPERTY IS ' || report_prop);
 report_prop := get_report_object_property(repid, REPORT_DESTYPE);
 message('REPORT DESTYPE PROPERTY IS ' || report_prop);
 report_prop := get_report_object_property(repid, REPORT_FILENAME);
 message('REPORT_FILENAME PROPERTY IS ' || report_prop);
END;
```

GET_TAB_PAGE_PROPERTY Built-in

Description

Returns property values for a specified tab page.

Syntax

FUNCTION GET_TAB_PAGE_PROPERTY
(*tab_page_id* TAB_PAGE*,*
property NUMBER);

FUNCTION GET_TAB_PAGE_PROPERTY
(*tab_page_name* VARCHAR2*,*
property NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

tab_page_id The unique ID Forms Developer assigned to the tab page object when you created it. Use the FIND_TAB_PAGE built-in to return the ID to a variable of datatype TAB_PAGE.

tab page_name The name you gave the tab page object when you created it. Note: if two tab pages in the same form module share the same name, you must provide the canvas and tab page (e.g., CVS_1.TAB_PG_1).

property The property the value of which you want to get for the given tab page. The possible properties are as follows:

BACKGROUND_COLOR The color of the object's background region.

CANVAS_NAME Returns the VARCHAR2 name of the canvas to which the tab page belongs.

ENABLED Returns the VARCHAR2 string TRUE if a tab page is enabled, FALSE if it is disabled (i.e., greyed out and unavailable).

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.

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_SPACING The width of the font, that is, the amount of space between characters (kerning).

FONT_STYLE The style of the font.

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.

LABEL Returns the VARCHAR2 string for the tab page label.

VISIBLE Returns the VARCHAR2 value TRUE if the tab page is visible, FALSE if it is not. A tab page is reported visible if it is currently mapped to the screen, even if it is entirely hidden behind another tab page.

VISUAL_ATTRIBUTE Returns the name of the visual attribute currently in force. If no named visual attribute is assigned to the tab page, returns CUSTOM for a custom visual attribute or DEFAULT for a default visual attribute.

GET_TAB_PAGE_PROPERTY Examples

```
/* Use FIND_TAB_PAGE and GET_TAB_PAGE_PROPERTY to check

** if a tab page is enabled:
*/
DECLARE
tp_id TAB_PAGE;
live VARCHAR2(32);
```

```
BEGIN
tp_id := FIND_TAB_PAGE('tab_page_1');
live := GET_TAB_PAGE_PROPERTY(tp_id, enabled);
END;
```

GET_TREE_NODE_PARENT Built-in

Description

Returns the parent of the specified node.

Syntax

FUNCTION GET_TREE_NODE_PARENT
(*item_name* VARCHAR2
node NODE);

FUNCTION GET_TREE_NODE_PARENT
(*item_id* ITEM
node NODE);

Returns FTREE.NODE

Built-in Type unrestricted function

Enter Query Mode no

Parameters

item_name	Specifies the name of the object created at design time. The data type of the name is VARCHAR2 string.
Item_id	Specifies the unique ID that Forms Developer 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.
node	Specifies a valid node.

GET_TREE_NODE_PARENT Example

```
/*

** Built-in: GET_TREE_NODE_PARENT
*/

-- This code could be used in a WHEN-TREE-NODE-SELECTED
-- trigger to locate the parent of the node that was
-- clicked on.
DECLARE
htree ITEM;
parent_node FTREE.NODE;
BEGIN
-- Find the tree itself.
htree := Find_Item('tree_block.htree3');
-- Get the parent of the node clicked on.
parent_node := Ftree.Get_Tree_Node_Parent(htree,
:SYSTEM.TRIGGER_NODE);
...
END;
```

GET_TREE_NODE_PROPERTY Built-in

Description

Returns the value of the specified property of the hierarchical tree node.

Syntax

FUNCTION GET_TREE_NODE_PROPERTY
(*item_name* VARCHAR2,
node NODE,
property NUMBER);

FUNCTION GET_TREE_NODE_PROPERTY
(*item_id* ITEM,
node NODE,
property NUMBER);

Returns VARCHAR2

Built-in Type unrestricted function

Enter Query Mode no

Parameters

item_name	Specifies the name of the object created at design time. The data type of the name is VARCHAR2 string.
Item_id	Specifies the unique ID that Forms Developer 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.
node	Specifies a valid node.

property	Specify one of the following properties:
	NODE_STATE Returns the state of the hierarchical tree node. This is either EXPANDED_NODE, COLLAPSED_NODE, or LEAF_NODE.
	NODE_DEPTH Returns the nesting level of the hierarchical tree node.
	NODE_LABEL Returns the label
	NODE_ICON Returns the icon name
	NODE_VALUE Returns the value of the hierarchical tree node.

GET_TREE_NODE_PROPERTY Example

```
/*

** Built-in: GET_TREE_NODE_PROPERTY
*/

-- This code could be used in a WHEN-TREE-NODE-SELECTED
-- trigger to return the value of the node that was
-- clicked on.
DECLARE
htree ITEM;
node_value VARCHAR2(100);
BEGIN
-- Find the tree itself.
htree := Find_Item('tree_block.htree3');
-- Find the value of the node clicked on.
node_value := Ftree.Get_Tree_Node_Property(htree,
:SYSTEM.TRIGGER_NODE, Ftree.NODE_VALUE);
...
END;
```

GET_TREE_PROPERTY Built-in

Description

Returns property values of the specified hierarchical tree.

Syntax

FUNCTION GET_TREE_PROPERTY
(*item_name* VARCHAR2,
property NUMBER);

FUNCTION GET_TREE_PROPERTY
(*item_id* ITEM,
property NUMBER);

Returns VARCHAR2

Built-in Type unrestricted function

Enter Query Mode no

Parameters

item_name	Specifies the name you gave the object when you created it. The data type of the name is VARCHAR2 string.
Item_id	Specifies the unique ID that Forms Developer 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.

property Specify one of the following properties:

DATASOURCE Returns the source used to initially populate the hierarchical tree, either in Forms Developer or by using the SET_TREE_PROPERTY built-in. Returns EXTERNAL if neither property was set in Forms Developer.

RECORD_GROUP Returns the RecordGroup used to initially populate the hierarchical tree, either in Forms Developer or by using the SET_TREE_PROPERTY built-in. This may be a null string.

QUERY_TEXT Returns the text of the query used to initially populate the hierarchical tree, either in Forms Developer or by using the SET_TREE_PROPERTY built-in.. This may be a null string.

NODE_COUNT Returns the number of rows in the hierarchical tree data set.

SELECTION_COUNT Returns the number of selected rows in the hierarchical tree.

ALLOW_EMPTY_BRANCHES Returns the character string TRUE or FALSE.

ALLOW_MULTI-SELECT Returns the character string TRUE or FALSE.

Usage Notes

The values returned by datasource RECORD_GROUP and QUERY_TEXT do not necessarily reflect the current data or state of the tree. The values returned are those that were set in Forms Developer and not those set using the SET_TREE_PROPERTY built-in.

GET_TREE_PROPERTY Example

```
/*

** Built-in: GET_TREE_PROPERTY
*/

-- This code could be used to find out how many nodes are
-- in a given tree.
DECLARE
htree ITEM;
node_count NUMBER;
BEGIN
-- Find the tree itself.
htree := FIND_ITEM('tree_block.htree3');
-- Get the node count of the tree.
```

```
node_count := FTREE.GET_TREE_PROPERTY(htree, FTREE.NODE_COUNT);
...
END;
```

GET_TREE_SELECTION Built-in

Description

Returns the data node indicated by *selection*. Selection is an index into the list of selected nodes.

Syntax

FUNCTION GET_TREE_SELECTION
(*item_name* VARCHAR2,
selection NUMBER);

FUNCTION GET_TREE_SELECTION
(*item_id* ITEM,
selection NUMBER);

Returns FTREE.NODE

Built-in Type unrestricted function

Enter Query Mode no

Parameters

item_name	Specifies the name of the object created at design time. The data type of the name is VARCHAR2 string.
Item_id	Specifies the unique ID that Forms Developer 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.
selection	Specifies the selection of a single node.

GET_TREE_SELECTION Examples

```
/*

** Built-in: GET_TREE_SELECTION
*/
-- This code will process all tree nodes marked as selected. See the
-- Ftree.Set_Tree_Selection built-in for a definition of "selected".
```

```
DECLARE
htree ITEM;
num_selected NUMBER;
current_node FTREE.NODE;
BEGIN
-- Find the tree itself.
htree := Find_Item('tree_block.htree3');
-- Find the number of nodes marked as selected.
num_selected := Ftree.Get_Tree_Property(htree,
Ftree.SELECTION_COUNT);
-- Loop through selected nodes and process them. If you are deleting
-- nodes, be sure to loop in reverse!
FOR j IN 1..num_selected LOOP
current_node := Ftree.Get_Tree_Selection(htree, j);
...
END LOOP;
END;
```

GET_VA_PROPERTY Built-in

Description

Returns visual attribute property values for the specified property.

Syntax

FUNCTION GET_VA_PROPERTY
(*va_id* VISUALATTRIBUTE
property NUMBER);

FUNCTION GET_VA_PROPERTY
(*va_name* VARCHAR2
property NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

va_id	The unique ID Forms Developer assinged to the visual attribute when you created it. The data type is VISUALATTRIBUTE.
va_name	The name you gave the visual attribute when you created it. The data type is VARCHAR2.
property	Specify one of the following properties: 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. 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 hundreds of points. For example, 8pt. would be 800.

FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

FONT_STYLE The style of the font.

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.

GET_VAR_BOUNDS Built-in

Description

Obtains the bounds of an OLE variant's array.

Syntax

PROCEDURE GET_VAR_BOUNDS
(var OLEVAR, bounds OLE_SAFEARRAYBOUNDS);

Built-in Type unrestricted function

Parameters

var

The variant.

bounds

The PL/SQL table that is populated with the bounds of the array.

GET_VAR_DIMS Built-in

Description

Determines if an OLE variant is an array, and if so, obtains the number of dimensions in that array.

Syntax

```
FUNCTION GET_VAR_DIMS
(var OLEVAR)
RETURN vardims PLS_INTEGER;
```

Built-in Type unrestricted function

Returns A value of zero (0) if the variant is not an array. Otherwise, the return value is the number of dimensions in the array.

Parameters

var

The variant.

GET_VAR_TYPE Built-in

Description

Obtains the type of an OLE variant.

Syntax

FUNCTION GET_VAR_TYPE
(var OLEVAR)
RETURN vartype VT_TYPE;

Built-in Type unrestricted function

Returns type of the variable.

vartype Type of the variant.

Parameters

var

The variant.

vartype

Type of the variant.

GET_VIEW_PROPERTY Built-in

Description

Returns the indicated property setting for the indicated canvas.

Syntax

FUNCTION GET_VIEW_PROPERTY
(*view_id* ViewPort,
property NUMBER);

FUNCTION GET_VIEW_PROPERTY
(*view_name* VARCHAR2,
property NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

view_id

Specifies the unique ID that Forms Developer assigns the canvas 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. 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.

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.

VIEWPORT_X_POS For a stacked canvas, 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 VARCHAR2 and is expressed in the units defined by the form module Coordinate System property.

VIEWPORT_Y_POS For a stacked canvas, 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 VARCHAR2 and is expressed in the units defined by the form module Coordinate System property.

VIEWPORT_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 VARCHAR2 and is expressed in the units defined by the form module Coordinate System property.

VIEWPORT_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 VARCHAR2 and is expressed in the units defined by the form module Coordinate System property.

VISIBLE Returns the VARCHAR2 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 is displayed.

GET_VIEW_PROPERTY Examples

```
/*

** 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,VIEWPORT_X_POS);
y := Get_View_Property(vw_id1,VIEWPORT_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,VIEWPORT_X_POS, x+w );
Set_View_Property(vw_id2,VIEWPORT_Y_POS, y );
END;
```

GET_WINDOW_PROPERTY Built-in

Description

Returns the current setting for the indicated window property for the given window.

Syntax

FUNCTION GET_WINDOW_PROPERTY
(*window_id* Window*,*
property NUMBER);

FUNCTION GET_WINDOW_PROPERTY
(*window_name* VARCHAR2,
property NUMBER);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Usage Notes

- You can reference the MDI application window with the constant `FORMS_MDI_WINDOW`.

Parameters

window_id

Specifies the unique ID that Forms Developer 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_WINDOW_PROPERTY 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:

BACKGROUND_COLOR The color of the object's background region.

DIRECTION Returns the layout direction for bidirectional objects. Valid return values are RIGHT_TO_LEFT, LEFT_TO_RIGHT.

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.

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_SPACING The width of the font, that is, the amount of space between characters (kerning).

FONT_STYLE The style of the font.

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.

HEIGHT Returns the height of the window.

HIDE_ON_EXIT Returns the VARCHAR2 value TRUE if the window has the Remove On Exit property set to Yes, otherwise, it is FALSE.

ICON_NAME Returns the file name of the icon resource associated with a window item when it is minimized.

TITLE Returns the title of the window.

VISIBLE Returns the VARCHAR2 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 VARCHAR2 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 VARCHAR2 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.

Related topics

GET_CANVAS_PROPERTY built-in

SET_VIEW_PROPERTY built-in

SET_WINDOW_PROPERTY built-in

GO_BLOCK Built-in

Description

GO_BLOCK navigates to an indicated block. If the target block is non-enterable, an error occurs.

Syntax

PROCEDURE GO_BLOCK
(*block_name* VARCHAR2);

Built-in Type restricted procedure

Enter Query Mode no

Parameters

block_name

Specifies the name you gave the block when defining it. The data type of the name is VARCHAR2.

GO_BLOCK Examples

```
/*

** 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 IF;
ELSIF :Global.Flag_Indicator = 'DAY' THEN
Go_Block('Day_Schedule');
/*
** Another way of checking that block navigation
** succeeds. If the block the cursor is in hasn't
** changed after a block navigation, something went
** wrong. This method is more reliable than simply
** checking FORM_SUCCESS.
*/
IF :System.Trigger_Block = :System.Cursor_Block THEN
RAISE Form_Trigger_Failure;
END IF;
END IF;
Execute_Query;
Go_Block('Main');
END;
```

GO_FORM Built-in

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.

Syntax

PROCEDURE GO_FORM
(*form_id* FORMMODULE);

PROCEDURE GO_FORM
(*form_name* VARCHAR2);

Built-in Type restricted procedure

Enter Query Mode no

Parameters

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.

form_name

> The name of the target form. The data type of name is VARCHAR2. The GO_FORM built-in attempts to search for the form module name, not the name of the .fmx file.

GO_FORM 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 Built-in

Description

GO_ITEM navigates to an indicated item. GO_ITEM succeeds even if the target item has the Keyboard Navigable property set to No.

Syntax

PROCEDURE GO_ITEM
(*item_id* Item);

PROCEDURE GO_ITEM
(*item_name* VARCHAR2);

Built-in Type restricted procedure

Enter Query Mode yes

Parameters

item_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

GO_ITEM Restrictions

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 VARCHAR2 item or display item.

GO_ITEM Examples

```
/*

** 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 Built-in

Description

Navigates to the record with the specified record number.

Syntax

PROCEDURE GO_RECORD
(*record_number* NUMBER);

Built-in Type restricted procedure

Enter Query Mode no

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.

GO_RECORD Restrictions

- If the query is open and the specified record number is greater than the number of records already fetched, Forms Developer fetches additional records to satisfy the call to this built-in.

GO_RECORD Examples

```
/*

** 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 Built-in

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.

Syntax

PROCEDURE HELP;

Built-in Type restricted procedure

Enter Query Mode yes

Parameters

none

HELP Examples

```
/*

** Built-in: HELP
** Example: Gives item-level hint/help.
*/
BEGIN
Help;
END;
```

HIDE_VIEW Built-in

Description

Hides the indicated canvas.

Syntax

PROCEDURE HIDE_VIEW
(*view_id* ViewPort);

PROCEDURE HIDE_VIEW
(*view_name* VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Description

Hides the indicated canvas.

Parameters

view_id

Specifies the unique ID that Forms Developer 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.

view_name

Specifies the name that you gave the view when creating it.

HIDE_VIEW Examples

```
/*

** Built-in: HIDE_VIEW
** Example: Programmatically dismiss a stacked view from the
** operator's sight.
*/
PROCEDURE Hide_Button_Bar IS
BEGIN
Hide_View('Button_Bar');
END;
```

HIDE_WINDOW Built-in

Description

Hides the given window. HIDE_WINDOW is equivalent to setting VISIBLE to No by calling SET_WINDOW_PROPERTY.

Syntax

PROCEDURE HIDE_WINDOW
(*window_id* Window);

PROCEDURE HIDE_WINDOW
(*window_name* VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

window_id

Specifies the unique ID that Forms Developer 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.

HIDE_WINDOW Examples

```
/*

** 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:
**
** Parent_Window Child_Window
** ------------- -------------
** MAIN DETAIL1
** MAIN DETAIL2
** DETAIL1 DETAIL3
** DETAIL1 DETAIL4
** DETAIL2 DETAIL5
** DETAIL3 DETAIL6
**
** 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.
*/
PROCEDURE Close_Window( wn_name VARCHAR2,
dest_item VARCHAR2 ) IS
rg_id RecordGroup;
gc_parent GroupColumn;
gc_child GroupColumn;
the_Rowcount NUMBER;
/*
** Local function called recursively to close children at
** all levels of the hierarchy.
*/
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
*/
```

```
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 Built-in

Description

Executes an indicated operating system command.

Syntax

PROCEDURE HOST
(*system_command_string* VARCHAR2);

PROCEDURE HOST
(*system_command_string* VARCHAR2*,*
screen_action NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

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 Forms Developer will:
clear the screen
prompt the operator to return from the command

NO_PROMPT Specifies that Forms Developer will:
clear the screen (does *not* prompt the operator to return from the command)

NO_SCREEN Specifies that Forms Developer 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.)

Usage notes

- Note that the command interpreter for Microsoft Windows NT is `cmd`, while on Windows 95 it is `command`. Before using the HOST built-in to run an external command, be sure to check for the operating system and pass the appropriate command string.
- On Windows 95 platforms the FORM_SUCCESS built-in will always return TRUE for HOST commands which fail. This includes calls to command.com or OS functions, any 16-bit DOS or GUI application, or an invalid command. 32-bit applications will correctly return TRUE if executed sucessfully and FALSE if failed.
- The host command operates on the application server machine. Any screen output that it performs is not visible to the user of the application.

HOST Examples

```
/*

** 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 Built-in

Description

Returns a BOOLEAN value that indicates whether the object ID is available.

Syntax

```
FUNCTION ID_NULL
(Alert ALERT);

FUNCTION ID_NULL
(Block BLOCK);

FUNCTION ID_NULL
(Canvas CANVAS);

FUNCTION ID_NULL
(Editor EDITOR);

FUNCTION ID_NULL
(FormModule FORMMODULE);

FUNCTION ID_NULL
(GroupColumn GROUPCOLUMN);

FUNCTION ID_NULL
(Item ITEM);

FUNCTION ID_NULL
(LOV LOV);

FUNCTION ID_NULL
(MenuItem MENUITEM);

FUNCTION ID_NULL
(ParamList PARAMLIST);

FUNCTION ID_NULL
(RecordGroup RECORDGROUP);

FUNCTION ID_NULL
(Relation RELATION);
```

FUNCTION ID_NULL
(TabPage TABPAGE);

FUNCTION ID_NULL
(Timer TIMER);

FUNCTION ID_NULL
(Viewport VIEWPORT);

FUNCTION ID_NULL
(VisualAttribute VISUALATTRIBUTE);

FUNCTION ID_NULL
(Window WINDOW);

Built-in Type unrestricted function

Returns BOOLEAN

Enter Query Mode yes

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`
- `RecordGroup`
- `Relation`

- `TabPage`
- `Timer`
- `Viewport`
- `VisualAttribute`
- `Window`

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.

If you are going to test for an object's existence at various times (that is, more than once during a run), then you need to reissue the appropriate FIND_ every time -- once preceding each use of ID_NULL.

ID_NULL Examples

See CREATE_GROUP Built-in.

IMAGE_SCROLL Built-in

Description

Scrolls the image item as close to the specified offset (the X,Y coordinates) as possible. This is useful if the image is bigger than the image item.

Syntax

PROCEDURE IMAGE_SCROLL
(*item_name* VARCHAR2,
X NUMBER,
Y NUMBER
);

PROCEDURE IMAGE_SCROLL
(*item_id* ITEM,
X NUMBER,
Y NUMBER
);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

item_id

Specifies the unique ID Forms Developer assigns when it creates the image item.

item_name

Specifies the name you gave the image when defining it.

X

The X coordinate of the offset.

Y

The Y coordinate of the offset.

IMAGE_SCROLL Examples

For example, suppose the image is twice the size of the image item, that is, the image coordinates are 0, 200, and the item coordinates are 0, 100. To roughly center the image, you can set IMAGE_SCROLL X, Y coordinates to 50, 50. This sets the top left corner of the item at 50 50 instead of 0, 0, which offsets the image so that it is displayed from its coordinates of 50 to 150.

IMAGE_ZOOM Built-in

Description

Zooms the image in or out using the effect specified in zoom_type and the amount specified in zoom_factor.

Syntax

PROCEDURE IMAGE_ZOOM
(*image_id* ITEM*,*
zoom_type NUMBER*);*

PROCEDURE IMAGE_ZOOM
(*image_name* VARCHAR2,
zoom_type NUMBER);

PROCEDURE IMAGE_ZOOM
(*image_id* ITEM,
zoom_type NUMBER,
zoom_factor NUMBER);

PROCEDURE IMAGE_ZOOM
(*image_name* VARCHAR2,
zoom_type NUMBER,
zoom_factor NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

image_id

Specifies the unique ID Forms Developer 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 Forms Developer 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.

IMAGE_ZOOM 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 Built-in

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.

Syntax

PROCEDURE INSERT_RECORD;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

none

INSERT_RECORD Restrictions

Valid only in an On-Insert trigger.

INSERT_RECORD Examples

```
/*

** Built-in: INSERT_RECORD
** Example : Perform Forms Developer 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 Built-in

Description

When called from an On-Rollback trigger, initiates the default Forms Developer 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.

Syntax

PROCEDURE ISSUE_ROLLBACK
(*savepoint_name* VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode no

Parameters

savepoint name Name of the savepoint to which you want to rollback. A null savepoint_name causes a full rollback.

ISSUE_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).

ISSUE_ROLLBACK Examples

```
/*

** Built-in: ISSUE_ROLLBACK
** Example: Perform Forms Developer 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 Developer 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 Built-in

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 Forms Developer 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.

Syntax

PROCEDURE ISSUE_SAVEPOINT
(*savepoint_name* VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode no

Parameters

savepoint _name

Name of the savepoint you want to be issued

ISSUE_SAVEPOINT 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 Forms Developer.

ISSUE_SAVEPOINT Examples

```
/*
```

```
** Built-in: ISSUE_SAVEPOINT
** Example: Perform Forms Developer 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 Developer 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;
```

LAST_RECORD Built-in

Description

Navigates to the last record in the block's list of records. If a query is open in the block, Forms Developer fetches the remaining selected records into the block's list of records, and closes the query.

Syntax

PROCEDURE LAST_RECORD;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

none

LAST_RECORD Examples

See FIRST_RECORD

LIST_VALUES Built-in

Description

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, LIST_VALUES uses the NO_RESTRICT parameter. This parameter causes Forms Developer *not* to use the automatic search and complete feature. If you use the RESTRICT parameter, Forms Developer uses the automatic search and complete feature.

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, Forms Developer checks to see if the item contains a value.

If the text item contains a value, Forms Developer 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, Forms Developer does not display the LOV, but automatically reads the correct value into the field.

Syntax

PROCEDURE LIST_VALUES
(*kwd* NUMBER);

Built-in Type restricted procedure

Enter Query Mode no

Parameters

kwd

Specifies one of the following constants:

NO_RESTRICT Specifies that Forms Developer will not use the automatic search and complete feature.

RESTRICT Specifies that Forms Developer will use the automatic search and complete feature.

Related topic

SHOW_LOV built-in

LOCK_RECORD Built-in

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.

Syntax

PROCEDURE LOCK_RECORD;

Built-in Type unrestricted procedure

Enter Query Mode no

Parameters

none

LOCK_RECORD Examples

```
/*

** Built-in: LOCK_RECORD
** Example: Perform Forms Developer 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;
```

Related topic

On-Lock Trigger

LOGON Built-in

Description

Performs the default Forms Developer logon processing with an indicated username and password. Call this procedure from an On-Logon trigger when you want to augment default logon processing.

Syntax

PROCEDURE LOGON
(*username* VARCHAR2,
password VARCHAR2);

PROCEDURE LOGON
(*username* VARCHAR2,
password VARCHAR2,
logon_screen_on_error VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

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 Forms Developer 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 FORM_FAILURE is set to TRUE so the designer can handle the condition in an appropriate manner.

Usage Notes:

When using LOGON to connect to an OPS$ database use a slash '/' for the user.name and the database name for the password..

LOGON Restrictions

- If you identify a remote database, a SQL*Net connection to that database must exist at runtime.
- Forms Developer can have a primary connection to only one database at a time. However, multiple connections can be established for use in PL/SQL using the EXEC_SQL packages. Also, database links may be used to access multiple databases with a single connection.

LOGON Examples

```
/*

** Built-in: LOGON
** Example: Perform Forms Developer standard logon to the ORACLE
** database. Decide whether to use Forms Developer
** 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

/*  ** Use the following to place a slash in the username field for
OPS$ logon  */  IF un IS NULL THEN  un:='/';  END IF  IF cn IS NOT
NULL THEN
LOGON(un,pw||'@'||cn);
ELSE
LOGON(un,pw);
END IF;
END IF;
END;
```

LOGON_SCREEN Built-in

Description

Displays the default Forms Developer 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.

Syntax

PROCEDURE LOGON_SCREEN;

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

none

LOGON_SCREEN Restrictions

- You must issue a call to the LOGON built-in to create the connection to your data source.

LOGON_SCREEN Examples

```
/*

** Built-in: LOGON_SCREEN
** Example: Use the default Forms Developer logon screen to prompt
** for username and password before logging on to
** the database. This uses the 'Get_Connect_Info'
** procedure from the GET_APPLICATION_PROPERTY
** example.
*/
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 Built-in

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 sessions, Forms Developer tries to re-establish all of those connections when you subsequently execute LOGON.

Syntax

PROCEDURE LOGOUT;

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

none

LOGOUT Examples

```
/*

** Built-in: LOGOUT
** Example: Perform Forms Developer standard logout. Decide
** whether to use Forms Developer 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;
```

MESSAGE_CODE Built-in

Description

Returns a message number for the message that Forms Developer most recently generated during the current Runform session. MESSAGE_CODE returns zero at the beginning of a session, before Forms Developer 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.

Syntax

FUNCTION MESSAGE_CODE;

Built-in Type unrestricted function

Returns NUMBER

Enter Query Mode yes

Parameters

none

MESSAGE_CODE Examples

```
/*

** 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_TEXT Built-in

Description

Returns message text for the message that Forms Developer most recently generated during the current Runform session. MESSAGE_TEXT returns NULL at the beginning of a session, before Forms Developer 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.

Syntax

FUNCTION MESSAGE_TEXT;

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

none

MESSAGE_TEXT Examples

```
/*

** 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 Built-in

Description

Returns a message type for the message that Forms Developer 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.

Syntax

FUNCTION MESSAGE_TYPE;

Built-in Type unrestricted function

Returns VARCHAR2

MESSAGE_TYPE returns one of three values for the message type:

- FRM Indicates that an Forms Developer message was generated.
- ORA Indicates that an ORACLE message was generated.
- NULL Indicates that Forms Developer has not yet issued any messages during the session.

Enter Query Mode yes

Parameters

none

MESSAGE_TYPE Examples

```
/*

** 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 Built-in

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.

Syntax

FUNCTION MOVE_WINDOW
(*window_id* Window*,*
x NUMBER*,*
y NUMBER);

FUNCTION MOVE_WINDOW
(*window_name* VARCHAR2*,*
x NUMBER*,*
y NUMBER);

Built-in Type unrestricted function

Enter Query Mode yes

Parameters

window_id

> Specifies the unique ID that Forms Developer 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.

x

Specifies the x coordinate on the screen where you want to place the upper left corner of a window.

y

Specifies the y coordinate on the screen where you want to place the upper left corner of a window.

MOVE_WINDOW Examples

```
/*

** 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 Built-in

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, Forms Developer evaluates the individual NAME_IN functions from the innermost one to the outermost one.

Syntax

FUNCTION NAME_IN
(*variable_name* VARCHAR2);

Built-in Type unrestricted function

Returns VARCHAR2

Enter Query Mode yes

Parameters

variable_name

Specifies a valid variable or text item. The data type of the name is VARCHAR2.

Usage Note

- If the returned value is a date string, NAME_IN will use the format mask specified in the Builtin Date Format Property.

NAME_IN Examples

```
/*

** 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 VARCHAR2, 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 PUSH'ed and then all values subsequently
** POP'ed. If no values were ever PUSH'ed 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 POP'ed 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 Built-in

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, Forms Developer keeps the higher call active and treats it as a call to the new form. Forms Developer releases memory (such as database cursors) that the terminated form was using.

Forms Developer runs the new form with the same Runform options as the parent form. If the parent form was a called form, Forms Developer runs the new form with the same options as the parent form.

Syntax

PROCEDURE NEW_FORM
(*formmodule_name* VARCHAR2);

PROCEDURE NEW_FORM
(*formmodule_name* VARCHAR2,
rollback_mode NUMBER);

PROCEDURE NEW_FORM
(*formmodule_name* VARCHAR2,
rollback_mode NUMBER,
query_mode NUMBER);

PROCEDURE NEW_FORM
(*formmodule_name* VARCHAR2,
rollback_mode NUMBER,
query_mode NUMBER,
data_mode NUMBER);

PROCEDURE NEW_FORM
(*formmodule_name* VARCHAR2,
rollback_mode NUMBER,
query_mode NUMBER,
paramlist_id PARAMLIST);

PROCEDURE NEW_FORM
(*formmodule_name* VARCHAR2,
rollback_mode NUMBER,

query_mode NUMBER,
paramlist_name VARCHAR2);

PROCEDURE NEW_FORM
(*formmodule_name* VARCHAR2,
rollback_mode NUMBER,
query_mode NUMBER,
data_mode NUMBER,
paramlist_id PARAMLIST);

PROCEDURE NEW_FORM
(*formmodule_name* VARCHAR2,
rollback_mode NUMBER,
query_mode NUMBER,
data_mode NUMBER,
paramlist_name VARCHAR2);

Built-in Type restricted procedure

Enter Query Mode no

Parameters

formmodule_name

Then name of the called form (must be enclosed in single quotes). Datatype is VARCHAR2.

rollback_mode

TO_SAVEPOINT (The default.) Forms Developer will roll back all uncommitted changes (including posted changes) to the current form's savepoint.

NO_ROLLBACK Forms Developer will exit 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 Forms Developer from an external 3GL program. The locks are still in effect when you regain control from Forms Developer.

FULL_ROLLBACK Forms Developer 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, Forms Developer prevents any commit processing in the called form.)

query_mode

NO_QUERY_ONLY (The default.) Runs the indicated form normally, allowing the end user to perform inserts, updates, and deletes in the form.

QUERY_ONLY Runs the indicated form in query-only mode; end users can query records, but cannot perform inserts, updates or deletes.

data_mode

NO_SHARE_LIBRARY_DATA (The default.) At runtime, Forms Developer will not share data between forms that have identical libraries attached (at design time).

SHARE_LIBRARY_DATA At runtime, Forms Developer will share data between forms that have identical libraries attached (at design time).

paramlist_id

The unique ID Forms Developer 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. Datatype is PARAMLIST. A parameter list passed to a form via NEW_FORM cannot contain parameters of type DATA_PARAMETER (a pointer to record group).

paramlist_name

The name you gave the parameter list object when you defined it. Datatype is VARCHAR2. A parameter list passed to a form via NEW_FORM cannot contain parameters of type DATA_PARAMETER (a pointer to record group).

NEW_FORM Examples

```
/* Create a generic procedure that 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;
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 Built-in

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.

Syntax

PROCEDURE NEXT_BLOCK;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

none

NEXT_BLOCK Examples

```
/*

** 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_Block;
ELSE
Next_Item;
END IF;
END;
```

NEXT_FORM Built-in

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.

Syntax

PROCEDURE NEXT_FORM;

Built-in Type restricted procedure

Enter Query Mode no

NEXT_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 Built-in

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, Forms Developer 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.

Syntax

PROCEDURE NEXT_ITEM;

Built-in Type restricted procedure

Enter Query Mode yes

Parameters

none

NEXT_ITEM Examples

```
/*

** Built-in: NEXT_ITEM
** Example: See NEXT_BLOCK
*/
```

NEXT_KEY Built-in

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.

Syntax

PROCEDURE NEXT_KEY;

Built-in Type restricted procedure

Enter Query Mode yes

Parameters

none

NEXT_KEY Examples

```
/*

** Built-in: NEXT_KEY
** Example: Jump the cursor to the next primary key item in
** in the current block.
*/
BEGIN
Next_Key;
END;
```

Related topic

Primary Key (Item) Property

NEXT_RECORD Built-in

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, Forms Developer will fetch or create a record. If the current record is a new record, NEXT_RECORD fails.

Syntax

PROCEDURE NEXT_RECORD;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

none

NEXT_RECORD Restrictions

Not allowed in Enter Query mode.

NEXT_RECORD Example

```
/*

** 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 Built-in

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.

Syntax

PROCEDURE NEXT_SET;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

none

NEXT_SET 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;
```

Related topic

Query Array Size Property

OPEN_FORM Built-in

Description

Opens the indicated form. Use OPEN_FORM to create multiple-form applications, that is, applications that open more than one form at the same time.

Syntax

PROCEDURE OPEN_FORM
(*form_name* VARCHAR2);

PROCEDURE OPEN_FORM
(*form_name* VARCHAR2,
activate_mode NUMBER);

PROCEDURE OPEN_FORM
(*form_name* VARCHAR2,
activate_mode NUMBER,
session_mode NUMBER);

PROCEDURE OPEN_FORM
(*form_name* VARCHAR2,
activate_mode NUMBER,
session_mode NUMBER,
data_mode NUMBER);

PROCEDURE OPEN_FORM
(*form_name* VARCHAR2,
activate_mode NUMBER,
session_mode NUMBER,
paramlist_name VARCHAR2);

PROCEDURE OPEN_FORM
(*form_name* VARCHAR2,
activate_mode NUMBER,
session_mode NUMBER,
paramlist_id PARAMLIST);

PROCEDURE OPEN_FORM
(*form_name* VARCHAR2,
activate_mode NUMBER,

session_mode NUMBER,
data_mode NUMBER,
paramlist_name VARCHAR2);

PROCEDURE OPEN_FORM
(*form_name* VARCHAR2,
activate_mode NUMBER,
session_mode NUMBER,
data_mode NUMBER,
paramlist_id PARAMLIST);

Built-in Type restricted procedure

Enter Query Mode no

Parameters:

form_name

The name of the form to open. Datatype is VARCHAR2. *Required*

activate_mode

ACTIVATE (The default.) 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 (The default.) Specifies that the opened form should share the same database session as the current form. POST and COMMIT operations in any form will cause posting, 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.

data_mode

NO_SHARE_LIBRARY_DATA (The default.) At runtime, Forms Developer will not share data between forms that have identical libraries attached (at design time).

SHARE_LIBRARY_DATA At runtime, Forms Developer will share data between forms that have identical libraries attached (at design time).

paramlist_name

The name of a parameter list to be passed to the opened form. Datatype is VARCHAR2.

paramlist_id

The unique ID that Forms Developer 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.

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. (However, see the usage note regarding SESSION-specified below.)
- 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.
- When you open a form with SESSION specified, the PRE-LOGON, ON-LOGON, and POST-LOGON triggers will not fire.
- If the form that issues the OPEN_FORM built-in is running in QUERY_ONLY mode, then the opened form will also run in QUERY_ONLY mode.
- 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, canvases 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 canvases in the original form partially or completely.

OPEN_FORM Restrictions

- You can set session On for all Runform invocations by setting the FORMSnn_SESSION environment variable to TRUE. When you set the FORMSnn_SESSION variable, all Runform invocations inherit its setting, unless you override the environment variable by setting the Session option from the Runform command line.
- If you set *session_mode* to SESSION when you use OPEN_FORM to create a multiple-form application, you cannot set *data_mode* to SHARE_LIBRARY_DATA (Forms Developer will display a runtime error message).

PASTE_REGION Built-in

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.

Syntax

PROCEDURE PASTE_REGION;

Built-in Type restricted procedure

Enter Query Mode yes

Parameters

none

Usage Notes

Use PASTE_REGION, as well as the other editing functions on text items only. The cut and copy functions transfer the selected region into the system clipboard until you indicate the paste target. At that time, the cut or copied content is pasted onto the target location.

Related topics

COPY_REGION built-in

CUT_REGION built-in

PAUSE Built-in

Description

Suspends processing until the end user presses a function key. PAUSE might display an alert.

Syntax

PROCEDURE PAUSE;

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

none

POPULATE_GROUP Built-in

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.

Syntax

FUNCTION POPULATE_GROUP
(*recordgroup_id* RecordGroup);

FUNCTION POPULATE_GROUP
(*recordgroup_name* VARCHAR2);

Built-in Type unrestricted function

Returns NUMBER

Enter Query Mode yes

Parameters

recordgroup_id

> The unique ID that Forms Developer 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

VARCHAR2.

POPULATE_GROUP 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)

POPULATE_GROUP Example

```
/*

** Built-in: POPULATE_GROUP
** Example: See GET_GROUP_ROW_COUNT and CREATE_GROUP_FROM_QUERY
*/
```

POPULATE_GROUP_FROM_TREE Built-in

Description

Populates a record group with the data from the hierarchical tree.

Syntax

PROCEDURE POPULATE_GROUP_FROM_TREE
(*group_name* VARCHAR2,
item_name VARCHAR2,
node FTREE.NODE);

PROCEDURE POPULATE_GROUP_FROM_TREE
(*group_name* VARCHAR2,
item_id ITEM,
node FTREE.NODE);

PROCEDURE POPULATE_GROUP_FROM_TREE
(*group_id* RECORDGROUP,
item_name VARCHAR2,
node FTREE.NODE);

PROCEDURE POPULATE_GROUP_FROM_TREE
(*group_id* RECORDGROUP,
item_id ITEM,
node FTREE.NODE);

Built-in Type unrestricted procedure

Enter Query Mode no

Parameters

group_name	Specifies the name of the group.
group_id	Specifies the ID assigned to the group.
item_name	Specifies the name of the object created at design time. The data type of the name is VARCHAR2 string.

Item_id	Specifies the unique ID that Forms Developer 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.
node	Specifies a valid node. If specified, indicates a sub-tree used to populate the RecordGroup, including the specified node.

Usage Note

- The record group is cleared prior to inserting the hierarchical tree's data set.

POPULATE_GROUP_FROM_TREE Example

```
/*

** Built-in: POPULATE_GROUP_FROM_TREE
*/
-- This code will transfer all the data from a hierarchical tree
-- that is parented by the node with a label of "Zetie" to a
-- pre-created record group.

DECLARE
htree ITEM;
find_node NODE;
BEGIN
-- Find the tree itself.
htree := Find_Item('tree_block.htree3');
-- Find the node with a label "Zetie".
find_node := FTREE.FIND_TREE_NODE(htree,
'Zetie',
FTREE.FIND_NEXT,
FTREE.NODE_LABEL,
FTREE.ROOT_NODE,
FTREE.ROOT_NODE);
-- Populate the record group with the tree data.
-- The record group must already exist.
FTREE.POPULATE_GROUP_FROM_TREE('tree_data_rg', htree, find_node);
END;
```

POPULATE_GROUP_WITH_QUERY Built-in

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, Forms Developer 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 20 records at a time. To improve network performance, you may want to restrict queries, thus limiting network traffic.

Syntax

FUNCTION POPULATE_GROUP_WITH_QUERY
(*recordgroup_id* RecordGroup*,*
query VARCHAR2);

FUNCTION POPULATE_GROUP_WITH_QUERY
(*recordgroup_name* VARCHAR2*,*
query VARCHAR2);

Built-in Type unrestricted function

Returns NUMBER

Enter Query Mode yes

Parameters

recordgroup_id

The unique ID that Forms Developer 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 VARCHAR2.

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 Forms Developer creates only those columns in the record group. The data type of the query is VARCHAR2.

POPULATE_GROUP_WITH_QUERY Restrictions

- The columns specified in the SELECT statement must match the record group columns in number and type.

POPULATE_GROUP_WITH_QUERY Example

```
/*

** Built-in: POPULATE_GROUP_WITH_QUERY
** Example: See CREATE_GROUP
*/
```

POPULATE_LIST Built-in

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 (VARCHAR2) structure:

Column 1: Column 2:

the list label the list value

Syntax

PROCEDURE POPULATE_LIST
(*list_id* ITEM*,*
recgrp_id RecordGroup);

PROCEDURE POPULATE_LIST
(*list_id* ITEM*,*
recgrp_name VARCHAR2);

PROCEDURE POPULATE_LIST
(*list_name* VARCHAR2*,*
recgrp_id RecordGroup);

PROCEDURE POPULATE_LIST
(*list_name* VARCHAR2*,*
recgrp_name VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

list_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

recgrp_id

Specifies the unique ID that Forms Developer assigns when it creates the record group. The data type of the ID is RecordGroup.

recgrp_name

The VARCHAR2 name you gave to the record group when you created it.

Usage Notes

- Do not use the POPULATE_LIST built-in if the Mapping of Other Values property is defined and there are queried records in the block. Doing so may cause Forms Developer 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 Mapping of 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 Forms Developer 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.

POPULATE_LIST 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.

POPULATE_LIST 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;
```

POPULATE_TREE Built-in

Description

Clears out any data already in the hierarchical tree, and obtains the data set specified by the RecordGroup or QueryText properties.

Syntax

PROCEDURE POPULATE_TREE
(*item_name* VARCHAR2);

PROCEDURE POPULATE_TREE
(*item_id* ITEM);

Built-in Type unrestricted procedure

Enter Query Mode no

Parameters

item_name	Specifies the name of the object created at design time. The data type of the name is VARCHAR2 string.
item_id	Specifies the unique ID that Forms Developer 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.

POPULATE_TREE Examples

```
/*

** Built-in: POPULATE_TREE
*/

-- This code will cause a tree to be re-populated using
-- either the record group or query already specified
-- for the hierarchical tree.

DECLARE
htree ITEM;
```

```
top_node FTREE.NODE;
find_node FTREE.NODE;
BEGIN
-- Find the tree itself.
htree := FIND_ITEM('tree_block.htree3');
-- Populate the tree with data.
FTREE.POPULATE_TREE(htree);
END;
```

POST Built-in

Description

Writes data in the form to the database, but does not perform a database commit. Forms Developer first validates the form. If there are changes to post to the database, for each block in the form Forms Developer 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.

Syntax

PROCEDURE POST;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

none

Usage Notes

If this form was called via OPEN_FORM with the NO_SESSION parameter specified, then the POST will validate and write the data both in this form and in the calling form.

POST Examples

```
/*

** Built-in: POST and 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
** successfully posted.
*/
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;
```

PREVIOUS_BLOCK Built-in

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.

Syntax

PROCEDURE PREVIOUS_BLOCK;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

none

PREVIOUS_BLOCK Examples

```
/*

** 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
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_Block;
ELSE
Previous_Item;
END IF;
END;
```

PREVIOUS_FORM Built-in

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.

Syntax

PROCEDURE PREVIOUS_FORM;

Built-in Type restricted procedure

Enter Query Mode no

PREVIOUS_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 Built-in

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.

Syntax

PROCEDURE PREVIOUS_ITEM;

Built-in Type restricted procedure

Enter Query Mode yes

Parameters

none

PREVIOUS_ITEM Examples

```
/*

** Built-in: PREVIOUS_ITEM
** Example: See PREVIOUS_BLOCK
*/
```

PREVIOUS_RECORD Built-in

Description

Navigates to the first enabled and navigable item in the record with the next lower sequence number than the current record.

Syntax

PROCEDURE PREVIOUS_RECORD;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

none

PREVIOUS_RECORD 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;
```

Related topic

NEXT_RECORD Built-in

PRINT Built-in

Description

Prints the current window to a file or to the printer.

Syntax

PROCEDURE PRINT;

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

none

PRINT Examples

```
/*

** Built-in: PRINT
** Example: Print the current window.
*/
BEGIN
Print;
END;
```

READ_IMAGE_FILE Built-in

Description

Reads an image of the given type from the given file and displays it in the Forms Developer image item.

Syntax

PROCEDURE READ_IMAGE_FILE
(*file_name* VARCHAR2*,*
file_type VARCHAR2*,*
item_id ITEM);

PROCEDURE READ_IMAGE_FILE
(*file_name* VARCHAR2*,*
file_type VARCHAR2*,*
item_name VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

file_name

> Valid file name. The file name designation can include a full path statement appropriate to your operating system.

file_type

> The valid image file type: BMP, CALS, GIF, JFIF, JPG, PICT, RAS, TIFF, or TPIC. (Note: File type is optional, as Forms Developer will attempt to deduce it from the source image file. To optimize performance, however, you should specify the file type.)

item_id

The unique ID Forms Developer assigns to the image item when it creates it. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. Datatype is ITEM.

item_name

The name you gave the image item when you created it. Datatype is VARCHAR2.

Usage Notes

- Forms Developer searches for the image file along the same default path as it searches for an .FMX file.
- READ_IMAGE_FILE Built-in reads images from the file system of the Application Server, not the client browser.

READ_IMAGE_FILE Examples

```
/* 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;
```

RECALCULATE Built-in

Description

Marks the value of the specified formula calculated item (in each record of the block) for recalculation. Typically you would invoke this when the formula (or function or procedure that it invokes) refers to a system variable or built-in function which now would return a different value.

Note that actual recalculation doesn't happen immediately; it occurs sometime after the item is marked but before the new value of the calculated item is referenced or displayed to the end user. Your application's logic should not depend on recalculation of a calculated item occurring at a specific time.

Syntax

PROCEDURE RECALCULATE
(*item_name* VARCHAR2);

PROCEDURE RECALCULATE
(*item_id* Item);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

item_name

The name you gave the item when you defined it. Datatype is VARCHAR2.

item_id

The unique ID Forms Developer assigned to the item when it created the item. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. Datatype is Item.

RECALCULATE Restrictions

You can use the RECALCULATE built-in to recalculate formula calculated items only; if you specify a summary item (or a non-calculated item) as the argument to RECALCULATE, Forms Developer will return an error message:

```
FRM-41379: Cannot recalculate non-formula item
<block_name.item_name>.
```

REDISPLAY Built-in

Description

Redraws the screen. This clears any existing system messages displayed on the screen.

Syntax

PROCEDURE REDISPLAY;

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

none

REPLACE_CONTENT_VIEW Built-in

Description

Replaces the content canvas currently displayed in the indicated window with a different content canvas.

Syntax

PROCEDURE REPLACE_CONTENT_VIEW
(*window_id* Window*,*
view_id ViewPort);

PROCEDURE REPLACE_CONTENT_VIEW
(*window_name* VARCHAR2*,*
view_id ViewPort);

PROCEDURE REPLACE_CONTENT_VIEW
(*window_id* Window*,*
view_name VARCHAR2);

PROCEDURE REPLACE_CONTENT_VIEW
(*window_name* VARCHAR2*,*
view_name VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

window_id

> Specifies the unique ID that Forms Developer 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 VARCHAR2.

view_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

REPLACE_CONTENT_VIEW Restrictions

- The canvas that replaces the window's current content canvas 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 that contains the item that currently has focus, Forms Developer will immediately undo the replacement to keep the focus item visible to the end user.

REPLACE_CONTENT_VIEW Examples

```
/*

** Built-in: REPLACE_CONTENT_VIEW
** Example: Replace the 'salary' view with the 'history'
** view in the 'employee_status' window.
*/
BEGIN
Replace_Content_View('employee_status','history');
END;
```

REPLACE_MENU Built-in

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, Forms Developer 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.

Syntax

REPLACE_MENU;

PROCEDURE REPLACE_MENU
(*menu_module_name* VARCHAR2);

PROCEDURE REPLACE_MENU
(*menu_module_name* VARCHAR2,
menu_type NUMBER);

PROCEDURE REPLACE_MENU
(*menu_module_name* VARCHAR2,
menu_type NUMBER,
starting_menu_name VARCHAR2);

PROCEDURE REPLACE_MENU
(*menu_module_name* VARCHAR2,
menu_type NUMBER,
starting_menu VARCHAR2,
group_name VARCHAR2);

PROCEDURE REPLACE_MENU
(*menu_module_name* VARCHAR2,
menu_type NUMBER,
starting_menu VARCHAR2,
group_name VARCHAR2,
use_file BOOLEAN);

Built-in Type unrestricted procedure

Enter Query Mode yes

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.
- If REPLACE_MENU is called from a menu item, code that follows the REPLACE_MENU call will not be executed because the context of that menu has been destroyed and replaced with the new menu.

Parameters

menu_module _name

Name of the menu module that should replace the current menu module. Datatype is VARCHAR2. This parameter is optional; if it is omitted, Forms Developer runs the form without a menu.

menu_type

The display style of the menu. The following constants can be passed as arguments for this parameter:

PULL_DOWN Specifies that you want Forms Developer to display the menus in a pull-down style that is characteristic of most GUI platforms.

starting_menu

Specifies the menu within the menu module that Forms Developer should use as the starting menu. The data type of the name is VARCHAR2.

group_name

Specifies the security role that Forms Developer is to use. If you do not specify a role name, Forms Developer uses the current username to determine the role.

use_file

Indicates how Forms Developer should locate the menu .MMX file to be run. The data type of *use_file* is BOOLEAN.

TRUE Specifies that Forms Developer should treat the menu_module value as a direct reference to a .MMX menu runfile in the file system. This value is the only one allowed.

REPLACE_MENU 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;
```

REPORT_OBJECT_STATUS Built-in

Description

Provides status of a report object run within a form by the RUN_REPORT_OBJECT built-in.

Syntax

FUNCTION REPORT_OBJECT_STATUS
(report_id VARCHAR2(20));

Built-in Type unrestricted function

Return Type VARCHAR2

Enter Query Mode yes

Parameters

report_id

The VARCHAR2 value returned by the RUN_REPORT_OBJECT built-in. This value uniquely identifies the report that is currently running either locally or on a remote report server.

Usage Notes

- There are eight possible return values for this built-in: finished, running, canceled, opening_report, enqueued, invalid_job, terminated_with_error, and crashed.

REPORT_OBJECT_STATUS Example

```
DECLARE
 repid REPORT_OBJECT;
 v_rep VARCHAR2(100);
 rep_status varchar2(20);
```

```
BEGIN
 repid := find_report_object('report4');
 v_rep := RUN_REPORT_OBJECT(repid);
 rep_status := REPORT_OBJECT_STATUS(v_rep);
 if rep_status = 'FINISHED' then
  message('Report Completed');
  copy_report_object_output(v_rep,'d:/temp/local.pdf');
  host('netscape d:/temp/local.pdf');
 else
  message('Error when running report.');
 end if;
END;
```

RESET_GROUP_SELECTION Built-in

Description

Deselects any selected rows in the given group. Use this built-in to deselect all record group rows that have been programmatically marked as selected by executing SET_GROUP_SELECTION on individual rows.

Syntax

PROCEDURE RESET_GROUP_SELECTION
(*recordgroup_id* RecordGroup);

PROCEDURE RESET_GROUP_SELECTION
(*recordgroup_name* VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

recordgroup_id

> The unique ID that Forms Developer 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 VARCHAR2.

RESET_GROUP_SELECTION Examples

```
/*
```

```
** 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 Built-in

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.

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`.

Syntax

PROCEDURE RESIZE_WINDOW
(*window_id* Window*,*
width NUMBER*,*
height NUMBER);

PROCEDURE RESIZE_WINDOW
(*window_name* VARCHAR2*,*
width NUMBER*,*
height NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

window_id

> Specifies the unique ID that Forms Developer 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 VARCHAR2.

width

Specifies the new width of the window, in form coordinate units.

height

Specifies the new height of the window, in form coordinate units.

RESIZE_WINDOW Examples

```
/*

** 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);
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 Built-in

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 (VARCHAR2) structure:

Column 1: Column 2:

the list label the list value

Storing the contents of a list item allows you to restore the list with its former contents.

Syntax

PROCEDURE RETRIEVE_LIST
(*list_id* ITEM*,*
recgrp_name VARCHAR2);

PROCEDURE RETRIEVE_LIST
(*list_id* ITEM,
recgrp_id RecordGroup);

PROCEDURE RETRIEVE_LIST
(*list_name* VARCHAR2*,*
recgrp_id RecordGroup);

PROCEDURE RETRIEVE_LIST
(*list_name* VARCHAR2*,*
recgrp_name VARCHAR2);

Built-in Type unrestricted procedure

Returns VARCHAR2

Enter Query Mode yes

Parameters

list_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

recgrp_id

Specifies the unique ID that Forms Developer assigns when it creates the record group. The data type of the ID is RecordGroup.

recgrp_name

The VARCHAR2 name you gave to the record group when you created it.

RETRIEVE_LIST Examples

```
/*

** Built-in: RETRIEVE_LIST
** Example: See POPULATE_LIST
*/
```

RUN_PRODUCT Built-in

Description

Invokes the Graphics product and specifies the name of the Graphics module or module to be run. If Graphics is unavailable at the time of the call, Forms Developer returns a message to the end user.

If you create a parameter list and then reference it in the call to RUN_PRODUCT, the form can pass text and data parameters to the Graphics 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 Forms Developer. You can pass DATA_PARAMETERs to Graphics, but not to Forms Developer.

To run a report from within a form, you must use the dedicated report integration built-in RUN_REPORT_OBJECT .

Syntax

PROCEDURE RUN_PRODUCT
(*product* NUMBER,
module VARCHAR2,
commmode NUMBER,
execmode NUMBER,
location NUMBER,
paramlist_id VARCHAR2,
display VARCHAR2);

PROCEDURE RUN_PRODUCT
(*product* NUMBER,
module VARCHAR2,
commmode NUMBER,
execmode NUMBER,
location NUMBER,
paramlist_name VARCHAR2,
display VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

product

Can specify GRAPHICS only.

module

Specifies the VARCHAR2 name of the module or module to be executed by Graphics. Valid value is the name of a Graphics display. The application looks for the module or module in the default paths defined for Graphics.

commmode

Specifies the communication mode to be used when running Graphics. Valid numeric constants for this parameter are:

SYNCHRONOUS

specifies that control returns to Forms Developer only after Graphics has been exited. The end user cannot work in the form while Graphics is running.

ASYNCHRONOUS

specifies that control returns to the calling application immediately, even if Graphics has not completed its display.

execmode

Specifies the execution mode to be used when running Graphics. Valid numeric constants for this parameter are BATCH and RUNTIME. When you run Graphics, execmode can be either BATCH or RUNTIME.

location

Specifies the location of the module or module you want Graphics to execute, either the file system or the database. Valid constants for this property are FILESYSTEM and DB.

Paramlist_name or paramlist_ID

Specifies the parameter list to be passed to Graphics. Valid values for this parameter are the VARCHAR2 name of the parameter list, the ID of the parameter list, or a null string (''). To specify a parameter list ID, use a variable of type PARAMLIST.

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 Graphics in SYNCHRONOUS mode. (SYNCHRONOUS mode is required when invoking Graphics to return an Graphics display that will be displayed in a form chart item.)

Note: You can prevent Graphics from logging on by passing a parameter list that includes a parameter with *key* set to LOGON and *value* set to NO.

display

Specifies the VARCHAR2 name of the Forms Developer chart item that will contain the display (such as a pie chart, bar chart, or graph) generated by Graphics. The name of the chart item must be specified in the format *block_name.item_name*. (This parameter is only required when you are using a Graphics chart item in a form.)

RUN_REPORT_OBJECT Built-in

Description

Use this built-in to run a report from within a form. You can run the report against either a local or remote database server.

Syntax

FUNCTION RUN_REPORT_OBJECT
(*report_id* REPORT_OBJECT*);*

FUNCTION RUN_REPORT_OBJECT
(*report_name* VARCHAR2*);*

FUNCTION RUN_REPORT_OBJECT
(*report_id* REPORT_OBJECT,
paramlist_name VARCHAR2);

FUNCTION RUN_REPORT_OBJECT
(*report_id* REPORT_OBJECT,
paramlist_id PARAMLIST);

FUNCTION RUN_REPORT_OBJECT
(*report_name* VARCHAR2,
paramlist_name VARCHAR2);

FUNCTION RUN_REPORT_OBJECT
(*report_name* VARCHAR2,
paramlist_id PARAMLIST);

Built-in Type unrestricted procedure

Returns VARCHAR2

Enter Query Mode yes

Parameters

report_id

Specifies the unique ID of the report to be run. You can get the report ID for a particular report using the built-in FIND_REPORT_OBJECT

report_name

The name of the report object to run.

paramlist_name

The name you gave the parameter list object when you defined it. Datatype is `VARCHAR2`.

paramlist_id

The unique ID Forms Developer assigns when it creates the parameter list. Datatype is `PARAMLIST`.

Usage Notes

- Returns a VARCHAR2 value that uniquely identifies the report that is running either locally or on a remote report server. You can use this report ID string as a parameter to REPORT_OBJECT_STATUS , COPY_REPORT_OBJECT , and CANCEL_REPORT_OBJECT.
- If you invoke RUN_REPORT_OBJECT with a blank `Reports Server` property, the return value will be `NULL`. In that case, you cannot then use the built-ins REPORT_OBJECT_STATUS and COPY_REPORT_OBJECT_OUTPUT because they require an actual ID value.

RUN_REPORT_OBJECT Example

```
DECLARE
 repid REPORT_OBJECT;
 v_rep VARCHAR2(100);
 rep_status VARCHAR2(20);
BEGIN
 repid := FIND_REPORT_OBJECT('report4');
 v_rep := RUN_REPORT_OBJECT(repid);
 ...
END;
```

SCROLL_DOWN Built-in

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, Forms Developer 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.

Syntax

PROCEDURE SCROLL_DOWN;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

none

SCROLL_DOWN Examples

```
/*

** Built-in: SCROLL_DOWN
** Example: Scroll records down some.
*/
BEGIN
Scroll_Down;
END;
```

Related topic

SCROLL_UP Built-in

SCROLL_UP Built-in

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.

Syntax

PROCEDURE SCROLL_UP;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

none

SCROLL_UP Examples

```
/*

** Built-in: SCROLL_UP
** Example: Scroll records up some.
*/
BEGIN
Scroll_Up;
END;
```

Related topics

SCROLL_DOWN built-in

SCROLL_VIEW Built-in

Description

Moves the view to a different position on its canvas by changing the Viewport X Position on Canvas and Viewport 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, the window in which the canvas is displayed represents the view for that canvas. For a stacked canvas, the view size is controlled by setting the Viewport Width and Viewport Height properties.

Syntax

PROCEDURE SCROLL_VIEW
(*view_id* ViewPort*,*
x NUMBER,
y NUMBER);

PROCEDURE SCROLL_VIEW
(*view_name* VARCHAR2*,*
x NUMBER,
y NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

view_id

> Specifies the unique ID that Forms Developer 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 VARCHAR2.

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.

SCROLL_VIEW Examples

```
/*
** 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,VIEWPORT_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,VIEWPORT_Y_POS_ON_CANVAS);
Scroll_View( vw_id, new_x , old_y );
END;
```

SELECT_ALL Built-in

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.

Syntax

PROCEDURE SELECT_ALL;

Built-in Type restricted procedure

Enter Query Mode yes

Parameters

none

Related topics

COPY_REGION built-in

CUT_REGION built-in

SELECT_RECORDS Built-in

Description

When called from an On-Select trigger, initiates default Forms Developer 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 Forms Developer transaction processing.

Syntax

PROCEDURE SELECT_RECORDS;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

none

SELECT_RECORDS Restrictions

Valid only within an On-Select trigger.

SELECT_RECORDS Examples

```
/*

** Built-in: SELECT_RECORDS
** Example: Perform Forms Developer 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 Built-in

Description

Changes the label on one of the buttons in an alert.

Syntax

PROCEDURE SET_ALERT_BUTTON_PROPERTY
(alert_id ALERT,
button NUMBER*,*
property VARCHAR2*,*
value VARCHAR2*)*;

PROCEDURE SET_ALERT_BUTTON_PROPERTY
(*alert_name*VARCHAR2,
button NUMBER,
property VARCHAR2*,*
value VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

alert_id

Specifies the unique ID (data type ALERT) that Forms Developer assigns to the alert when it is created. Use FIND_ALERT to return the ID to an appropriately typed variable.

alert_name

Specifies the VARCHAR2 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 VARCHAR2 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_ALERT_PROPERTY Built-in

Description

Changes the message text for an existing alert.

Syntax

SET_ALERT_PROPERTY
(alert_id ALERT*,*
property NUMBER*,*
message VARCHAR2*)*;

SET_ALERT_PROPERTY
(*alert_name*VARCHAR2,
property NUMBER*,*
message VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

alert_id

Specifies the unique ID (data type ALERT) that Forms Developer assigns to the alert when it is created. Return the ID to an appropriately typed variable.

alert_name

Specifies the VARCHAR2 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 Forms Developer 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.

SET_ALERT_PROPERTY Restrictions

If the message text exceeds 200 characters, it will be truncated.

SET_ALERT_PROPERTY Example

```
/*

** Built-in: SET_ALERT_PROPERTY
** 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 Built-in

Description

Sets (or resets) the application property for the current application.

Syntax

SET_APPLICATION_PROPERTY
(*property* NUMBER,
value VARCHAR2)

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

property

Specifies the property you want to set for the given application. The possible properties are:

BUILTIN_DATE_FORMAT Specifies the Builtin date format mask.

CURSOR_STYLE Specifies the cursor style for the given application.

DATETIME_LOCAL_TZ Specifies the local time zone region for `DATETIME` items.

FLAG_USER_VALUE_TOO_LONG Specifies how Forms Developer should handle user-entered values that exceed an item's Maximum Length property. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

PLSQL_DATE_FORMAT Specifies the PLSQL date format mask.

value

The new value to be set for this property.

Related topic

GET_APPLICATION_PROPERTY built-in

SET_BLOCK_PROPERTY Built-in

Description

Sets the given block characteristic of the given block.

Syntax

SET_BLOCK_PROPERTY
(*block_id* Block,
property VARCHAR,
value VARCHAR);

SET_BLOCK_PROPERTY
(*block_id* Block,
property VARCHAR,
x NUMBER);

SET_BLOCK_PROPERTY
(*block_id* Block,
property VARCHAR,

x NUMBER
y NUMBER);

SET_BLOCK_PROPERTY
(*block_name* VARCHAR2,
property VARCHAR,
value VARCHAR);

SET_BLOCK_PROPERTY
(*block_name* VARCHAR2,
property VARCHAR,
x NUMBER);

SET_BLOCK_PROPERTY
(*block_name* VARCHAR2,
property VARCHAR,
x NUMBER,
y NUMBER);

SET_BLOCK_PROPERTY
(*block_name*,
property,
value);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

block_id

The unique ID Forms Developer assigned to the block when you created it. Datatype is BLOCK.

block_name

The name you gave the block when you created it. Datatype is VARCHAR2.

property

Specify one of the following constants:

ALL_RECORDS Specifies whether all the records matching the query criteria should be fetched into the data block when a query is executed.

BLOCKSCROLLBAR_POSITION Specifies both the x and y positions of the block's scroll bar in the form coordinate units indicated by the Coordinate System form property.

BLOCKSCROLLBAR_X_POS Specifies the x position of the block's scroll bar in the form coordinate units indicated by the Coordinate System form property.

BLOCKSCROLLBAR_Y_POS Specifies the y position of the block scroll bar in the form coordinate units indicated by the Coordinate System form 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 VARCHAR2 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.

CURRENT_ROW_BACKGROUND_COLOR The color of the object's background region.

CURRENT_ROW_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.

CURRENT_ROW_FONT_NAME The font family, or typeface, that should be used for text in the object. The list of fonts available is system-dependent.

CURRENT_ROW_FONT_SIZE The size of the font, specified in points.

CURRENT_ROW_FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

CURRENT_ROW_FONT_STYLE The style of the font.

CURRENT_ROW_FONT_WEIGHT The weight of the font.

CURRENT_ROW_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.

DEFAULT_WHERE sets the WHERE clause on the block and OVERRIDES the value set at design time.

The WHERE reserved word is optional. The default WHERE clause can include references to global variables, form parameters, system 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.

DML_DATA_TARGET_NAME Specifies the name of the block's DML data source.

ENFORCE_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.

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 Forms Developer against non-ORACLE data sources. Valid values are UPDATEABLE_PRIMARY_KEY and NON_UPDATEABLE_PRIMARY_KEY.

LOCKING_MODE Specifies the block's LOCKING_MODE property. Valid values are DELAYED or IMMEDIATE.

MAX_QUERY_TIME Specifies the maximum query time. The operator can abort a query when the elapsed time of the query exceeds the value of this property.

MAX_RECORDS_FETCHED Specifies the maximum number of records that can be fetched. This property is only useful when the Query All Records property is set to Yes.

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.

ONETIME_WHERE Specifies a one time where clause for the block, overriding any previous ONETIME_WHERE clause. The ONETIME_WHERE clause can include references to global variables, forms parameters, and item values, specified with standard bind variable syntax.

OPTIMIZER_HINT Specifies a hint that Forms Developer 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. Do not include the actual words 'ORDER BY'. Forms Developer automatically prefixes the statement you supply with "ORDER BY."

PRECOMPUTE_SUMMARIES Specifies that the value of any summarized item in a data block is computed before the normal query is issued on the block. Forms Developer issues a special query that selects all records (in the database) of the summarized item and performs the summary operation (sum, count, etc.) over all the records.

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.

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_DATA_SOURCE_NAME Specifies the name of the block's query data source. Note: You cannot set a blocks' QUERY_DATA_SOURCE_NAME when the block's datasource is a procedure.

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.

UPDATE_CHANGED_COLUMNS Specifies that only those columns updated by an operator will be sent to the database. When Update Changed Columns Only is set to No, 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.

value 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 Forms Developer to lock detail records only at the execution of a commit action.

IMMEDIATE Specifies that you want Forms Developer 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 Forms Developer 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 Forms Developer to process records in the block on the basis that the underlying data source uses some form of unique key, or ROWID.

UPDATEABLE_PRIMARY_KEY Specifies that you want Forms Developer to process records in the block on the basis that the underlying data source allows for primary keys to be updated.

x

The NUMBER value of the axis coordinate specified in form coordinate system units. If setting both x and y positions this value refers to the x coordinate. When setting the y position only, this value refers to the y coordinate.

y

The NUMBER value of the y axis coordinate specified in form coordinate system units. This value applies when setting both x and y positions, and can be ignored for all other properties.

SET_BLOCK_PROPERTY Examples

```
/*

** 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;
```

Using BLOCKSCROLLBAR_POSITION:

```
/*

** Built-in: SET_BLOCK_PROPERTY
** Example: Set the x and y position of the block's scrollbar
**  to the passed x and y coordinates

*/
PROCEDURE Set_Scrollbar_Pos( blk_name IN VARCHAR2, xpos IN
NUMBER, ypos IN NUMBER )
IS
BEGIN
Set_Block_Property(blk_name, BLOCKSCROLLBAR_POSITION, xpos, ypos);
```

```
END;
```

Related topic

GET_BLOCK_PROPERTY built-in

SET_CANVAS_PROPERTY Built-in

Description

Sets the given canvas property for the given canvas.

Syntax

SET_CANVAS_PROPERTY
(*canvas_id* CANVAS,
property NUMBER,
value VARCHAR2);

SET_CANVAS_PROPERTY
(*canvas_id* CANVAS,
property NUMBER,
x NUMBER);

SET_CANVAS_PROPERTY
(*canvas_id* CANVAS,
property NUMBER,
x NUMBER,
y NUMBER);

SET_CANVAS_PROPERTY
(*canvas_name* VARCHAR2,
property NUMBER,
value VARCHAR2);

SET_CANVAS_PROPERTY
(*canvas_name* VARCHAR2,
property NUMBER,
x NUMBER);

SET_CANVAS_PROPERTY
(*canvas_name* VARCHAR2,
property NUMBER,
x NUMBER,
y NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

canvas_id

The unique ID Forms Developer assigned to the canvas object when you created it. Use the FIND_CANVAS built-in to return the ID to a variable of datatype CANVAS.

canvas_name

The name you gave the canvas object when you defined it. Datatype is VARCHAR2.

property

The property you want to set for the given canvas. Possible properties are:

BACKGROUND_COLOR The color of the object's background region.

CANVAS_SIZE The dimensions of the canvas (width, height).

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.

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_SPACING The width of the font, that is, the amount of space between characters (kerning).

FONT_STYLE The style of the font.

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.

HEIGHT The height of the canvas in characters.

TOPMOST_TAB_PAGE The name of the tab page that will appear to operators as the top-most (i.e., overlaying all other tab pages in the tab canvas).

VISUAL_ATTRIBUTE A valid named visual attribute that exists in the current form that you want Forms Developer to apply to the canvas.

WIDTH The width of the canvas in characters.

value

The VARCHAR2 value to be applied to the property you specified.

x

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

The NUMBER value of the y coordinate or the height, depending on the property you specified. Specify the argument in form coordinate system units.

SET_CANVAS_PROPERTY Restrictions

- You cannot enter a non-existent named visual attribute.

SET_CANVAS_PROPERTY Examples

```
/* Change the "background color" by dynamically setting the

** canvas color at runtime to the name of a visual attribute
** you created:
```

```
*/
BEGIN
SET_CANVAS_PROPERTY('my_cvs', visual_attribute, 'blue_txt');
END;
```

SET_CUSTOM_ITEM_PROPERTY Built-in

Note:

This built-in has been replaced by the SET_CUSTOM_PROPERTY Built-in You should use that built-in in any new form. The following information is provided only for maintenance purposes.

Description

Sets the value of a property of a JavaBean associated with a Bean Area item.

Syntax

The built-in is available for types `VARCHAR2`, `NUMBER`, or `BOOLEAN`.

SET_CUSTOM_ITEM_PROPERTY
(*item_id* ITEM,
property_name VARCHAR2,
value VARCHAR2);

SET_CUSTOM_ITEM_PROPERTY
(*item_id* ITEM,
property_name VARCHAR2,
value NUMBER);

SET_CUSTOM_ITEM_PROPERTY
(*item_id* ITEM,
property_name VARCHAR2,
value BOOLEAN);

SET_CUSTOM_ITEM_PROPERTY
(*item_name* VARCHAR2,
property_name VARCHAR2,
value VARCHAR2);

SET_CUSTOM_ITEM_PROPERTY
(*item_name* VARCHAR2,
property_name VARCHAR2,
value NUMBER);

```
SET_CUSTOM_ITEM_PROPERTY
(item_name VARCHAR2,
property_name VARCHAR2,
value BOOLEAN);
```

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

item_id

Specifies the unique ID that Forms Developer assigns to the item when created. The data type of the ID is `ITEM`.

item_name

The name of the Bean Area item associated with the target JavaBean. The name can be in the form of either a `VARCHAR2` literal or a variable set to the value of the name.

property_name

The particular property of the JavaBean container associated with this Bean Area.

value

The value for the specified property. Value must be of type `VARCHAR2`, `INTEGER`, or `BOOLEAN`.

Usage Notes

- In the JavaBean container, each property type must be represented by a single instance of the ID class, created by using `ID.registerProperty`.

- For each SET_CUSTOM_ITEM_PROPERTY Built-in executed in the form, the JavaBean container's `setProperty` method is called.
- The ID of the Bean Area item can be gained through FIND_ITEM('item_name').

SET_CUSTOM_PROPERTY Built-in

Description

Sets the value of a user-defined property in a Java pluggable component.

Syntax

The built-in is available for types VARCHAR2, NUMBER, or BOOLEAN.

SET_CUSTOM_PROPERTY
(*item* ITEM,
row_number NUMBER,
property_name VARCHAR2,
value VARCHAR2);

SET_CUSTOM_PROPERTY
(*item* ITEM,
row_number NUMBER,
property_name VARCHAR2,
value NUMBER);

SET_CUSTOM_PROPERTY
(*item* ITEM,
row_number NUMBER,
property_name VARCHAR2,
value BOOLEAN);

SET_CUSTOM_PROPERTY
(*item_name* VARCHAR2,
row_number NUMBER,
property_name VARCHAR2,
value VARCHAR2);

SET_CUSTOM_PROPERTY
(item_name VARCHAR2,
row_number NUMBER,
property_name VARCHAR2,
value NUMBER);

SET_CUSTOM_PROPERTY
(item_name VARCHAR2,
row_number NUMBER,
property_name VARCHAR2,
value BOOLEAN);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

item_id

Specifies the unique ID that Forms Developer assigns to the item when created. The data type of the ID is `ITEM`.

item_name

The name of the item associated with the target Java pluggable component. The name can be in the form of either a `VARCHAR2` literal or a variable set to the value of the name.

row_number

The row number of the instance of the item that you want to set (instance row numbers begin with 1). If you want to set all the instances, specify the constant ALL_ROWS.

property_name

The particular property of the Java component that you want to set.

value

The value for the specified property. Value must be of type `VARCHAR2`, `INTEGER`, or

`BOOLEAN`.

Usage Notes

- In the Java pluggable component, each custom property must be represented by a single instance of the ID class, created by using ID.registerProperty.
- For each Set_Custom_Property built-in executed in the form, the Java component's setProperty method is called.
- The name of the item can be gained through either Find_Item('Item_Name'), or simply via 'Item_Name'.

SET_CUSTOM_PROPERTY Built-in Example

In this example, the Java pluggable component is a JavaBean.

In the container (or wrapper) for the JavaBean:

```
private static final ID SETRATE =
ID.registerProperty("SetAnimationRate");
```

In the form, as part of the PL/SQL code activated by a When_Button_Pressed trigger on a "faster" button on the end-user's screen:

```
NewAnimationRate := gb.CurAnimationRate + 25;
  . . .
SET_CUSTOM_PROPERTY('Juggler_Bean', ALL_ROWS, 'SetAnimationRate',
NewAnimationRate);
```

In this SET_CUSTOM_PROPERTY built-in:

- `Juggler_Bean` is the name of the Bean Area item in the form. The item is associated with the container of the JavaBean.
- `SetAnimationRate` is a property in the container for the JavaBean.
- `NewAnimationRate` is a variable holding the new value for that property that is being passed to the JavaBean container.

SET_FORM_PROPERTY Built-in

Description

Sets a property of the given form.

Syntax

SET_FORM_PROPERTY
(*formmodule_id* FormModule,
property NUMBER,
value NUMBER);

SET_FORM_PROPERTY
(*formmodule_name* VARCHAR2,
property NUMBER,
value NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

formmodule_id

Specifies the unique ID that Forms Developer assigns to the form when created. The data type of the ID is FormModule.

formmodule_name

Specifies the name of the form module that you gave the form when creating it. The data type of the name is VARCHAR2.

property

Specifies the property you want to set for the form:

CURRENT_RECORD_ATTRIBUTE Specify the VARCHAR2 name of a named visual attribute to be associated with the given form. If the named visual attribute does not exist, you will get an error message.

CURRENT_ROW_BACKGROUND_COLOR The color of the object's background region.

CURRENT_ROW_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.

CURRENT_ROW_FONT_NAME The font family, or typeface, that should be used for text in the object. The list of fonts available is system-dependent.

CURRENT_ROW_FONT_SIZE The size of the font, specified in points.

CURRENT_ROW_FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

CURRENT_ROW_FONT_STYLE The style of the font.

CURRENT_ROW_FONT_WEIGHT The weight of the font.

CURRENT_ROW_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.

CURSOR_MODE Specifies the cursor state Forms Developer 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, PROPERTY_4_5, 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 Forms Developer 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 Forms Developer is to issue savepoints. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

VALIDATION Specifies whether Forms Developer 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.

value

The following constants can be passed as arguments to the property values described earlier:

BLOCK_SCOPE Specify when you want Forms Developer to validate data only at the block level. This means, for instance, that Forms Developer 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 Forms Developer to validate at the item level. This means, for instance, that Forms Developer 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 Forms Developer to validate at the record level. This means that Forms Developer validates each changed record when, for instance, it leaves the record.

SET_FORM_PROPERTY Examples

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;
```

Related topics

GET_FORM_PROPERTY built-in

SET_GROUP_CHAR_CELL Built-in

Description

Sets the value for the record group cell identified by the given row and column.

Syntax

SET_GROUP_CHAR_CELL
(*groupcolumn_id* GroupColumn*,*
row_number NUMBER*,*
cell_value VARCHAR2);

SET_GROUP_CHAR_CELL
(*groupcolumn_name* VARCHAR2*,*
row_number NUMBER*,*
cell_value VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

groupcolumn_id

The unique ID that Forms Developer assigns when it creates the column for the record group. 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

The name you gave to the column when you created it, preceded by the record group name and a dot, as in recordgroup_name.groupcolumn_name. The data type of the name is VARCHAR2.

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 VARCHAR2 column, specifies the VARCHAR2 value you intend to enter into a cell; for a LONG column, specifies the LONG value you intend to enter into a cell.

SET_GROUP_CHAR_CELL Restrictions

- You must create the specified row before setting the value of a cell in that row. Forms Developer 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.

SET_GROUP_CHAR_CELL Example

```
/*

** Built-in: SET_GROUP_CHAR_CELL
** Example: See ADD_GROUP_ROW
*/
```

SET_GROUP_DATE_CELL Built-in

Description

Sets the value for the record group cell identified by the given row and column.

Syntax

SET_GROUP_DATE_CELL
(*groupcolumn_id* GroupColumn*,*
row_number NUMBER*,*
cell_value DATE);

SET_GROUP_DATE_CELL
(*groupcolumn_name* VARCHAR2*,*
row_number NUMBER*,*
cell_value DATE);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

groupcolumn_id

> The unique ID that Forms Developer assigns when it creates the column for the record group. 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

> The name you gave to the column when you created it, preceded by the record group name and a dot, as in recordgroup_name.groupcolumn_name. The data type of the name is VARCHAR2.

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.

SET_GROUP_DATE_CELL Restrictions

- You must create the specified row before setting the value of a cell in that row. Forms Developer 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.

SET_GROUP_DATE_CELL Examples

```
/*

** 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 Built-in

Description

Sets the value for the record group cell identified by the given row and column.

Syntax

SET_GROUP_NUMBER_CELL
(*groupcolumn_id* GroupColumn*,*
row_number NUMBER*,*
cell_value NUMBER);

SET_GROUP_NUMBER_CELL
(*groupcolumn_name* VARCHAR2*,*
row_number NUMBER*,*
cell_value NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

groupcolumn_id

The unique ID that Forms Developer assigns when it creates the column for the record group. 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

The name you gave to the column when you created it, preceded by the record group name and a dot, as in recordgroup_name.groupcolumn_name. The data type of the name is VARCHAR2.

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.

SET_GROUP_NUMBER_CELL Restrictions

- 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.

SET_GROUP_NUMBER_CELL Example

```
/*

** Built-in: SET_GROUP_NUMBER_CELL
** Example: See ADD_GROUP_ROW
*/
```

SET_GROUP_SELECTION Built-in

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 Forms Developer 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.

Syntax

SET_GROUP_SELECTION
(*recordgroup_id* RecordGroup*,*
row_number NUMBER);

SET_GROUP_SELECTION
(*recordgroup_name* VARCHAR2*,*
row_number NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

recordgroup_id

> Specifies the unique ID that Forms Developer 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 VARCHAR2.

row_number

Specifies the number of the record group row that you want to select. The value you specify is a NUMBER.

SET_GROUP_SELECTION Examples

```
/*

** 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_ITEM_INSTANCE_PROPERTY Built-in

Description

Modifies the current item instance in a block by changing the specified item property. SET_ITEM_INSTANCE_PROPERTY does not change the appearance of items that mirror the current instance.

You can reference any item in the current form. Note that SET_ITEM_INSTANCE_PROPERTY 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, SET_ITEM_INSTANCE_PROPERTY 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 SET_ITEM_INSTANCE_PROPERTY remains in effect until they::

- same item instance is referenced by another SET_ITEM_INSTANCE_PROPERTY, or
- same item instance is referenced by the DISPLAY_ITEM built-in, or
- instance of the item is removed (e.g., through a CLEAR_RECORD or a query), or
- current form is exited

Syntax

SET_ITEM_INSTANCE_PROPERTY
(*item_id* ITEM,
record_number NUMBER,
property NUMBER,
value VARCHAR2);

SET_ITEM_INSTANCE_PROPERTY
(*item_id* ITEM,
record_number NUMBER,
property NUMBER,
value NUMBER);

SET_ITEM_INSTANCE_PROPERTY
(*item_name* VARCHAR2,

record_number NUMBER,
property NUMBER,
value VARCHAR2);

SET_ITEM_INSTANCE_PROPERTY
(*item_name* VARCHAR2,
record_number NUMBER,
property NUMBER,
value NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

item_id

The unique ID that Forms Developer assigned to the object when it created it. Use the FIND_ITEM built-in to return the ID to a variable with datatype of ITEM.

record_number

The record number that you want to set. The record number is the record's position in the block. Specify as a whole number. You can specify CURRENT_RECORD if you want to set the block's current record.

item_name

The name you gave the item when you created it. Datatype is VARCHAR2.

property

The property you want to set for the given item. Possible properties are:

BORDER_BEVEL Specifies the item border bevel for the specified item instance. Valid values are RAISED, LOWERED, PLAIN (unbeveled), or " ". A value of " " causes the border bevel to be determined by the value specified at the item level at design-time or by SET_ITEM_PROPERTY at runtime.

Note: You cannot set BORDER_BEVEL if the item's Bevel property is set to None in Forms Developer.

INSERT_ALLOWED Applies only to records not retrieved from the database. When set to PROPERTY_TRUE at the item instance, item, and block levels, allows the end user to modify the item instance. Setting this property to PROPERTY_FALSE at the item instance, item, or block levels, prohibits the end user from modifying the item instance.

NAVIGABLE When set to PROPERTY_TRUE at the item instance and item levels, allows the end user to be able to navigate to the item instance using default keyboard navigation. Setting this property to PROPERTY_FALSE at the item instance or item levels, disables default keyboard navigation to the item instance.

REQUIRED Specify the constant PROPERTY_TRUE if you want to force the end user to enter a non-null value for the item instance. Setting this property to PROPERTY_FALSE at the item instance and item levels, indicates that the item instance is not required.

UPDATE_ALLOWED Applies only to records retrieved from the database. When set to PROPERTY_TRUE at the item instance, item, and block levels, allows the end user to modify the item instance. When set to PROPERTY_FALSE at the instance, item, or block levels, prohibits the end user from modifying the item instance.

VISUAL_ATTRIBUTE Specify a valid named visual attribute that exists in the current form or ''. Specifying '' leaves visual attribute unspecified at the item instance level.

Usage Notes

When working with properties specified at multiple levels (item instance, item, and block), consider the following guidelines:

- Required properties specified at multiple levels are ORed together
- Other boolean properties specified at multiple levels are ANDed together

The value derived from combining properties specified at the item instance, item, and block levels is called the *effective value*. Some of the effects of these two rules are as follows:

- setting INSERT_ALLOWED to true has no effect at the item instance level unless it is set consistently at the block and item levels. For example, your user cannot type data into an item instance if INSERT_ALLOWED is true at the instance level, but not at the item or block levels.
- setting NAVIGABLE to true has no effect at the item instance level unless it is set consistently at the item and item instance levels
- Setting NAVIGABLE to true may affect whether the block is considered enterable. A block's read-only Enterable property will be true if and only if its current record contains an item instance whose effective value for the NAVIGABLE property is true.
- setting REQUIRED to false has no effect at the item instance level unless it is set consistently at the item and item instance levels.
- setting UPDATE_ALLOWED to true has no effect at the item instance level unless it is set consistently at the block, item, and item instance levels.
- setting BORDER_BEVEL at the item instance level will override the item level BORDER_BEVEL setting, except when the item instance BORDER_BEVEL property is unspecified (that is, set to " ").
- setting VISUAL_ATTRIBUTE at the item instance level will override the properties at the item and block levels unless you specify a partial visual attribute, in which case a merge will occur between the partial visual attribute and the item's current visual attribute. If VISUAL_ATTRIBUTE is set to " " at the item instance level, the item-level settings of this property are used.
- When a new record is created, its item instance properties are set to values that do not override the values specified at higher levels. For example, the BORDER_BEVEL and VISUAL_ATTRIBUTE properties get set to " ", REQUIRED is set to false, and other boolean properties are set to true.
- Setting an item instance property does not affect the item instance properties of any items that mirror the specified item.
- An instance of a poplist will, when selected, display an extra null value if its current value is NULL or if its Required property is set to false. When selecting the current value of an instance of a text list (t-list), it will be unselected (leaving the t-list with no selected value) if its Required property is set to false. If its Required property is set to true, selecting a t-list instance's current value will have no effect, that is, the value will remain selected.

SET_ITEM_INSTANCE_PROPERTY Examples

```
/*

** Built-in: SET_ITEM_INSTANCE_PROPERTY
** Example: Change the visual attribute of each item instance 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;
Set_Item_Instance_Property( cur_itm, CURRENT_RECORD,
VISUAL_ATTRIBUTE,'My_Favorite_Named_Attribute');
cur_itm := Get_Item_Property( cur_itm, NEXTITEM );
END LOOP;
END;
```

SET_ITEM_PROPERTY Built-in

Description

Modifies all instances of an item in a block by changing a specified item property. Note that in some cases you can *get* but not *set* certain object properties.

Syntax

SET_ITEM_PROPERTY
(*item_id* ITEM,
property NUMBER,
value VARCHAR2);

SET_ITEM_PROPERTY
(*item_name* VARCHAR2,
property NUMBER,
value VARCHAR2);

SET_ITEM_PROPERTY
(*item_id* ITEM,
property NUMBER,
x NUMBER);

SET_ITEM_PROPERTY
(*item_name* VARCHAR2,
property NUMBER,
x NUMBER);

SET_ITEM_PROPERTY
(*item_id* ITEM,
property NUMBER,
x NUMBER,
y NUMBER);

SET_ITEM_PROPERTY
(*item_name* VARCHAR2,
property NUMBER,
x NUMBER,
y NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

item_id

The unique ID that Forms Developer assigned to the object when it created it. Use the FIND_ITEM built-in to return the ID to a variable with datatype of ITEM.

item_name

The name you gave the item when you created it. Datatype is VARCHAR2.

property

The property you want to set for the given item. Possible properties are:

ALIGNMENT The text alignment (text and display items only). Valid values are ALIGNMENT_START, ALIGNMENT_END, ALIGNMENT_LEFT, ALIGNMENT_ CENTER, ALIGNMENT_RIGHT.

AUTO_HINT Determines if Forms Developer will display help hints on the status line automatically when input focus is in the specified item. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

AUTO_SKIP Specifies whether the cursor should skip to the next item automatically when the end user enters the last character in a text item. Valid only for a text item. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

BACKGROUND_COLOR The color of the object's background region.

BORDER_BEVEL Specifies the item border bevel for the specified item instance. Valid values are RAISED, LOWERED, or PLAIN (unbeveled).

Note: You cannot set BORDER_BEVEL if the item's Bevel property is set to None in Forms Developer.

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.

CONCEAL_DATA Specify the constant PROPERTY_TRUE if you want the item to remain blank or otherwise obscured when the end user enters a value. Specify the constant PROPERTY_FALSE if you want any value that is typed into the text item to be visible.

CURRENT_RECORD_ATTRIBUTE Specifies the VARCHAR2 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.

CURRENT_ROW_BACKGROUND_COLOR The color of the object's background region.

CURRENT_ROW_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.

CURRENT_ROW_FONT_NAME The font family, or typeface, that should be used for text in the object. The list of fonts available is system-dependent.

CURRENT_ROW_FONT_SIZE The size of the font, specified in points.

CURRENT_ROW_FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

CURRENT_ROW_FONT_STYLE The style of the font.

CURRENT_ROW_FONT_WEIGHT The weight of the font.

CURRENT_ROW_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.

DIRECTION Specifies the layout direction for bidirectional objects. Valid values are

DIRECTION_DEFAULT, RIGHT_TO_LEFT, LEFT_TO_RIGHT.

DISPLAYED Specifies whether the item will be displayed/enabled or hidden/disabled.

ECHO Specifies whether characters an end user 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 end users 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.

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.

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 hundredths of a point (i.e., for a font size of 8 points, the value should be set to 800).

FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

FONT_STYLE The style of the font.

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.

FORMAT_MASK Specifies the display format and input accepted for data in text items.

HEIGHT Specifies the height of the item.

HINT_TEXT Specifies the item-specific help text displayed on the message line at runtime. If the text specified is NULL, the original hint text, specified in Forms Developer, will be restored.

ICON_NAME Specifies the file name of the icon resource associated with a button item having the Iconic property set to YES.

IMAGE_DEPTH Specifies the depth of color to be applied to an image item.

INSERT_ALLOWED In a new record, allows end user 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_Allowed 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 Cursor Position property should be true or false. When Keep Cursor Position is true, the cursor returns to the same position it was in when it left the text item. When Keep Cursor 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 VARCHAR2 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 VARCHAR2 name of an LOV to be associated with the given item. If the LOV name does not exist, you will get an error message.

MERGE_CURRENT_ROW_VA Merges the contents of the specified visual attribute with the current row's visual attribute (rather than replacing it).

MERGE_TOOLTIP_ATTRIBUTE Merges the contents of the specified visual attribute with the tooltip's current visual attribute (rather than replacing it).

MERGE_VISUAL_ATTRIBUTE Merges the contents of the specified visual attribute with the object's current visual attribute (rather than replacing it).

MOUSE_NAVIGATE Specifies whether Forms Developer should navigate and set focus to the item when the end user activates the item with the mouse. Specify the constant PROPERTY_TRUE if you want the end user 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 end user 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. (Keyboard Navigable does not propagate changes to the Enabled property.)

NEXT_NAVIGATION_ITEM Specifies the name of the item that is defined as the "next navigation item" with respect to this current item.

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 Specifies 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.

PROMPT_ALIGNMENT_OFFSET Determines the distance between the item and its prompt.

PROMPT_BACKGROUND_COLOR The color of the object's background region.

PROMPT_DISPLAY_STYLE Determines the prompt's display style, either PROMPT_FIRST_RECORD, PROMPT_HIDDEN, or PROMPT_ALL_RECORDS.

PROMPT_EDGE Determines which edge the item's prompt is attached to, either START_EDGE, END_EDGE, TOP_EDGE, or BOTTOM_EDGE.

PROMPT_EDGE_ALIGNMENT Determines which edge the item's prompt is aligned to, either ALIGNMENT_START, ALIGNMENT_END, or ALIGNMENT_CENTER.

PROMPT_EDGE_OFFSET Determines the distance between the item and its prompt as a VARCHAR2 value.

PROMPT_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.

PROMPT_FONT_NAME The font family, or typeface, that should be used for text in the object. The list of fonts available is system-dependent.

PROMPT_FONT_SIZE The size of the font, specified in points.

PROMPT_FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

PROMPT_FONT_STYLE The style of the font.

PROMPT_FONT_WEIGHT The weight of the font.

PROMPT_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.

PROMPT_TEXT Determines the text label that displays for an item.

PROMPT_TEXT_ALIGNMENT Determines how the prompt is justified, either ALIGNMENT_START, ALIGNMENT_LEFT, ALIGNMENT_RIGHT, ALIGNMENT_CENTER, or ALIGNMENT_END.

PROMPT_VISUAL_ATTRIBUTE Specifies the named visual attribute that should be applied to the prompt at runtime.

QUERYABLE Specify the constant PROPERTY_TRUE if you want the end user 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 end user to enter a value for the item. Specify the constant PROPERTY_FALSE if the item is not to be required.

TOOLTIP_BACKGROUND_COLOR The color of the object's background region.

TOOLTIP_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.

TOOLTIP_FONT_NAME The font family, or typeface, that should be used for text in the object. The list of fonts available is system-dependent.

TOOLTIP_FONT_SIZE The size of the font, specified in points.

TOOLTIP_FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

TOOLTIP_FONT_STYLE The style of the font.

TOOLTIP_FONT_WEIGHT The weight of the font.

TOOLTIP_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.

TOOLTIP_TEXT Determines the item’s tooltip text.

UPDATE_ALLOWED Specify the constant PROPERTY_TRUE if you want the end user to be able to update the item. Specify the constant PROPERTY_FALSE if you want the item protected from update.

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 end user to be able to update the item only if its value is NULL. Specify the constant PROPERTY_FALSE if you want the end user 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.

VALIDATE_FROM_LIST Specifies that Forms Developer 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 Forms Developer should not use the LOV for validation.

VISIBLE Specifies whether the indicated item should be visible or hidden. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

Note: Setting Visible to false will cause other item property settings to change. Consult the "Propagation of Property Changes" section for details.

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.

value

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

VISIBLE 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 VARCHAR2 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.

If you want to **reset** the value of the property to be the value originally established for it at design time, enter two single quotes with no space between: ‘’. For example, SET_ITEM_PROPERTY(‘DEPTNO’, FORMAT_MASK, ‘’); would reset that format mask to its design-time value.

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, Forms Developer validates the change before it adjusts the property. If the change is validated, Forms Developer makes the change and leaves it in effect until another SET_ITEM_PROPERTY changes the same property or the current form is exited.

Illegal Settings

If the change is not validated, Forms Developer issues an error message. You cannot use SET_ITEM_PROPERTY to set the following item properties true or false, given the following target item conditions.

You cannot set this property parameter...	**To this restricted setting**	**If this target item condition is true:**
(All)	true/false	• NULL-canvas item (item's canvas property is null)
ENABLED	true/false	• current item
	true	• Visible item property is false
INSERT_ALLOWED	true	• Enabled item property is false
	true	• Visible item property is false
NAVIGABLE	true/false	• current item
	true	• Visible item property is false
QUERYABLE (Query Allowed)	true	• Visible item property is false
UPDATE_ALLOWED	true	• Enabled item property is false
	true	• Conceal Data item property is true
UPDATE_NULL (Update if NULL)	true	• Enabled item property is false
	true	• Conceal Data item property is true
VISIBLE	true/false	• current item

Forms Developer 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 Forms Developer 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 Forms Developer next validates it under normal circumstances.

For example, suppose the application has a required text item, such as Employee ID. In the application, the end user 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, Forms Developer marks an existing NULL value as VALID. Later in the application, when you set the REQUIRED property to true again, Forms Developer 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 Forms Developer 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.

The following table shows the SET_ITEM_PROPERTY settings that cause Forms Developer to propagate changes across item properties:

Setting this property parameter...	**To this setting**	**Also causes these propagated changes:**
ENABLED	False	• sets the Navigable item property to False • sets the Update_Null item property to False • sets the Updateable item property to False • sets the Required item property to False

DISPLAYED	False	• sets the Enabled and Navigable item properties to False • sets the Updateable item property to False • sets the Update_Null item property to False • sets the Required item property to False • sets the Queryable item property to False
UPDATEABLE	True	• sets the Update_Null item property to False
UPDATE_NULL	True	• sets the Updateable item property to False

SET_ITEM_PROPERTY Examples

```
/*

** 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 Built-in

Description

Sets the given LOV property for the given LOV.

Syntax

SET_LOV_COLUMN_PROPERTY
(*lov_id* LOV,
colnum NUMBER,
property NUMBER,
value VARCHAR2*);*

SET_LOV_COLUMN_PROPERTY
(lov_name VARCHAR2,
colnum NUMBER,
property NUMBER,
value VARCHAR2*);*

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

lov_id

Specifies the unique ID that Forms Developer 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 VARCHAR2).

colnum

Specifies the column to be modified (as a NUMBER). 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.

Note: Setting the column title to NULL resets the column title to the title specified at design time.

WIDTH Specifies the width to be reserved in the LOV for displaying column values.

Note: Setting the column width to NULL results in a hidden, or non-displayed, column.

value

The VARCHAR2 or NUMBER value that represents the desired property setting.

SET_LOV_PROPERTY Built-in

Description

Sets the given LOV property for the given LOV.

Syntax

SET_LOV_PROPERTY
(*lov_id* LOV,
property NUMBER,
value NUMBER*);*

SET_LOV_PROPERTY
(lov_name VARCHAR2,
property NUMBER,
value NUMBER*);*

SET_LOV_PROPERTY
(lov_id LOV,
property NUMBER,
x NUMBER,
y NUMBER*);*

SET_LOV_PROPERTY
(lov_name VARCHAR2,
property NUMBER,
x NUMBER,
y NUMBER*);*

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

lov_id

Specifies the unique ID that Forms Developer 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 VARCHAR2).

property

Specifies the property you want to set for the given LOV. The possible properties are as follows:

AUTO_REFRESH Specifies whether Forms Developer 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 Forms Developer 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 VARCHAR2 name of the record group you are setting. You can create this record group in Forms Developer 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.

SET_LOV_PROPERTY Restrictions

- You can set only one property per call to the built-in.

SET_LOV_PROPERTY Examples

```
/*

** 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 Built-in

Description

Modifies the given properties of a menu item.

Syntax

SET_MENU_ITEM_PROPERTY
(menuitem_id MenuItem,
property NUMBER,
value NUMBER*);*

SET_MENU_ITEM_PROPERTY
(*menu_name.menuitem_name* VARCHAR2,
property NUMBER,
value NUMBER);

SET_MENU_ITEM_PROPERTY
(menuitem_id MenuItem,
property NUMBER,
value VARCHAR2*);*

SET_MENU_ITEM_PROPERTY
(*menu_name.menuitem_name* VARCHAR2,
property NUMBER,
value VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

menuitem_id

> Specifies the unique ID Forms Developer 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

Specifies the VARCHAR2 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.

ENABLED Specifies whether the menu item is enabled (thus active) or disabled (thus greyed out and unavailable to the operator).

ICON_NAME Specifies the file name of the icon resource associated with a menu item having the Icon in Menu property set to TRUE.

LABEL Specifies the character string for the menu item label.

VISIBLE Specifies whether the menu item is visibly displayed.

value

Specify one of the following constants or an appropriate character 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.

SET_MENU_ITEM_PROPERTY Restrictions

These restrictions apply only if the menu module's Use Security property is set to Yes:

- If the menu module Use Security property is Yes, 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 Forms Developer, Runform displays the item in a disabled state. In this case, you *can* set the menu item properties programmatically.

SET_MENU_ITEM_PROPERTY Examples

```
/*

** Built-in: SET_MENU_ITEM_PROPERTY
** Example: See GET_MENU_ITEM_PROPERTY
*/
```

SET_PARAMETER_ATTR Built-in

Description

Sets the type and value of an indicated parameter in an indicated parameter list.

Syntax

SET_PARAMETER_ATTR
(*list* PARAMLIST*,*
key VARCHAR2*,*
paramtype NUMBER*,*
value VARCHAR2);

SET_PARAMETER_ATTR
(*name* VARCHAR2*,*
key VARCHAR2*,*
paramtype NUMBER*,*
value VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

list or name

Specifies the parameter list. The actual parameter can be either a parameter list ID of type PARAMLIST, or the VARCHAR2 name of the parameter list.

key

The VARCHAR2 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 VARCHAR2 string.

Related topic

GET_PARAMETER_ATTR built-in

SET_RADIO_BUTTON_PROPERTY Built-in

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*.

Syntax

SET_RADIO_BUTTON_PROPERTY
(*item_id* ITEM,
button_name VARCHAR2,
property NUMBER,
value NUMBER);

SET_RADIO_BUTTON_PROPERTY
(*item_name* VARCHAR2,
button_name VARCHAR2,
property NUMBER,
value NUMBER);

SET_RADIO_BUTTON_PROPERTY
(*item_id* ITEM,
button_name VARCHAR2,
property NUMBER,
value VARCHAR2);

SET_RADIO_BUTTON_PROPERTY
(*item_name* VARCHAR2,
button_name VARCHAR2,
property NUMBER,
value VARCHAR2);

SET_RADIO_BUTTON_PROPERTY
(*item_id* ITEM,
button_name VARCHAR2,
property NUMBER,
x NUMBER,
y NUMBER);

SET_RADIO_BUTTON_PROPERTY

(*item_name* VARCHAR2,
button_name VARCHAR2,
property NUMBER,
x NUMBER,
y NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

item_id

Specifies the radio group item ID. Forms Developer 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 VARCHAR2.

button_name

Specifies the name of the radio button whose property you want to set. The data type of the name is VARCHAR2.

property

Specifies the property you want to set. The possible property constants you can set are as follows:

BACKGROUND_COLOR The color of the object's background region.

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.

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.

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_SPACING The width of the font, that is, the amount of space between characters (kerning).

FONT_STYLE The style of the font.

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.

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.

PROMPT_BACKGROUND_COLOR The color of the object's background region.

PROMPT_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.

PROMPT_FONT_NAME The font family, or typeface, that should be used for text in the object. The list of fonts available is system-dependent.

PROMPT_FONT_SIZE The size of the font, specified in points.

PROMPT_FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

PROMPT_FONT_STYLE The style of the font.

PROMPT_FONT_WEIGHT The weight of the font.

PROMPT_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.

VISIBLE 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.

VISUAL_ATTRIBUTE Specifies a valid named visual attribute that exists in the current form that you want Forms Developer 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 VARCHAR2 value. The data type of the value you enter is determined by the data type of the property you specified. If you enter a VARCHAR2 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.

SET_RADIO_BUTTON_PROPERTY Examples

```
/*

** 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 Built-in

Description

Sets the specified record property to the specified value.

Syntax

SET_RECORD_PROPERTY
(*record_number* NUMBER*,*
block_name VARCHAR2,
property NUMBER,
value NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

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 VARCHAR2.

property

Use the following property:

STATUS Specifies that you intend to change the record status. STATUS is a constant.

value

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. Changed but uncleared or uncommitted records cannot be assigned a status of NEW.

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.

SET_RECORD_PROPERTY Restrictions

The following table illustrates the valid transition states of a record.

Current Status	**Target Status**			
	NEW	*QUERY*	*INSERT*	*CHANGED*
NEW	yes	yes1	yes2	no
QUERY	yes4	yes	no	yes
INSERT	yes4	yes3	yes	no
CHANGED	yes4	no	no	yes

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 Forms Developer returns an error.
4. Records that have been changed but not yet committed or cleared cannot be assigned a status of NEW.

SET_RECORD_PROPERTY Examples

```
/*

** 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 Built-in

Description

Sets the given relation property in a master-detail relationship.

Syntax

SET_RELATION_PROPERTY
(*relation_id* Relation,
property NUMBER,
value NUMBER);

SET_RELATION_PROPERTY
(*relation_name* VARCHAR2,
property NUMBER,
value NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

relation_id

Specifies the unique ID that Forms Developer assigns the relation when it creates the relation object. This can occur automatically when you define a master-detail relationship in Forms Developer, or you can explicitly create the relation. The data type of the ID is Relation.

relation_name

Specifies the name you or Forms Developer gave the relation object when defining it. The data type of the name is VARCHAR2.

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.

value 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, Forms Developer 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, Forms Developer 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.

SET_RELATION_PROPERTY Restrictions

You can only set one property per call to this built-in.

SET_RELATION_PROPERTY Examples

```
/*

** 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_REPORT_OBJECT_PROPERTY Built-in

Description

Programmatically sets the value of a report property.

Syntax

PROCEDURE SET_REPORT_OBJECT_PROPERTY
(*report_id* REPORT_OBJECT,
property NUMBER,
value VARCHAR2
);

PROCEDURE SET_REPORT_OBJECT_PROPERTY
(*report_name* VARCHAR2,
property NUMBER,
value VARCHAR2
);

PROCEDURE SET_REPORT_OBJECT_PROPERTY
(*report_id* REPORT_OBJECT,
property NUMBER,
value NUMBER
);

PROCEDURE SET_REPORT_OBJECT_PROPERTY
(*report_name* VARCHAR2,
property NUMBER,
value NUMBER
);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

report_id

Specifies the unique ID of the report. You can get the report ID for a particular report using FIND_REPORT_OBJECT .

report_name

Specifies the unique name of the report.

property

One of the following constants:

REPORT_EXECUTION_MODE: The report execution mode, either BATCH or RUNTIME

REPORT_COMM_MODE: The report communication mode, either SYNCHRONOUS or ASYNCHRONOUS

REPORT_DESTYPE: The report destination type, either PREVIEW, FILE, PRINTER, MAIL, CACHE or SCREEN

One of the following strings:

REPORT_FILENAME: The report filename

REPORT_SOURCE_BLOCK: The report source block name

REPORT_QUERY_NAME: The report query name

REPORT_DESNAME: The report destination name

REPORT_DESFORMAT: The report destination format

REPORT_SERVER: The report server name

REPORT_OTHER: The other user-specified report properties

value

One of the following constants:

REPORT_EXECUTION_MODE: Value must be BATCH or RUNTIME

REPORT_COMM_MODE: Value must be SYNCHRONOUS or ASYNCHRONOUS

REPORT_DESTYPE: Value must be PREVIEW, FILE, PRINTER, MAIL, CACHE, or SCREEN

One of the following strings:

REPORT_FILENAME: Value must be of type VARCHAR2

REPORT_SOURCE_BLOCK: Value must be of type VARCHAR2

REPORT_QUERY_NAME: Value must be of type VARCHAR2

REPORT_DEST_NAME: Value must be of type VARCHAR2

REPORT_DEST_FORMAT: Value must be of type VARCHAR2

REPORT_SERVER: Value must be of type VARCHAR2

REPORT_OTHER: Value must be of type VARCHAR2

Usage Notes

- SET_REPORT_OBJECT_PROPERTY sets properties using constant or string values. The value type depends on the particular property being set, as specified above. In contrast, GET_REPORT_OBJECT_PROPERTY returns a string value for all properties.

SET_REPORT_OBJECT_PROPERTY Examples

```
DECLARE
 repid REPORT_OBJECT;
 report_prop VARCHAR2(20);
BEGIN
 repid := find_report_object('report4');
 SET_REPORT_OBJECT_PROPERTY(repid, REPORT_EXECUTION_MODE, BATCH);
 SET_REPORT_OBJECT_PROPERTY(repid, REPORT_COMM_MODE, SYNCHRONOUS);
 SET_REPORT_OBJECT_PROPERTY(repid, REPORT_DESTYPE, FILE);
END;
```

SET_TAB_PAGE_PROPERTY Built-in

Description

Sets the tab page properties of the specified tab canvas page.

Syntax

SET_TAB_PAGE_PROPERTY
(*tab_page_id* TAB_PAGE,
property NUMBER,
value NUMBER);

SET_TAB_PAGE_PROPERTY
(*tab_page_name* VARCHAR2,
property NUMBER,
value NUMBER);

SET_TAB_PAGE_PROPERTY
(*tab_page_id* TAB_PAGE,
property NUMBER,
value VARCHAR2);

SET_TAB_PAGE_PROPERTY
(*tab_page_name* VARCHAR2,
property NUMBER,
value VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

tab_page_id

The unique ID Forms Developer assigns to the tab page when it creates it. Datatype is TAB_PAGE.

tab_page_name

The name you gave the tab page when you defined it. Datatype is VARCHAR2.

property

The property you want to set for the given tab page. Possible values are:

BACKGROUND_COLOR The color of the object's background region.

ENABLED Specify TRUE to enable the tab page, FALSE to disable it (i.e., make it greyed out and unavailable).

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.

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_SPACING The width of the font, that is, the amount of space between characters (kerning).

FONT_STYLE The style of the font.

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.

LABEL The character string for the tab page label.

VISIBLE Specify TRUE to make the tab page visible, FALSE to make it not visible. A tab page is reported visible if it is currently mapped to the screen, even if it is entirely hidden behind another tab page.

VISUAL_ATTRIBUTE Specifies the name of the visual attribute currently in force.

value

You can pass the following constants or the appropriate character strings as arguments to the property values described above:

PROPERTY_TRUE (sets the property to the TRUE state)

PROPERTY_FALSE (sets the property to the FALSE state)

SET_TAB_PAGE_PROPERTY Example

```
/* Example: Use SET_TAB_PAGE_PROPERTY to set the
** ENABLED property to TRUE for a tab page (if it currently
** is set to FALSE:
*/

DECLARE
tb_pg_id TAB_PAGE;

BEGIN
tb_pg_id := FIND_TAB_PAGE('tab_page_1');
IF GET_TAB_PAGE_PROPERTY(tb_pg_id, enabled) = 'FALSE' THEN
SET_TAB_PAGE_PROPERTY(tb_pg_id, enabled, property_true);
END IF;
END;
```

SET_TIMER Built-in

Description

Changes the settings for an existing timer. You can modify the interval, the repeat parameter, or both.

Syntax

SET_TIMER
(*timer_id* Timer,
milliseconds NUMBER,
iterate NUMBER);

SET_TIMER
(*timer_name* VARCHAR2,
milliseconds NUMBER,
iterate NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

timer_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

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.

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.

SET_TIMER Restrictions

- 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, Forms Developer 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.

SET_TIMER Example

```
/*

** Built-in: SET_TIMER
** Example: See FIND_TIMER
*/
```

SET_TREE_NODE_PROPERTY Built-in

Description

Sets the state of a branch node.

Syntax

PROCEDURE SET_TREE_NODE_PROPERTY
(*item_name* VARCHAR2,
node FTREE.NODE,
property NUMBER,
value NUMBER);

PROCEDURE SET_TREE_NODE_PROPERTY
(*item_name* VARCHAR2,
node FTREE.NODE,
property NUMBER,
value VARCHAR2);

PROCEDURE SET_TREE_NODE_PROPERTY
(*item_id* ITEM,
node FTREE.NODE,
property NUMBER,
value NUMBER);

PROCEDURE SET_TREE_NODE_PROPERTY
(*item_id* ITEM,
node FTREE.NODE,
property NUMBER,
value VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode no

Parameters

item_name

Specifies the name of the object created at design time. The data type of the name is

VARCHAR2 string.

Item_id

Specifies the unique ID that Forms Developer 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.

node

Specifies a valid node.

property

Specify one of the following properties:

NODE_STATE Possible values are EXPANDED_NODE, COLLAPSED_NODE, and LEAF_NODE.

NODE_LABEL Sets the label of the node.

NODE_ICON Sets the icon of the node.

NODE_VALUE Sets the value of the node.

value

The actual value you intend to pass.

SET_TREE_NODE_PROPERTY Example

This code could be used in a WHEN-TREE-NODE-SELECTED trigger to change the icon of the node clicked on.

```
/*

** Built-in: SET_TREE_NODE_PROPERTY
*/

DECLARE
htree ITEM;
current_node FTREE.NODE;
find_node FTREE.NODE;
BEGIN
```

```
-- Find the tree itself.
htree := FIND_ITEM('tree_block.htree3');
-- Change the icon of the clicked node.
-- The icon file will be located
-- in the virtual directory.
FTREE.SET_TREE_NODE_PROPERTY(htree, :SYSTEM.TRIGGER_NODE,
Ftree.NODE_ICON, 'Open');
END;
```

SET_TREE_PROPERTY Built-in

Description

Sets the value of the indicated hierarchical tree property.

Syntax

PROCEDURE SET_TREE_PROPERTY
(*item_name* VARCHAR2,
property NUMBER,
value NUMBER);

PROCEDURE SET_TREE_PROPERTY
(*item_name* VARCHAR2,
property NUMBER,
value VARCHAR2);

PROCEDURE SET_TREE_PROPERTY
(*item_name* VARCHAR2,
property NUMBER,
value RECORDGROUP);

PROCEDURE SET_TREE_PROPERTY
(*item_id* ITEM,
property NUMBER,
value NUMBER);

PROCEDURE SET_TREE_PROPERTY
(*item_id* ITEM,
property NUMBER,
value VARCHAR2);

PROCEDURE SET_TREE_PROPERTY
(*item_id* ITEM,
property NUMBER,
value RECORDGROUP);

Built-in Type unrestricted procedure

Enter Query Mode no

Parameters

item_name

Specifies the name of the object created at design time. The data type of the name is VARCHAR2 string.

Item_id

Specifies the unique ID that Forms Developer 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.

property

Specify one of the following properties:

RECORD_GROUP Replaces the data set of the hierarchical tree with a record group and causes it to display.

QUERY_TEXT Replaces the data set of the hierarchical tree with an SQL query and causes it to display.

ALLOW_EMPTY_BRANCHES Possible values are PROPERTY_TRUE and PROPERTY_FALSE.

value

Specify the value appropriate to the property you are setting:

PROPERTY_TRUE The property is to be set to the TRUE state.

PROPERTY_FALSE The property is to be set to the FALSE state.

SET_TREE_PROPERTY Examples

This code could be used in a WHEN-NEW-FORM-INSTANCE trigger to initially populate the hierarchical tree with data.

```
/*

** Built-in: SET_TREE_PROPERTY
*/

DECLARE
htree ITEM;
v_ignore NUMBER;
rg_emps RECORDGROUP;

BEGIN

-- Find the tree itself.
htree := FIND_ITEM('tree_block.htree3');
-- Check for the existence of the record group.
rg_emps := FIND_GROUP('emps');
IF NOT ID_NULL(rg_emps)
THEN  DELETE_GROUP(rg_emps);
END IF;
-- Create the record group.
rg_emps := CREATE_GROUP_FROM_QUERY('rg_emps',
'select 1, level, ename, NULL, to_char(empno) '
||  'from emp '
||  'connect by prior empno = mgr '
||  'start with job = ''PRESIDENT''');
-- Populate the record group with data.
v_ignore := POPULATE_GROUP(rg_emps);
-- Transfer the data from the record group to the hierarchical
-- tree and cause it to display.
FTREE.SET_TREE_PROPERTY(htree, FTREE.RECORD_GROUP, rg_emps);
END;
```

SET_TREE_SELECTION Built-in

Description

Specifies the selection of a single node.

Syntax

```
PROCEDURE SET_TREE_SELECTION
(item_name VARCHAR2,
node NODE,
selection_type NUMBER);

PROCEDURE SET_TREE_SELECTION
(item_id ITEM,
node NODE,
selection_type NUMBER);
```

Built-in Type unrestricted procedure

Enter Query Mode no

Parameters

item_name

> Specifies the name of the object created at design time. The data type of the name is VARCHAR2 string.

Item_id

> Specifies the unique ID that Forms Developer 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.

node

Specifies a valid node.

selection_type

Specifies the type of selection:

SELECT_ON Selects the node.

SELECT_OFF Deselects the node.

SELECT_TOGGLE Toggles the selection state of the node.

SET_TREE_SELECTION Example

This code could be used in a WHEN-TREE-NODE-EXPANDED trigger and will mark the clicked node as selected.

```
/*

** Built-in: SET_TREE_SELECTION
*/

DECLARE
htree ITEM;
BEGIN
-- Find the tree itself.
htree := FIND_ITEM('tree_block.htree3');
-- Mark the clicked node as selected.
FTREE.SET_TREE_SELECTION(htree, :SYSTEM.TRIGGER_NODE,
FTREE.SELECT_ON);
END;
```

SET_VA_PROPERTY Built-in

Description

Modifies visual attribute property values for the specified property.

Syntax

SET_VA_PROPERTY
(*va_id* VISUALATTRIBUTE
property NUMBER
value VARCHAR2);

SET_VA_PROPERTY
(*va_name* VARCHAR2
property NUMBER
value VARCHAR2);

SET_VA_PROPERTY
(*va_id* VISUALATTRIBUTE
property NUMBER
value NUMBER);

SET_VA_PROPERTY
(*va_name* VARCHAR2
property NUMBER
value NUMBER);

Built-in Type unrestricted function

Enter Query Mode yes

Parameters

va_id

The unique ID Forms Developer assinged to the visual attribute when you created it. The data type is VISUALATTRIBUTE.

va_name

The name you gave the visual attribute when you created it. The data type is VARCHAR2.

Property value

Specify one of the following properties:

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.

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 hundreds of points.

FONT_SPACING The width of the font, that is, the amount of space between characters (kerning).

FONT_STYLE The style of the font.

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.

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 change the FONT_NAME for the item, specify the value, in other words, the label, as a VARCHAR2 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.

If you want to reset the value of the property to the value originally established for it at design time, enter two single quotes `''` with no space between. For example, SET_VA_PROPERTY('OVERDRAWN', FONT_SIZE, ''); resets that format size to its design-time value.

SET_VIEW_PROPERTY Built-in

Description

Sets a property for the indicated canvas. 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.

Syntax

SET_VIEW_PROPERTY
(*view_id* ViewPort,
property NUMBER,
value NUMBER);

SET_VIEW_PROPERTY
(*view_id* ViewPort,
property NUMBER,
x NUMBER,
y NUMBER);

SET_VIEW_PROPERTY
(*view_name* VARCHAR2,
property NUMBER,
value NUMBER);

SET_VIEW_PROPERTY
(*view_name* ViewPort,
property NUMBER,
x NUMBER,
y NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

view_id

The unique ID Forms Developer assigned the view when you created the canvas/view. Use the FIND_VIEW built-in to return the ID to an appropriately typed variable. Datatype is VIEWPORT.

view_name

The name you gave the canvas object when you defined it. Datatype is VARCHAR2.

property

Specifies one of the following properties:

DIRECTION The layout direction for bidirectional objects. Valid values are DIRECTION_DEFAULT, RIGHT_TO_LEFT, LEFT_TO_RIGHT.

DISPLAY_POSITION For a stacked view, 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.

HEIGHT For a stacked canvas, the height of the view. To change the size of the canvas itself, use SET_CANVAS_PROPERTY.

POSITION_ON_CANVAS An X, Y pair indicating the location of the view's upper-left corner relative to its canvas.

VIEWPORT_X_POS For a stacked view, the X coordinate for the view's upper-left corner relative to the window's content view.

VIEWPORT_Y_POS For a stacked view, the Y coordinate for the view's upper-left corner relative to the window's content view.

VIEWPORT_X_POS_ON_CANVAS The X coordinate for the view's upper-left corner relative to its canvas.

VIEWPORT_Y_POS_ON_CANVAS The Y coordinate for the the view's upper-left corner relative to its canvas.

VIEW_SIZE For a stacked canvas, the size of the view, as a width, height pair. To change the size of the canvas itself, use SET_CANVAS_PROPERTY.

VISIBLE Whether the view is to be displayed. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

WIDTH For a stacked canvas, the width of the view. To change the size of the canvas itself, use SET_CANVAS_PROPERTY.

value

Specify the value appropriate to the property you are setting:

PROPERTY_TRUE The property is to be set to the TRUE state.

PROPERTY_FALSE The property is to be set to the FALSE state.

x

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

The NUMBER value of the Y coordinate or the height, depending on the property you specified. Specify the argument in form coordinate system units.

Related topics

SET_CANVAS_PROPERTY Built-in

SET_VIEW_PROPERTY Built-in

SET_WINDOW_PROPERTY Built-in

SET_WINDOW_PROPERTY Built-in

Description

Sets a property for the indicated window.

Syntax

SET_WINDOW_PROPERTY
(*window_id* Window,
property NUMBER,
value VARCHAR2);

SET_WINDOW_PROPERTY
(*window_id* Window,
property NUMBER,
x NUMBER);

SET_WINDOW_PROPERTY
(*window_id* Window,
property NUMBER,
x NUMBER,
y NUMBER);

SET_WINDOW_PROPERTY
(*window_name* VARCHAR2,
property NUMBER,
value VARCHAR2);

SET_WINDOW_PROPERTY
(*window_name* VARCHAR2,
property NUMBER,
x NUMBER);

SET_WINDOW_PROPERTY
(*window_name* VARCHAR2,
property NUMBER,
x NUMBER,
y NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

window_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

property

Specify one of the following window properties:

BACKGROUND_COLOR The color of the object's background region.

DIRECTION Specifies the layout direction for bidirectional objects. Valid values are DIRECTION_DEFAULT, RIGHT_TO_LEFT, LEFT_TO_RIGHT.

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.

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_SPACING The width of the font, that is, the amount of space between characters (kerning).

FONT_STYLE The style of the font.

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.

HEIGHT Specifies the height of the window.

HIDE_ON_EXIT Specifies whether Forms Developer hides the current window automatically when the operator navigates to an item in another window. Valid values are PROPERTY_TRUE and PROPERTY_FALSE.

ICON_NAME Specifies the file name of the icon resource associated with a window item when the window is minimized.

POSITION Specifies an x, y pair indicating the location for the window on the screen.

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.

value 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.

Usage Notes

In normal operation (`USESDI=YES` parameter is not passed to the application), Forms run inside an MDI *application window* which may be represented by the Applet are in the HTML page or a separate window if the separateFrame applet parameter has ben set to True. 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:

- Set_Window_Property(FORMS_MDI_WINDOW, POSITION, 5,10)
- Set_Window_Property(FORMS_MDI_WINDOW, WINDOW_STATE, MINIMIZE)

SET_WINDOW_PROPERTY Restrictions

- 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.

SET_WINDOW_PROPERTY Example

```
/*

** Built-in: SET_WINDOW_PROPERTY
** Example: See FIND_WINDOW
*/
```

SHOW_ALERT Built-in

Description

Displays the given alert, and returns a numeric value when the operator selects one of three alert buttons.

Syntax

SHOW_ALERT
(*alert_id* Alert);

SHOW_ALERT
(*alert_name* VARCHAR2);

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.

If the operator selects...	**Forms Developer returns**
Button 1	ALERT_BUTTON1
Button 2	ALERT_BUTTON2
Button 3	ALERT_BUTTON3

Enter Query Mode yes

Parameters

alert_id

The unique ID that Forms Developer 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 VARCHAR2.

SHOW_ALERT Example

```
/*

** Built-in: SHOW_ALERT
** Example: See FIND_ALERT and SET_ALERT_PROPERTY
*/
```

SHOW_EDITOR Built-in

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.

Syntax

SHOW_EDITOR
(*editor_id* Editor,
message_in VARCHAR2,
message_out VARCHAR2,
result BOOLKAN);

SHOW_EDITOR
(*editor_id* Editor,
message_in VARCHAR2,
x NUMBER,
y NUMBER,
message_out VARCHAR2,
result BOOLEAN);

SHOW_EDITOR
(*editor_name* VARCHAR2,
message_in VARCHAR2,
message_out VARCHAR2,
result BOOLEAN);

SHOW_EDITOR
(*editor_name* VARCHAR2,
message_in VARCHAR2,
x NUMBER,
y NUMBER,
message_out VARCHAR2,
result BOOLEAN);

Built-in Type unrestricted procedure that returns two OUT parameters (*result* and *message_out*)

Enter Query Mode yes

Parameters

editor_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

message_i

Specifies a required IN parameter of VARCHAR2 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 VARCHAR2 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_ou*t is NULL.

SHOW_EDITOR Restrictions

- *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 VARCHAR2 by Forms Developer when passed to the editor. However, if you are passing *message_out* to something other than a VARCHAR2 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.

SHOW_EDITOR Examples

```
/*

** 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 Built-in

Description

Displays the Keys screen. When the operator presses a function key, Forms Developer redisplays the form as it was before invoking the SHOW_KEYS built-in.

Syntax

SHOW_KEYS;

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

none

SHOW_KEYS Examples

```
/*

** Built-in: SHOW_KEYS
** Example: Display valid function key bindings
*/
BEGIN
Show_Keys;
END;
```

Related topic

Display in Keyboard Help property

SHOW_LOV Built-in

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.

Syntax

SHOW_LOV
(*lov_id* LOV);

SHOW_LOV
(*lov_id* LOV,
x NUMBER,
y NUMBER);

SHOW_LOV
(*lov_name* VARCHAR2);

SHOW_LOV
(*lov_name* VARCHAR2,
x NUMBER,
y NUMBER);

Built-in Type unrestricted function

Returns BOOLEAN

Enter Query Mode yes

Parameters

lov_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

x

Specifies the x coordinate of the LOV.

y

Specifies the y coordinate of the LOV.

Usage Notes

- When SHOW_LOV is used to display an LOV, Forms Developer ignores the LOV's Automatic Skip property.
- If you want to move the cursor to the next navigable item, use the LIST_VALUES built-in.

SHOW_LOV Restrictions

- If the lov_name argument is not supplied *and* there is no LOV associated with the current item, Forms Developer issues an error.
- If the record group underlying the LOV contains 0 records, the BOOLEAN return value for SHOW_LOV will be FALSE.

SHOW_LOV Examples

```
/*

** 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_VIEW Built-in

Description

Displays the indicated canvas at the coordinates specified by the canvas'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.

Syntax

SHOW_VIEW
(*view_id* ViewPort);

SHOW_VIEW
(*view_name* VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

view_id

> Specifies the unique ID that Forms Developer 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 VARCHAR2.

SHOW_VIEW Examples

```
/*
```

```
** 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 Built-in

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.

Syntax

SHOW_WINDOW
(*window_id* Window);

SHOW_WINDOW
(*window_id* Window*,*
x NUMBER*,*
y NUMBER);

SHOW_WINDOW
(*window_name* VARCHAR2);

SHOW_WINDOW
(*window_name* VARCHAR2*,*
x NUMBER*,*
y NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

window_id

Specifies the unique ID that Forms Developer 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 VARCHAR2.

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.

SHOW_WINDOW Examples

```
/*

** 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 Built-in

Description

Synchronizes the display's screen with the internal state of the form. That is, SYNCHRONIZE updates the screen display to reflect the information that Forms Developer has in its internal representation of the screen.

Syntax

SYNCHRONIZE;

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

none

SYNCHRONIZE Restrictions

SYNCHRONIZE only updates the screen display if both of the following conditions are true:

- Forms Developer is at the item level in the forms hierarchy (i.e., SYSTEM.CURRENT_ITEM is not NULL).

Note: Use SYNCRHONIZE with caution. Each call to SYNNCRHONIZE generates a network round-trip between the application server and the Forms Java client.

SYNCHRONIZE Example

```
/*

** Built-in: SYNCHRONIZE
** Example: Achieve an odometer effect by updating the
** screen as an items value changes quickly.
** Without synchronize, the screen is typically
** only updated when Forms Developer 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;
```

UNSET_GROUP_SELECTION Built-in

Syntax

UNSET_GROUP_SELECTION
(*recordgroup_id* RecordGroup,
row_number NUMBER);

UNSET_GROUP_SELECTION
(*recordgroup_name* VARCHAR2,
row_number 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 Forms Developer 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 Forms Developer 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 VARCHAR2.

row_number

Specifies the number of the record group row that you want to select. The value you specify is a NUMBER.

UNSET_GROUP_SELECTION 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 Built-in

Description

Navigates to the instance of the current item in the record with the next lowest sequence number.

Syntax

UP;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

None

Related topic

DOWN Built-in

PREVIOUS_RECORD Built-in

NEXT_RECORD Built-in

UPDATE_CHART Built-in

Description

A data block is updated whenever it is queried or when changes to it are committed. By default, when the block is updated, any charts based on the data block are automatically updated. You can use the UPDATE_CHART built-in to explicitly cause a chart item to be updated, even if the data block on which it is based has not been updated. For example, you may want update the chart to reflect uncommitted changes in the data block.

Syntax

PROCEDURE UPDATE_CHART
(*chart_name* VARCHAR2*,*
param_list_id PARAMLIST
);

PROCEDURE UPDATE_CHART
(*chart_name* VARCHAR2*,*
param_list_name VARCHAR2
);

PROCEDURE UPDATE_CHART
(*chart_id* ITEM*,*
param_list_id PARAMLIST
);

PROCEDURE UPDATE_CHART
(*chart_id* ITEM*,*
param_list_name VARCHAR2
);

PROCEDURE UPDATE_CHART
(*chart_id* ITEM
);

PROCEDURE UPDATE_CHART
(*chart_name* VARCHAR2
);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

chart_id

Specifies the unique ID of the chart.

chart_name

Specifies the unique name of the chart.

param_list_id

Specifies the unique ID of the chart parameter list.

param_list_name

Specifies the unique name of the chart parameter list.

UPDATE_RECORD Built-in

Description

When called from an On-Update trigger, initiates the default Forms Developer 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.

Syntax

UPDATE RECORD;

Built-in Type restricted procedure

Enter Query Mode no

Parameters

none

UPDATE_RECORD Restrictions

Valid only in an On-Update trigger.

USER_EXIT Built-in

Description

Calls the user exit named in the user_exit_string.

Syntax

USER_EXIT
(*user_exit_string* VARCHAR2);

USER_EXIT
(*user_exit_string* VARCHAR2*,*
error_string VARCHAR2);

Built-in Type unrestricted procedure

Enter Query Mode yes

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 Forms Developer should display if the user exit fails.

USER_EXIT Examples

```
/*

** Built-in: USER_EXIT
** Example: Invoke a 3GL program by name which has been
** properly linked into your current Forms Developer
** 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 Built-in

Description

VALIDATE forces Forms Developer to immediately execute validation processing for the indicated validation scope.

Syntax

VALIDATE
(validation_scope NUMBER*)*;

Built-in Type:

unrestricted procedure

Enter Query Mode yes

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, Forms Developer 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.

Note on runtime behavior

If an invalid field is detected when validation is performed, the cursor does not move to that field. Instead, the cursor remains in its previous position.

VALIDATE Examples

```
/*
** 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;
Execute_Query;
END IF;
END;
```

Related topic

Validation Unit property

WEB.SHOW_DOCUMENT Built-in

Syntax:

SHOW_DOCUMENT *(url, target);*

Built-in Type: unrestricted procedure

Enter Query Mode: yes

Description:

Specifies the URL and target window of a Web application.

Parameters:

url

Datatype is VARCHAR2. Specifies the Uniform Resource Locator of the document to be loaded.

target

Datatype is VARCHAR2. Specifies one of the following targets:

"_self" Causes the document to load into the same frame or window as the source document.

"_parent" Causes the target document to load into the parent window or frameset containing the hypertext reference. If the reference is in a window or top-level frame, it is equivalent to the target _self**.**

"_top" Causes the document to load into the window containing the hypertext link, replacing any frames currently displayed in the window.

"_blank" Causes the document to load into a new, unnamed top-level window.

Note that these targets are lowercase and enclosed in quotation marks.

Example

```
/*

** Built-in: WEB.SHOW_DOCUMENT
** Example: Display the specified URL in the target window.
*/

BEGIN
WEB.SHOW_DOCUMENT(‘http://www.abc.com’, ‘_self’);
END;
```

WRITE_IMAGE_FILE File Built-in

Description

Writes the image from a Forms Developer image item into the specified file.

Syntax

WRITE_IMAGE_FILE
(*file_name* VARCHAR2*,*
file_type VARCHAR2*,*
item_id ITEM,
compression_quality NUMBER,
image_depth NUMBER);

WRITE_IMAGE_FILE
(*file_name* VARCHAR2*,*
file_type VARCHAR2*,*
item_name VARCHAR2,
compression_quality NUMBER,
image_depth NUMBER);

Built-in Type unrestricted procedure

Enter Query Mode yes

Parameters

file_name

> The name of the file where the image is stored. The file name must adhere to your operating system requirements.

file_type

> The file type of the image: BMP, CALS, GIF, JFIF, JPEG, PICT, RAS, TIFF, or TPIC. The parameter takes a VARCHAR2 argument.

item_id

The unique ID Forms Developer assigned to the image item when you created it. Use the FIND_ITEM built-in to return the ID to an appropriately typed variable. Datatype is ITEM.

item_name

The name you gave the image item when you defined it. Datatype is VARCHAR2.

compression_quality

The degree of compression Forms Developer will apply to the image when it stores it to the file (optional). Datatype is VARCHAR2. Valid values are:NO_COMPRESSION, MINIMIZE_COMPRESSION, LOW_COMPRESSION, MEDIUM_COMPRESSION, HIGH_COMPRESSION, MAXIMIZE_COMPRESSION

image_depth

The degree of depth Forms Developer will apply to the image when it stores it to the file (optional). Datatype is VARCHAR2. Valid values are:ORIGINAL_DEPTH, MONOCHROME, GRAYSCALE, LUT (Lookup Table), RGB (Red, Green, Blue)

WRITE_IMAGE_FILE Restrictions

- The indicated file type must be compatible with the actual file type of the image.
- 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.
- Though you can read PCD and PCX files from the filesystem or the database, you cannot write image files to the filesystem in PCD or PCX format (using WRITE_IMAGE_FILE). (If you use a restricted file type when writing images to the filesystem, Forms Developer defaults the image file to TIFF format.)
- Writing a JPEG file from a Forms Developer image item will result in loss of resolution.
- The image is written to the middle tier server.

WRITE_IMAGE_FILE Examples

```
/* Built-in: WRITE_IMAGE_FILE
**
** 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',
maximize_compression,
original_depth);
END;
```

About Properties

Each object in a Forms Developer application possesses characteristics known as *properties*. An object's properties determine its appearance and functionality.

Related topic

About Setting and Modifying Properties

Properties

Alert properties

Application properties

Block properties

Canvas properties

Chart properties

Editor properties

Form Parameter properties

Form Module properties

Frame properties

Graphics properties

Item properties

LOV properties

Menu properties

Named Visual Attribute Properties

Prompt properties

Record properties

Record Group properties

Relation properties

Report properties

Tab Page properties

Trigger properties

Window properties

About setting and modifying properties

Each property description includes a **Set** heading that describes how you can set the property; either declaratively in Forms Developer (using the Property Palette), programmatically at runtime, or both.

Setting Properties Programmatically

To dynamically modify object properties programmatically, use the following Forms Developer built-ins subprograms:

SET_APPLICATION_PROPERTY

SET_BLOCK_PROPERTY

SET_CANVAS_PROPERTY

SET_FORM_PROPERTY

SET_ITEM_PROPERTY

SET_ITEM_INSTANCE_PROPERTY

SET_LOV_PROPERTY

SET_MENU_ITEM_PROPERTY

SET_PARAMETER_ATTR

SET_RADIO_BUTTON_PROPERTY

SET_RECORD_PROPERTY

SET_RELATION_PROPERTY

SET_REPORT_OBJECT_PROPERTY

SET_TAB_PAGE_PROPERTY

SET_VIEW_PROPERTY

SET_WINDOW_PROPERTY

You can programmatically determine the settings of most properties by using the set of corresponding built-ins to get properties (e.g., GET_ITEM_PROPERTY).

Reading Property Descriptions

Description

The property descriptions follow a general pattern. The property name is printed in a bold typeface and is followed by a brief description.

Applies to

The object class or classes for which this property is meaningful.

Set

Where you can set the property: in Forms Developer (using the Property Palette), programmatically at runtime, or both.

Refer to Built-in

Built-in(s) you can use to set the property, if you can set the property programmatically

Default

The default value of the property.

Required/Optional

Whether the property is required or optional.

Restrictions

Any restrictions potentially affecting usage of the property.

Usage Notes

Any particular usage considerations you should keep in mind when using the property.

Alert Properties

Alert style

Button 1 Label, Button 2 Label, Button 3 Label

Comments

Default Alert Button

Default Button

Message

Name

Property Class

Title

Application Properties

Calling_Form

Comments

Connect_String

Current_Form

Current_Form_Name

Cursor_Style

Datasource

Display_Height

Display_Width

Interaction_Mode

Isolation_Mode

Maximum_Query_Time

Maximum_Records_Fetched

Name

Operating_System

Password

Savepoint_Name

Timer_Name

User_Interface

Username

User_NLS_Lang

Automatic Skip Properties

Automatic Skip (Item) property

Automatic Skip (LOV) property

Block Properties

Application Instance

Block Description

Comments

Current_Record

Current Record Visual Attribute Group

Data Block Description

Database Block

Delete Allowed

Delete Procedure Arguments

Delete Procedure Name

Delete Procedure Result Set Columns

DML Array Size

DML Data Target Name

DML Data Target Type

Enforce Column Security

Enforce Primary Key

Enterable

First_Block

First_Detail_Relation

First_Item

First_Master_Relation

Insert Allowed (Block)

Insert Procedure Arguments

Insert Procedure Name

Insert Procedure Result Set Columns

Join Condition

Key Mode

Last_Item

Locking Mode

Lock Procedure Name

Lock Procedure Result Set Columns

Name

Navigation Style

NextBlock

Next Navigation Block

Number of Records Buffered

Number of Records Displayed

Optimizer_Hint

Order By

Precompute Summaries

Previous Navigation Block

PreviousBlock

Property Class

Query All Records

Query Allowed (Block)

Query Array Size

Query Data Source Arguments

Query Data Source Columns

Query Data Source Name

Query Data Source Type

Query_Hits

Query_Options

Record Orientation

Records_To_Fetch

Show Scroll Bar

Single Record

Status (Block)

Top_Record

Transactional Triggers

Update Allowed (Block)

Update Changed Columns Only

Update Procedure Arguments

Update Procedure Name

Update Procedure Result Set Columns

Where Clause/Order By Clause

Window_Handle

Canvas Properties

Bevel

Canvas Type

Comments

Form Horizontal Toolbar Canvas

Form Vertical Toolbar Canvas

Name

Popup Menu

Property Class

Raise on Entry

Size

Tab Attachment Edge

Tab Style

Topmost_Tab_Page

Viewport Height, Viewport Width

Viewport X Position, Viewport Y Position

Viewport X Position on Canvas, Viewport Y Position on Canvas

Visible (Canvas)

Visual Attribute Group

Width/Height

Window

X Position, Y Position

Chart Properties

Communication Mode (Chart)

Data Source Data Block (Chart)

Execution Mode (Chart)

Communication Mode Properties

Communication Mode (Chart) property

Communication Mode (Report) property

Data Source Data Block properties

Data Source Data Block (Chart) property

Data Source Data Block (Report) property

Editor Properties

Bottom Title (Editor)

Comments

Editor X Position, Editor Y Position

Name

Property Class

Show Horizontal Scroll Bar

Size

Top Title

Width/Height

Wrap Style

X Position, Y Position

Execution Mode Properties

Execution Mode (Chart) property

Execution Mode (Report) property

Forms Developer Properties

Properties

A

Access Key

Alert Style

Alias

Allow Expansion

Allow Multi-Line Prompts

Allow Start-Attached Prompts

Allow Top-Attached Prompts

Application Instance

Arrow Style

Associated Menus

Automatical Column Width

Automatic Display

Automatic Position

Automatic Query

Automatic Refresh

Automatic Select

Automatic Skip (Item)

Automatic Skip (LOV)

B

Background_Color

Bevel

Block Description

Bottom Title (Editor)

Builtin Date Format

Button 1 Label, Button 2 Label, Button 3 Label

C

Calculation Mode

Calling_Form

Canvas

Canvas Type

Cap Style

Case Insensitive Query

Case Restriction

Character Cell WD/HT

Check Box Mapping of Other Values

Checked

Clip Height

Clip Width

Clip X Position

Clip Y Position

Close Allowed

Closed

Column Mapping Properties

Column Name

Column Specifications

Column Title

Column Value

Command Text

Comments

Communication Mode (Chart)

Communication Mode (Report)

Compression_Quality

Conceal Data

Connect_String

Console Window

Coordinate System

Coordination

Coordination_Status

Copy Value from Item

Current_Form

Current_Form_Name

Current_Record

Current Record Visual Attribute

Current_Row_Background_Color

Current_Row_Fill_Pattern

Current_Row_Font_Name

Current_Row_Font_Size

Current_Row_Font_Spacing

Current_Row_Font_Style

Current_Row_Font_Weight

Current_Row_Foreground_Color

Cursor Mode

Cursor_Style

Custom Spacing

D

Dash Style

Data Source Data Block (Chart)

Data Source Data Block (Report)

Data Source X Axis

Data Source Y Axis

Data Type

Data Type (Record Group)

Database Block

Database_Value

Datasource

Default Alert Button

Default Button

Default Font Scaling

Deferred

Defer_Required_Enforcement

Delete Allowed

Delete Procedure Arguments

Delete Procedure Name

Delete Procedure Result Set Columns

Delete Record Behavior

Detail Block

Detail Reference Item

Direction

Display without Privilege

Display_Height

Display in Keyboard Help

Display Hint Automatically

Display Quality

Display_Width

Display Width (LOV)

Distance Between Records

Dither

DML Array Size

DML Data Target Name

DML Data Target Type

E

Edge Background Color

Edge Foreground Color

Edge Pattern

Editor

Editor X Position, Editor Y Position

Elements in List

Enabled (Item)

Enabled (Menu Item)

Enabled (Tab Page)

End Angle

Enforce Column Security

Enforce Primary Key

Enterable

Execution Mode (Chart)

Execution Mode (Report)

Execution Style

F

File_Name

Fill

Fill_Pattern

Filter Before Display

Fire in Enter-Query Mode

First_Block

First_Detail_Relation

First_Item

First_Master_Relation

First Navigation Block

Fixed Bounding Box

Font Spacing

Font_Name

Font_Size

Font_Spacing

Font_Style

Font_Weight

Foreground_Color

Form Horizontal Toolbar Canvas

Form Vertical Toolbar Canvas

Form_Name

Format Mask

Formula

Frame Alignment

Frame Title

Frame Title Alignment

Frame Title Font Name

Frame Title Font Size

Frame Title Font Spacing

Frame Title Font Style

Frame Title Font Weight

Frame Title Font Foreground Color

Frame Title Offset

Frame Title Reading Order

Frame Title Spacing

Frame Title Visual Attribute Group

G

Graphics Type

Group_Name

H

Hide on Exit

Highest Value Allowed

Hint (Item)

Horizontal Justification

Horizontal Margin

Horizontal Object Offset

Horizontal Origin

Horizontal Toolbar Canvas

I

Icon Filename

Icon in Menu

Iconic

Image_Depth

Image_Format

Implementation Class

Include REF Item

Inherit Menu

Initial Keyboard State

Initial Menu

Initial Value

Insert Allowed (Block)

Insert Allowed (Item)

Insert Procedure Arguments

Insert Procedure Name

Insert Procedure Result Set Columns

Interaction Mode

Isolation Mode

Item_Is_Valid

Item Roles

Item_Tab_Page (item)

Item Type

J

Join Condition

Join Style

Justification

K

Keep Cursor Position

Keyboard Accelerator

Keyboard Help Description

Keyboard Navigable

Keyboard State

Key Mode

L

Label (Item)

Label (Menu Item)

Label (Tab Page)

Last_Block

Last_Item

Last Query

Layout Data Block

Layout Style

Length

Line Spacing

Line Width

List Item Value

List of Values

List Style

List Type

List X Position

List Y Position

Lock Procedure Arguments

Lock Procedure Name

Lock Procedure Result Set Columns

Lock Record

Locking Mode

Lowest Value Allowed

M

Magic Item

Main Menu

Mapping of Other Values

Maximize Allowed

Maximum Length (Form Parameter)

Maximum Length (Item)

Maximum Objects Per Line

Maximum Query Time

Maximum Records Fetched

Menu Directory

Menu Filename

Menu Item Code

Menu Item Radio Group

Menu Item Type

Menu Module

Menu Role

Menu Style

Message

Minimize Allowed

Minimized Title

Modal

Module_NLS_Lang

Module Roles

Mouse Navigate

Mouse Navigation Limit

Move Allowed

Multi-Line

N

Name

Navigation Style

NextBlock

NextItem

Next_Detail_Relation

Next_Master_Relation

Next Navigation Block

Next Navigation Item

Number of Items Displayed

Number of Records Buffered

Number of Records Displayed

O

Operating_System

Optimizer_Hint

Order By

Other Reports Parameters

P

Parameter Data Type

Parameter Initial Value

Password

PLSQL Date Format

Popup Menu (Item, Canvas)

Position (X, Y)

Precompute Summaries

Prevent Masterless Operation

Previous Navigation Block

PreviousBlock

PreviousItem

Previous Navigation Item

Primary Canvas

Primary Key (Item)

Program Unit Text

Prompt

Prompt Alignment

Prompt Alignment Offset

Prompt Attachment Edge

Prompt Attachment Offset

Prompt Background Color

Prompt Display Style

Prompt Font Name

Prompt Font Size

Prompt Font Spacing

Prompt Font Style

Prompt Font Weight

Prompt Foreground Color

Prompt Fill Pattern

Prompt Justification

Prompt Reading Order

Prompt Visual Attribute Group

Prompt_Fill_Pattern

Prompt_Font_Name

Prompt_Font_Size

Prompt_Font_Spacing

Prompt_Font_Style

Prompt_Font_Weight

Prompt_Foreground_Color

Property Class

Q

Query All Records

Query Allowed (Block)

Query Allowed (Item)

Query Array Size

Query Data Source Arguments

Query Data Source Columns

Query Data Source Name

Query Data Source Type

Query Length

Query Name

Query Only

Query_Hits

Query_Options

R

Radio Button Value

Raise on Entry

Reading Order

Real Unit

Record Group

Record Group Fetch Size

Record Group Query

Record Group Type

Record Orientation

Records_To_Fetch

Relation Type

Rendered

Report Destination Format

Report Destination Name

Report Destination Type

Report Server

Required (Item)

Resize Allowed

Return Item (LOV)

Rotation Angle

Runtime Compatibility Mode

S

Savepoint Mode

Savepoint_Name

Scroll Bar Alignment

Scroll Bar Height

Scroll Bar Width

Share Library with Form

Show Horizontal Scroll Bar

Show Palette

Show Scroll Bar

Show Vertical Scroll Bar

Shrinkwrap

Single Object Alignment

Single Record

Size

Sizing Style

Start Angle

Start Prompt Alignment

Start Prompt Offset

Startup Code

Status (Block)

Status (Record)

Subclass Information

Submenu Name

Summarized Block

Summarized Item

Summary Function

Synchronize with Item

T

Tab Attachment Edge

Tab Page

Tab Page X Offset

Tab Page Y Offset

Tab Style

Tear-off Menu

Timer_Name

Title

Tooltip

Tooltip_Background_Color

Tooltip_Fill_Pattern

Tooltip_Font_Name

Tooltip_Font_Size

Tooltip_Font_Spacing

Tooltip_Font_Style

Tooltip_Font_Weight

Tooltip_Foreground_Color

Tooltip Visual Attribute Group

Topmost_Tab_Page (Canvas)

Top Prompt Alignment

Top Prompt Offset

Top Title

Top_Record

Transactional Triggers

Trigger Text

Trigger Type

U

Update Allowed (Block)

Update Allowed (Item)

Update Changed Columns Only

Update_Column

Update_Commit

Update Layout

Update Only if NULL

Update_Permission

Update Procedure Arguments

Update Procedure Name

Update Procedure Result Set Columns

Update_Query

Use 3D Controls

Use Security

User_Interface

User_NLS_Date_Format

User_NLS_Lang

Username

V

Validate from List

Validation

Validation Unit

Value When Checked

Value When Unchecked

Vertical Fill

Vertical Justification

Vertical Margin

Vertical Object Offset

Vertical Origin

Vertical Toolbar Canvas

Viewport Height, Viewport Width

Viewport X Position, Viewport Y Position

Viewport X Position on Canvas, Viewport Y Position on Canvas

Visible

Visible (Canvas)

Visible (Item)

Visible in Menu

Visible (Tab Page)

Visible In Horizontal/Vertical Menu Toolbar

Visual_Attribute

Visual Attribute Group

Visual Attribute Type

W

Where Clause/Order By Clause

Width/Height

Window

Window_Handle

Window_State

Window Style

Wrap Style

Wrap Text

X

X Corner Radius

X Position, Y Position

Y

Y Position, X Position

Z

Frame Properties

Allow Expansion

Allow Multi-Line Prompts

Allow Start-Attached Prompts

Allow Top-Attached Prompts

Distance Between Records

Frame Alignment

Frame Title

Frame Title Alignment

Frame Title Font Name

Frame Title Font Size

Frame Title Font Spacing

Frame Title Font Style

Frame Title Font Weight

Frame Title Font Foreground Color

Frame Title Offset

Frame Title Reading Order

Frame Title Spacing

Frame Title Visual Attribute Group

Horizontal Object Offset

Layout Style

Maximum Objects Per Line

Number of Records Displayed

Scroll Bar Alignment

Scroll Bar Width

Show Scroll Bar

Shrinkwrap

Single Object Alignment

Start Prompt Alignment

Start Prompt Offset

Top Prompt Alignment

Top Prompt Offset

Update Layout

Vertical Fill

Vertical Margin

Vertical Object Offset

Graphics Properties

Cap Style

Dash Style

Edge Background Color

Edge Foreground Color

Edge Pattern

Graphics Type

Join Style

Line Width

Rotation Angle

Hint properties

Hint (Item) property

Item Properties

Access Key

Application Instance

Automatic Skip (Item)

Bevel

Calculation Mode

Canvas

Case Insensitive Query

Case Restriction

Check Box Mapping of Other Values

Column Name

Comments

Compression_Quality

Conceal Data

Copy Value from Item

Current Record Visual Attribute

Data Source X Axis

Data Source Y Axis

Database_Value

Default Button

Display Hint Automatically

Display Quality

Distance Between Records

Editor

Editor X Position, Editor Y Position

Elements in List

Enabled (Item)

Format Mask

Formula

Highest Value Allowed

Hint (Item)

Icon Filename

Iconic

Image_Depth

Image_Format

Implementation Class

Initial Value

Initial Keyboard State

Insert Allowed (Item)

Item_Is_Valid

Item_Tab_Page (item)

Item Type

Justification

Keep Cursor Position

Keyboard Navigable

Keyboard State

Label (Item)

List Item Value

List of Values

List Style

List X Position

List Y Position

Lock Record

Lowest Value Allowed

Mapping of Other Values

Maximum Length (Item)

Mouse Navigate

Multi-Line

Name

NextItem

Next Navigation Item

Number of Items Displayed

Parameter Data Type

Popup Menu

PreviousItem

Previous Navigation Item

Primary Key (Item)

Property Class

Query Allowed (Item)

Query Length

Query Only

Radio Button Value

Rendered

Required (Item)

Size

Sizing Style

Show Horizontal Scroll Bar

Show Palette

Summarized Block

Summarized Item

Summary Function

Synchronize with Item

Tab Page

Tooltip

Tooltip Visual Attribute Group

Update Allowed (Item)

Update_Column

Update_Commit

Update Only if NULL

Update_Permission

Validate from List

Value When Checked

Value When Unchecked

Visible (Item)

Visual Attribute Group

Width/Height

Window_Handle

Wrap Style

Wrap Text

X Position, Y Position

Label Properties

Label (Item) property

Label (Menu Item) property

Label (Tab Page) property

LOV Properties

Automatic Display

Automatic Refresh

Automatic Select

Automatic Skip (LOV)

Column Mapping Properties

Column Name

Column Specifications

Column Title

Comments

Display Width (LOV)

Filter Before Display

Group_Name

Name

Property Class

Record Group

Return Item (LOV)

Size

Title

X Position, Y Position

Menu Properties

Bottom Title

Case Restriction

Checked

Command Text

Comments

Display without Privilege

Enabled (Menu Item)

Icon Filename

Icon in Menu

Item Roles

Keyboard Accelerator

Label (Menu Item)

Magic Item

Menu Directory

Menu Filename

Menu Item Radio Group

Menu Item Type

Menu Module

Module Roles

Property Class

Share Library with Form

Startup Code

Tear-off Menu

Title

Use Security

Visible In Horizontal/Vertical Menu Toolbar

Visible in Menu

Visual Attribute Group

Module Properties

Application Instance

Character Cell WD/HT

Comments

Console Window

Coordinate System

Current Record Visual Attribute

Cursor Mode

Defer_Required_Enforcement

File_Name

First_Block

First Navigation Block

Font Spacing

Form Horizontal Toolbar Canvas

Form Vertical Toolbar Canvas

Form_Name

Initial Menu

Last_Block

Menu Module

Menu Role

Menu Style

Module_NLS_Lang

Mouse Navigation Limit

Name

Property Class

Real Unit

Runtime Compatibility Mode

Savepoint Mode

Title

Use 3D Controls

Validation

Validation Unit

Window_Handle

Named Visual Attribute Properties

Background_Color

Foreground_Color

Name

Property Class

Size

Visible

Visual_Attribute

Visual Attribute Type

Width/Height

Parameter Properties

Comments

Maximum Length (Form Parameter)

Name

Parameter Data Type

Parameter Initial Value

Property Class

Prompt Properties

Prompt

Prompt Alignment

Prompt Alignment Offset

Prompt Attachment Edge

Prompt Attachment Offset

Prompt Background Color

Prompt Display Style

Prompt Fill Pattern

Prompt Font Name

Prompt Font Size

Prompt Font Spacing

Prompt Font Style

Prompt Font Weight

Prompt Foreground Color

Prompt Justification

Prompt Reading Order

Prompt Visual Attribute Group

Record Group Properties

Column Name

Column Specifications

Column Value

Comments

Data Type

Delete Record Behavior

Length

Name

Property Class

Record Group Fetch Size

Record Group Query

Record Group Type

Relation Properties

Comments

Coordination

Coordination_Status

Deferred

Detail Block

Join Condition

Name

Next_Detail_Relation

Next_Master_Relation

Prevent Masterless Operation

Property Class

Report Properties

Communication Mode (Report)

Data Source Data Block (Report)

Execution Mode (Report)

Other Reports Parameters

Query Name

Report Destination Format

Report Destination Name

Report Destination Type

Tab Page Properties

Canvas

Enabled (Tab Page)

Label (Tab Page)

Tab Page X Offset

Tab Page Y Offset

Visible (Tab Page)

Visual Attribute Group

Trigger Properties

Comments

Display in Keyboard Help

Execution Style

Fire in Enter-Query Mode

Keyboard Help Description

Name

Property Class

Trigger Text

Trigger Style

Window Properties

Comments

Hide on Exit

Horizontal Toolbar Canvas

Icon Filename

Inherit Menu

Maximize Allowed

Minimize Allowed

Minimized Title

Modal

Move Allowed

Name

Primary Canvas

Property Class

Resize Allowed

Show Horizontal Scroll Bar

Show Vertical Scroll Bar

Size

Title

Vertical Toolbar Canvas

Visible

Visual_Attribute

Visual Attribute Group

Window_State

Window Style

X Position, Y Position

Access Key Property

Description

Specifies the character that will be used as 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), Forms Developer executes the "Commit" command.

Applies to button, radio button, and check box

Set Forms Developer

Default No

Required/Optional Optional

Usage Notes

- 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.

Access Key Restrictions

- Buttons with the Iconic property set to Yes cannot have an access key.

Alert Style Property

Description

Specifies the alert style: note, stop, or caution. The alert style determines which icon is displayed in the alert.

Applies to alert

Set Forms Developer

Default

Stop

Alias Property

Description

Establishes an alias for the table that the data block is associated with.

Applies to table/columns associated with a data block

Set Forms Developer

Default

The Data Block wizard sets the Alias property to the first letter of the table name. (For example, a table named DEPT would have a default alias of D.)

Required/Optional required for Oracle tables that contain column objects or REFs

Usage Notes

For Oracle tables, SELECT statements that include column objects or REF columns must identify both the table name and its alias, and must qualify the column name by using that alias as a prefix.

For example:

```
CREATE TYPE ADDRESS_TYPE AS OBJECT  (STREET VARCHAR2(30),  CITY
VARCHAR2(30),  STATE VARCHAR2(2));  CREATE TABLE EMP  (EMPNO NUMBER,
ADDRESS ADDRESS_TYPE);
```

If the alias for this EMP table were E, then a SELECT statement would need to be qualified as follows:

```
SELECT EMPNO, E.ADDRESS.CITY FROM EMP E;
```

In this case, the alias is E. The column object ADDRESS.CITY is qualified with that alias, and the alias is also given after the table name. (The column EMPNO, which is a normal relational column, requires no such qualification.)

In most situations, Forms Developer will handle this alias naming for you. It will establish an alias name at design-time, and then automatically use the qualified name at runtime when it fetches the data from the Oracle Server. You only need to concern yourself with this alias naming if you are doing such things as coding a block WHERE clause.

Allow Empty Branches Property

Description

Specifies whether branch nodes may exist with no children. If set to FALSE, branch nodes with no children will be converted to leaf nodes. If set to TRUE, an empty branch will be displayed as a collapsed node.

Applies to hierarchical tree

Set Forms Developer, programmatically

Default

False

Required/Optional required

Allow Expansion Property

Description

Specifies whether Forms Developer can automatically expand a frame when the contents of the frame extend beyond the frame's borders.

Applies to frame

Set Forms Developer

Default

Yes

Required/Optional required

Allow Multi-Line Prompts Property

Description

Specifies whether Forms Developer can conserve space within a frame by splitting a prompt into multiple lines. Prompts can only span two lines.

Applies to frame

Set Forms Developer

Default

Yes

Required/Optional required

Allow Start-Attached Prompts Property

Description

Specifies whether space usage can be optimized when arranging items in tablular-style frames.

By default, this property is set to No, and prompts are attached to the item's top edge. Setting Allow Start-Attached Prompts to Yes allows you to attach prompts to the item's start edge if there is enough space.

Applies to frame

Set Forms Developer

Default

No

Required/Optional required

Allow Top-Attached Prompts Property

Description

Specifies whether space usage can be optimized when arranging items in form-style frames.

By default, this property is set to No, and prompts are attached to the item's start edge. Setting Allow Top-Attached Prompts to Yes allows you to attach prompts to the item's top edge if there is enough space.

Applies to frame

Set Forms Developer

Default

No

Required/Optional required

Application Instance Property

Description

Specifies a reference to an instance of a Forms Services application on the middle tier server machine, when Forms Services is running on a Microsoft Windows server. Other platforms always return NULL.

Note: This is the handle to the part of the application on the middle tier, NOT the application in the client browser. It cannot be used to interact with the clinet browser machine.

Applies to form, block, or item

Refer to Built-in

GET_APPLICATION

Default

Null

Usage Notes

Specify the APPLICATION_INSTANCE property in GET_APPLICATION_PROPERTY to obtain the pointer value of an instance handle. To use the instance handle when calling the Windows API, this pointer value must be converted with TO_PLS_INTEGER.

Application Instance Restrictions

Valid only on Microsoft Windows (Returns NULL on other platforms).

Arrow Style Property

Description

Specifies the arrow style of the line as None, Start, End, Both ends, Middle to Start, or Middle to End.

Applies to graphic line

Set Forms Developer

Default

None

Required/Optional required

Automatic Column Width Property

Description

Specifies whether LOV column width is set automatically.

When Automatic Column Width is set to Yes, the width of each column is set automatically to the greater of the two following settings:

- the width specified by the Display Width property

or

- the width necessary to display the column's title as specified in the Column Title property.

When Automatic Column Width is set to No, the width of each column is set to the value specified by the Display Width property.

Applies to LOV

Set Forms Developer

Default

No

Related topic

Column Mapping Properties property

Automatic Display Property

Description

Specifies whether Forms Developer 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 Forms Developer

Default

No

Automatic Position Property

Description

Specifies whether Forms Developer automatically positions the LOV near the field from which it was invoked.

Applies to LOV

Set Forms Developer

Default

No

Automatic Query Property

Description

See Coordination.

Automatic Refresh Property

Description

Determines whether Forms Developer re-executes the query to populate an LOV that is based on a query record group. By default, Forms Developer 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 Forms Developer validates a text item that has the Use LOV for Validation property set to Yes.

- When Automatic Refresh is set to Yes (the default), Forms Developer 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 Automatic Refresh is set to No, Forms Developer 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, Forms Developer does not re-execute the query, but instead displays the LOV using the records currently stored in the record group.

The Automatic Refresh property also determines how long records retrieved by the query remain stored in the underlying record group:

- When Automatic Refresh is set to Yes, 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 Automatic Refresh is set to No, 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 LOV

Set Forms Developer, programmatically

Refer to Built-in

GET_LOV_PROPERTY

SET_LOV_PROPERTY

Default

Yes

Usage Notes

- When multiple LOVs are based on the same record group, it is usually appropriate to use the same Automatic Refresh setting for each one. This is not, however, a strict requirement; the following scenario describes refresh behavior when one LOV has Automatic Refresh set to Yes and another has Automatic Refresh set to No.
    - LOV1 and LOV2 are based on the same record group; LOV1 has Automatic Refresh set to Yes, LOV2 has Automatic Refresh set to No. When LOV1 is invoked, Forms Developer executes the query to populate the underlying record group. When the operator dismisses LOV1, Forms Developer destroys the record cache, and clears the record group.
    - When LOV2 is subsequently invoked, Forms Developer again executes the query to populate the record group, even though LOV2 has Automatic Refresh set to No. Because LOV2's underlying record group was cleared when LOV1 was dismissed, Forms Developer 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 Automatic Refresh set to No, Forms Developer 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 Automatic Refresh is set to No, you can programmatically replace the rows that were returned by the initial query with POPULATE_GROUP. Forms Developer ignores this operation when deciding whether to re-execute the query. (Forms Developer looks only at the internal flag that indicates whether a query has occurred, not at the actual rows returned by that query.)

Automatic Refresh Restrictions

Valid only for an LOV based on a query record group, rather than a static or non-query record group.

Automatic Select Property

Description

Specifies what happens when an LOV has been invoked and the user reduces the list to a single choice when using auto-reduction or searching:

- When Automatic Confirm is set to Yes, the LOV is dismissed automatically and column values from the single row are assigned to their corresponding return items.
- When Automatic Confirm is set to No, the LOV remains displayed, giving the operator the option to explicitly select the remaining choice or dismiss the LOV.

Applies to LOV

Set Forms Developer

Default

No

Automatic Skip (Item) Property

Description

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 Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

No

Usage Notes

Combine the Automatic 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.

Automatic Skip (Item) Restrictions

- Valid only for single-line text items.
- The KEY-NEXT-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].

Automatic Skip (LOV) Property

Description

Moves the cursor to the next navigable item when the operator makes a selection from an LOV to a text item. When Automatic Skip is set to No, the focus remains in the text item after the operator makes a selection from the LOV.

Applies to LOV

Set Forms Developer, programmatically

Refer to Built-in

SET_ITEM_PROPERTY

Default

No

Automatic Skip (LOV) Restrictions

- The KEY-NEXT-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].

Background Color Property

Description

Specifies the color of the object’s background region.

Applies to item, tab page, canvas, window, radio button

Set Programmatically

Default

Unspecified

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

GET_TAB_PAGE_PROPERTY

SET_TAB_PAGE_PROPERTY

GET_CANVAS_PROPERTY

SET_CANVAS_PROPERTY

GET_WINDOW_PROPERTY

SET_WINDOW_PROPERTY

Block Description Property

Description

See Listed in Block Menu/Block Description.

Border Bevel Property

Description

Specifies the appearance of the object border, either RAISED, LOWERED, INSET, OUTSET, PLAIN, or NONE. Can also be set programmatically at the item instance level to indicate the property is unspecified at this level. That is, if you set this property programmatically at the item instance level using SET_ITEM_INSTANCE_PROPERTY, the border bevel is determined by the item-level value specified at design-time or by SET_ITEM_PROPERTY at runtime.

Applies to chart item, image item, custom item, stacked canvases, text items

Set Forms Developer, programmatically [BORDER_BEVEL]

Refer to Built-in

GET_ITEM_INSTANCE_PROPERTY

GET_ITEM_PROPERTY

SET_ITEM_INSTANCE_PROPERTY

SET_ITEM_PROPERTY

Default

LOWERED

Usage Notes

- To create a scrolling window, the Bevel property should be set to RAISED or LOWERED.

Bevel Restrictions

- If the item's Bevel property is set to None in Forms Developer, you cannot set BORDER_BEVEL programmatically.
- You cannot programmatically set BORDER_BEVEL to NONE.

Bottom Title (Editor) Property

Description

Specifies a title of up to 72 characters to appear at the bottom of the editor window.

Applies to editor

Set Forms Developer

Required/Optional optional

Bounding Box Scalable Property

Description

Specifies whether the text object's bounding box should be scaled when the text object is scaled.

Applies to graphic text

Set Forms Developer

Default

Yes

Required/Optional required

Builtin Date Format Property

Description

This property establishes the format mask used in converting a date value to or from a string that is not potentially visible to the end user. This format mask is most commonly used when executing a built-in subprogram.

Applies to application (global value)

Set programmatically

Refer to Built-in

GET_APPLICATION_PROPERTY Built-in

Required/Optional optional. However, it is STRONGLY RECOMMENDED that, for a new application, you set this property to a format mask containing full century and time information. It is also recommended that this format mask be the same as the one specified in the PLSQL_DATE_FORMAT property .

Default

As noted above, it is strongly recommended that you explicitly set this value for a new application. However, if you do not, the default value used will depend on the context.

Forms first determines whether the item is a DATE2, DATE4, or DATETIME object, and then tries a series of format masks accordingly. (These default masks are used for compatibility with prior releases.)

Object types are determined as shown in the following table:

Date object	Type
Item of datatype DATETIME	DATETIME
Item of datatype DATE:	
…having a format mask that contains yyyy, YYYY, rrrr, or RRRR	DATE4
…having a format mask that does not contain yyyy, YYYY, rrrr, or RRRR	DATE2
…not having a format mask, and its length (Maximum Length) is 10 or more	DATE4
…not having a format mask, and its length (Maximum Length) is 9 or less	DATE2

Parameter (as in :PARAMETER.myparam) of datatype DATE. (Note that there are no DATETIME parameters, and that a parameter's Maximum Length property applies only to CHAR parameters.)	DATE2
LOV column of datatype DATE. (Note that there are no DATETIME LOV columns.)	DATE2
Internal value of system variables CURRENT_DATETIME and EFFECTIVE_DATE	DATETIME

After determining the object type of the item to be converted, Forms uses one of the masks listed below. There are two sets of masks -- one set for YY operations, and another set for RR operations.

For a date-to-string operation, only the first (primary) format mask is used. For a string-to-date operation, Forms Developer first tries the first/primary format mask. If that conversion is unsuccessful, it tries the other (secondary) masks, in the order shown

For YY:

Object Type	**Format Masks Used**
	DD-MON-YY
DATE2	
	DD-MM-SYYYY HH24:MI:SS
	DD-MON-YYYY
DATE4	
	DD-MM-SYYYY HH24:MI:SS
	DD-MON-YYYY HH24:MI:SS
DATETIME	DD-MON-YYYY HH24:MI
	DD-MM-SYYYY HH24:MI:SS

For RR:

Object Type	**Format Masks Used**
	DD-MON-RR
DATE2	
	DD-MM-SYYYY HH24:MI:SS
	DD-MON-RRRR
DATE4	
	DD-MM-SYYYY HH24:MI:SS

	DD-MON-RRRR HH24:MI:SS
DATETIME	DD-MON-RRRR HH24:MI
	DD-MM-SYYYY HH24:MI:SS

Button 1 Label, Button 2 Label, Button 3 Label Property

Description

Specifies the text labels for the three available alert buttons.

Applies to alert

Set Forms Developer, programmatically

Refer to Built-in

SET_ALERT_BUTTON_PROPERTY

Required/Optional At least one button must have a label.

Default

Button 1 Label: OK, Button 2 Label: Cancel, Button 3 Label: NULL

Related topics

SET_ALERT_BUTTON_PROPERTY built-in

Calculation Mode Property

Description

Specifies the method of computing the value of a calculated item. Valid values are:

None

The default. Indicates the item is not a calculated item.

Formula

Indicates the item's value will be calculated as the result of a user-written formula. You must enter a single PL/SQL expression for an item's formula. The expression can compute a value, and also can call a Forms Developer or user-written subprogram.

Summary

Indicates the item's value will be calculated as the result of a summary operation on a single form item. You must specify the summary type, and the item to be summarized.

Applies to item

Set Forms Developer

Required/Optional optional

Default

None

Calling Form Property

Description

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 Property

Description

Specifies the canvas on which you want the item to be displayed.

Applies to item

Set Forms Developer

Default

The item's current canvas assignment.

Required/Optional optional

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 Form Editor or at runtime.
- If you change the name of a canvas in Forms Developer, Forms Developer automatically updates the Canvas property for all items assigned to that canvas.

Canvas Restrictions

The canvas specified must already exist in the form.

Canvas Type Property

Description

Specifies the type of canvas, either Content, Stacked, Vertical Toolbar Canvas, or Horizontal Toolbar Canvas. The type determines how the canvas is displayed in the window to which it is assigned, and determines which properties make sense for the canvas.

Content	The default. Specifies that the canvas should occupy the entire content area of the window to which it is assigned. Most canvases are content canvases.
Stacked	Specifies that the canvas should be displayed in its window at the same time as the window's content canvas. Stacked views are usually displayed programmatically and overlay some portion of the content view displayed in the same window.
Vertical Toolbar Canvas	Specifies that the canvas 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.
Horizontal Toolbar Canvas	Specifies that the canvas should be displayed as a horizontal toolbar at the left side of the window to which it is assigned.
Tab Canvas	Specifies that this canvas will be a container for one or more tab pages.

Applies to canvas

Set Forms Developer

Default

Content

Usage Notes

In the Property Palette, the properties listed under the Stacked View heading are valid only for a canvas with the Canvas Type property set to Stacked.

Cap Style Property

Description

Specifies the cap style of the graphic object's edge as either Butt, Round, or Projecting.

Applies to graphic physical

Set Forms Developer

Default

Butt

Required/Optional required

Case Insensitive Query Property

Description

Determines whether the operator can perform case-insensitive queries on the text item.

Applies to text item

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

No

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 Yes.
- In Enter Query mode, enter the name 'BLAKE' into :ENAME.
- Execute the query.

Forms Developer 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 Insensitive Query Restrictions

If you set this property to Yes, queries may take longer to execute.

Case Restriction Property

Description

Specifies the case for text entered in the text item. The allowable values for this property are as follows:

Value	Result
MIXED	Text appears as typed.
UPPER	Lower case text converted to upper case as it is typed.
LOWER	Upper case text converted to lower case as it is typed.

Applies to text item

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Case Restriction Restrictions

- Values assigned to the text item through triggers are not effected.
- 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 what the end user sees on the screen may differ from the internal value of that text item. For example, if Case Restriction is set to UPPER and the data retrieved from the data source is in mixed case, the form will display it in UPPER, but the actual value of the data will remain mixed case. However, If the data is subsequently modified in that field and the change is committed, the value of the data will change to upper case.

Character Cell WD/HT Property

Description

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 Forms Developer

Required/Optional optional

Usage Notes

The character cell size is specified in the Coordinate System dialog in pixels but displayed in the Layout Editor in points.

Related topics

Coordinate System property

Real Unit property

Chart Subtype Property

Description

Specifies a variation of the chart type. Each variation is based on the specified chart type, with various properties set to achieve a different look.

Applies to chart item

Set Forms Developer

Default

Column

Check Box Mapping of Other Values Property

Description

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 Forms Developer

Default

NOT ALLOWED

Usage Notes

The following settings are valid for this property:

Setting	Description
Not Allowed	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.
Checked	Any value other than the user-defined unchecked value is interpreted as the checked state.
Unchecked	Any value other than the user-defined checked value is interpreted as the unchecked state.

Related topic

Value when Unchecked property

Checked Property

Description

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

Checked Restrictions

Valid only for a menu item with the Menu Item Type property set to Check or Radio.

Clip Height Property

Description

Specifies the height of a clipped (cropped) image in layout units. If you specify a value less than the original image height, the image clips from the bottom.

Applies to graphic image

Set Forms Developer

Default

original image height

Required/Optional required

Clip Width Property

Description

Specifies the width of a clipped (cropped) image in layout units. If you specify a value less than the original image's width, the image clips from the right.

Applies to graphic image

Set Forms Developer

Default

original image width

Required/Optional required

Clip X Position Property

Description

Specifies how much (in layout units) to clip off the left side of the image.

Applies to graphic image

Set Forms Developer

Default

0

Required/Optional required

Clip Y Position Property

Description

Specifies how much (in layout units) to clip off the top of the image.

Applies to graphic image

Set Forms Developer

Default

0

Required/Optional required

Closed Property

Description

Specifies whether an arc is closed.

Applies to graphic arc

Set Forms Developer

Default

Yes

Required/Optional required

Column Mapping Properties Property

Description

The Column Mapping Properties group includes Column Name, Column Title, Display Width, and Return Item.

Applies to LOV

Set Forms Developer

Column Name

Specifies the names of the columns in an LOV.

Required/Optional At least one column must be defined.

Default

The 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 for the column currently selected in the Column Name list.

Required/Optional optional

Usage Notes

- Set the Display Width property to the width in appropriate units (points, pixels, centimeters, inches, or characters as specified by the form's Coordinate System property) that you want Forms Developer to reserve for the column in the LOV window. Column value truncation may occur if the Display Width is smaller than the width of the column value. To avoid this situation, increase the Display Width for the column.

- 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, set the Display Width wider than the column's default width. Note, however, that as an exception to this rule, you cannot increase the width between a NUMBER column and a non-NUMBER column by increasing the display width for the NUMBER column because LOVs display numbers right-justified. For example, assume that your LOV contains 3 columns: column 1 and 3 are type CHAR and column 2 is type NUMBER. To increase the width between each column, increase the Display Width for columns 1 and 3.

Return Item

Specifies the name of the form item or variable to which Forms Developer 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 Property

Description

Establishes that an item corresponds to a column in the table associated with the data block.

Applies to any item except button, or chart.

Set Forms Developer

Refer to Built-in

GET_ITEM_PROPERTY

Default

Yes

Required/Optional optional

Usage notes

When a selected item is from a column object or REF column in a table, Forms Developer creates a compound name for it using dot notation: O*bjectColumnName.AttributeName*.

For example, assume dept_type were an object type having attributes of dnum, dname, and dloc, and we had a column object called dept based on dept_type. If we then selected dname to become an item in the data block, its column name property would become dept.dname.

Related topics

Primary Key (Item) property

Column Mapping Properties property

Column Specifications property

Column Specifications Property

Description

The Column Specifications group of properties include Column Name, Column Value, Data Type, Length.

Applies to record group

Set Forms Developer

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

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, 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 Specifications Restrictions

- You cannot reference an uninitialized variable or an item for this property, as that action constitutes a forward reference that Forms Developer 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.

Column Title (LOV) Property

Description

See Column Mapping Properties.

Column Value (Record Group) Property

Description

See Column Specifications.

Command Text Property

Description

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 Forms Developer

Required/Optional Required for all command types except NULL.

Valid values Null, PL/SQL, Menu

Usage Notes

Null, PL/SQL, and Menu are the only valid values for Oracle9i Forms Developer. Previous versions allowed the use of Plus, Form, and Macro, which will be automatically converted, where <old_code> is the value of the Command Text property before the upgrade. The replacement PL/SQL code will be commented out to allow developers to replace their previous code with new PL/SQL code.

For Plus, the code is upgraded as follows:

```
/* HOST('plus80 <old_code> '); */ null;
```

For Form, the code is upgraded as follows:

```
/* CALL_FORM(<old_code>); */ null;
```

For Macro, the code is upgraded as follows:

```
/* MACRO: <old_code> ; */ null;
```

Command Type Property

Description

Specifies the nature of the menu item command. This property determines how Form Builder interprets the text in the Command Text property.

Applies to menu item

Set Form Builder

Default NULL

Required/Optional required

Command Type Descriptions

Null

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.

Menu

Invokes a submenu. Valid command text is the name of the submenu to be invoked.

PL/SQL

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 NAME_IN and COPY Built-ins to indirectly reference such values.

Plus*

Avoid. To invoke SQL*Plus, use the PL/SQL command type, and execute the HOST built-in to launch SQL*Plus. (On Windows platforms, use plus80.exe as the executable name.)

Current Forms*

Avoid. To invoke Form Builder, use the PL/SQL command type, and execute the HOST or RUN_PRODUCT Built-ins to execute a valid Form Builder login.

Macro*

Avoid. Executes a SQL*Menu macro.

*This command type is included for compatibility with previous versions. Do not use this command type in new applications.

Comments Property

Description

The Comments property specifies general information about any Forms Developer 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 Forms Developer

Required/Optional optional

Communication Mode (Chart) Property

Description

When calling Graphics from Forms Developer to create a chart, specifies the communication mode to be used as either Synchronous or Asynchronous. Synchronous specifies that control returns to the calling application only after the called product has finished. The end user 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.

When data is returned from the called product, such as when updating a chart item, communication mode must be synchronous.

Applies to chart items

Set Forms Developer

Default

Synchronous

Required/Optional required

Communication Mode (Report) Property

Description

For report/form integration, specifies communication mode between the form and the report as either Synchronous or Asynchronous. Synchronous specifies that control returns to the calling application only after the called product has finished. The end user 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.

When data is returned from the called product, communication mode must be synchronous.

Applies to report integration

Set Forms Developer

Default

Synchronous

Required/Optional required

Compression Quality Property

Description

Specifies whether an image object being read into a form from a file, or written to a file (with the WRITE_IMAGE_FILE built-in) should be compressed, and if so, to what degree. Valid values are:

- None
- Minimum
- Low
- Medium
- High
- Maximum

Applies to image item

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

None

Conceal Data Property

Description

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:

Yes Disables the echoing back of data entered by the operator.

No Enables echoing of data entered by the operator.

Applies to text item

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

No

Conceal Data Restrictions

Valid only for single-line text items.

Console Window Property

Description

Specifies the name of the window that should display the Forms Developer console. The console includes the status line and message line, and is displayed at the bottom of the window.

On the Web, 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 Forms Developer

Default

WINDOW1

Required/Optional optional

Coordinate System Property

Description

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 Form Editor rulers, but does not change the grid spacing and snap-points settings.

Applies to form

Set Forms Developer

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, Forms Developer 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 you want to optimize for GUIs, the Real setting provides maximum flexibility for proportional fonts.

The Character setting provides less flexibility for the proportional fonts used on GUIs, but lets you line up character cell boundaries exactly.

For this type of application...	**Set Coordinate System to...**
GUI only	Real: inches, centimeters, or points
Optimize for GUI	Real

Related topic

Real Unit property

Coordination Property

Description

Specifies how and when the population phase of block coordination should occur. Specify the coordination desired by setting the Deferred and Automatic Query properties. When you set these properties at design time, Forms Developer creates or modifies the appropriate master-detail triggers to enforce the coordination setting you choose.

Applies to:

relation

Set:

Forms Developer, programmatically

Refer to Built-in

GET_RELATION_PROPERTY

SET_RELATION_PROPERTY

Default

Immediate coordination (Deferred No, Automatic Query No)

Usage Notes

Whenever the current record in the master block changes at runtime (a coordination-causing event), Forms Developer 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=No,Automatic Query ignored	The default setting. When a coordination-causing event occurs in the master block, the detail records are fetched immediately.
Deferred=Yes, Automatic Query=Yes	When a coordination-causing event occurs, Forms Developer defers fetching the associated detail records until the operator navigates to the detail block.

Deferred=Yes, Automatic Query=No	When a coordination-causing event occurs, Forms Developer defers fetching the associated detail records until the operator navigates to the detail block and explicitly executes a query.
Deferred=No,Automatic Query=Yes	Not a valid setting.

Coordination 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, Forms Developer generates the appropriate triggers to enforce coordination, and setting the coordination properties at runtime has no effect on the default trigger text.

Copy Value from Item Property

Description

Specifies the source of the value that Forms Developer uses to populate the item. When you define a master-detail relation, Forms Developer 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 Forms Developer

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 No for the foreign key items.
- To get the Copy Value from Item property programmatically with GET_ITEM_PROPERTY, use the constant ENFORCE_KEY.

Current Form Name Property

Description

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.

Related topics

Form_Name property

GET_FORM_PROPERTY built-in

Current Form Property

Description

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.

Related topics

GET_FORM_PROPERTY built-in

Current Record Property

Description

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_BLOCK_PROPERTY

Current Record Visual Attribute Group Property

Description

Specifies the named visual attribute used when an item is part of the current record.

Applies to form, block, item

Set Forms Developer, 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.

Note that if you define a form-level Current Record Visual Attribute, any toolbars in the form will be displayed using that Current Record Visual Attribute. You can avoid this by defining block-level Current Record Visual Attributes for the blocks that need them instead of defining them at the form level. If you wish to retain the form-level Current Record Visual Attribute, you can set the block-level Current Record Visual Attribute for the toolbar to something acceptable.

Current Record Visual 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 Visual 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 Visual 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 Row Background Color Property

Description

Specifies the color of the object’s background region.

Applies to item, block, form

Set Programmatically

Default

NULL

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_BLOCK_PROPERTY

SET_BLOCK_PROPERTY

GET_FORM_PROPERTY

SET_FORM_PROPERTY

Current Row Fill Pattern Property

Description

Specifies 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.

Applies to item, block, form

Set Programmatically

Default

Unspecified

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_BLOCK_PROPERTY

SET_BLOCK_PROPERTY

GET_FORM_PROPERTY

SET_FORM_PROPERTY

Current Row Font Name Property

Description

Specifies the font family, or typeface, to be used for text in the object. The list of fonts available is system-dependent.

Applies to item, block, form

Set Programmatically

Default

NULL

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_BLOCK_PROPERTY

SET_BLOCK_PROPERTY

GET_FORM_PROPERTY

SET_FORM_PROPERTY

Current Row Font Size Property

Description

Specifes the size of the font in points.

Applies to item, block, form

Set Programmatically

Default

NULL

Current Row Font Spacing Property

Description

Specifies the width of the font (i.e., the amount of space between characters, or kerning).

Applies to item, block, form

Set Programmatically

Default

NULL

Current Row Font Style Property

Description

Specifies the style of the font.

Applies to item, block, form

Set Programmatically

Default

NULL

Current Row Font Weight Property

Description

Specifies the weight of the font.

Applies to item, block, form

Set Programmatically

Default

NULL

Current Row Foreground Color Property

Description

Specifies the color of the object's foreground region. For items, defines the color of the text displayed in the item.

Applies to item, block, form

Set Programmatically

Default

NULL

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_BLOCK_PROPERTY

SET_BLOCK_PROPERTY

GET_FORM_PROPERTY

SET_FORM_PROPERTY

Cursor Mode Property

Note:

In Release 5.0 and later, cursor mode is handled automatically by Forms Developer. This property is now obsolete, and should not be used. In particular, cursor mode should never be set to Close. The following information is provided only for historical and maintenance purposes.

Description

Defines the cursor state across transactions. The cursor refers to the memory work area in which SQL statements are executed. This property is useful for applications running against a non-ORACLE data source.

The following settings are valid for the Cursor_Mode property:

Setting	**Description**
Open (the default)	Specifies that cursors should remain open across transactions.
Close	Specifies that cursors should be closed when a commit is issued.

Applies to form

Set 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, Forms Developer 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

- Forms Developer does not explicitly close cursors following commit processing if you set the property to CLOSE_AT_COMMIT. This setting is primarily a hint to Forms Developer that the cursor state can be undefined after a commit.

  Forms Developer maintains a transaction ID during all types of transaction processing. For instance, Forms Developer increments the transaction ID each time it opens a cursor, performs a commit, or performs a rollback.

  When Forms Developer attempts to re-execute a cursor, it checks the transaction ID. If it is not the current transaction ID, then Forms Developer opens, parses, and executes a new cursor. Only the last transaction ID is maintained.

- If you query, change data, then commit, Forms Developer increments the transaction ID. Subsequent fetches do not re-open and execute the cursor, for the following reasons:
- Forms Developer does not attempt to handle read consistency issues, nor does it handle re-positioning in the cursor.
- Forms Developer 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, Forms Developer 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.

Cursor Style Property

Description

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:

ARROW

Displays a GUI-specific arrow symbol.

BUSY

Displays a GUI-specific busy symbol.

CROSSHAIR

Displays a GUI-specific crosshair symbol.

DEFAULT

Displays a GUI-specific arrow symbol.

HELP

Displays a GUI-specific help symbol.

INSERTION

Displays a GUI-specific insertion symbol.

HAND

Pointing finger symbol.

MOVE

Crossed Arrow move window symbol.

Resize Arrows

The first two letters indicate direction, e.g. MWRESIZE - North-West Resize

NRESIZE
SRESIZE
ERESIZE
WRESIZE
NERESIZE
NWRESIZE
SERESIZE
SWRESIZE

Applies to application

Set Programmatically

Refer to Built-in

GET_APPLICATION_PROPERTY

SET_APPLICATION_PROPERTY

Default

Arrow symbol

Usage Notes

When Forms Developer 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 Forms Developer executes the query, the BUSY cursor reverts to the HELP cursor.

Note, however, if you change the cursor style *while* Forms Developer is displaying the BUSY cursor, the cursor style changes immediately rather than waiting for Forms Developer to complete the operation before changing cursor styles.

Custom Spacing Property

Description

Specifies the custom spacing for the text object in layout units.

Applies to graphic text

Set Forms Developer

Default

0

Required/Optional required

Dash Style Property

Description

Specifies the dash style of the graphic object's edge as Solid, Dotted, Dashed, Dash Dot, Double Dot, Long dash, or dash Double Dot.

Applies to graphic physical

Set Forms Developer

Default

Solid

Required/Optional required

Data Length Semantics Property

Description

Determines the semantics of the `MAX_LENGTH` and `QUERY_LENGTH` properties, when it is used to enforce the maximum size of the internal value of a character item.

Values

`CHAR`, `BYTE`, `null`

Applies to: Items of datatype `CHAR`, `ALPHA`, or `LONG`.

Required: No

Set At Runtime: No

Default: Null

Refer to Built-in

GET_ITEM_PROPERTY

Usage Notes

- If a null value is specified, then byte semantics will be used unless the environment variable `NLS_LENGTH_SEMANTICS` is set to `CHAR` when the form is compiled.
- When the `Synchronize with Item` property is set, `DATA_LENGTH_SEMANTICS` will be ignored in a subordinate mirror item. The `DATA_LENGTH_SEMANTICS` property is always taken from the master mirror item. A compiler (generator) warning will be issued if a non-null value is specified in a subordinate mirror item.
- A compiler (generator) warning will also be issued if a non-null value is specified in an item whose datatype is neither `CHAR`, `ALPHA`, nor `LONG`.

Related topics

Maximum Length Property

Query Length Property

Synchronize with Item Property

Data Query Property

Description

Specifies the query-based data source.

Applies to hierarchical tree

Set Forms Developer, programmatically

Refer to Built-in

ADD_TREE_DATA

Default

NULL

Required/Optional optional

Data Source Data Block (Chart) Property

Description

When running Graphics from Forms Developer to create a chart, specifies the data block to be used as the source of a chart item.

Applies to chart item

Set Forms Developer

Default

Null

Required/Optional optional

Related topics

Data Source X Axis property

Data Source Y Axis property

Data Source Data Block (Report) Property

Description

For report/form integration, specifies the data block to be used as the source of the report as either Null or a block name.

Applies to report integration

Set Forms Developer

Default

Null

Required/Optional optional

Related topics

Data Source X Axis property

Data Source Y Axis property

Data Source X Axis Property

Description

Specifies the data block column to be used as the basis of the X axis of a chart item.

Applies to chart item

Set Forms Developer

Related topics

Data Source Data Block (Chart) property

Data Source Y Axis property

Data Source Y Axis Property

Description

Specifies the data block column to be used as the basis of the Y axis of a chart item.

Applies to chart item

Set Forms Developer

Related topics

Data Source Data Block (Chart) property

Data Source X Axis property

Data Type (Record Group) Property

Description

See Column Specifications.

Data Type Property

Description

Specifies what kinds of values Forms Developer allows as input and how Forms Developer displays those values.

Applies to check box, display item, list item, radio group, text item, custom item, and form parameter (form parameter supports CHAR, DATE, DATETIME, and NUMBER only)

Note: All data types do not apply to each item type.

Set Forms Developer

Usage Notes

- In Forms Developer 6.0 and later, it is recommended that you use only the standard data types CHAR, DATE, DATETIME, and NUMBER for data. 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.
- Forms Developer 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	Null
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	Null
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 is not intended to store a 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 characters. Stored in ORACLE as variable-length character strings. Forms allows a LONG field to be up to 65,534 bytes. However, PL/SQL has a maximum of 32,760 bytes. If a LONG variable is to be used as a bind variable in a PL/SQL statement, it cannot exceed that 32,760 byte limit.

Default	Null
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 9.99x10124, 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 or FLOAT columns in the database, and Forms Developer 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.

Apply a format mask such as 999 to produce a right-justified number.

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

Related topic

Format Mask property

Database Value Property

Description

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 end user.

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_ITEM_PROPERTY

Datetime Local TZ Property

Description

Specifies the local time zone region for DATETIME items.

Set At Runtime Yes

Default

Value of the FORMS90_DATETIME_LOCAL_TZ environment variable (if set); otherwise the time zone region extracted from Java client (if FORMS90_TZFILE is set); otherwise GMT.

Valid Values

Any string listed as a valid time zone region in the time zone file.

Usage Note

Specifying an invalid timezone_region_string in SET_APPLICATION_PROPERTY(DATETIME_LOCAL_TZ, timezone_region_string); results in the generic error:

```
FRM-41340 Invalid property or property value for
Set_Application_Property
```

Refer to Built-In

GET_APPLICATION_PROPERTY

SET_APPLICATION_PROPERTY

Related topics

Datetime_Server_TZ Property

ADJUST_TZ Built-In

Datetime Server TZ Property

Description

Specifies the server time zone region for `DATETIME` items.

Set At Runtime No

Default

Value of the `FORMS90_DATETIME_SERVER_TZ` environment variable (if set); otherwise `GMT`.

Valid Values

Any string listed as a valid time zone region in the time zone file.

Usage Note

Issuing SET_APPLICATION_PROPERTY(DATETIME_SERVER_TZ, timezone_region_string); will result in the generic error:

FRM-41340 Invalid property or property value for Set_Application_Property

whether or not timezone_region_string is valid.

Refer to Built-In

GET_APPLICATION_PROPERTY

Related topics

Datetime_Local_TZ Property

ADJUST_TZ Built-In

Default Alert Button Property

Description

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 Forms Developer

Default

Button 1

Required/Optional optional

Default Button Property

Description

Specifies that the button should be identified as the default button. At runtime, the end user can invoke the default button by pressing [Select] if focus is within the window 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 Forms Developer

Default

No

Required/Optional optional

Default Font Scaling Property

Description

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 Forms Developer

Default

Yes

Default Font Scaling Restrictions

Valid only when the Coordinate System property is set to Character Cell.

Defer Required Enforcement Property

Description

For an item that has the Required property set to true, it specifies whether Forms Developer should defer enforcement of the Required item attribute until the record is validated.

There are three settings for this property: Yes, 4.5, and No.

Applies to form

Set Forms Developer, programmatically

Refer to Built-in

GET_FORM_PROPERTY

SET_FORM_PROPERTY

Default

No

Usage Notes

- This property applies only when item-level validation is in effect. By default, when an item has Required set to true, Forms Developer will not allow navigation out of the item until a valid value is entered. This behavior will be in effect if you set Defer Required Enforcement to No. (An exception is made when the item instance does not allow end-user update; in this unusual case, a Defer Required Enforcement setting of No is ignored and item-level validation does not take place.)
- If you set Defer Required Enforcement to Yes (PROPERTY_TRUE for runtime) or to 4.5 (PROPERTY_4_5 for runtime), you allow the end user to move freely among the items in the record, even if they are null, postponing enforcement of the Required attribute until validation occurs at the record level.
- When Defer Required Enforcement is set to Yes, null-valued Required items are not validated when navigated out of. That is, the WHEN-VALIDATE-ITEM trigger (if any) does not fire, and the item's Item Is Valid property is unchanged. If the item value is still

null when record-level validation occurs later, Forms Developer will issue an error.

- When Defer Required Enforcement is set to 4.5, null-valued Required items are not validated when navigated out of, and the item's Item Is Valid property is unchanged. However, the WHEN-VALIDATE-ITEM trigger (if any) does fire. If it fails (raises Form_Trigger_Failure), the item is considered to have failed validation and Forms Developer will issue an error. If the trigger ends normally, processing continues normally. If the item value is still null when record-level validation occurs later, Forms Developer will issue an error at that time.
- Setting a value of 4.5 for Defer Required Enforcement allows you to code logic in a WHEN-VALIDATE-ITEM trigger that will be executed immediately whenever the end-user changes the item’s value (even to null) and then navigates out. Such logic might, for example, update the values of other items. (The name "4.5" for this setting reflects the fact that in Release 4.5, and subsequent releases running in 4.5 mode, the WHEN-VALIDATE-ITEM trigger always fired during item-level validation.)
- Migration note: If your Forms application used "4.5" as the Runtime Compatibility Mode property setting, the Oracle9i Forms Migration Assistant will automatically set the Defer Required Enforcement property to "4.5" because the Runtime Compatibility Mode property is obsolete in Oracle9i Forms.

Related topic

Required (Item) property

Deferred Property

Description

See Coordination.

Delete Allowed Property

Description

Specifies whether records can be deleted from the block.

Applies to block

Set Forms Developer, programmatically

Default

Yes

Refer to Built-in

GET_BLOCK_PROPERTY

SET_BLOCK_PROPERTY

Delete Procedure Arguments Property

Description

Specifies the names, datatypes, and values of the arguments that are to be passed to the procedure for deleting data. The Delete Procedure Arguments property is valid only when the DML Data Target Type property is set to Procedure.

Applies to block

Set Forms Developer

Default

NULL

Required/Optional optional

Related topics

Delete Procedure Name property

Delete Procedure Result Set Columns property

DML Data Target Name property

DML Data Target Type property

Delete Procedure Name Property

Description

Specifies the name of the procedure to be used for deleting data. The Delete Procedure Name property is valid only when the DML Data Target Type property is set to Procedure.

Applies to block

Set Forms Developer

Default

NULL

Required/Optional optional

Related topics

Delete Procedure Arguments property

Delete Procedure Result Set Columns property

DML Data Target Name property

DML Data Target Type property

Delete Procedure Result Set Columns Property

Description

Specifies the names and the datatypes of the result set columns associated with the procedure for deleting data. The Delete Procedure Result Set Columns property is valid only when the DML Data Target Type property is set to Procedure.

Applies to block

Set Forms Developer

Default

NULL

Required/Optional optional

Related topics

Delete Procedure Arguments property

Delete Procedure Name property

DML Data Target Name property

DML Data Target Type property

Delete Record Behavior Property

Description

(Note : this property was formerly called the Master Deletes property.)

Specifies how the deletion of a record in the master block should affect records in the detail block:

Setting	Description
Non-Isolated	The default setting. Prevents the deletion of a master record when associated detail records exist in the database.
Isolated	Allows the master record to be deleted and does not affect associated detail records in the database.
Cascading	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).

Applies to relation

Set Forms Developer, programmatically

Refer to Built-in

GET_RELATION_PROPERTY

SET_RELATION_PROPERTY

Default

Non-Isolated

Delete Record Behavior Restrictions

- Setting this property at runtime has no effect for a default master-detail relation. At design time, Forms Developer creates the appropriate triggers to enforce the relation, and changing the Delete Record Behavior 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.

Detail Block Property

Description

Specifies the name of the detail block in a master-detail block relation.

Applies to relation

Set Forms Developer

Refer to Built-in

GET_RELATION_PROPERTY (Detail_Name)

Default:

NULL

Required/Optional required

Detail Block Restrictions

The block specified must exist in the active form.

Detail Reference Item Property

Description

Identifies the REF item in the relation's detail data block that forms the link to the master data block. This property applies only when the Relation Type property is set to REF.

Applies to Relation

Set Forms Developer

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

Null

Usage Notes

This property applies only when the Relation type property is set to REF

Direction Property

Description

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 Direction 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.

	Default Setting Derives Value From This Object
Form	NLS environment variable
All objects, such as Alert, Block, LOV, Window, and Canvas	Form
All items, such as Text Item, Display Item, Check Box, Button, Radio Group, and List Item	Canvas

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 Forms Developer, you can specifically set Justification, Reading Order, and Initial Keyboard Direction properties for these items. However, programmatically, you can get and set the Direction property only for all items, including text items and display items.)

	Layout Direction	Text Reading Order	Text Alignment	Scrollbar Position	Initial Keyboard Direction
Form	X				
Alert	X	X			X
Block(for future use)					
LOV(for future use)					
Window	X(of menu)	X			X
Canvas	X(also point of origin)	X(boilerplate text)		X(and rulers)	
Check Box	X	X			X
Button	X	X			X
Radio Group	X	X			X
List Item	X	X	X	X	

Note: The headings listed above represent functions, not properties: for example, the Direction property for alerts does not set the Initial Keyboard Direction property, it controls the initial keyboard state function.

The allowable values for this property are:

Value	Description
Default	Direction based on the property shown in the table.

Right-To-Left	Direction is right-to-left.
Left-To-Right	Direction is left-to-right.

Applies to all objects listed in the table

Set Forms Developer, programmatically

Refer to Built-in

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 Language 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 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)

Specifies the layout direction of the canvas, 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 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. This will provide:
- automatic layout of blocks in the canvas Direction property
- boilerplate text reading order will default to the canvas Direction property
- If a block spans multiple canvases, keep the canvas Direction property the same for all canvases, unless you intend to have part of the block displayed with a different Direction.

- In Forms Developer, if you change the canvas 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)

Related topics

Bidirectional support in Forms Developer

Reading Order property

Defining a prompt's alignment offset

Display Hint Automatically Property

Description

Determines when the help text specified by the item property, Hint, is displayed:

- Set Display Hint Automatically to Yes to have Forms Developer display the hint text whenever the input focus enters the item.
- Set Display Hint Automatically to No to have Forms Developer display the hint text only when the input focus is in the item and the end user presses [Help] or selects the Help command on the default menu.

Applies to all items except chart item, display item, and custom item

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM

SET_ITEM

Default

No

Usage Notes

If a trigger causes Forms Developer 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.

Display Hint Automatically Restrictions

Not applicable when the `Hint` property is NULL.

Display in 'Keyboard Help'/'Keyboard Text' Property

Description

Specifies whether a key trigger description is displayed in the runtime Keys help screen. An entry in the 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 Forms Developer

Default

No

Usage Notes

- If you do not want the name or the description to appear in the Show Keys window, set the Display Keyboard Help property to No. 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 Keys window, set the Display Keyboard Help property to Yes and leave the Keyboard Help Text blank.
- If you want to replace the default key description, set the Display Keyboard Help property to Yes, then enter the desired description in the Keyboard Help Text property.

Display in Keyboard Help Restrictions

Valid only for key triggers.

Related topic

Record Group Type property

Display_Height Property

Description

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_APPLICATION_PROPERTY

Displayed Property

Description

Enables/unhides or disables/hides an item. When an item is disabled and hidden it is not navigable, queryable, or updateable.

Values: TRUE/FALSE

Applies to: item

Set: programmatically

Usage notes

You should make sure an item is not selected before setting the Displayed property to FALSE. Setting a selected item's Displayed property to false will generate an error: FRM-41016.

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Displayed property restrictions

You cannot set the Displayed property of an item that is selected or has focus.

Display Quality Property

Description

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 Forms Developer

Default

High

Display Quality Restrictions

none

Display_Width Property

Description

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_APPLICATION_PROPERTY

Display Width (LOV) Property

Description

See Column Mapping Properties.

Display without Privilege Property

Description

Determines whether the current menu item is displayed when the current form end user is not a member of a security role that has access privileges to the item:

- When Display without Privilege is No, Forms Developer does not display the item if the end user does not have access to it.
- When Display without Privilege is Yes, Forms Developer displays the item as a disabled (grayed) menu item. The end user 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 Forms Developer online help system.

Applies to menu item

Set Forms Developer

Default

No

Display without Privilege Restrictions

Valid only when the name of at least one database role has been specified in the roles list.

Distance Between Records Property

Description

Specifies the amount of space between instances of items in a multi-record block. A multi-record block is a block that has the Number of Records Displayed property set to greater than 1.

Applies to item

Set Forms Developer

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 Distance Between Records property to at least 4, one cell for each space between item instances.

Related topics

Number of Records Displayed property

Number of Items Displayed property

Dither Property

Description

Specifies the whether the image is dithered when it is displayed.

Applies to graphic image

Set Forms Developer

Default

No

Required/Optional required

DML Array Size Property

Description

Specifies the maximum array size for inserting, updating, and deleting records in the database at one time.

Applies to block

Set Forms Developer

Default

1

Usage Notes

A larger size reduces transaction processing time by reducing network traffic to the database, but requires more memory. The optimal size is the number of records a user modifies in one transaction.

DML Array Size Restrictions

Minimium number of records is 1; there is no maximum.

- When the DML Array Size is greater than 1 and Insert Allowed is Yes, you must specify one or more items as a primary key, because you cannot get the ROWID of the records. ROWID is the default construct ORACLE uses to identify each record. With single record processing, the ROWID of a record is obtained for future reference (update or delete). During array processing, the ROWID of each record in the array is not returned, resulting in the need to designate one or more primary key items in the block. The primary key is used to specify the row to lock, and the ROWID is used for updating and deleting. BLOCK.ROWID is not available until the record is locked. You should specify one or more items in the block as the primary key even if the Key Mode value is Unique (the default).
- When DML Array Size is greater than 1, Update Changed Columns Only is always set to No at runtime, even if Update Changed Columns Only is Yes in Forms Developer. Update Changed Columns Only specifies that only columns whose values are actually changed should be included in the UPDATE statement during a COMMIT.

- If a long raw item (such as an image) appears in the block, the DML Array Size is always set to 1 at runtime.

DML Data Target Name Property

Description

Specifies the name of the block's DML data target. The DML Data Target Name property is valid only when the DML Data Target Type property is set to Table.

Applies to block

Set Forms Developer, programmatically

Refer to Built-in

GET_BLOCK_PROPERTY

SET_ITEM_PROPERTY

Default

NULL

Required/Optional optional

DML Data Target Name Restrictions

Prior to setting the DML Data Target Name property you must perform a COMMIT_FORM or a CLEAR_FORM.

DML Data Target Type Property

Description

Specifies the block's DML data target type. A DML data target type can be a Table, Procedure, or Transactional Trigger.

Applies to block

Set Forms Developer, programmatically

Refer to Built-in

GET_BLOCK_PROPERTY

Default

Table

Required/Optional required

Related topics

DML Data Target Name property

DML Returning Value Property

Description

Specifies whether Forms should use new or old behavior when updating client-side data with changed values after a database update or insert. A Yes setting for this property selects new behavior (new as of Release 6). A No setting selects old behavior (behavior of Release 5 and earlier).

A database update or insert action may initiate server-side triggers that cause alterations or additional changes in the data. In Release 6, when using an Oracle8 database server, Forms uses the DML Returning clause to immediately bring back any such changes. When this property is set to Yes, Forms will automatically update the client-side version of the data, and the user will not need to re-query the database to obtain the changed values.

When this property is set to No, Forms will not automatically update the client-side version of the data. (This is its pre-Release 6 behavior.) In this case, if the user subsequently tries to update a row whose values were altered on the server side, the user receives a warning message and is asked to re-query to obtain the latest values. This No setting is available as a compatibility option.

Applies to block

Set Forms Developer

Valid values Yes/No

Default No

Required/Optional required

Restrictions

- Forms uses the DML Returning clause only with an Oracle8 database server. This property is ignored when using a non-Oracle8 server.
- Forms uses the Returning clause with Insert and Update statements, but (currently) not with Delete statements.
- Forms does not use the Returning clause when processing LONGs.
- The updating of unchanged columns is controlled by the setting of the Update Changed

Columns Only property, which in turn is affected by the setting of the DML Array Size property.

Edge Background Color Property

Description

Specifies the background color of the graphic object's edge.

Applies to graphic font & color

Set Forms Developer

Default

Null

Required/Optional optional

Edge Foreground Color Property

Description

Specifies the foreground color of the graphic object's edge.

Applies to graphic font & color

Set Forms Developer

Default

Null

Required/Optional optional

Edge Pattern Property

Description

Specifies the pattern of the graphic object's edge.

Applies to graphic font & color

Set Forms Developer

Default

Null

Required/Optional optional

Editor Property

Description

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 Forms Developer that you specified by setting the SYSTEM_EDITOR environment variable

Applies to text item

Set Forms Developer

Refer to Built-in

GET_ITEM_PROPERTY

Default

blank, indicating the default Forms Developer editor

Required/Optional optional

Usage Notes

To specify a system editor:

- Define the system editor by setting the FORMS60_EDITOR environment variable.
- Enter the value SYSTEM_EDITOR in the Editor Name field.

Editor Restrictions

The editor specified must exist in the active form.

Editor X Position, Editor Y Position

Description

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 Forms Developer

Refer to Built-in

GET_ITEM_PROPERTY

Default

0, 0; indicating that Forms Developer should use the default editor display coordinates, as specified by the editor Position property.

Required/Optional optional

Elements in List Property

Description

The Elements in List property group includes the List Item and List Item Value properties.

Applies to list item

Set Forms Developer

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

Usage Notes

When you leave the List Item Value field blank, the value associated with the element is NULL.

Elements in List Restrictions

- Must be unique among values associated with element values.

Enabled (Item) Property

Description

Determines whether end users can use the mouse to manipulate an item.

On most window managers, Enabled set to No grays out the item.

Applies to all items except buttons, chart items, and display items

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

Default

Yes

Usage Notes

When Enabled is set to Yes, Keyboard Navigable can be set to Yes or No. When Enabled is No, an item is always non-Keyboard Navigable. At runtime, when the Enabled property is set to PROPERTY_FALSE, the Keyboard_Navigable property is also set to PROPERTY_FALSE.

Enabled set to No grays out the item. If you want the item to appear normally so the user can inspect it but without being able to change it, set the following properties:

- Insert Allowed (Item) to No
- Update Allowed (Item) to No
- Enabled to Yes

Related topics

Insert Allowed (Item) property

Update Allowed (Item) property

About making items navigable and enabled

Enabled (Menu Item) Property

Description

Specifies whether the menu item should be displayed as an enabled (normal) item or disabled (grayed) item.

Applies to menu item

Set Forms Developer, programmatically

Refer to Built-in

GET_MENU_ITEM_PROPERTY

Default

Yes

Enabled (Menu Item) 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 Yes.
- The menu item Display without Privilege property is set to No.
- The current end user is not a member of a role that has access to the menu item.

Enabled (Tab Page) Property

Description

Specifies whether the tab page should be displayed as enabled (normal) or disabled (greyed out).

Applies to tab page

Set Forms Developer, programmatically

Refer to Built-in

GET_TAB_PAGE_PROPERTY

SET_TAB_PAGE_PROPERTY

Default

Yes

Enabled properties

Enabled (Item) property

Enabled (Menu Item) property

Enabled (Tab Page) property

End Angle Property

Description

Specifies the ending angle of the arc, using the horizontal axis as an origin.

Applies to graphic arc

Set Forms Developer

Default

180

Required/Optional required

Enforce Column Security Property

Description

Specifies when Forms Developer should enforce update privileges on a column-by-column basis for the block's base table. If an end user does not have update privileges on a particular column in the base table, Forms Developer makes the corresponding item non-updateable for this end user 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:

State	Effect
Yes	Forms Developer enforces the update privileges that are defined in the database for the current end user.
No	Forms Developer does not enforce the defined update privileges.

Applies to block

Set Forms Developer

Refer to Built-in

GET_BLOCK_PROPERTY

Default

No

Related topics

Update Allowed (Block) property

Update Allowed (Item) property

Update_Permission property

Enforce Primary Key (Block) Property

Description

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 Forms Developer, programmatically

Refer to Built-in

GET_BLOCK_PROPERTY

SET_BLOCK_PROPERTY

Default

No

Enforce Primary Key (Block) Restrictions

- The Primary Key item property must be set to Yes for one or more items in the block.

Related topic

Primary Key (Item) property

Enterable Property

Description

Specifies whether the block is enterable.

Applies to block

Set not settable

Refer to Built-in

GET_BLOCK_PROPERTY

Usage Notes

- A block is enterable when its current record contains an item instance whose Keyboard Navigable property has an *effective* value of true. See the Keyboard Navigable property and SET_ITEM_INSTANCE_PROPERTY built-in for information about effective Keyboard Navigable values.

Related topics

About making items navigable and enabled

About making items navigable and enabled

Error_Date/Datetime_Format Property

Description

Holds the current error date or datetime format mask established by the environment variable FORMSnn_ERROR_DATE_FORMAT or FORMSnn_ERROR_DATETIME_FORMAT. Forms uses these format masks as defaults in its runtime error processing.

There are two separate properties: Error_Date_Format and Error_Datetime_Format.

Applies to application

Set Not settable from within Forms Developer.

Refer to Built-in

GET_APPLICATION_PROPERTY

Execution Hierarchy Property

Description

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 Forms Developer

Default

Override

Execution Mode (Chart) Property

Description

When running Graphics from Forms Developer to create a chart, this property specifies the execution mode to be used as either Batch or Runtime. Batch mode executes the report or graphic without user interaction. Runtime mode enables user interaction during the execution.

Applies to chart items

Set Forms Developer

Default

Batch

Required/Optional required

Execution Mode (Report) Property

Description

For report integration with a form, this property specifies the execution mode of the report as either Batch or Runtime. Batch mode executes the report or graphic without user interaction. Runtime mode enables user interaction during the execution.

Applies to report Developer integration

Set Forms Developer

Default

Batch

Required/Optional required

Filename Property

Description

Specifies the name of the file where the named object is stored.

Applies to form, report

Set not settable

Refer to Built-in

GET_FORM_PROPERTY

Required/Optional optional

Usage Notes

Filename at the form level corresponds to Current_Form at the application level. Current_Form is gettable with GET_APPLICATION_PROPERTY.

Filename property Restrictions

If two or more forms share the same name, Filename supplies the name of the file where the most recently-accessed form is stored.

Related topic

Current_Form property

Fill Pattern Property

Description

Specifies 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.

Applies to item, tab page, canvas, window, radio button

Set Programmatically

Default

Unspecified

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

GET_TAB_PAGE_PROPERTY

SET_TAB_PAGE_PROPERTY

GET_CANVAS_PROPERTY

SET_CANVAS_PROPERTY

GET_WINDOW_PROPERTY

SET_WINDOW_PROPERTY

Fill Property

Description

Specifies the fill shape of the arc as either Pie or Chord. Pie renders the arc from the center point of the circle described by the arc. Chord renders the arc from a line segment between the arc's two end points.

Applies to graphic arc

Set Forms Developer

Default

Pie

Required/Optional required

Filter Before Display Property

Description

When Filter Before Display is set to Yes, Forms Developer displays a query criteria dialog before displaying the LOV. End users can enter a value in the query criteria dialog to further restrict the rows that are returned by the default SELECT statement that populates the LOV's underlying record group. Forms Developer uses the value entered in the query criteria dialog to construct a 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 Forms Developer appends the wildcard symbol to the value entered by the end user. For example, if the end user enters 7, the WHERE clause reads LIKE '7%' and would return 7, 712, and 7290.

Keep in mind that once the end user enters a value in the query criteria dialog and the LOV is displayed, the LOV effectively contains only those rows that correspond to both the the default SELECT statement and the WHERE clause created by the value in the query criteria dialog. For example, consider an LOV whose default SELECT statement returns the values FOO, FAR, and BAZ. If the end user enters the value F or F% in the query criteria dialog, the resulting LOV contains only the values FOO and FAR. If the user then enters the value B% in the LOV's selection field, nothing will be returned because BAZ has already been selected against in the query criteria dialog.

Applies to LOV

Set Forms Developer

Default

No

Filter Before Display Restrictions

- 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 end user attempts to use the Filter Before Display 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 end users 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 end user. Instead, Forms Developer 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.
- This property is required for cancelling a query in a long list LOV

Fire in Enter-Query Mode Property

Description

Specifies that the trigger should fire when the form is in Enter-Query mode, as well as in Normal mode.

Applies to trigger

Set Forms Developer

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

First Detail Relation Property

Description

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.

Related topics

First_Master_Relation property

Next_Detail_Relation property

Next_Master_Relation property

First_Block Property

Description

Specifies the block that is the first block in the form, as indicated by the sequence of blocks in the Object Navigator. At startup, Forms Developer navigates to the first item in the first block.

Applies to form

Set not settable

Refer to Built-in

GET_FORM_PROPERTY

First Item Property

Description

Specifies the item that is the first item in the block, as indicated by the sequence of items in the Object Navigator. At startup, Forms Developer navigates to the first item in the first block.

Applies to block

Set not settable

Refer to Built-in

GET_BLOCK_PROPERTY

Related topic

Last_Item property

First Master Relation Property

Description

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_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.

Related topics

First_Detail_Relation property

Next_Detail_Relation property

Next_Master_Relation property

First Navigation Block Property

Description

Specifies the name of the block to which Forms Developer 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 Forms Developer, 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 Pre-Form trigger, which fires at form startup, before Forms Developer navigates internally to the first block in the form.

Related topic

When-New-Form-Instance Trigger

Fixed Bounding Box Property

Description

Specifies whether the text object's bounding box should remain a fixed size. If this property is set to Yes, the values of the Width and Height properties determine the size of the bounding box.

Applies to graphic text

Set Forms Developer

Default

No

Required/Optional required

Flag User Value Too Long Property

Description

Specifies how Forms should handle a user-entered value that exceeds the item's Maximum Length property.

This property applies only in a 3-tier environment in which the middle tier (the Forms Servlet) specifies a multi-byte character set other than UTF8.

Applies to application

Set programmatically

Default

Property_False ('FALSE')

Refer to Built-in

GET_APPLICATION_PROPERTY

SET_APPLICATION_PROPERTY

Usage Notes

In a 3-tier, non-UTF8 multi-byte character set environment, it is possible for an end user to type more bytes into an item than the item's Maximum Length property specifies.

When the Flag User Value Too Long property has been set or defaulted to **FALSE** and this situation arises, then the user's typed-in value is truncated (on a character boundary) so that its size in bytes does not exceed the item's Maximum Length. When item-level validation is performed, the truncated value is validated. If validation (and any navigational triggers) succeeds, then the end user is allowed to navigate out of the item. No error or warning message is displayed.

When the Flag User Value Too Long property has been set to **TRUE** and this situation arises, then the user's typed-in value is not truncated. When item-level validation is performed, it will fail (with an error message indicating that truncation would be necessary). This means that the end user is not allowed to leave the current validation unit (as specified by the current form’s Validation Unit property).

Font Name Property

Description

Specifies the font family, or typeface, to be used for text in the object. The list of fonts available is system-dependent.

Applies to item, tab page, canvas, window, radio button

Set Programmatically

Default

Unspecified

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

GET_TAB_PAGE_PROPERTY

SET_TAB_PAGE_PROPERTY

GET_CANVAS_PROPERTY

SET_CANVAS_PROPERTY

GET_WINDOW_PROPERTY

SET_WINDOW_PROPERTY

Font_Size Property

Description

Specifes the size of the font in points.

Applies to item, tab page, canvas, window, radio button

Set Programmatically

Default

Unspecified

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

GET_TAB_PAGE_PROPERTY

SET_TAB_PAGE_PROPERTY

GET_CANVAS_PROPERTY

SET_CANVAS_PROPERTY

GET_WINDOW_PROPERTY

SET_WINDOW_PROPERTY

Font Spacing Property

Description

Specifies the width of the font (i.e., the amount of space between characters, or kerning). Valid values are:

FONT_NORMAL

FONT_ULTRADENSE

FONT_EXTRADENSE

FONT_DENSE

FONT_SEMIDENSE

FONT_SEMIEXPAND

FONT_EXPAND

FONT_EXTRAEXPAND

FONT_ULTRAEXPAND

Applies to item, tab page, canvas, window, radio button

Set Programmatically

Default

FONT_NORMAL

Refer to Built-in

GET_CANVAS_PROPERTY

SET_CANVAS_PROPERTY

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

GET_TAB_PAGE_PROPERTY

SET_TAB_PAGE_PROPERTY

GET_VA_PROPERTY

SET_VA_PROPERTY

GET_WINDOW_PROPERTY

SET_WINDOW_PROPERTY

Font Style Property

Description

Specifies the style of the font. Valid values are:

FONT_PLAIN

FONT_ITALIC

FONT_OBLIQUE

FONT_UNDERLINE

FONT_OUTLINE

FONT_SHADOW

FONT_INVERTED

FONT_OVERSTRIKE

FONT_BLINK

Applies to item, tab page, canvas, window, radio button

Set Programmatically

Default

FONT_PLAIN

Refer to Built-in

GET_CANVAS_PROPERTY

SET_CANVAS_PROPERTY

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

GET_TAB_PAGE_PROPERTY

SET_TAB_PAGE_PROPERTY

GET_VA_PROPERTY

SET_VA_PROPERTY

GET_WINDOW_PROPERTY

SET_WINDOW_PROPERTY

Font Weight Property

Description

Specifies the weight of the font. Valid values are:

FONT_MEDIUM

FONT_ULTRALIGHT

FONT_EXTRALIGHT

FONT_LIGHT

FONT_DEMILIGHT

FONT_DEMIBOLD

FONT_BOLD

FONT_EXTRABOLD

FONT_ULTRABOLD

Applies to item, tab page, canvas, window, radio button

Set Programmatically

Default

FONT_MEDIUM

Refer to Built-in

GET_CANVAS_PROPERTY

SET_CANVAS_PROPERTY

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

GET_TAB_PAGE_PROPERTY

SET_TAB_PAGE_PROPERTY

GET_VA_PROPERTY

SET_VA_PROPERTY

GET_WINDOW_PROPERTY

SET_WINDOW_PROPERTY

Foreground Color Property

Description

Specifies the color of the object’s foreground region. For items, defines the color of the text displayed in the item.

Applies to item, tab page, canvas, window, radio button

Set Programmatically

Default

Unspecified

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

GET_TAB_PAGE_PROPERTY

SET_TAB_PAGE_PROPERTY

GET_CANVAS_PROPERTY

SET_CANVAS_PROPERTY

GET_WINDOW_PROPERTY

SET_WINDOW_PROPERTY

Form Horizontal Toolbar Canvas Property

Description

On Microsoft Windows, specifies the canvas that should be displayed as a horizontal toolbar on the MDI application window. The canvas specified must have the Canvas Type property set to Horizontal Toolbar.

Applies to form

Set Forms Developer

Default

Null

Required/Optional optional

Form Horizontal Toolbar Canvas Restrictions

Valid only on Microsoft Windows. On other platforms, the Form Horizontal Toolbar Canvas property is ignored and the canvas is mapped to the window indicated by its Window property setting.

Form Name Property

Description

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.

Related topic

Current_Form_Name property

Form Vertical Toolbar Canvas Property

Description

On Microsoft Windows, specifies the toolbar canvas that should be displayed as a vertical toolbar on the MDI application window. The canvas specified must have the Canvas Type property set to Vertical Toolbar.

Applies to form

Set Forms Developer

Default

Null

Required/Optional optional

Form Vertical Toolbar Canvas Restrictions

Valid only on Microsoft Windows. On other platforms, the Form Vertical Toolbar Canvas property is ignored and the toolbar canvas is mapped to the window indicated by its Window property setting.

Format Mask Property

Description

Specifies the display format and input accepted for data in text items.

Applies to text item

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Required/Optional optional

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.

Element	Example	Description
FM	FMXX99	Fill mode: accept string as typed, do not right justify. Allows end user input string to be shorter than the format mask.
X	XXXX	Any alphabetic, numeric, or special character. End user input string must be exact length specified by format mask.
9	9999	Numeric characters only. End user input string must be exact length specified by format mask.

A	AAAA	Alphabetic characters only. End user input string must be exact length specified by format mask.

Character String Examples

Format Mask	Description
XXAA	Will accept: --ab, abcd, 11ab; will *not* accept: --11, ab11, or ab--(must use XX to accept hyphens and other special characters).
XXXX	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.)
FMXX99	Will accept ab12, ab1, ab followed by two spaces; will *not* accept 12ab or abcd. (To produce the Forms Developer Version 3.0 Alpha datatype, use FMAAAAAA.)

- 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 end user enters them.

NUMBERS

The following table describes valid format masks for numbers.

Element	Example	Description
9	9999	Number of nines determines display width. Any leading zeros will be displayed as blanks.
0	0999	Display leading zeros.
0	9990	Display zero value as zero, not blank.
$	$9999	Prefix value with dollar sign.

B	B9999	Display zero value as blank, not "0".
MI	9999MI	Display "-" after a negative value.
PR	9999PR	Display a negative value in <angle brackets>.
comma	9,999	Display a comma in this position. For correct behavior in multilingual applications, substitute G to return the appropriate group (thousands) separator.
period	99.99	Display a decimal point in this position. For correct behavior in multilingual applications, substitute D to return the appropriate decimal separator.
E	9.999EEEE	Display in scientific notation (format must contain exactly four "E"s).
FM	FM999	Fill mode: accept string as typed, do not right justify.

- When you mask a number with nines (9), Forms Developer 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 Forms Developer 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 end user enters them.

NUMBER Examples

Format Mask	**Description**
FM099"-"99"-"9999	Displays the social security number as formatted, including hyphens, even if end user 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).
99999PR	Accepts -123; reformats as <123>.

999MI	Accepts -678; reformats as 678-.
9.999EEEE	Displays as 1.00E+20.

How Forms handles length mismatches

If a runtime user enters a numeric string that exceeds the format mask specification, the value will be rejected. For example:

Format Mask	User enters	Result
99.9	321.0	Invalid
99.9	21.01	Invalid
99.9	21.1	21.1
99.9	01.1	1.1

In contrast, if a numeric value fetched from the database exceeds the format mask specification for its display field, the value is displayed, but truncated, with rounding, to fit the mask. (The item itself within the Forms application retains its full value.) For example, if the database held the value 2.0666, and the format mask was 99.9, the value displayed to the user would be 2.1. However, the value of the item within the form would be the full 2.0666.

Dates

The following table describes valid format masks for dates.

Element	Description
YYYY or SYYYY	4-digit year; "S" prefixes "BC" date with "-".
YYY or YY or Y	Last 3, 2, or 1 digits of year.
Y,YYY	Year with comma in this position.

BC or AD	BC/AD indicator.
B.C. or A.D.	BD/AD indicator with periods.
RR	Defaults to correct century. Deduces the century from a date entered by comparing the 2 digit year entered with the year and century to which the computer's internal clock is set. Years 00-49 will be given the 21st century (the year 2000), and years from 50-99 will be given the 20th century (the year 1900).
MM	Month (01-12; JAN = 01).
MONTH	Name of month, padded with blanks to length of 9 characters.
MON	Name of month, 3-letter abbreviation.
DDD	Day of year (1-366).
DD	Day of month (1-31).
D	Day of week (1-7; Sunday=1).
DAY	Name of day, padded with blanks to length of 9 characters.
DY	Name of day, 3-letter abbreviation.
J	Julian day; the number of days since January 1, 4712 BC.
AM or PM	Meridian indicator.
A.M. or P.M.	Meridian indicator with periods.
HH or HH12	Hour of day (1-12).
HH24	Hour of day (0-23).
MI	Minute (0-59).
SS	Second (0-59).

SSSSS	Seconds past midnight (0-86399).
/. , .	Punctuation is reproduced in the result.
"..."	Quoted string is reproduced in the result.
FM	Fill mode: assumes implied characters such as O or space; displays significant characters left justified. Allows end user input to be shorter than the format mask. (Use in conjunction with FX to require specific delimiters.)
FX	All date literals must match the format mask exactly, including delimiters.

- When you prefix a date mask with FX, the end user must enter the date exactly as you define the mask, including the specified delimiters:

Date Examples

Format Mask	**Description**
FXDD-MON-YY	Will accept 12-JAN-94, but will *not* accept 12.JAN.94 or 12/JAN/94 because the delimiters do not match the mask. Will not accept 12JAN94 because there are no delimiters. Will accept 01-JAN-94 but will not accept 1-JAN-94.
FMDD-MON-YY	Will accept 01-JAN-94. Will also accept the entry of other delimiters, for example 01/JAN/94 and 01 JAN 94. However, will not accept 01JAN94. Will accept 1-JAN-94, converting it to 01-JAN-94.
DD-MON-YY	Will accept 12.JAN.94, 12/JAN/94 or 12-JAN-94. Note: Any delimiter characters will be accepted, but if delimiters are omitted by the end user, this mask will interpret date characters as a delimiters. Will accept 12-JAN94, (but will erroneously interpret as 12-JAN-04); but will *not* accept 12JAN94, because "AN" is not a valid month name.

- Use of a format mask only affects how the data looks. Forms Developer 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 end user enters them.
- To embed additional characters such as a hyphen (-) or a comma (,), surround the character with double-quotes ("). Note, however, that double-quotes themselves cannot

be used as a character. In other words, trying to achieve output of DD"MM by specifying a mask of DD""""MM would not work.

Format Mask	Description
FMMONTH" "DD", "YYYY	Displays the text item data in the specified date format: JANUARY 12, 1994, including the appropriate blank spaces and comma.
FMDD-MONTH-YYYY	Displays as 12-JANUARY-1994.
DY-DDD-YYYY	Displays as WED-012-1994. Note: for input validation including day of the week, a mask that allows specific determination of the day is required, such as this example or DY-DD-MM-YY.

- 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.

Format Mask	Description
DD-MONTH-YYYY	Displays as 12-JANUARY-1994.
DY-DDD-YYYY	Displays as WED-012-1994.
DY-DD-MON-YY	Displays as WED-12-JAN-94. Be sure to include month. Avoid masks such as DY-DD-YY, which could generate an error.

NLS Format Masks

The following table describes valid National Language Support (NLS) format masks.

Element	Example	Description
C	C999	Returns the international currency symbol.

L	L9999	Returns the local currency symbol.
D	99D99	Returns the decimal separator.
G	9G999	Returns the group (thousands) separator.
comma	9,999	Displays a comma in this position.
period	9.999	Displays a decimal point in this position. Displays a decimal point in this position.

NLS Format Mask Examples

Format Mask	Description
L99G999D99	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.
C99G999D99	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.

Format Mask Restrictions

- 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.
- Forms Developer supports only ORACLE format masks that are used for both input and output. Output-only format masks, such as WW, are not supported.

Related topic

Data Type property

Formula Property

Description

Specifies a single PL/SQL expression that determines the value for a formula calculated item. The expression can reference built-in or user-written subprograms.

Applies to item

Set Forms Developer

Refer to Built-in

RECALCULATE

Usage Notes

You *cannot* enter an entire PL/SQL statement as your formula; accordingly, do not terminate your calculation expression with a semicolon. Forms Developer adds the actual assignment code to the formula internally do not code it yourself. For example, instead of coding an entire assignment statement, code just the expression

```
:emp.sal + :emp.comm
```

Forms Developer will internally convert this into a complete statement, e.g.,

```
:emp.gross_comp := (:emp.sal + :emp_comm);
```

Required/Optional required if Calculation Mode property is set to Formula

Frame Alignment Property

Description

Specifies how objects should be aligned within the width of the frame, either Start, End, Center, Fill, or Column. This property is valid when the Layout Style property is set to Form.

Applies to frame

Set Forms Developer

Default

Fill

Required/Optional required

Frame Title Alignment Property

Description

Specifies the title alignment for a frame, either Start, End, or Center.

Note: Title alignment is relative to the Direction of the canvas on which the canvas appears.

Applies to frame

Set Forms Developer

Default

Start

Required/Optional required

Frame Title Font Name Property

Description

Specifies the name of the font (typeface) to apply to the frame title.

Applies to frame

Set Forms Developer

Default

Defaults to the standard operating system font

Required/Optional required

Frame Title Font Size property

Description

Specifies the size of the font (typeface) to apply to the frame title.

Applies to frame

Set Forms Developer

Default

Defaults to the standard operating system font size

Required/Optional required

Frame Title Font Spacing property

Description

Specifies the spacing to apply to the frame title text.

Applies to frame

Set Forms Developer

Default

Defaults to the standard operating system font spacing

Required/Optional required

Frame Title Font Style Property

Description

Specifies the typographic style (for example, *Italic*) to apply to the frame title text.

Applies to frame

Set Forms Developer

Default

Defaults to the standard operating system font style

Required/Optional required

Frame Title Font Weight Property

Description

Specifies the typographic weight to apply to the frame title text.

Applies to frame

Set Forms Developer

Default

Defaults to the standard operating system font weight.

Required/Optional required

Frame Title Foreground Color Property

Description

Specifies the color to apply to the frame title text.

Applies to frame

Set Forms Developer

Default

Defaults to the standard operating system font color (usually black).

Required/Optional required

Frame Title Offset Property

Description

Specifies the distance between the frame and its title.

Applies to frame

Set Forms Developer

Default

2 char cells (or the equivalent depending on the form coordinate system)

Required/Optional required

Frame Title Property

Description

Specifies the frame's title.

Applies to frame

Set Forms Developer

Default

blank

Required/Optional optional

Frame Title Visual Attribute Group Property

Description

Specifies how the frame title'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 Group 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.

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 Group 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 Default.

Applies to frame title

Set Forms Developer

Default

Default

Usage Notes

- Default and named visual attributes can include the following individual attributes, listed in the order they appear in the Property Palette:

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 Spacing 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.

- 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. Override the default font for new items and boilerplate by setting the optional FORMS60_DEFAULTFONT environment variable. For example, On Microsoft Windows, set this variable in the ORACLE.INI file, as follows: FORMS60_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 Group 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 Forms Developer.
- When the Use 3D Controls form property is set to Yes 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 canvases that have Visual Attribute

Group 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 named visual attributes to the canvas.

- An item that has Visual Attribute Group set to Default, or that has individual attribute settings left unspecified, inherits those settings from the canvas to which it is assigned. Similarly, a canvas that has Visual Attribute Group 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 assigned to the window, at runtime, that canvas 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 Group property, or one or more of the individual attribute properties. (If a property class contains both Visual Attribute Group and individual attributes, the Visual Attribute Group 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.

Graphics Type Property

Description

A read-only property that specifies the type of the graphic object. Possible values include:

- Arc
- Chart
- Group
- Image
- Line
- Polygon
- Rectangle
- Rounded Rectangle
- Symbol
- Text

(Same possible values as Graphics Property Object Type.)

Applies to graphics general

Set Forms Developer

Default

Rectangle

Required/Optional required

Group Name Property

Description

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.

Hide on Exit Property

Description

For a modeless window, determines whether Forms Developer hides the window automatically when the end user navigates to an item in another window.

Applies to window

Set Forms Developer, programmatically

Refer to Built-in

GET_WINDOW_PROPERTY

SET_WINDOW_PROPERTY

Default

No

Hide on Exit Restrictions

- Cannot be set for a root window: a root window always remains visible when the end user navigates to an item in another window.

Highest Allowed Value/Lowest Allowed Value Property

Description

Determines the maximum value or minimum value, inclusive, that Forms Developer allows in the text item.

Applies to text item

Set Forms Developer

Refer to Built-in

GET_ITEM_PROPERTY

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)

- Forms Developer 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

- For compatibility with prior releases, a reference to a form item or to a sequence may be specified with a leading ampersand (&) instead of a leading colon (:).
- To specify a raw value that begins with a leading ampersand ('&') or a leading colon (':'), specify two of them (that is, '&&' or '::'). (This is a change in Forms behavior, beginning with Release 6.5.)

Related topics

$$DATE$$ examples

$$DBDATE$$ examples

$$DATETIME$$ examples

$$DBDATETIME$$ examples

Hint (Item) Property

Description

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 Forms Developer

Refer to Built-in

GET_ITEM_PROPERTY (HINT_TEXT)

Default

"Enter value for: <item name>"	For an item that was created by using the Data Block Wizard
NULL	For all other items

Required/Optional optional

Usage Notes

Leave the Hint property NULL if you do not want the item to have hint text.

Related topic

Display Hint Automatically property

Horizontal Justification Property

Description

Specifies the horizontal justification of the text object as either Left, Right, Center, Start, or End.

Applies to graphic text

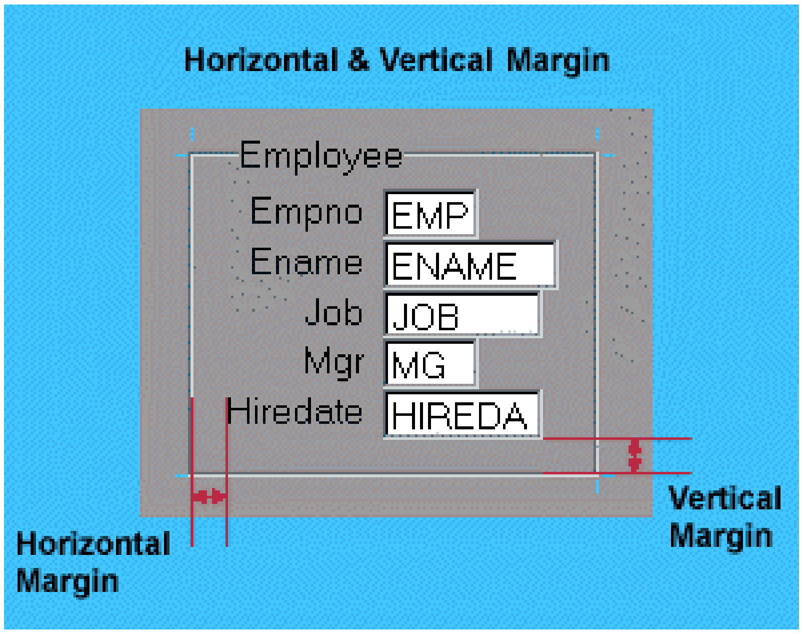
Set Forms Developer

Default

Start

Required/Optional required

Horizontal Margin



Related topic

Horizontal Margin Property

Horizontal Object Offset Property

Description

Specifies the horizontal distance between the objects within a frame.

Applies to frame

Set Forms Developer

Default

2 char cells (or the equivalent depending on the form coordinate system)

Required/Optional required

Horizontal Toolbar Canvas Property

Description

Specifies the canvas that should be displayed as a horizontal toolbar on the window. The canvas specified must be a horizontal toolbar canvas (Canvas Type property set to Horizontal Toolbar) and must be assigned to the current window by setting the Window property.

Applies to window

Set Forms Developer

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 canvases that have the Canvas Type property set to Horizontal Toolbar.
- At runtime, Forms Developer 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 Window property to point to the specified window), Forms Developer may display a different toolbar in response to navigation events or programmatic control.
- On Microsoft Windows, the specified horizontal toolbar canvas will not be displayed on the window if you have specified that it should be displayed on the MDI application window by setting the Form Horizontal Toolbar Canvas form property.

Related topics

Canvas Type property

Vertical Toolbar Canvas property

Icon Filename Property

Description

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 Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_MENU_ITEM_PROPERTY

SET_MENU_ITEM_PROPERTY

GET_WINDOW_PROPERTY

SET_WINDOW_PROPERTY

Default

NULL

Required/Optional optional

Usage Notes

When defining the Icon Filename property, do not include the icon file extension (.ico, .xpm, etc.). For example, enter my_icon, not my_icon.ico.

The icon filename should not end in any of the following five letters: A, L, M, S, and X. (Neither upper-case nor lower-case are allowed.) These are reserved letters used internally for icon sizing. Unexpected icon placement results may occur if your icon filename ends in any of these letters.

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 UI60_ICON.

Icon Filename Restrictions

- For a window, it is only valid when Minimize Allowed property set to Yes.
- 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.

Related topic

Defining an iconic button

Icon in Menu Property

Description

Specifies whether an icon should be displayed in the menu beside the menu item. If Yes, the Icon Filename property specifies the icon that will be displayed.

Applies to menu item

Set Forms Developer

Default

No

Required/Optional optional

Related topics

Icon Filename property

Iconic Property

Description

Specifies that a button is to be an iconic button.

Applies to button

Set Forms Developer

Refer to Built-in

GET_ITEM_PROPERTY

Default

No

Required/Optional optional

Usage Notes

When Iconic is Yes, the button's Icon Filename property specifies the icon resource that Forms Developer should display for the button.

Iconic Restrictions

A valid icon resource file name must be supplied.

Image Depth Property

Description

Specifies the image depth setting Forms Developer applies to an image being read from or written to a file in the filesystem. Valid values are:

- Original
- Monochrome
- Gray
- LUT (Lookup Table)
- RGB (Red, Green, Blue)

Applies to image item

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

WRITE_IMAGE_FILE

Default

Original

Required/Optional required

Image Format Property

Description

Specifies the format in which an image item will be stored in the database. Valid values are:

- BMP
- CALS
- GIF
- JFIF
- PICT
- RAS
- TIFF
- TPIC

Applies to image item

Set Forms Developer

Refer to Built-in

GET_ITEM_PROPERTY

WRITE_IMAGE_FILE

Default

TIFF

Required/Optional required

Usage Notes

- The default Oracle image storage format no longer is valid.
- The value you set for this property will override the original format of an image when the

record containing the image item is stored in the database.

For example, if an image item's Image Format property is set to GIF, and a TIFF image is pasted into the image item at runtime, the pasted image will be stored in GIF format when the record is saved to the database

Implementation Class Property

Description

Identifies the class name of a container of a JavaBean, or of a custom implementation for a control item type when you want to supply an alternate to the standard Forms Developer control.

Applies to

The following control item types:

- Bean Area
- Check Box
- List Item
- Push Button
- Radio Group
- Text Item

Set

Forms Developer

Default

None.

Required/Optional

Always required for Bean Area. This property identifies the class name of the container of the JavaBean you are adding to the application. (If this property is not supplied, the form's end user will see an empty square.)

Also required for any other control item type listed above if the form is to use a customized, user-supplied implementation of that control. This identifies the class name of the alternate control you are supplying.

Set at Runtime

No.

Usage Notes

- The Implementation Class property is only available for those control item types listed above, not for all control item types.

Related topic

Item Type property

Include REF Item Property

Description

Creates a hidden item called REF for this block. This item is used internally to coordinate master-detail relationships built on a REF link. This item also can be used programmatically to access the object Id (OID) of a row in an object table.

Applies to

Blocks based on object tables; master-detail REF links, in particular.

Set Forms Developer

Default

Default is No. However, when creating a relationship based on a REF pointer, Forms Developer sets this property to Yes.

Required/Optional

Required for a master block in a master-detail relationship based on a REF pointer.

Usage Notes

This REF item is used to obtain the object-ids (OIDs) of the rows in an object table.

Each row in an object table is identified by a unique object id (OID). The OID is a unique identifier for that row. These OIDs form an implicit column in an object table.

In a REF pointer relationship between two tables, the REF column in the pointing table holds copies of the OID values (addresses) of the pointed-to table. This forms the link between the two tables.

The Data Block wizard sets this property to Yes and creates this REF item when you build a master-detail relationship based on a REF pointer. The item is named REF, and is in the master block. It is not visible to the end user. In addition, the wizard sets the Copy_Value_From_Item property in the detail block to access this new REF. This is how Forms Developer coordinates the master-detail relationship at runtime.

Inherit Menu Property

Description

Specifies whether the window should display the current form menu on window managers that support this feature.

Applies to window

Set Forms Developer

Default

Yes

Required/Optional optional

Inherit Menu Restrictions

- Not valid on Microsoft Windows.

Initial Value (Item) Property

Description

Specifies the default value that Forms Developer 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 check boxes, display items, list items, radio groups, text items, and user areas

Set Forms Developer

Default

Null

Required/Optional Optional for all items except radio groups, check boxes, and list items.

For a radio group, a valid Initial Value is required unless

a) the radio group specifies Mapping of Other Values or,

b) the value associated with one of the radio buttons in the group is NULL.

For a list item, a valid Initial Value is required unless

a) the list item specifies Mapping of Other Values or,

b) the value associated with one of the list elements is NULL.

For a check box, a valid Initial Value is required unless

a) the check box specifies Mapping of Other Values or,

b) the value associated with Checked or Unchecked is NULL.

Usage Notes

- When 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 Forms Developer creates a record in the block.
- Subordinate mirror items are initialized from the master mirror item's Initial Value property. The ON-SEQUENCE-NUMBER trigger is also taken from the master item. If the subordinate mirror item specifies Initial Value and ON-SEQUENCE-NUMBER, Forms Developer ignores them and issues a warning.
- At runtime, the initial value set by this property will be ignored if all of the following are true for the item (or an item that mirrors it):
    - the item is a poplist, T-list, radio group, or check box
    - there is no element corresponding to the initial value
    - the item does not allow other values
- For compatibility with prior releases, a reference to a form item or to a sequence may be specified with a leading ampersand (&) instead of a leading colon (:).

To specify a raw value that begins with a leading ampersand ('&') or a leading colon (':'), specify two of them (that is, '&&' or '::'). (This is a change in Forms behavior, beginning with Release 6.5.)

Initial Value (Item) property Restrictions

- For a text item, the value cannot be outside the range defined by the Lowest Allowed Value and Highest Allowed 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. Forms Developer 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. Forms Developer checks first for a list element name.

Insert Allowed (Block) Property

Description

Specifies whether records can be inserted in the block.

Applies to block

Set Forms Developer , programatically

Refer to Built-in

GET_BLOCK_PROPERTY

SET_BLOCK_PROPERTY

Default

Yes

Related topics

Insert Allowed (Item) property

Query Allowed (Block) property

Query Allowed (Item) property

Update Allowed (Block) property

Update Allowed (Item) property

Insert Allowed (Item) Property

Description

Determines whether an end user 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 No 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 Forms Developer, programmatically

Refer to Built-in

GET_ITEM_INSTANCE_PROPERTY

GET_ITEM_PROPERTY

SET_ITEM_INSTANCE_PROPERTY

SET_ITEM_PROPERTY

Default

Yes

Usage Notes

Set Insert Allowed to No 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 No to prevent modification of the key while still displaying it normally (not grayed out).

Set the *Enabled* property to No 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 No 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 end user can enter or change the value displayed by an item. For items in non-database blocks, setting Insert Allowed to No lets you create a display-only item without disabling it.

If Enabled or Visible is set to No (or PROPERTY_FALSE for runtime), then the items' or item instance's Insert Allowed property is effectively false.

- Setting INSERT_ALLOWED to Yes (or PROPERTY_TRUE for runtime) has no effect at the item instance level unless it is set consistently at the block and item levels. For example, your user cannot type data into an item instance if INSERT_ALLOWED is true at the instance level, but not at the item or block levels.

Insert Allowed (Item) Restrictions

- If you are using SET_ITEM_PROPERTY to set Insert Allowed to true, then you must set item properties as follows:

  Enabled to Yes (PROPERTY_TRUE for runtime)
  Visible to Yes (PROPERTY_TRUE for runtime)
- When Insert Allowed is specified at multiple levels (item instance, item, and block), the values are "ANDed" together; meaning that setting INSERT_ALLOWED to Yes (PROPERTY_TRUE for runtime) has no effect at the item instance level unless it is set consistently at the block and item levels. For example, your user cannot type data into an item instance if INSERT_ALLOWED is true at the instance level, but not at the item or block levels.

Insert Procedure Arguments Property

Description

Specifies the names, datatypes, and values of the arguments to pass to the procedure for inserting data into the data block. The Insert Procedure Arguments property is valid only when the DML Data Target Type property is set to Procedure.

Applies to block

Set Forms Developer

Default

NULL

Required/Optional optional

Related topics

DML Data Target Name property

DML Data Target Type property

Insert Procedure Name property

Insert Procedure Arguments property

Insert Procedure Name Property

Description

Specifies the name of the procedure to be used for inserting data into the data block. The Insert Procedure Name property is valid only when the DML Data Target Type property is set to Procedure.

Applies to block

Set Forms Developer

Default

NULL

Required/Optional optional

Related topics

DML Data Target Name property

DML Data Target Type property

Insert Procedure Arguments property

Insert Procedure Arguments property

Isolation Mode Property

Description

Specifies whether or not transactions in a session will be serializable. If Isolation Mode has the value Serializable, the end user sees a consistent view of the database for the entire length of the transaction, regardless of updates committed by other users from other sessions. If the end user queries and changes a row, and a second user updates and commits the same row from another session, the first user sees Oracle error (ORA-08177: Cannot serialize access.).

Applies to form module

Set Forms Developer

Usage Notes

Serializable mode is best suited for an implementation where few users are performing a limited number of transactions against a large database; in other words, an implementation where there is a low chance that two concurrent transactions will modify the same row, and where long-running transactions are queries. For transaction-intensive implementations, leave Isolation Mode set to Read Committed (the default). Serializable mode is best used in conjunction with the block-level property Locking Mode set to Delayed.

Default

Read Committed

Required/Optional required

Item Is Valid Property

Description

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: No; item in a queried record: Yes

Usage Notes

- Use Item_Is_Valid to check whether the current status of a text item is valid.
- Set Item_Is_Valid to Yes to instruct Forms Developer to treat any current data in an item as *valid* and skip any subsequent validation. Set Item_Is_Valid to No to instruct Forms Developer to treat any current data in a text item as *invalid* and subject it to subsequent validation.

Item Roles Property

Description

Specifies which menu roles have access to a menu item.

Applies to menu item

Set Forms Developer

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 Module Roles.

Item Roles Restrictions

Valid only when the name of at least one role has been specified in the menu module roles list.

Item Tab Page Property

Description

Specifies the tab page on which an item is placed.

Applies to item

Refer to Built-in

GET_ITEM_PROPERTY

Default

None

Item Tab Page Restrictions

Related topic

Tab Page property

Join Condition Property

Description

Defines the relationship that links a record in the detail block with a record in the master block.

Applies to relation

Set Forms Developer

Required/Optional required for a relation object

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

Join Condition Restrictions

- Maximum length for a join condition is 255 characters.

Join Condition Examples

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.

Join Style Property

Description

Specifies the join style of the graphic object as either Mitre, Bevel, or Round.

Applies to graphic physical

Set Forms Developer

Default

Mitre

Required/Optional optional

Justification Property

Description

Specifies the text justification within the item. The allowable values for this property are as follows:

Value	Description
Left	Left-justified, regardless of Reading Order property.
Center	Centered, regardless of Reading Order property.
Right	Right-justified, regardless of Reading Order property.
Start	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.
End	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.

Applies to display item, text item

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM

SET_ITEM

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**.

Keep Cursor Position Property

Description

Specifies that the cursor position be the same upon re-entering the text item as when last exited.

Applies to text item

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

No

Usage Notes

- Use this property if you want to give the end user 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.

Keep Cursor Position Restrictions

- Unsupported on some window managers.

Key Mode Property

Description

Specifies how Forms Developer 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.

Value	**Description**
Automatic (default)	Specifies that Forms Developer should use ROWID constructs to identify unique rows in the datasource but only if the datasource supports ROWID.
Non-Updateable	Specifies that Forms Developer 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.
Unique	Instructs Forms Developer to use ROWID constructs to identify unique rows in an ORACLE database.
Updateable	Specifies that Forms Developer 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.

Applies to block

Set Forms Developer, 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 Enforce Primary Key block property to Yes for the block, and the Primary Key item property to Yes for at least one item in the block.

Keyboard Accelerator Property

Description

Specifies a logical function key to be associated with a menu item. Accelerator keys are named ACCELERATOR1, ACCELERATOR2, and so on, through ACCELERATOR5. End users 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 Forms Developer

Required/Optional optional

Usage Notes

The mappings of logical accelerator keys to physical device keys is defined in the runtime resource file.

Keyboard Accelerator Restrictions

- Not valid for separator menu items.
- Key mappings must not interfere with standard Forms Developer key mappings.
- When running with bar-style menus, accelerator keys can be used only for items on the menu that is currently displayed.

Keyboard Help Description Property

Description

Specifies the key trigger description that is displayed in the runtime Keys help screen if the Display in Keyboard Help property is set to Yes. An entry in the 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 Forms Developer

Default

blank

Usage Notes

- If you do not want the name or the description to appear in the Keys window, set the Display Keyboard Help property to No. 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 Keys window, set the Display Keyboard Help property to Yes and leave the Keyboard Help Description blank.
- If you want to replace the default key description, set the Display Keyboard Help property to Yes, then enter the desired description in the Keyboard Help Description field.

Keyboard Help Description Restrictions

Valid only for key triggers.

Related topic

Record Group Type property

Keyboard Navigable Property

Description

Determines whether the end user or the application can place the input focus in the item during default navigation. When set to Yes for an item, the item is navigable. When set to No, Forms Developer 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 Forms Developer, programmatically [NAVIGABLE]

Refer to Built-in

GET_ITEM_INSTANCE_PROPERTY

GET_ITEM_PROPERTY

SET_ITEM_INSTANCE_PROPERTY

SET_ITEM_PROPERTY

Default

Yes

Usage Notes

If Enabled or Visible is set to No (PROPERTY_FALSE for runtime), then the items' or item instance's Keyboard navigable property is effectively false. At runtime, when the Enabled property is set to PROPERTY_FALSE, the Keyboard_Navigable property is also set to PROPERTY_FALSE. However, if the Enabled property is subsequently set back to PROPERTY_TRUE, the keyboard Navigable property is NOT set to PROPERTY_TRUE, and must be changed explicitly.

- When Keyboard Navigable is specified at multiple levels (item instance, item, and block), the values are ANDed together. This means that setting Keyboard Navigable to Yes (or

NAVIGABLE to PROPERTY_TRUE for runtime) has no effect at the item instance level unless it is set consistently at the item level. For example, your user cannot navigate to an item instance if Keyboard Navigable is true at the instance level, but not at the item level.

- You can use the GO_ITEM built-in procedure to navigate to an item that has its Keyboard Navigable property set to No (PROPERTY_FALSE) for runtime.

Keyboard Navigable Restrictions

- If you are using SET_ITEM_PROPERTY to set NAVIGABLE to true, then you must set item properties as follows:

  Enabled to Yes (PROPERTY_TRUE for runtime)

  Visible to Yes (PROPERTY_TRUE for runtime)

Label (Item) Property

Description

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 Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_ITEM_INSTANCE_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

Default

blank

Required/Optional optional

Label (Menu Item) Property

Description

Specifies the text label for each menu item.

Applies to menu item

Set Forms Developer, 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, Forms Developer 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 Object Navigator or the Properties Palette

Label (Tab Page) Property

Description

The label identifying the tab page. End users click the labelled tab to display the tab pages of a tab canvas.

Applies to tab page

Set Forms Developer, programmatically

Refer to Built-in

GET TAB PAGE PROPERTY

SET TAB PAGE PROPERTY

Required/Optional optional

Label (Tab Page) Restrictions

none

Last_Block Property

Description

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

Related topics

First_Block property

Last_Item Property

Description

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_BLOCK_PROPERTY

Related topics

First_Item property

Last Query Property

Description

Specifies the SQL statement for the last query of the specified block.

Applies to block

Set not settable

Refer to Built-in

GET_BLOCK_PROPERTY

Layout Data Block Property

Description

Specifies the name of the data block which the frame is associated with; the items within this block are arranged within the frame. A block can only be associated with one frame. You cannot arrange a block's items within multiple frames.

Applies to frame

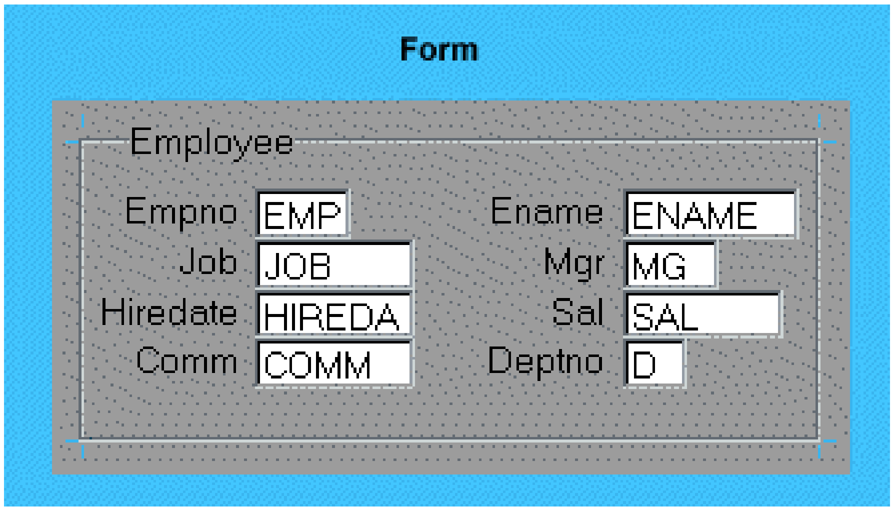
Set Forms Developer

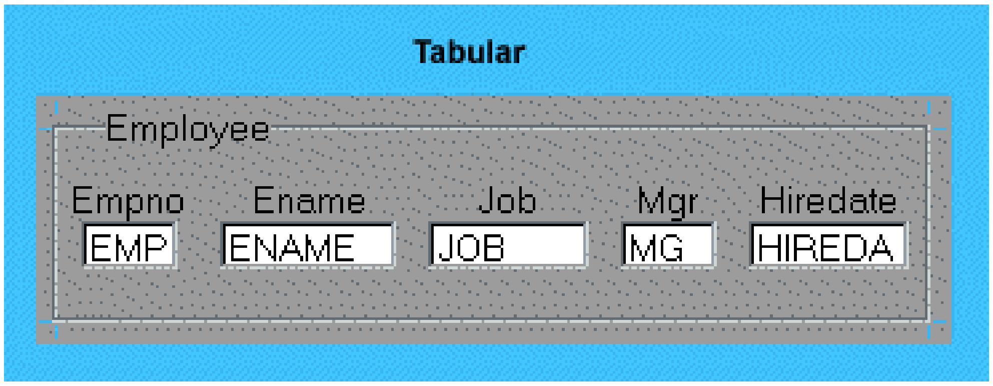
Default

NULL

Required/Optional required

Layout Styles





Related topic

Layout Style Property

Layout Style Property

Description

Specifies the layout style for the items within the frame.

Form The default frame style. When Frame Style is set is to Form, Forms Developer arranges the items in a two-column format, with graphic text prompts positioned to the left of each item.

Tabular When Frame Style is set to Tabular, Forms Developer arranges the items next to each other across a single row, with graphic text prompts above each item.

Applies to frame

Set Forms Developer

Default

Form

Required/Optional required

Related topic

Layout Style Property Showme

Length (Record Group) Property

Description

See Column Specifications.

Line Spacing Property

Description

Specifies the line spacing of the text objext as Single, One-and-a-half, Double, Custom. If Custom is selected, the Custom Spacing property determines the line spacing.

Applies to graphic text

Set Forms Developer

Default

Single

Required/Optional required

Line Width Property

Description

Specifies the width of the object's edge in points (1/72 inch). (Same as Graphics property Edge Width.)

Applies to graphic physical

Set Forms Developer

Required/Optional optional

List of Values Property

Description

Specifies the list of values (LOV) to attach to the text item. When an LOV is attached to a text item, end users can navigate to the item and press [List of Values] to invoke the LOV. To alert end users that an LOV is available, Forms Developer 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 Forms Developer

Refer to Built-in

GET_ITEM_PROPERTY

Required/Optional optional

List of Values Restrictions

The LOV must exist in the active form module.

List Style Property

Description

Specifies the display style for the list item, either poplist, combo box, or Tlist. The poplist and combo box styles take up less space than a Tlist, but end users must open the poplist or combo box to see list elements. A Tlist remains "open," and end users can see more than one value at a time if the list is large enough to display multiple values.

Applies to list item

Set Forms Developer

Default

Poplist

Usage Notes

The display style you select for the list item has no effect on the data structure of the list item.

List X Position Property

Description

Specifies the horizontal (X) coordinate of the upper left corner of the LOV relative to the screen. When you attach an LOV to a text item by setting the List of Values property, you can also set the List X Position and List Y Position properties to override the default display coordinates specified for the LOV.

Applies to text item

Set Forms Developer

Refer to Built-in

GET_ITEM_PROPERTY

Default

0; indicating that Forms Developer should use the default LOV display horizontal (X) coordinate, as specified by the List X Position property.

Required/Optional required

Usage Notes

- If you leave the List X Position property set to 0 and the List Y Position property set to 0, Forms Developer 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 List of Values property must be specified.

List Y Position Property

Description

Specifies the vertical (Y) coordinate of the upper left corner of the LOV relative to the screen. When you attach an LOV to a text item by setting the List of Values property, you can also set the List Y Position and List X Position properties to override the default display coordinates specified for the LOV.

Applies to text item

Set Forms Developer

Refer to Built-in

GET_ITEM_PROPERTY

Default

0; indicating that Forms Developer should use the default LOV display vertical (Y) coordinate, as specified by the List Y Position property.

Required/Optional required

Usage Notes

- If you leave the List X Position property set to 0 and the List Y Position property set to 0, Forms Developer 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 List of Values property must be specified.

Lock Procedure Arguments Property

Description

Specifies the names, datatypes, and values of the arguments that are to be passed to the procedure for locking data. The Lock Procedure Arguments property is valid only when the DML Data Target Type property is set to Procedure.

Applies to block

Set Forms Developer

Default

NULL

Required/Optional optional

Related topics

DML Data Target Name property

DML Data Target Type property

Lock Procedure Name property

Lock Procedure Arguments property

Lock Procedure Name Property

Description

Specifies the name of the procedure to be used for locking data. The Lock Procedure Name property is valid only when the DML Data Target Type property is set to Procedure.

Applies to block

Set Forms Developer

Default

NULL

Required/Optional optional

Related topics

DML Data Target Name property

DML Data Target Type property

Lock Procedure Arguments property

Lock Procedure Arguments property

Lock Procedure Result Set Columns Property

Description

Specifies the names and datatypes of the result set columns associated with the procedure for locking data. The Lock Procedure Result Set Columns property is valid only when the DML Data Target Type property is set to Procedure.

Applies to block

Set Forms Developer

Default

NULL

Required/Optional optional

Related topics

DML Data Target Name property

DML Data Target Type property

Lock Procedure Arguments property

Lock Procedure Name property

Lock Record Property

Description

Specifies that Forms Developer 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 end user or programmatically.

Applies to text item

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

No

Usage Notes

- Set this property to Yes when the text item is a control item (an item not associated with a base table column), but you still want Forms Developer 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.

Lock Record Restrictions

- Valid only when the item is a control item (Base Table Item property set to No) in a data block.

Locking Mode Property

Description

Specifies when Forms Developer tries to obtain database locks on rows that correspond to queried records in the form. The following table describes the allowed settings for the Locking Mode property:

Value	Description
Automatic (default)	Identical to *Immediate* if the datasource is an Oracle database. For other datasources, Forms Developer determines the available locking facilities and behaves as much like *Immediate* as possible.
Immediate	Forms Developer locks the corresponding row as soon as the end user presses a key to enter or edit the value in a text item.
Delayed	Forms Developer locks the row only while it posts the transaction to the database, not while the end user is editing the record. Forms Developer prevents the commit action from processing if values of the fields in the block have changed when the user causes a commit action.

Applies to block

Set Forms Developer, programmatically

Refer to Built-in

GET_BLOCK_PROPERTY

SET_BLOCK_PROPERTY

Default

Automatic

Usage Notes

For most applications use the default setting of `Automatic`.

The `Immediate` setting remains for compatibility with existing applications, but there is no reason to use it in new applications. Use `Automatic` instead.

The `Delayed` setting is useful for applications that must minimize the number of locks or the amount of time that rows remain locked. Use delayed locking if the form's Isolation Mode property has the value *Serializable*.

The main drawbacks of delayed locking are

- The changes an end user makes to a block may need to be redone at commit time.
- Another user's lock can force the first end user to choose between waiting indefinitely or abandoning the changes.

Magic Item Property

Description

Specifies one of the 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 Forms Developer, while the other magic menu items can have commands associated with them.

Applies to menu item

Set Forms Developer

Default

Cut

Required/Optional optional

Usage Notes

The following settings are valid for this property:

Setting	**Description**
Cut, Copy, Paste, Clear	These items perform the usual text-manipulation operations. Forms Developer supplies their functionality, so the designer may not enter a command for these items.
Undo, About	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.
Help	The command for the Help menu item must be Menu. The designer provides the functionality of items on this submenu.

Quit	Quit also has built-in functionality, so the designer may not assign a command to this item.
Window	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 Forms Developer. The command type for a magic Window item is Null or Menu.

Magic Item Restrictions

- 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, Keyboard Accelerator, and Hint properties blank.

Main Menu Property

Description

Specifies the name of the individual menu in the document 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.

Applies to menu module

Set Forms Developer

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 document to be the main menu by setting the Initial Menu property.

Related topic

Initial Menu property

Mapping of Other Values Property

Description

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 Forms Developer

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

Maximize Allowed Property

Description

Specifies that end users can resize the window by using the zooming capabilities provided by the runtime window manager.

Applies to window

Set Forms Developer

Default

Yes

Maximize Allowed Restrictions

- Only valid when Resize Allowed is set to No

Maximum Length (Form Parameter) Property

Description

Specifies the maximum length, in characters, of a form parameter of type CHAR.

Applies to form parameter

Set Forms Developer

Default

For a parameter of type CHAR, 30

Required/Optional required

Maximum Length (Form Parameter) Restrictions

- Maximum length of a CHAR parameter is 2000 bytes.

Maximum Length Property

Description

This property can potentially limit the amount of data that an item is allowed to contain internally (in the Forms server) when it's in native format. It can also potentially limit the number of character (or bytes) displayed or entered by the end user. That is, it can specify a data limit and/or a user interface (UI) limit. The exact effect depends upon several factors, including the item's Data type, Data Length Semantics, and Format Mask properties.

For a character item (datatypes `CHAR`, `ALPHA`, `LONG`), the `Maximum Length` property specifies a data limit. The property value specifies either the maximum number of bytes or else the maximum number of characters, depending upon the value of the `Data Length Semantics` property. Within a group of character mirror items, the `Maximum Length` property is always taken from the master mirror item. A compiler (generator) warning is issued if a non-zero non-default value is specified in a subordinate mirror item.

If a character item has no format mask (the `Format Mask` property is null), then the `Maximum Length` property (taken from the master mirror item) also specifies a UI limit. When the Forms server is using the UTF8 character set , this UI limit has the same semantics (byte versus character) as the data limit. When the Forms server is using a multi-byte character set other than UTF8, and the `Data Length Semantics` property specifies byte semantics, Forms cannot prevent the end user from typing in too many bytes. Instead, it uses character semantics for the UI limit.

In such a case, the end user is allowed to type in too many bytes. When the end user attempts to navigate out of the current validation unit, one of two things will happen. If the application property `FLAG_USER_VALUE_TOO_LONG` is set, an error will be flagged and the excess characters on the right will be selected. If this application property is not set, the excess characters will be quietly truncated, and the navigation will succeed.

If a character item has a format mask, then the item's UI limit is derived from the format mask. This limit is always a character limit.

For a number item (datatypes `NUMBER`, `INT`, `MONEY`, `RNUMBER`, `RINT`, `RMONEY`), the `Maximum Length` property specifies a user interface limit. This limit is always a character limit. There is no data limit. The data item can always internally hold up to 23 bytes (which is the maximum size of an Oracle number in native format), regardless of the value of the `Maximum Length` property.

For a date or time item (datatypes `DATETIME`, `DATE`, `TIME`, `JDATE`, `EDATE`), the `Maximum Length` property specified in Forms Builder has almost no effect (See the Query Length property for a discussion of the only effect). As with "number" items, there is no data limit. For `DATE` and `DATETIME` items, the internal value is `7` bytes long, and for `TIME`, `JDATE` and `EDATE` items, the internal value is `4` bytes long . The user interface limit is derived from the format mask. This limit is always a character limit which is what is returned by `GET_ITEM_PROPERTY(item, MAX_LENGTH)`.

Applies to: all items except buttons, image items, and chart items.

Set At Runtime: No

Get At Runtime: Yes, as `MAX_LENGTH`.

Required: No

Values

A number from `0` to `32767`

Default

30

Usage Notes

- The limits established by the `Maximum Length` property do not apply to "enter-query mode" items (items in the navigationally current block, when the form is in enter-query mode). The `Query Length` property is used instead.
- For List Items of type Combobox, the labels will be truncated if the label length exceeds the Max Length of the Item

---

Related topic

Query Length Property

Maximum Objects Per Line Property

Description

Specifies the maximum number of objects that can appear on each line within a frame.

When the Maximum Number of Frame Objects is set to 0 (the default), there is no maximum--Forms Developer arranges the maximum number of objects per line within a frame.

This property is valid when the Frame Style property is set to Form and the Vertical Fill property is set to No.

Applies to frame

Set Forms Developer

Default

0

Required/Optional required

Maximum Query Time Property

Description

Provides the option to abort a query when the elapsed time of the query exceeds the value of this property.

Applies to form, block

Set Forms Developer, Programmatically

Refer to Built-in

GET_FORM_PROPERTY

GET_BLOCK_PROPERTY

Required/Optional optional

Usage Notes

This property is only useful when the Query All Records property is set to Yes.

Maximum Records Fetched Property

Description

Specifies the number of records fetched when running a query before the query is aborted.

Applies to form, block

Set Forms Developer, Programmatically

Refer to Built-in

GET_FORM_PROPERTY

GET_BLOCK_PROPERTY

Required/Optional optional

Usage Notes

- Maximum Records Fetched is only useful when the properties Query Allowed and Query All Records are set to Yes. Set the Maximum Records Fetched property to limit the records returned by a user's query.

Menu Directory Property

Description

Specifies the directory in which Forms Developer should look for the .MMX runtime menu file. This property is applicable only when you want Forms Developer 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, Forms Developer queries the menu module definition stored in the database to find out the directory and filename of the menu .MMX runfile. The Menu Directory and Menu Filename menu module properties specify the path where Forms Developer should look for the .MMX menu file.

Applies to menu module

Set Forms Developer

Default

blank

Required/Optional optional

Usage Notes

If you leave the directory path unspecified, Forms Developer first searches the default directory for the file, then searches any predefined search paths.

Menu Directory Restrictions

Not valid when using direct reference to specify the location of the menu .MMX runfile.

Menu Filename Property

Description

Specifies the name of the .MMX runtime menu file that Forms Developer should look for at form startup. This property is applicable only when you want Forms Developer 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, Forms Developer queries the menu module definition stored in the database to find out the directory and filename of the menu .MMX runfile. The Menu Directory and Menu Filename menu module properties specify the path where Forms Developer should look for the .MMX menu file.

Applies to menu module

Set Forms Developer

Default

Module Name property

Required/Optional required

Usage Notes

If you leave the directory unspecified, Forms Developer first searches the default directory for the file, then searches any predefined search paths.

Menu Filename Restrictions

- The .MMX file extension is not required.

Related topics

Creating an archive directory

Menu Item Code Property

Description

Contains the PL/SQL commands for a menu item.

Applies to menu items

Set Forms Developer

Required/Optional required

Usage Notes

Clicking the More… button opens the PL/SQL Editor for the menu item, allowing you to edit the PL/SQL commands.

Menu Item Radio Group Property

Description

Specifies the name of the radio group to which the current radio menu item belongs.

Applies to menu item

Set Forms Developer

Required/Optional required

Usage Notes

Specify the same Radio Group for all radio items that belong to the same logical set.

Menu Item Radio Group Restrictions

- Radio items must be adjacent to each other on the same menu.
- Only one radio group per individual menu is allowed.

Menu Item Type Property

Description

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 Forms Developer

Default

Plain

Usage Notes

- The following menu item types are available:
  - Plain
    Default. Standard text menu item.

  - Check
    Indicates a Boolean menu item that is either Yes or No, checked or unchecked.

    Whenever the end user selects a Check menu item Forms Developer 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 Forms Developer, 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 end user selects a Radio menu item, Forms Developer 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 Forms Developer. 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.

Menu Item Type Restrictions

The top-level menu should only have plain or magic menu items.

Menu Module Property

Description

Specifies the name of the menu to use with this form. When this property is set to Default, Forms Developer runs the form with the default menu that is built in to every form. When left NULL, Forms Developer runs the form without any menu.

Applies to form module

Set Forms Developer

Default

Default, indicating that Forms Developer should run the form with the default form menu.

Required/Optional optional

Menu Style Property

Description

Specifies the menu display style Forms Developer should use to run the custom menu specified by the Menu Module property. The only valid option is pull-down.

Applies to form module

Set Forms Developer

Default

Pull-down

Required/Optional optional

Menu Style Restrictions

Not valid when the Menu Module property is set to DEFAULT. (The default menu runs only in pull-down display style.)

Message Property

Description

Specifies the message that is to be displayed in an alert.

Applies to alert

Set Forms Developer, programmatically

Refer to Built-in

SET_ALERT_PROPERTY

Required/Optional optional

Message 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.

Minimize Allowed Property

Description

Specifies that a window can be iconified on window managers that support this feature.

Applies to window

Set Forms Developer

Default

Yes

Required/Optional optional

Minimize Allowed Restrictions

Cannot be set for a root window: a root window is always iconifiable.

Minimized Title Property

Description

Specifies the text string that should appear below an iconified window.

Applies to window

Set Forms Developer

Default

NULL

Required/Optional optional

Minimized Title Restrictions

Only applicable when the Minimize Allowed property is set to Yes.

Modal Property

Description

Specifies whether a window is to be modal. Modal windows require an end user to dismiss the window before any other user interaction can continue.

Applies to window

Set Forms Developer

Default

No

Modal Restrictions

- When Modal is set to Yes, the following window properties are ignored:
  - Close Allowed
  - Icon Filename
  - Inherit Menu
  - Maximize Allowed
  - Minimize Allowed
  - Minimized Title
  - Move Allowed
  - Resize Allowed
  - Show Vertical/Horizontal Scroll Bar

Module Roles Property

Description

Specifies which database roles are available for items in this menu module.

Applies to menu module

Set Forms Developer

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.

Related topic

Item Roles property

Use Security property

Module NLS Lang Property

Description

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)

Applies to form

Set Cannot set within Forms Developer. Set at your operating system level.

Refer to Built-in

GET_FORM_PROPERTY

Default

Default is usually "America_American.WE8ISO8859P1," but all the defaults can be port-specific.

Mouse Navigate Property

Description

Specifies whether Forms Developer should perform navigation to the item when the end user activates the item with a mouse.

Applies to button, check box, list item, radio group

Set Forms Developer

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

Yes

Usage Notes

When Mouse Navigate is No, Forms Developer does not perform navigation to the item when the end user activates it with the mouse. For example, a mouse click in a button or check box is *not* treated as a navigational event. Forms Developer 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 Yes, Forms Developer navigates to the item, firing any appropriate navigation and validation triggers on the way.

Mouse Navigate Restrictions

Applies only in mouse-driven environments.

Mouse Navigation Limit Property

Description

Determines how far outside the current item an end user can navigate with the mouse. Mouse Navigation Limit can be set to the following settings:

Form (The default.) Allows end users to navigate to any item in the current form.

Block Allows end users to navigate only to items that are within the current block.

Record Allows end users to navigate only to items that are within the current record.

Item Prevents end users from navigating out of the current item. This setting prevents end users from navigating with the mouse at all.

Applies to form

Set Forms Developer

Default

Form

Move Allowed Property

Description

Specifies whether or not the window can be moved .

Windows can be moved from one location to another on the screen by the end user or programmatically by way of the appropriate built-in subprogram.

Applies to window

Set Forms Developer

Default

Yes

Required/Optional optional

Usage Notes

In general, it is recommended that windows always be movable.

Move Allowed Restrictions

Cannot be set to NO for a window with the name of ROOT_WINDOW. Such a window is always movable.

Multi-Line Property

Description

Determines whether the text item is a single-line or multi-line editing region.

Applies to text item

Set Forms Developer

Refer to Built-in

GET_ITEM_PROPERTY

Default

No

Usage Notes

Setting the Multi-line property Yes 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.

Multi-Line Restrictions

Valid only for text items with data type CHAR, ALPHA, or LONG.

Multi-Selection Property

Description

Indicates whether multiple nodes may be selected at one time. If set to FALSE, attempting to select a second node will deselect the first node, leaving only the second node selected.

Applies to hierarchical tree

Set Forms Developer

Default

False

Required/Optional required

Name Property

Description

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 Forms Developer

Default

OBJECT_CLASS_N, where `OBJECT_CLASS` is the type of object, and N is the next available number in the document; for example, BLOCK5 or EDITOR3.

Required/Optional required

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.

Name 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

Name Examples

Example

ENAME, ADDRESS1, PHONE_NO1

Navigation Style Property

Description

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 Forms Developer, 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, Forms Developer creates a new record. If there is an open query in the block (the block contains queried records), Forms Developer 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.

Related topics

About making items navigable and enabled

About making items navigable and enabled

NextBlock Property

Description

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.

Related topics

First_Block property

Last_Block property

Next Navigation Block property

Next Detail Relation Property

Description

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.

Related topic

First_Detail_Relation property

Next Item Property

Description

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

Related topics

First_Block property

Last_Item property

Next Navigation Item property

Next Master Relation Property

Description

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.

Related topic

First_Master_Relation property

Next Navigation Block Property

Description

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 Forms Developer, 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 Next Block property.

Related topics

First Navigation Block property

NextBlock property

Previous Navigation Item property

Next Navigation Item Property

Description

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 Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

NULL. NULL indicates the default sequence, which is the name of the item with the next higher sequence number.

Next Navigation Item Restrictions

- The item specified as Next Navigation Item must be in the same block as the current item.

Number of Items Displayed Property

Description

Specifies the number of item instances displayed for the item when the item is in a multi-record block.

Setting Number of Items Displayed > 0 overrides the Number of Records Displayed block property.

Applies to item

Set Forms Developer

Default

Zero. Zero indicates that the item should display the number of instances specified by the Number of Records Displayed block property.

Required/Optional optional

Usage

Use Number of Items Displayed to create a single button, chart, or image as part of a multi-record block. For instance, if Number of 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 Number of Items Displayed to 1.

Number of Items Displayed Restrictions

Number of Items Displayed must be <= Number of Records Displayed block property setting.

Number of Records Buffered Property

Description

Specifies the minimum number of records buffered in memory during a query in the block.

Applies to block

Set Forms Developer

Default

NULL; which indicates the minimum setting allowed (the value set for the Number of Records Displayed property plus a constant of 3).

Required/Optional optional

Usage Notes

- Forms Developer 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 Number of Records Buffered property to increase performance.
- Consider lowering the Number of Records Buffered property if you anticipate retrieving large items, such as image items, because of the amount of memory each item buffered may require.

Number of Records Buffered Restrictions

- If you specify a number lower than the minimum, Forms Developer returns an error when you attempt to accept the value.

Number of Records Displayed Property

Description

Specifies the maximum number of records that the block can display at one time. The default is 1 record. Setting Number of Records Displayed greater than 1 creates a multi-record block.

Applies to block

Set Forms Developer

Refer to Built-in

GET_BLOCK_PROPERTY

Default

1

Required/Optional required

Onetime Where Property

Description

Specifies a one time where clause for the block, overriding any previous ONETIME_WHERE clause. The ONETIME_WHERE clause can include references to global variables, forms parameters, and item values, specified with standard bind variable syntax.

Any WHERE condition specified in the ONETIME_WHERE clause is appended to the query for the next execution only, i.e is a throw-away addition to the WHERE clause. It can be used in place of a Pre-Query trigger to temporarily amend the where clause. It has the advantage over additions to the where clause made in the Pre-Query trigger in that bind variable references (Items, Global
variables and System Variables) can be made in the additional WHERE clause. This behavior premotes cursor reuse and reduces database resource usage.

Applies to form, block

Set Forms Developer, programmatically

Refer to Built-in

GET_BLOCK_PROPERTY

SET_BLOCK_PROPERTY

Required/Optional optional

Usage Notes

1. Enclose in single quotes.
2. The WHERE reserved word is optional.

Onetime Where property examples

Allowing this property to be set for a particular block from any where before a query is executed makes it easier and more user friendly. For example, in a multiblock form, you can set the ONETIME_WHERE clause of a block from a different block followed by an execute-query of the block:

Block1

```
When-button-pressed
Set_Block_Property('Block2', ONETIME_WHERE, '<condition>');

Go_block('Block2');

Execute_Query();
```

In this case you don't have to worry about having a Pre-Query trigger for Block2 and having to set the Set_Block_Property.

For an Emp/Dept form with a default-where clause for the Emp block set to 'empno > 7800' and a push button, do the following:

```
When-Button-Pressed:
Set_Block_Property('emp', ONETIME_WHERE, 'deptno <= :dept.deptno');
```

If a query is performed after pushing the button, the generated query would like the following using the :SYSTEM.LAST_QUERY:

```
Select empno, ename, ... From Emp Where empno> 7800 And depnto <= 10
(in this case the deptno of the DEPT block is 10).
```

Get_Block_Property('emp', DEFAULT_WHERE) would return:

```
empno > 7800
```

Get_Block_Property('emp', ONETIME_WHERE) would return:

```
deptno <= :dept.deptno
```

If a new query is performed, the generated query would be like:

```
Select empno, ename, ... From Emp Where empno> 7800
```

Get_Block_Property('emp', ONETIME_WHERE) would return a NULL string in this case.

Operating System Property

Description

Specifies the name of the current operating system, such as Microsoft WINDOWS, WIN32COMMON, UNIX, Sun OS, MACINTOSH, VMS, and HP-UX.

Applies to application

Set cannot set

Refer to Built-in

GET_APPLICATION_PROPERTY

Related topic

User_Interface property

Optimizer Hint Property

Description

Specifies a hint string that Forms Developer passes on to the RDBMS optimizer when constructing queries. Using the optimizer can improve the performance of database transactions.

Applies to block

Set Designer, programmatically

Refer to Built-in

GET_BLOCK_PROPERTY

SET_BLOCK_PROPERTY

Restrictions:

Valid only for applications running against the Oracle7 Server or Oracle8 Server.

Usage Notes

- Consider a form that contains a block named *DeptBlock* based on the DEPT table. If the end user enters a criteria of " > 25 " for the DEPTNO column and executes the query, the default SELECT statement that Forms Developer 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)
```

Order By Property

Description

See WHERE CLAUSE/ORDER BY CLAUSE.

Other Reports Parameters Property

Description

A <keyword>=<value> list of parameters to include in the running of the report. For a list of valid parameters, see the keyword list in the Reports online help.

Applies to Reports reports

Set Forms Developer

Default

blank

Required/Optional optional

Usage Notes:

When passing multi-word parameter values in the where-clause, the entire where-clause should be enclosed in single quotes. When a name appears in such a multi-word parameter, then two single quotes should also be used to begin and to end that name. For example, in order to pass the parameter value where ename = ’MILLER’ it is necessary to code this as:

‘where ename = ‘‘MILLER’’’

Output Date/Datetime Format Property

Description

Holds the current output date or datetime format mask established by the environment variable FORMS90_OUTPUT_DATE_FORMAT or FORMS90_OUTPUT_DATETIME_FORMAT. Forms uses these format masks as defaults in its runtime output processing. These properties will return values even when these environment variables aren't set.

There are two separate properties: Output_Date_Format and Output_Datetime_Format.

Applies to application

Set Not settable from within Forms Developer.

Refer to Built-in

GET_APPLICATION_PROPERTY

Parameter Data Type Property

Description

Specifies what kinds of values Forms Developer allows as input and how Forms Developer 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 Forms Developer

Usage Notes

- It is recommended that you use only the standard data types CHAR, DATE, LONG, and NUMBER. 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.
- Forms Developer 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 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 9.99x10124, 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 Forms Developer 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.
	Apply a format mask such as 999 to produce a right-justified number.

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

Related topic

Format Mask property

Parameter Initial Value (Form Parameter) Property

Description

Specifies the value that Forms Developer assigns the parameter at form startup.

Applies to Form Parameter

Set Forms Developer

Default

NULL

Required/Optional optional

Usage Notes

Any valid constant is a valid value for this property.

Related topics

$$DATE$$ examples

$$DATETIME$$ examples

$$TIME$$ examples

Password Property

Description

Specifies the password of the current end user.

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.

Related topic

Connect_String property

PL/SQL Date Format Property

Description

This property establishes the format mask used in converting date values when executing PL/SQL (for a trigger or called function or procedure) within Forms, in the following cases:

- evaluating TO_DATE (char_value) or TO_DATE (date_value) with no explicit format mask
- assigning a CHAR value to a date variable, or vice versa.

Applies to entire Forms application (global value)

Set programmatically

Refer to Built-in

GET_APPLICATION_PROPERTY built-in

SET_APPLICATION_PROPERTY built-in

Required/Optional optional. However, it is STRONGLY RECOMMENDED that, for a new application, you set this property to a format mask containing full century and time information. It is also recommended that this format mask be the same as the one specified in the Builtin_Date_Format property.

Default

As noted above, it is strongly recommended that you explicitly set this value for a new application. If you do not, the default value will be DD-MON-YY. (This value is used for compatibility with Release 4.5 and earlier.)

Compatibility with other Oracle products

In Oracle products other than Forms Developer, PL/SQL version 2 does not necessarily use a default date format mask of DD-MON-YY. Instead, it typically uses a format mask derived from the current NLS environment. If for some reason you want your Forms application to exhibit the same behavior, you can use the USER_NLS_DATE_FORMAT application property to get the current NLS date format mask, and then assign it to the application's PLSQL_DATE_FORMAT property.

PL/SQL Library Location Property

Description

Shows the location of the attached PL/SQL library.

This property is set when you attach a PL/SQL library to a Forms module. If you requested that the directory path be retained, this property will be the full pathname of the library. If you requested that the directory path be removed, this property will be just the library name.

This property is displayed only for your information. You cannot set or change it through the property palette.

Applies to PL/SQL libraries

Set Forms Developer

Required/Optional display only.

Default none

Related topic

PL/SQL Library Source property

PL/SQL Library Source Property

Description

This property is set when you attach a PL/SQL library to a Forms module. It shows the source of this PL/SQL library – either File or Database.

This property is displayed only for your information. You cannot set or change it through the property palette.

Applies to PL/SQL libraries

Set Forms Developer

Required/Optional display only

Default File

Related topic

PL/SQL Library Location property

Popup Menu Property

Description

Specifies the popup menu to display for the canvas or item.

Applies to canvases and items

Set not settable

Required/Optional Optional

Default

NULL

Refer to Built-in

GET_MENU_ITEM_PROPERTY

SET__MENU_ITEM_PROPERTY

Note: ENABLED, DISABLED, and LABEL are the only properties valid for popup menus.

Popup Menu Restrictions

- The popup menu must be defined within the current form module.
- You cannot attach a popup menu to individual radio buttons, but you can assign a popup menu to a radio group.

Precompute Summaries Property

Description

Specifies that the value of any summarized item in a data block is computed before the normal query is issued on the block. Forms Developer issues a special query that selects all records (in the database) of the summarized item and performs the summary operation (sum, count, etc.) over all the records.

Applies to block

Set Forms Developer

Required/Optional Required if the block contains summarized items and the block's Query All Records property is set to No.

Default

No

Usage Notes

When an end user executes a query in a block with Precompute Summaries set to Yes, Forms Developer fires the Pre-Query trigger (if any) once before it executes the special query. Forms Developer fires the Pre-Select trigger (if any) twice: once just before executing the special query, and again just before executing the normal query.

Precompute Summaries Restrictions

- You cannot set Precompute Summaries to Yes if any of the following are true: (1) the block contains a summarized control item, (2) a minimize or maximize operation is performed on a summarized item in the block, (3) the block's Query Data Source is a stored procedure or transactional triggers (must be a table or sub-query), or (4) the block contains a checkbox item, list item, or radio group with an empty Other Values property.
- Read consistency cannot be guaranteed unless (1) the form is running against an Oracle7.3 database, and (2) the form-level Isolation Mode property is set to Serializable.

Prevent Masterless Operations Property

Description

Specifies whether end users should be allowed to query or insert records in a block that is a detail block in a master-detail relation. When set to Yes, Forms Developer 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 Yes, Forms Developer displays an appropriate message when end users 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 Forms Developer, programmatically

Refer to Built-in

GET_RELATION_PROPERTY

SET_RELATION_PROPERTY

Default

No

Previous Block Property

Description

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.

Related topic

Previous Navigation Block property

Previous Item Property

Description

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

Related topic

Previous Navigation Item property

Previous Navigation Block Property

Description

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 Forms Developer, programmatically

Refer to Built-in

GET_BLOCK_PROPERTY

SET_BLOCK_PROPERTY

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 Previous Block property.

Related topics

Next Navigation Block property

PreviousBlock property

Previous Navigation Item Property

Description

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 Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

NULL. NULL indicates the default, which is the name of the item with the next lower sequence in the form.

Required/Optional optional

Previous Navigation Item Restrictions

The item specified as Previous Navigation Item must be in the same block as the current item.

Primary Canvas Property

Description

Specifies the canvas that is to be the window's primary content view. At runtime, Form Builder 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, Form Builder displays the window with its primary content view.

However, if 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 Form Builder

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.

Primary Key (Item) Property

Description

Indicates that the item is a base table item in a data block and that it corresponds to a primary key column in the base table. Forms Developer requires values in primary key items to be unique.

Applies to all items except buttons, chart items, and image items

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

No

Required/Optional optional

Primary Key (Item) Restrictions

The Enforce Primary Key block property must be set to Yes for the item's owning block.

Program Unit Text Property

Description

Specifies the PL/SQL code that a program unit contains. When you click on More… in the Property Palette the Program Unit Editor is invoked.

Applies to program unit

Set Forms Developer

Required/Optional required

Prompt Alignment Offset Property

Description

Specifies the prompt's alignment offset.

Applies to item prompt

Set Forms Developer, programmatically (Note: When you use this property in a PL/SQL program, replace the spaces with underscores: Prompt_Alignment_Offset.)

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

blank

Required/Optional optional

Prompt Alignment Property

Description

Specifies how the prompt is aligned along the item's edge, either Start, End, or Center.

Applies to item prompt

Set Forms Developer, programmatically (Note: When you use this property in a PL/SQL program, replace the spaces with underscores: Prompt_Alignment.)

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

Start

Required/Optional required

Prompt Attachment Edge Property

Description

Specifies which edge the prompt should be attached to, either Start, End, Top, or Bottom.

Applies to item prompt

Set Forms Developer, programmatically (Note: When you use this property in a PL/SQL program, replace the spaces with underscores: Prompt_Attachment_Edge.)

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

Start

Required/Optional required

Prompt Attachment Offset Property

Description

Specifies the distance between the item and its prompt.

Applies to item prompt

Set Forms Developer, programmatically.

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

blank

Required/Optional optional

Prompt Background Color Property

Description

The color of the object's or background region.

Applies to item prompt, radio button

Set Forms Developer, programmatically (Note: When you use this property in a PL/SQL program, replace the spaces with underscores: Prompt_Background_Color.)

Default

Unspecified

Required/Optional optional

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

Prompt Display Style Property

Description

Specifies the prompt's display style.

First Record

Forms Developer displays a prompt beside the first record.

Hidden

Forms Developer does not display a prompt.

All Records

Forms Developer displays a prompt beside each record.

Applies to item prompt

Set Forms Developer, programmatically (Note: When you use this property in a PL/SQL program, replace the spaces with underscores: Prompt_Display_Style.)

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

First Record

Required/Optional required

Prompt Fill Pattern Property

Description

Specifies 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.

Applies to item, item prompt's, radio button

Set Forms Developer, programmatically (Note: When you use this property in a PL/SQL program, replace the spaces with underscores: Prompt_Fill_Pattern.)

Default

Unspecified

Required/Optional optional

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

Prompt Font Name Property

Description

Specifies the font family, or typeface, to be used for text in the object. The list of fonts available is system-dependent.

Applies to item item prompt, radio button

Set Forms Developer, programmatically (Note: When you use this property in a PL/SQL program, replace the spaces with underscores: Prompt_Font_Name.)

Default

Unspecified

Required/Optional optional

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

Prompt Font Size Property

Description

The size of the font, specified in points.

Applies to item, item prompt, radio button

Set Forms Developer, programmatically (Note: When you use this property in a PL/SQL program, replace the spaces with underscores: Prompt_Font_Size.)

Default

Unspecified

Required/Optional optional

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

Prompt Font Spacing Property

Description

Specifies the width of the font (i.e., the amount of space between characters, or kerning).

Applies to item, item prompt, radio button

Set Forms Developer, programmatically (Note: When you use this property in a PL/SQL program, replace the spaces with underscores: Prompt_Font_Spacing.)

Default

Unspecified

Required/Optional optional

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

Prompt Font Style Property

Description

Specifies the style of the font.

Applies to item, item prompt, radio button

Set Forms Developer, programmatically (Note: When you use this property in a PL/SQL program, replace the spaces with underscores: Prompt_Font_Style.)

Default

Unspecified

Required/Optional optional

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

Prompt Font Weight Property

Description

Specifies the weight of the font.

Applies to item, item prompt, radio button

Set Forms Developer, programmatically (Note: When you use this property in a PL/SQL program, replace the spaces with underscores: Prompt_Font_Weight.)

Default

Unspecified

Required/Optional optional

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

Prompt Foreground Color Property

Description

Specifies the color of the object’s foreground region. For items, defines the color of the text displayed in the item.

Applies to item, item prompt, radio button

Set Forms Developer, programmatically (Note: When you use this property in a PL/SQL program, replace the spaces with underscores: Prompt_Foreground_Color.)

Default

Unspecified

Required/Optional optional

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

Prompt Justification Property

Description

Specifies justification of the prompt as either Left, Right, Center, Start, or End.

Applies to item prompt

Set Forms Developer

Default

Start

Required/Optional required

Prompt Property

Description

Specifies the text label that displays for an item.

Applies to item prompt

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

blank

Required/Optional optional

Prompt Reading Order Property

Description

Specifies the prompt's reading order, either Default, Left to Right, or Right to Left.

Applies to item prompt

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

Default

Required/Optional required

Prompt Visual Attribute Group Property

Description

Specifies the named visual attribute that should be applied to the prompt at runtime.

Applies to item prompt

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

Default

Required/Optional required

Property Class Property

Description

Specifies the name of the property class from which the object can inherit property settings.

Applies to all objects

Set Forms Developer

Default

Null

Query All Records Property

Description

Specifies whether all the records matching the query criteria should be fetched into the data block when a query is executed.

Yes - Fetches all records from query; equivalent to executing the EXECUTE_QUERY (ALL_RECORDS) built-in.

No - Fetches the number of records specified by the Query Array Size block property.

Applies to block

Set Forms Developer

Refer to Built-in

GET_BLOCK_PROPERTY

SET_BLOCK_PROPERTY

Default

No

Required/Optional Required if a data block contains summarized items, and the block's Precompute Summaries property is set to No.

Related topics

Precompute Summaries property

DML Array Size property

EXECUTE_QUERY built-in

Query Array Size property

Query Allowed (Block) Property

Description

Specifies whether Forms Developer should allow the end user or the application to execute a query in the block. When Query Allowed is No, Forms Developer displays the following message if the end user attempts to query the block:

FRM-40360: Cannot query records here.

Applies to block

Set Forms Developer, programmatically

Refer to Built-in

GET_BLOCK_PROPERTY

SET_BLOCK_PROPERTY

Default

Yes

Restrictions:

When the Query Allowed block property is Yes, the Query Allowed item property must be set to Yes for at least one item in the block.

Query Allowed (Item) Property

Description

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 Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

Yes; however if the item is part of the foreign key in the detail block of a master-detail block relation, Forms Developer sets this property to No.

Usage Notes

To set the Query Allowed (Item) property programmatically, use the constant QUERYABLE.

Query Allowed (Item) Restrictions

- The Visible property must also be set to Yes.
- Items with the data type LONG cannot be directly queried.

Query Array Size Property

Description

Specifies the maximum number of records that Forms Developer should fetch from the database at one time.

Applies to block

Set Forms Developer

Refer to Built-in

GET_BLOCK_PROPERTY

Default

The number of records the block can display, as indicated by the Number of Records Displayed block property.

Required/Optional required

Usage Notes

- A size of 1 provides the fastest perceived response time, because Forms Developer 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.

Query Array Size Restrictions

- There is no maximum.

Query Data Source Arguments Property

Description

Specifies the names, datatypes, and values of the arguments that are to be passed to the procedure for querying data. The Query Procedure Arguments property is valid only when the Query Data Source Type property is set to Procedure.

Applies to block

Set Forms Developer

Default

NULL

Required/Optional optional

Related topics

Query Data Source Columns property

Query Data Source Name property

Query Data Source Type property

Query Data Source Columns Property

Description

Specifies the names and datatypes of the columns associated with the block's query data source. The Query Data Source Columns property is valid only when the Query Data Source Type property is set to Table, Sub-query, or Procedure.

Applies to block

Set Forms Developer

Default

NULL

Required/Optional optional

Related topics

Query Data Source Arguments property

Query Data Source Name property

Query Data Source Type property

Query Data Source Name Property

Description

Specifies the name of the block's query data source.

The Query Data Source Name property is valid only when the Query Data Source Type property is set to Table, Sub-Query, or Procedure.

Applies to block

Set Forms Developer, programmatically

Refer to Built-in

GET_BLOCK_PROPERTY

SET_ITEM_PROPERTY

Default

NULL

Required/Optional optional

Query Data Source Name Restrictions

- Prior to setting the Query Data Source Name property you must perform a COMMIT_FORM or a CLEAR_FORM.

Query Data Source Type Property

Description

Specifies the query data source type for the block. A query data source type can be a Table, Procedure, Transactional Trigger, FROM clause query, or None.

Applies to block

Set Forms Developer, programmatically

Refer to Built-in

GET_BLOCK_PROPERTY

Default

Table

Required/Optional required

Related topics

Query Data Source Arguments property

Query Data Source Columns property

Query Data Source Name property

Query Hits Property

Description

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

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 Forms Developer of the number of records that a query will return. This allows you to implement count query processing equivalent to Forms Developer 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.

Query_Hits Restrictions

Set this property greater than or equal to 0.

Query Length Property

Description

Specifies the maximum length value that can be stored in an item during Enter Query mode.

Get At Runtime: Yes, as `QUERY_LENGTH`.

Set At Runtime: No

Required: No

Valid Values

Any integer between 0 to 32767

Default

0

Query Length Restrictions

- The maximum `QUERY_LENGTH` is 255 characters.

Usage Notes

- The `QUERY_LENGTH` property has essentially the same semantics as the `MAXIMUM_LENGTH` property. The primary difference between the two is that `QUERY_LENGTH` applies to "enter-query mode" items (items in the navigationally current block, when the form is in enter-query mode), and `MAXIMUM_LENGTH` applies in all other cases.
- Another difference is that for date/time items, the enter-query mode UI limit is computed by taking the non-enter-query mode UI limit (which is computed from the format mask), and adding the delta between the item’s `QUERY_LENGTH` and `MAXIMUM_LENGTH` properties. This computed UI limit is what is returned by `GET_ITEM_PROPERTY`(item, `QUERY_LENGTH`).

- A value of zero indicates that the effective `QUERY_LENGTH` is always the same as the effective `MAXIMUM_LENGTH`; it indicates that `MAXIMUM_LENGTH` applies to enter-query mode items as well as non-enter-query mode items. A non-zero value must always be greater than or equal to the `MAXIMUM_LENGTH`.

Related topic

Maximimum Length Property

Query Name Property

Description

Specifies the name of the query in the report with which to associate the forms block.

Applies to report integration

Set Forms Developer

Default

blank

Required/Optional optional

Query Only Property

Description

Specifies that an item can be queried but that it should not be included in any INSERT or UPDATE statement that Forms Developer issues for the block at runtime.

Applies to check box, radio group, list item, image item, text item

Set programmatically

Default

No

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Related topics

Query Allowed (Block) property

Query Allowed (Item) property

Query Options Property

Description

Specifies the type of query operation Forms Developer 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_BLOCK_PROPERTY

Radio Button Value Property

Description

Specifies the value associated with a radio button item in a radio group.

Applies to radio button

Set Forms Developer

Default

blank

Raise on Entry Property

Description

For a canvas that is displayed in the same window with one or more other canvases, Raise on Entry specifies how Forms Developer should display the canvas when the end user or the application navigates to an item on the canvas.

- When Raise on Entry is No, Forms Developer 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 Yes, Forms Developer *always* raises the view to the front of the window when the end user or the application navigates to *any* item on the view.

Applies to canvas

Set Forms Developer

Default

No

Raise on Entry Restrictions

Applicable only when more than one canvas is assigned to the same window.

Reading Order Property

Description

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:

Value	**Description**
Default	Text item inherits the reading order specified by its canvas Language Direction property setting.
Right-To-Left	Item reading order is right-to-left.
Left-To-Right	Item reading order is left-to-right.

Applies to display item, text item

Set Forms Developer

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

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 Language 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.

Related topics

Bidirectional support in Forms Developer

Defining a prompt's alignment offset

Real Unit Property

Description

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, points, or decipoints. (A point is 1/72nd of an inch.)

Forms Developer 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 Forms Developer

Default

Centimeter

Required/Optional optional

Real Unit Restrictions

Valid only when the coordinate system property is set to Real.

Record Group Property

Description

Specifies the name of the record group from which the LOV or hierarchical tree derives its values.

Applies to:

LOV, hierarchical tree

Set Forms Developer, programmatically

Refer to Built-in

GET_LOV_PROPERTY (GROUP_NAME)

SET_LOV_PROPERTY (GROUP_NAME)

POPULATE_TREE

POPULATE_GROUP_FROM_TREE

Default

Null

Required/Optional Required for LOV, Optional for hierarchical tree

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).

Related topics

Record Group Query property

Record Group Type property

Record Group Query Property

Description

Specifies the SELECT statement for query associated with the record group.

Applies to record group

Set Forms Developer, programmatically

Refer to Built-in

POPULATE_GROUP_WITH_QUERY

Required/Optional optional

Record Group Type Property

Description

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 Forms Developer

Default

Query

Record Orientation Property

Description

Determines the orientation of records in the block, either horizontal records or vertical records. When you set this property, Forms Developer adjusts the display position of items in the block accordingly.

Applies to block

Set Forms Developer

Default

Vertical records

Required/Optional optional

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.

Record Orientation Restrictions

Valid only for a multi-record block (Number of Records Displayed property set greater than 1).

Records To Fetch Property

Description

Returns the number of records Forms Developer 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 Forms Developer 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 Query Array Size 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, Forms Developer will fire the On-Fetch trigger again. Records_To_Fetch will be the 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).

Records To Fetch Examples

Example

```
/*

** Call a client-side package function to retrieve
** the proper number of rows from a package cursor.
*/
DECLARE
j NUMBER := Get_Block_Property(blk_name, 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;
```

Relation Type Property

Description

Specifies whether the link between the master block and detail block is a relational join or an object REF pointer.

Applies to master-detail relations

Set Forms Developer

Default

Join

Usage Notes

- Valid values are Join (indicating a relational join) or REF (indicating a REF column in one block pointing to referenced data in the other block).
- When the link is via a REF, see also the Detail Reference property.
- When the link is via a join, see also the Join Condition property.

Rendered Property

Description

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 Forms Developer

Default

Yes

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.

Report Destination Format Property

Description

In bit-mapped environments, this property specifies the printer driver to be used when the Report Destination Type property is File.

Possible valus are any valid destination format not to exceed 1K in length. Examples of valid values for this keyword are hpl, hplwide, dec, decwide, decland, dec180, dflt, wide, etc. Ask your System Administrator for a list of valid destination formats. In addition, Reports supports the following destination formats:

PDF	Means that the report output will be sent to a file that can be read by a PDF viewer. PDF output is based upon the currently configured printer for your system. The drivers for the currently selected printer is used to produce the output; you must have a printer configured for the machine on which you are running the report.
HTML	Means that the report output will be sent to a file that can be read by an HTML 3.0 compliant browser (e.g., Netscape 2.2).
HTMLCSS	Means that the report output sent to a file will include style sheet extensions that can be read by an HTML 3.0 compliant browser that supports cascading style sheets.
HTMLCSSIE	Means that the report output sent to a file will include style sheet extensions that can be read by Microsoft Internet Explorer 3.x.
RTF	Means that the report output will be sent to a file that can be read by standard word processors (such as Microsoft Word). Note that when you open the file in MS Word, you must choose **View | Page Layout** to view all the graphics and objects in your report.
DELIMITED	Means that the report output will be sent to a file that can be read by standard spreadsheet utilities, such as Microsoft Excel. Note that you must also specify a DELIMITER.

For more information about this property, see DESFORMAT under the index category Command Line Arguments in the Reports online help.

Applies to report reports

Set Forms Developer

Default blank

Required/Optional optional

Report Destination Name Property

Description

Name of the file, printer, Interoffice directory, or email user ID (or distribution list) to which the report output will be sent. Possible values are any of the following not to exceed 1K in length:

- a filename (if Report Destination Type property is File or Localfile)
- a printer name (if Report Destination Type property is Printer)
- email name or distribution name list (if Report Destination Type property is Mail).

To send the report output via email, specify the email ID as you do in your email application (any MAPI-compliant application on Windows, such as Oracle InterOffice, or your native mail application on UNIX). You can specify multiple usernames by enclosing the names in parentheses and separating them by commas (e.g., (name, name, . . .name)). For printer names, you can optionally specify a port. For example:

printer,LPT1:

printer,FILE:

Or, if the Report Destination Type property is Interoffice:

/FOLDERS/directory/reportname

For more information about this property, see DESNAME under the index category Command Line Arguments in the Reports online help.

Applies to report reports

Set Forms Developer

Default

blank

Required/Optional optional

Report Destination Type Property

Description

Destination to which you want the output to be sent. Possible values are Screen, File, Printer, Preview, Mail, and Interoffice. For more information about this property, see DESTYPE under the index category Command Line Arguments in the Reports online help.

SCREEN	Screen routes the output to the Previewer for interactive viewing. This value is valid only for when running the report in Runtime mode (not Batch). Font aliasing is not performed.
FILE	File saves the output to a file named in Report Destination Name.
PRINTER	Printer routes the output to the printer named in Report Destination Name.
PREVIEW	Preview routes the output to the Previewer for interactive viewing. However, Preview causes the output to be formatted as Postscript output. The Previewer will use the Report Destination Name property to determine which printer's fonts to use to display the output. Font aliasing is performed for Preview.
MAIL	Mail routes the output to the mail users specified in Report Destination Name. You can send mail to any mail system that is MAPI-compliant or has the service provider driver installed. The report is sent as an attached file.
INTEROFFICE	routes the output to the Oracle InterOffice mail users specified in Report Destination Name. By specifying this value, you store the report output in InterOffice as a repository.

Applies to report reports

Set Forms Developer

Default

File

Required/Optional required

Report Server Property

Description

Specifies the Report Server against which you can run your Report.

Applies to report reports

Set Forms Developer

Required/Optional optional

Default

blank

Required (Item) Property

Description

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 Forms Developer, programmatically

Refer to Built-in

GET_ITEM_INSTANCE_PROPERTY

GET_ITEM_PROPERTY

SET_ITEM_INSTANCE_PROPERTY

SET_ITEM_PROPERTY

Default

No

Usage Notes

- When an item has Required set to Yes, and item-level validation is in effect, by default Forms Developer will not allow navigation out of the item until a valid value is entered. To allow the end user to move freely among the items in the record, set the Defer Required Enforcement property to Yes. This will postpone enforcement of the Required attribute from item validation to record validation.
- Even when Required is set to Yes, there are circumstances when an item's value could be NULL. Forms Developer 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.
- Setting a poplist's or T-list's Required property may affect the values the list will display:

When selected, an instance of a poplist will display an extra null value if its current value is NULL or if its effective Required property is No (false). When selecting the current value of an instance of a T-list, it will be unselected (leaving the T-list with no selected value) if its effective Required property is No (false). But if its effective Required property is Yes (true), selecting a T-list instance’s current value will have no effect. The value will stay selected.

Related topic

Defer Required Enforcement property

Resize Allowed Property

Description

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 Forms Developer

Default

Yes

Usage Notes

The Resize Allowed property prevents an end user from resizing the window, but it does not prevent you from resizing the window programmatically with RESIZE_WINDOW or SET_WINDOW_PROPERTY.

Resize Allowed Restrictions

- Resize Allowed is only valid when the Maximize Allowed property is set to No.

Return Item (LOV) Property

Description

See Column Mapping Properties .

Rotation Angle Property

Description

Specifies the graphic object's rotation angle. The angle at which the object is initially created is considered to be 0, and this property is the number of degrees clockwise the object currently differs from that initial angle. You can rotate an object to an absolute angle by setting this property.

Applies to graphics physical

Set Forms Developer

Default 0

Required/Optional required

Runtime Compatibility Mode Property

Description

This property setting is ignored starting with Oracle9i Forms Developer. Forms 5.0 behavior is used in all cases.

Applies to forms compatibility

Set Forms Developer

Default

5.0. All other values are ignored.

Required/Optional required

Savepoint Mode Property

Description

Specifies whether Forms Developer 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:

Setting	Description
Yes (the default)	Specifies that Forms Developer should issue a savepoint at form startup and at the start of each Post and Commit process.
No	Specifies that Forms Developer is to issue no savepoints, and that no rollbacks to savepoints are to be performed.

Applies to form module

Set Forms Developer, programmatically

Refer to Built-in

GET_FORM_PROPERTY

SET_FORM_PROPERTY

Default

Yes

Required/Optional optional

Savepoint Mode Restrictions

- When Savepoint Mode is No, Forms Developer does not allow a form that has uncommitted changes to invoke another form with the CALL_FORM procedure.

Savepoint Name Property

Description

Specifies the name of the savepoint Forms Developer 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 Forms Developer should roll back by way of a call to ISSUE_ROLLBACK. A NULL savepoint name implies that a full rollback is expected.

Scroll Bar Alignment Property

Description

Specifies whether the scroll bar is displayed at the start or the end of the frame.

Applies to frame

Set Forms Developer

Default

End

Required/Optional optional

Scroll Bar Height Property

Description

Specifies the height of the scroll bar.

Applies to scroll bar

Set Forms Developer

Required/Optional optional

Scroll Bar Width Property

Description

Specifies the width of the scroll bar.

Applies to scroll bar

Set Forms Developer

Required/Optional optional

Share Library with Form Property

Description

Forms that have identical libraries attached can share library package data. (For more information, see the *data_mode* parameter for the CALL_FORM, OPEN_FORM, and NEW_FORM built-ins, under Related Topics above.) The Share Library with Form property enables menus associated with the forms to share the library package data as well.

Applies to menus

Set Forms Developer

Default

Yes

Usage Notes

1 If two forms share an object, and both forms are open at design time and you make changes to the object in Form A, those changes will not be seen in Form B until the changes are first saved by Form A, and Form B is then closed and reopened.

2 If you use OPEN_FORM to open a form in a different database session, you cannot share library data with the form or its associated menus. Attempts to share library data by setting the property to Yes will be ignored.

Related topics

NEW_FORM built-in

OPEN_FORM built-in

CALL_FORM built-in

Show Horizontal Scroll Bar Property

Description

Determines whether a canvas, secondary window, or image item is displayed with a scroll bar.

Applies to canvas, window, editor, image item

Set Forms Developer

Default

No

Required/Optional optional

Show Horizontal Scroll Bar Restrictions

- For a window, only valid when the Modal property is set to No.
- Valid on window managers that support horizontal scroll bars.

Show Lines Property

Description

Determines whether a hierarchical tree displays lines leading up to each node.

Applies to hierarchical tree

Set Forms Developer

Default

True

Required/Optional required

Show Palette Property

Description

Determines whether Forms Developer will display an image-manipulation palette adjacent to the associated image item at runtime. The palette provides three tools that enable end users to manipulate a displayed image:

- **Zoom**—click the tool, then repeatedly click the image to incrementally reduce the amount of the source image displayed within the image item's borders.
- **Pan**—click the tool and use the grab hand to pan unseen portions of the source image into view (valid only if the source image extends beyond at least one border of the image item).
- **Rotate**—click the tool, then repeatedly click the image to rotate it clockwise in 90-degree increments.

Applies to image item

Set Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Default

No

Required/Optional Required

Show Scroll Bar Property

Description

The Show Scroll Bar option specifies whether Forms Developer should create a block scroll bar for the block you are defining. When Show Scroll Bar is set to Yes, Forms Developer creates the scroll bar on the canvas 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 Group.

Applies to block

Set Forms Developer

Default:

No

Required/Optional optional

Usage Notes

Setting Reverse Direction to Yes causes Forms Developer to fetch the next set of records when the end user scrolls upward. If the end user scrolls downward, Forms Developer displays already fetched records.

Property	**Description**
Scroll Bar Canvas	Specifies the canvas on which the block's scroll bar should be displayed. The specified canvas must exist in the form.
Scroll Bar Orientation	Specifies whether the block scroll bar should be displayed horizontally or vertically.
Scroll Bar X Position	Specifies the x position of a block scroll bar measured at the upper left corner of the scrollbar. The default value is 0.
Scroll Bar Y Position	Specifies the width of a block scroll bar measured at the upper left corner of the scrollbar. The default value is 0.
Scroll Bar Width	Specifies the width of a block scroll bar. The default value is 2.
Scroll Bar Height	Specifies the height of a block scroll bar. The default value is 10.
Reverse Direction	Specifies that the scroll bar scrolls in reverse. The default value is No.

Visual Attribute Group	Specifies the font, color, and pattern attributes to use for scroll bar. Refer to the Visual Attribute Group property for more information. The default setting is determined by the platform and resource file definition.

Related topic

Visual Attribute Group property

Show Symbols Property

Description

Indicates whether a hierarchical tree should display + or - symbols in front of each branch node. The + symbol indicates that the node has children but is not expanded. The - symbol indicates that the node is expanded.

Applies to hierarchical tree

Set Forms Developer

Default

True

Required/Optional required

Show Vertical Scroll Bar Property

Description

Specifies that a vertical scroll bar is to appear on the side of a canvas or window.

Applies to canvas, window, image item, editor, item

Set Forms Developer

Default

No

Required/Optional Optional

Show Vertical Scroll Bar Restrictions

- Valid on window managers that support vertical scroll bars.
- 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 YES

Shrinkwrap Property

Description

Specifies whether blank space should be automatically removed from the frame. When Shrinkwrap is set to Yes, Forms Developer automatically reduces, or "shrinkwraps", the frame around the items within the frame.

Note: Reisizing a frame has no effect when Shrinkwrap is set to Yes; when you, for example, increase the size of the frame, Forms Developer automatically reduces the frame to its shrinwrap size. If you want to resize a frame, set Shrinkwrap to No.

Applies to frame

Set Forms Developer

Default

Yes

Required/Optional Optional

Single Object Alignment Property

Description

Specifies the alignment for single line objects when the Frame Alignment property is set to Fill.

Applies to frame

Set Forms Developer

Default

Start

Required/Optional Required

Single Record Property

Description

Specifies that the control block always should *contain* one record. Note that this differs from the number of records *displayed* in a block.

Applies to block

Set Forms Developer

Default

No

Usage Notes

- Set Single Record to Yes for a control block that contains a summary calculated item. Conversely, Single Record must be set to No for the block that contains the item whose values are being summarized and displayed in the calculated item.
- You cannot set Single Record to Yes for a data block.

Related topic

Query All Records property

Size Property

Description

Specifies the width and height of the canvas in the current form coordinate units specified by the Coordinate System form property.

Applies to canvas

Set Forms Developer, programmatically

Refer to Built-in

GET_CANVAS_PROPERTY

SET_CANVAS_PROPERTY

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 Forms Developer, programmatically

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

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 Forms Developer, programmatically

Refer to Built-in

SHOW_EDITOR

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.

Size (LOV)

- Specifies the width and height of the LOV in the current form coordinate units specified by the Coordinate System form property.
- 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 Forms Developer, programmatically

Refer to Built-in

GET_LOV_PROPERTY

SET_LOV_PROPERTY

Restrictions

Forms Developer 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 Forms Developer, programmatically

Default

80 characters by 24 characters

Refer to Built-in

GET_WINDOW_PROPERTY

SET_WINDOW_PROPERTY

Size Restrictions

- Forms Developer 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.)

Sizing Style Property

Description

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 Forms Developer

Default

Crop

SSO USERID Property

Description

Returns a string containing the Single Sign On user ID if the user has been authenticated via the Login Server; NULL otherwise.

Syntax

get_application_property(sso_userid)

Set not settable

Refer to Built-in

GET_APPLICATION_PROPERTY

Start Angle Property

Description

Specifies the starting angle of the arc, using the horizontal axis as an origin.

Applies to graphic arc

Set Forms Developer

Default

90

Required/Optional required

Start Prompt Alignment Property

Description

Specifies how the prompt is aligned to the item's horizontal edge, either Start, Center, or End. This property is valid when the Layout Style property is set to Form.

Applies to frame

Set Forms Developer

Default

Start

Required/Optional required

Start Prompt Offset Property

Description

Specifies the distance between the prompt and its item when the Start Prompt Alignment property is set to Start.

Applies to frame

Set Forms Developer

Default

0 (character cell)

Required/Optional required

Startup Code Property

Description

Specifies optional PL/SQL code that Forms Developer 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 Forms Developer

Required/Optional optional

Usage Notes

Startup code does not execute when Forms Developer is returning from a called form.

Status (Block) Property

Description

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.

Status (Record) Property

Description

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.
- In general, any assignment to a database item will change a record's status from QUERY to CHANGED (or from NEW to INSERT), even if the value being assigned is the same as the previous value. Passing an item to a procedure as OUT or IN OUT parameter counts as an assignment to it.

Subclass Information Property

Description

Specifies the following information about the source object and source module for a referenced objects.

Module The name of the source module.

Storage The source module type (Form or Menu) and location (File System or Database)

Name 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.)

Applies to any reference object

Set Forms Developer

Required/Optional optional

Tab Page Property

Description

The name of the tab page on which the item is located.

Applies to item

Set Forms Developer

Default

none

Refer to Built-in

GET_ITEM_PROPERTY (programmatic property name is `Item_Tab_Page`)

Required/Optional required (if the item is located on a tab canvas)

Related topic

Item_Tab_Page property

Tab Page X Offset Property

Description

The distance between the left edge of the tab canvas and the left edge of the tab page. The value returned depends on the Form coordinate system: pixel, centimeter, inch, or point.

Applies to tab canvas

Refer to Built-in

- GET CANVAS PROPERTY

Tab Page X Offset Restrictions

- you can *get* the property value, but you cannot *set* it
- valid only for tab canvas. 0 is returned for all other canvas types

---

Related topic

Tab Page Y Offset property

Tab Page Y Offset Property

Description

Specifies the distance between the top edge of the tab canvas and the top edge of the tab page. The value returned depends on the Form coordinate system used—pixel, centimeter, inch, or point.

Applies to tab canvas

Refer to Built-in

GET CANVAS PROPERTY

Tab Page Y Offset Restrictions

- you can *get* the property value, but you cannot *set* it
- valid only for tab canvas. 0 is returned for all other canvas types

---

Related topic

Tab Page X Offset property

Tab Style Property

Description

Specifies the shape of the labelled tab(s) on a tab canvas.

Applies to tab canvas

Set Forms Developer

Default

Chamfered

Required/Optional required

Title Property

Description

Specifies the title to be displayed for the object.

Applies to alert, form module, LOV, window

Set Forms Developer

Required/Optional optional

Title (LOV)

Default

NULL

Required/Optional optional

Title (Window)

Refer to Built-in

GET WINDOW PROPERTY

SET WINDOW PROPERTY

Required/Optional optional

Usage Notes

- Length limit of a window title depends on the display driver used. (For example, for an SVGA 1280 x 1024, running under NT, the limit is 78 characters.)
- If you do not specify a title for a window that is not a root window, Forms Developer 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, Forms Developer 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, Forms Developer uses the name of the main menu in the module for the root window title, as indicated by the menu module Main property.

Tooltip Background Color Property

Description

Specifies the color of the object’s background region.

Applies to item

Set Programmatically

Default

Unspecified

Refer to Built-in

GET ITEM PROPERTY

SET ITEM PROPERTY

Tooltip Fill Pattern Property

Description

Specifies 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.

Applies to item

Set Programmatically

Default

Unspecified

Refer to Built-in

GET ITEM PROPERTY

SET ITEM PROPERTY

Tooltip Font Name Property

Description

Specifies the font family, or typeface, to be used for text in the object. The list of fonts available is system-dependent.

Applies to item

Set Programmatically

Default

Unspecified

Refer to Built-in

GET ITEM PROPERTY

SET ITEM PROPERTY

Tooltip Font Size Property

Description

Specifes the size of the font in points.

Applies to item

Set Programmatically

Default

Unspecified

Refer to Built-in

GET ITEM PROPERTY

SET ITEM PROPERTY

Tooltip Font Spacing Property

Description

Specifies the width of the font (i.e., the amount of space between characters, or kerning).

Applies to item

Set Programmatically

Default

Unspecified

Refer to Built-in

GET ITEM PROPERTY

SET ITEM PROPERTY

Tooltip Font Style Property

Description

Specifies the style of the font.

Applies to item

Set Programmatically

Default

Unspecified

Refer to Built-in

GET ITEM PROPERTY

SET ITEM PROPERTY

Tooltip Font Weight Property

Description

Specifies the weight of the font.

Applies to item

Set Programmatically

Default

Unspecified

Refer to Built-in

GET ITEM PROPERTY

SET ITEM PROPERTY

Tooltip Foreground Color Property

Description

Specifies the color of the object’s foreground region. For items, defines the color of the text displayed in the item.

Applies to item

Set Programmatically

Default

Unspecified

Refer to Built-in

GET_ITEM_PROPERTY

SET_ITEM_PROPERTY

Tooltip Property

Description

Specifies the help text that should appear in a small box beneath the item when the mouse enters the item.

Applies to item

Set Forms Developer, programmtically

Refer to Built-in

GET ITEM PROPERTY

SET ITEM PROPERTY

Default

blank

Required/Optional optional

Tooltip Visual Attribute Group Property

Description

Specifies the named visual attribute that should be applied to the tooltip at runtime.

Applies to item tooltip

Set Forms Developer

Default

Default

Required/Optional required

Top Prompt Alignment Property

Description

Specifies how the prompt is aligned to the item's top edge, either Start, End, or Center. This property is valid when the Layout Style property is set to Tabular.

Applies to frame

Set Forms Developer

Default

Start

Required/Optional required

Top Prompt Offset Property

Description

Specifies the distance between the prompt and its item when the Top Prompt Alignment property is set to Top.

Applies to frame

Set Forms Developer

Default

0 (character cell)

Required/Optional required

Related topic

Top Prompt Offset Showme

Top Record Property

Description

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.

Top Title Property

Description

Specifies a title of up to 72 characters to appear at the top of the editor window.

Applies to editor

Set Forms Developer

Required/Optional optional

Topmost Tab Page Property

Description

Specifies the top most tab page in a tab canvas.

Applies to tab canvas

Set Forms Developer, programmatically

Refer to Built-in

GET CANVAS PROPERTY

SET CANVAS PROPERTY

Default

First tab page that appears under the Tab Page node.

Topmost Tab Page Restrictions

Valid only for tab canvas.

Transactional Triggers Property

Description

Identifies a block as a *transactional control block*; that is, a non-database block that Forms Developer 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 No.

When you create a non-ORACLE data source application, you are essentially simulating the functionality of a data 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 data block.

For more information, see *Forms Developer Advanced Techniques*, Chapter 4, "Connecting to Non-ORACLE Data Sources."

Applies to block

Set Forms Developer

Default

No

Usage Notes

- Transactional Triggers applies only when the Base Table property is NULL.
- Setting Transactional Triggers to Yes enables the Enforce Primary Key and Enforce Column Security properties.

Trigger Style Property

Description

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 Forms Developer

Default

PL/SQL

Usage Notes

Choosing V2-Style Trigger enables the Zoom button, which opens the Trigger Step property sheet.

Trigger Text Property

Description

Specifies the PL/SQL code that Forms Developer executes when the trigger fires.

Applies to trigger

Set Forms Developer

Required/Optional required

Trigger Type Property

Description

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 Forms Developer

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 Forms Developer and corresponds to a specific, pre-defined runtime event.

User-named Specifies that the trigger is not provided by Forms Developer. A user-named trigger can only be executed by a call to the EXECUTE TRIGGER built-in procedure.

Update Allowed (Block) Property

Description

Determines whether end users can modify the values of items in the block that have the Update Allowed item property set to Yes. (Setting Update Allowed to No for the block overrides the Update Allowed setting of any items in the block.)

Applies to block

Set Forms Developer, programmatically

Refer to Built-in

GET BLOCK PROPERTY

SET BLOCK PROPERTY

Default

Yes

Update Allowed (Block) Restrictions

When the Update Allowed block property is set to Yes, at least one item in the block must have the Update Allowed item property set to Yes for the block to be updateable.

Update Allowed (Item) Property

Description

Specifies whether end users should be allowed to change the value of the base table item in a queried record. When Update Allowed is set to No, end users can navigate to the item in a queried record, but if they attempt to change its value, Forms Developer displays error FRM-40200: Field is protected against update.

Setting Update Allowed to Yes does not prevent end users from entering values in a NEW (INSERT) record.

Applies to all items except buttons, chart items, and image items

Set Forms Developer, programmatically

Refer to Built-in

GET ITEM INSTANCE PROPERTY

GET ITEM PROPERTY

SET ITEM INSTANCE PROPERTY

SET ITEM PROPERTY

Default

Yes

Usage Notes

- To set the Update Allowed (Item) property programmatically, you can use the constant UPDATE ALLOWED or UPDATEABLE. The constant UPDATEABLE is for compatibility with prior releases.
- If Enabled is set to PROPERTY FALSE at runtime, then the items' or item instance's Update Allowed property is also set to PROPERTY FALSE.
- When Update Allowed is specified at multiple levels (item instance, item, and block), the

values are ANDed together. This means that setting Update Allowed to Yes (PROPERTY TRUE for runtime) has no effect at the item instance level unless it is set consistently at the block and item levels. For example, your user cannot update an item instance if Update Allowed is true at the instance level, but not at the item or block levels.

Update Allowed (Item) Restrictions

- If you are using SET ITEM PROPERTY to set UPDATE ALLOWED to true, then you must set item properties as follows:
    - Enabled to Yes (PROPERTY TRUE for runtime)
    - Visible to Yes (PROPERTY TRUE for runtime)
    - Base Table Item to Yes (PROPERTY TRUE for runtime)
    - Update Only If Null to No (PROPERTY FALSE for runtime)

Update Changed Columns Only Property

Description

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 Only is set to No, and all columns are included in the UPDATE statement.

Applies to block

Set Forms Developer, programmatically

Refer to Built-in

GET BLOCK PROPERTY

SET BLOCK PROPERTY

Default

No

Required/Optional optional

Usage Notes

- If the DML Array Size property is set to a value greater than 1, this Update Changed Columns Only property will be ignored at runtime. That is, a DML Array Size greater than 1 causes all columns to be updated – even if Update Changed Columns Only was set to Yes.
- When Update Changed Columns Only is No, Forms Developer can reuse the same SQL statement for multiple updates, without having to reparse each time. Setting Update Changed Columns Only to Yes can degrade performance because the UPDATE statement must be reparsed each time. In general, you should only set Update Changed Columns Only to Yes 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 Only to Yes 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.

Update Column Property

Description

When set to Yes, forces Forms Developer to treat this item as updated.

If the Update Changed Columns Only block property is set to Yes, 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 Only block property is set to Yes, 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 No, 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 Only. However, whether or not Update Changed Columns Only is set to Yes, 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.

Related topics

Update Allowed (Block) property

Update Changed Columns Only property

Update Commit Property

Description

Specifies whether a chart item is updated to reflect changes made by committing new or updated records to its source block.

Applies to chart item

Set Forms Developer

Default

Yes

Required/Optional required

Update Layout Property

Description

Specifies when the frame's layout is updated.

Automatically

> The layout is updated whenever the frame is moved or resized or whenever any frame layout property is modified.

Manually

> The layout is updated whenever the Layout Wizard is used to modify a frame or whenever the user clicks the Update Layout button or menu option.

Locked

> The layout is locked and cannnot be updated.

Applies to frame

Set Forms Developer

Default

Yes

Required/Optional required

Update Only if NULL Property

Description

Indicates that operators can modify the value of the item only when the current value of the item is NULL.

Applies to image items, list items, text items

Set Forms Developer, programmatically

Refer to Built-in

GET ITEM PROPERTY

SET ITEM PROPERTY

Default

No

Required/Optional optional

Usage Notes

To set the Update Only if NULL property programmatically, use the constant UPDATE NULL.

Update Only if NULL Restrictions

Item properties must be set as follows:

- Enabled set to Yes
- Visible set to Yes
- Update Allowed set to No

Update Permission Property

Description

`Setting Update Permission to No performs the following three actions:`

- Sets the Update If Null property to No.
- Sets the Update Allowed property to No.
- Specifies that this column should not be included in any UPDATE statements issued by Forms Developer, 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

Yes

Required/Optional optional

Usage Notes

- Update Permission allows form developers to implement their own security mechanism, overriding the Forms Developer default Enforce 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.

Related topic

Enforce Column Security property

Update Procedure Arguments Property

Description

Specifies the names, datatypes, and values of the arguments that are to be passed to the procedure for updating data. The Update Procedure Arguments property is valid only when the DML Data Target Type property is set to Procedure.

Applies to block

Set Forms Developer

Default

NULL

Required/Optional optional

Related topics

DML Data Target Name property

DML Data Target Type property

Update Procedure Name property

Update Procedure Arguments property

Update Procedure Name Property

Description

Specifies the name of the procedure to be used for updating data. The Update Procedure Name property is valid only when the DML Data Target Type property is set to Procedure.

Applies to block

Set Forms Developer

Default

NULL

Required/Optional optional

Related topics

DML Data Target Name property

DML Data Target Type property

Update Procedure Arguments property

Update Procedure Arguments property

Update Procedure Result Set Columns Property

Description

Specifies the names and datatypes of the result set columns associated with the procedure for updating data. The Update Procedure Result Set Columns property is valid only when the DML Data Target Type property is set to Procedure.

Applies to block

Set Forms Developer

Default

NULL

Required/Optional optional

Related topics

DML Data Target Name property

DML Data Target Type property

Update Procedure Arguments property

Update Procedure Name property

Update Query Property

Description

Specifies whether a chart item is updated to reflect changes made by querying records in its source block.

Applies to chart item

Set Forms Developer

Default

Yes

Required/Optional required

Use 3D Controls Property

Description

On Microsoft Windows, specifies that Forms Developer displays items with a 3-dimensional, beveled look.

When Use 3D Controls is set to Yes, any canvas that has Visual Attribute Group set to Default will automatically be displayed with background color grey.

In addition, when Use 3D Controls is set to Yes, the bevel for each item automatically appears lowered, even if an item-level property is set, for example, to raised.

Applies to form

Set Forms Developer

Default

For a new form, Yes. For a form upgraded from a previous version of Forms Developer, No.

Use 3D Controls Restrictions

- Valid only on Microsoft Windows.

Related topic

Visual Attribute Group property

Use Security Property

Description

Specifies that Forms Developer should enforce the security scheme defined for the menu module, using the Menu Module Roles property.

Applies to menu module

Set Forms Developer

Default

No

Usage Notes

This property can be set to No so that developers can test a menu module without having to be members of any database role. Use Security can then be set to Yes at production to enforce those roles.

Use Security Restrictions

none

User Date/Datetime Format Property

Description

Holds the current date or datetime format mask established by the environment variable FORMS90 USER DATE FORMAT or FORMS90 USER DATETIME FORMAT.

There are two separate properties: User Date Format and User Datetime Format. These properties
will return values even when these environment variables aren't set.

Applies to application

Set Not settable from within Forms Developer.

Refer to Built-in

GET APPLICATION PROPERTY

User Interface Property

Description

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:

- BLOCKMODE
- MACINTOSH
- MOTIF
- MSWINDOWS
- MSWINDOWS32
- PM
- WIN32COMMON
- WEB
- X

User NLS Date Format Property

Description

Obtains the current NLS date format mask. This is equal to the value of the NLS_DATE_FORMAT environment variable if it is set. If it is not set, a default value is derived based on the current NLS territory.

Applies to application

Set Not settable from within Forms Developer.

Refer to Built-in

GET APPLICATION PROPERTY

Note that this property is read-only. That is, you cannot specify it in SET APPLICATION PROPERTY.

For example, if you wanted to set the PLSQL DATE FORMAT property to the current NLS date format mask value, you could code the following in a WHEN-NEW-FORM-INSTANCE trigger in your application:

```
SET APPLICATION PROPERTY(PLSQL DATE FORMAT,
GET APPLICATION PROPERTY(USER NLS DATE FORMAT));
```

Default

None.

User NLS Lang Property

Description

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)

Applies to application

Set Not settable from within Forms Developer. 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.

Username Property

Description

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.

Validate From List Property

Description

Specifies whether Forms Developer should validate the value of the text item against the values in the attached LOV.

Applies to text item

Set Forms Developer

Default

No

Required/Optional optional

Restrictions:

List of Values property must be specified.

Usage Notes

When Validate from List is Yes, Forms Developer 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, Forms Developer 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, Forms Developer 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, Forms Developer dismisses the LOV and assigns the selected values to their corresponding return items.
- When you use an LOV for validation, Forms Developer generally 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

However, when a When-Validate-Item trigger on the item still fires, any validation checks you perform in the trigger still occur.

Also, 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.

Validation Property

Description

Specifies whether default Forms Developer validation processing has been enabled or disabled for a form.

Applies to form module

Set programmatically

Refer to Built-in

GET FORM PROPERTY

SET FORM PROPERTY

Default

Yes

Usage Notes

- Use this property with caution, because when you set Validation to No all internal form validation will be bypassed and no WHEN-VALIDATE triggers will fire.
- You can programmatically set Validation to No for only brief periods of time when you specifically want to avoid all default Forms Developer validation behavior. Once you set Validation to Yes again, any text items left in an unvalidated state will be validated according to normal processing rules.
- When Validation is set to No, the Post-Change trigger will fire during query processing but will *not* fire elsewhere.

Validation Unit Property

Description

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 Forms Developer initiates validation. For most applications, the Validation Unit is Item (default setting on most platforms), which means that Forms Developer validates data in an item as soon as the operator attempts to navigate out of the item.

Applies to form module

Set Forms Developer, programmatically

Refer to Built-in

GET FORM PROPERTY

SET FORM PROPERTY

Default

Default

Usage Notes

The following settings are valid for this property:

- Default
- Form
- Block
- Record
- Item

Value When Checked Property

Description

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 Forms Developer

Default

NULL

Required/Optional optional

Value when Checked Restrictions

The value must be compatible with the datatype specified by the Data Type property.

Value When Unchecked Property

Description

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 Forms Developer

Default

NULL

Required/Optional Optional; leaving this property blank makes the Unchecked value NULL.

Value When Unchecked Restrictions

The value must be compatible with the datatype specified by the Parameter Data Type property.

Vertical Fill Property

Description

Specifies whether the Layout Wizard uses the empty space surrounding an object when the Layout Style property is set to Form.

Yes Specifies that the Layout Wizard should use all available space when arranging frame objects. Consequently, Forms Developer ignores the Maximum Objects Per Line property.

No Specifies that the Layout Wizard should not use all available space when arranging frame objects. When objects wrap, they begin on the next frame line.

Applies to frame

Set Forms Developer

Default

Yes

Required/Optional required

Vertical Justification Property

Description

Specifies the vertical justification of the text object as either Top, Center, or Bottom.

Applies to graphic text

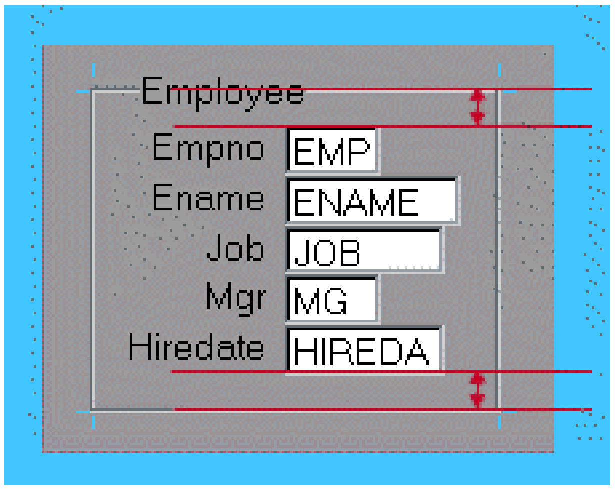
Set Forms Developer

Default

Top

Required/Optional required

Vertical Margin



Related topic

Vertical Margin Property

Vertical Margin Property

Description

Specifies the distance between the frame's top and bottom borders and the objects within the frame.

Applies to frame

Set Forms Developer

Default

1 character cell (or the equivalent based on the form coordinate system)

Required/Optional required

Vertical Object Offset Property

Description

Specifies the vertical distance between the objects within a frame.

Applies to frame

Set Forms Developer

Default

0

Required/Optional required

Vertical Origin Property

Description

Specifies the vertical position of the text object relative to its origin point as either Top, Center, or Bottom.

Applies to graphic text

Set Forms Developer

Default

Top

Required/Optional required

Vertical Toolbar Canvas Property

Description

Specifies the canvas that should be displayed as a vertical toolbar on the window. The canvas you specify must be a vertical toolbar canvas (Canvas Type property set to Vertical Toolbar) and must be assigned to the current window by setting the Window property.

Applies to window

Set Forms Developer

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 canvases that have the Canvas Type property set to Vertical Toolbar.
- At runtime, Forms Developer 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 Window property to point to the specified window), Forms Developer may display a different toolbar in response to navigation events or programmatic control.
- On Microsoft Windows, the specified vertical toolbar canvas 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 Form Vertical Toolbar Canvas form property.

Viewport Height, Viewport Width Property

Description

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 is assigned, and so the Viewport Height and Viewport Width properties do not apply.

Applies to canvas

Set Forms Developer, programmatically

Refer to Built-in

SET VIEW PROPERTY

Default

0,0

Required/Optional optional

Viewport Height, Viewport Width 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.

Viewport X Position on Canvas, Viewport Y Position on Canvas Property

Description

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

Set Forms Developer, programmatically

Refer to Built-in

GET_VIEW_PROPERTY

SET_VIEW_PROPERTY

Default

0,0

Visible (Canvas) Property

Description

Determines whether a stacked canvas is initially shown or hidden in the window to which it is assigned.

Applies to:

stacked canvas

Set:

Forms Developer, programmatically

Refer to Built-in

GET VIEW PROPERTY (VISIBLE)

SET VIEW PROPERTY (VISIBLE)

Default

Yes

Visible (Canvas) Restrictions

- A displayed view may not be visible if it is behind the content view or another stacked view assigned to the same window.

Visible (Tab Page) Property

Description

Determines whether a tab page is shown or hidden at runtime.

Applies to:

tab page

Applies to:

Forms Developer, programmatically

Refer to Built-in

GET TAB PAGE PROPERTY

SET TAB PAGE PROPERTY

Default

Yes

Visible In Horizontal/Vertical Menu Toolbar Property

Description

Specifies whether the menu item should appear (represented by an icon) on the horizontal or vertical menu toolbar (or both) of a form.

Applies to menu item

Set Forms Developer

Default

No

Required/Optional optional

Visible in Horizontal/Vertical Menu Toolbar Restrictions

Developers must provide icons to associate with each menu item that appears on a menu toolbar.

Visible In Menu Property

Description

Determines whether the menu item is shown or hidden at runtime.

Applies to:

menu item

Set:

Forms Developer, programmatically

Refer to Built-in

GET MENU ITEM PROPERTY

SET MENU ITEM PROPERTY

Default**:**

Yes

Visible Property

Description

Indicates whether the object is currently displayed or visible. Set Visible to Yes or No to show or hide a canvas or window.

Applies to canvas, window

Set programmatically

Refer to Built-in

GET VIEW PROPERTY

GET WINDOW PROPERTY

SET VIEW PROPERTY

SET WINDOW PROPERTY

Default

TRUE

Usage Notes

- You cannot hide the canvas 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 WINDOW PROPERTY to determine window visibility, Forms Developer 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 VIEW PROPERTY to determine canvas visibility, Forms Developer 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 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.

Related topics

Visible (Canvas) property

Visible (Item) property

Visible (Tab Page) property

Visual Attribute Group Property

Description

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 Group 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.
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 Group 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 Default.

Applies to all interface objects

Set Forms Developer

Default

Default

Usage Notes

- Default and named visual attributes can include the following individual attributes, listed in the order they appear in the Property Palette:
    - **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 Spacing** 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.

- 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 FORMS60 DEFAULTFONT environment variable. For example, on Microsoft Windows, you can set this variable in the registry, as follows: FORMS60 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 Group 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 Forms Developer.
- When the Use 3D Controls form property is set to Yes 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 canvases that have Visual Attribute Group 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 named visual attributes to the canvas.
- An item that has Visual Attribute Group set to Default, or that has individual attribute settings left unspecified, inherits those settings from the canvas to which it is assigned. Similarly, a canvas that has Visual Attribute Group 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 assigned to the window, at runtime, that canvas will inherit the CYAN background from its window. Visual attribute settings derived through window--canvas or canvas--item inheritance are apparent at design time if the Layout Editor is reopened.
- You can apply property classes to objects to specify visual attribute settings. A property class can contain either the Visual Attribute Group property, or one or more of the

individual attribute properties. (If a property class contains both Visual Attribute Group and individual attributes, the Visual Attribute Group 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.

Visual Attribute Property

Description

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, tab page, item, radio button

Set programmatically

Refer to Built-in

GET ITEM INSTANCE PROPERTY

GET ITEM PROPERTY

GET RADIO BUTTON PROPERTY

GET TAB PAGE PROPERTY

SET CANVAS PROPERTY

SET ITEM INSTANCE PROPERTY

SET ITEM PROPERTY

SET RADIO BUTTON PROPERTY

SET TAB PAGE PROPERTY

Usage Notes

When you execute the appropriate GET built-in 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

- DEFAULT (the item uses the default attributes defined in the resource file)

Visual Attribute Restrictions

- The visual attribute must be a named visual attribute defined in the form module.

Visual Attribute Type Property

Description

Specifies the type of the visual attribute during design time as either Common, Prompt, or Title.

Applies to visual attribute general

Set Forms Developer

Default Common

Required/Optional required

WHERE Clause/ORDER BY Clause Properties

Description

The default WHERE Clause and default ORDER BY Clause properties specify standard SQL clauses for the default SELECT statement associated with a data block. These clauses are automatically appended to the SELECT statement that Forms Developer constructs and issues whenever the operator or the application executes a query in the block.

Applies to block

Set Forms Developer, programmatically

Refer to Built-in

GET BLOCK PROPERTY

SET BLOCK PROPERTY

Required/Optional optional

Usage Notes

1. The reserved words WHERE and ORDER BY are optional. If you do not include them, Forms Developer automatically prefixes the statement with these words.
2. WHERE Clause can reference the following objects:
   columns in the block's data block table (except LONG columns)
   form parameters (:PARAMETER.my parameter)
3. ORDER BY Clause can reference the following objects:
   columns in the block's data block table (except LONG columns)
4. Embedded comments are not supported in WHERE Clause and ORDER BY Clause.

WHERE Clause/ORDER BY Clause Restrictions

- Maximum length for WHERE Clause is 32,000 bytes.
- ORDER BY clause cannot reference global variables or form parameters.

WHERE Clause/ORDER BY Clause Examples

Example

In the following example from an order tracking system, the WHERE Clause limits the retrieved records to those whose *shipdate* column is NULL. The ORDER BY Clause arranges the selected records from the lowest (earliest) date to the highest (latest) date.

```
WHERE shipdate IS NULL
ORDER BY orderdate
```

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.

Width/Height (WD, HT) Property

Description

See Size

Window Handle Property

Description

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

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 Handle Restrictions

- Valid only on Microsoft Windows. (Returns NULL on other platforms.)

Window Property

Description

Specifies the window in which the canvas will be displayed at runtime.

Applies to canvas

Set Forms Developer

Refer to Built-in

GET VIEW PROPERTY

Default

ROOT WINDOW, if there is a root window in the form, else the first window listed under the Windows node in the Object Navigator.

Required/Optional required for the canvas to be displayed at runtime

Window State Property

Description

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 s 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

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 State Restrictions

- Setting Window State to MAXIMIZE is not supported on Motif.

Window Style Property

Description

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 Forms Developer

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 is 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 Property

Description

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 Forms Developer

Refer to Built-in

GET ITEM PROPERTY

Default

WORD

Wrap Style Restrictions

- Valid only for multi-line text items.

Wrap Text Property

Description

Specifies whether the text in the text object wraps to the next line to fit within the bounding box.

Applies to graphic text

Set Forms Developer

Default

Yes

Required/Optional required

X Corner Radius Property

Description

Specifies the amount of horizontal rounding (in layout units) of the corners of the rounded rectangle.

Applies to graphic rounded rectangle

Set Forms Developer

Default

10

Required/Optional required

X Position, Y Position Property

Description

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, canvases

Set Forms Developer, programmatically

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_ITEM_PROPERTY

SET_ITEM_PROPERTY

GET_RADIO_BUTTON_PROPERTY

SET_RADIO_BUTTON_PROPERTY

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_LOV_PROPERTY

SET_LOV_PROPERTY

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_WINDOW_PROPERTY

SET_WINDOW_PROPERTY

Default

x,y(0,0)

X Position, Y Position Restrictions

- 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.

Y Corner Radius Property

Description

Specifies the amount of vertical rounding (in layout units) of the corners of the rounded rectangle.

Applies to graphic rounded rectangle

Set Forms Developer

Default

10

Required/Optional required

About System Variables

A system variable is an Forms Developer variable that keeps track of an internal Forms Developer state. You can reference the value of a system variable to control the way an application behaves.

Forms Developer 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 names of the available system variables are:

- 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.TAB_NEW_PAGE
- SYSTEM.TAB_PREVIOUS_PAGE
- 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. Forms Developer also supplies 6 default values for date and time. (See Date and Time System Default Values).

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 instead of getting the system variable value each time.

Date and Time System Default Values

Forms Developer supplies six special default values $$DATE$$, $$DATETIME$$, $$TIME$$, $$DBDATE$$, $$DBDATETIME$$, and $$DBTIME$$ that supply date and time information. These variables have the and the following special restrictions on their use:

- When 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 when, for example, connecting to a remote database in a different time zone.
- Use these variables only to set the value of the Initial Value, Highest Allowed Value or Lowest Allowed Value property.

About system variables Examples

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:

```
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.

$$DATE$$ System Variable

Syntax

$$DATE$$

Description

$$DATE$$ retrieves the current operating system date (client-side). Use $$DATE$$ to designate a default value or range for a text item using the Initial Value or Lowest/Highest Allowed Value properties. The text item must be of the CHAR, DATE, or DATETIME data type.

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, to use the default DD-MON-YY format, 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:MM: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.

$$DATE$$ Examples

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 Initial 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:

```
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;
```

$$DATETIME$$ System Variable

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 Initial Value or Lowest/Highest Allowed Value properties. The text item must be of the CHAR or DATETIME data type.

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:MM: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 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.

$$DATETIME$$ Examples

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 Initial Value property.

$$DBDATE$$ System Variable

Syntax

$$DBDATE$$

Description

$$DBDATE$$ retrieves the current database date. Use $$DBDATE$$ to designate a default value or range for a text item using the Initial Value or Lowest/Highest Allowed Value properties. The text item must be of the CHAR, DATE, or DATETIME data type.

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 $$DBDATE$$ 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:MM:SS: for example, 01-DEC-1994 00:00:00 (although only 01-DEC-1994 will be displayed).

$$ DBDATE$$ Restrictions

- 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.

$$ DBDATE$$ Examples

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 Initial Value property.

$$DBDATETIME$$ System Variable

Syntax

$$DBDATETIME$$

Description

$$DBDATETIME$$ retrieves the current date and time from the local database. Use $$DBDATETIME$$ to designate a default value or range for a text item using the Initial Value or Lowest/Highest Allowed Value properties. The text item must be of the CHAR or DATETIME data type.

Usage Notes

Use $$DBDATETIME$$ 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:MM:SS format, you must specify a DATETIME or CHAR data type. (Note that the default format mask depends on the value of NLS_LANG.)

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.

$$DBDATETIME$$ Restrictions

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.

$$DBDATETIME$$ Examples

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 Lowest/Highest Allowed Value properties.

$$DBTIME$$ System Variable

Syntax

$$DBTIME$$

Description

$$DBTIME$$ retrieves the current time from the local database. Use $$DBTIME$$ to designate a default value or range for a text item using the Initial Value or Lowest/Highest Allowed Value properties. The text item must be of the CHAR or TIME data type.

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:MM:SS format, you must specify a TIME data type. (Note that the default format mask depends on the value of NLS_LANG.)

$$DBTIME$$ Restrictions

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.

$$DBTIME$$ Examples

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 Initial Value property.

$$TIME$$ System Variable

Syntax

$$TIME$$

Description

$$TIME$$ retrieves the current operating system time. Use $$TIME$$ to designate a default value or range for a text item using the Initial Value or Lowest/Highest Allowed Value properties. 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.

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 HH:MM:SS format, you must specify a TIME data type. (Note that the default format mask depends on the value of NLS_LANG.)

$$TIME$$ Examples

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 Initial Value property.

SYSTEM.BLOCK_STATUS System Variable

Syntax

SYSTEM.BLOCK_STATUS

Description

SYSTEM.BLOCK_STATUS represents the status of a Data block where the cursor is located, or the current data 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.

Usage Notes

Each time this value is referenced, it must be constructed by Forms Developer. If a block contains a large number of records, using SYSTEM.BLOCK_STATUS could adversely affect performance.

SYSTEM.BLOCK_STATUS Examples

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.

```
IF :System.Block_Status = 'CHANGED'
THEN Commit_Form;
END IF;
Clear_Block;
```

Related topics

SYSTEM.RECORD_STATUS examples

SYSTEM.FORM_STATUS examples

SYSTEM.COORDINATION_OPERATION System Variable

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:

: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:

: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.

Value	**Description**	**Caused By**
MOUSE	Mouse to non-current record	Mouse
UP	Move up a record	Menu, key, PL/SQL
DOWN	Move down a record	Menu, key, PL/SQL
SCROLL_UP	Scroll up records	Menu, key, PL/SQL
SCROLL_DOWN	Scroll down records	Mouse, key, PL/SQL
CLEAR_BLOCK	Clear current block	Menu, key, PL/SQL
CLEAR_RECORD	Clear current record	Menu, key, PL/SQL
CREATE_RECORD	Create new record	Mouse, menu, key, PL/SQL
DELETE_RECORD	Delete current record	Menu, key, PL/SQL
DUPLICATE_RECORD	Duplicate current record	Menu, key, PL/SQL
FIRST_RECORD	Move to first record	PL/SQL
LAST_RECORD	Move to last record	PL/SQL

NEXT_RECORD	Move to next record	Mouse, menu, key, PL/SQL
PREVIOUS_RECORD	Move to previous record	Mouse, menu, key, PL/SQL
GO_RECORD	Jump to record by number	PL/SQL
ENTER_QUERY	Enter Query mode	Menu, key, PL/SQL
EXECUTE_QUERY	Execute query	Menu, key, PL/SQL
COUNT_QUERY	Count queried records	Menu, key, PL/SQL
NEXT_SET	Fetch next set of records	Menu, key, PL/SQL
SYNCHRONIZE_ BLOCKS	Resume after commit error	Internal only

Related topic

About system variables

SYSTEM.CURRENT_BLOCK System Variable

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 Forms Developer 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 System Variable

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:MM: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.

SYSTEM.CURRENT_DATETIME Examples

```
/*

**
** 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 System Variable

Syntax

SYSTEM.CURRENT_FORM

Description

`SYSTEM.CURRENT_FORM` represents the name of the form that Forms Developer 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.

SYSTEM.CURRENT_FORM Examples

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 Forms Developer issues a call. This procedure stores the name of the current form in a global variable named CALLING_FORM.

```
PROCEDURE STORE_FORMNAME IS
BEGIN
:GLOBAL.Calling_Form := :System.Current_Form;
END;
```

SYSTEM.CURRENT_ITEM System Variable

Syntax

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 Forms Developer 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 System Variable

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 System Variable

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.

SYSTEM.CURSOR_BLOCK Examples

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.

```
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 System Variable

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.

Usage Notes

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.

SYSTEM.CURSOR_ITEM Restrictions

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.

SYSTEM.CURSOR_ITEM Examples

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.

```
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 System Variable

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.

SYSTEM.CURSOR_RECORD Examples

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 System Variable

Syntax

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.

Usage Notes

Be aware that in triggers where the current navigation unit is *not* the item, such as 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*.

SYSTEM.CURSOR_VALUE Restrictions

- Avoid using SYSTEM.CURSOR_VALUE in Pre-Form and Post-Form triggers, where the value of SYSTEM.CURSOR_VALUE is NULL.

SYSTEM.CURSOR_VALUE Examples

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.

```
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.DATE_THRESHOLD System Variable

Syntax

SYSTEM.DATE_THRESHOLD

Description

SYSTEM.DATE_THRESHOLD represents the database date requery threshold. This variable works in conjunction with the three system variables $$DBDATE$$, $$DBDATETIME$$, and $$DBTIME$$, and controls how often Forms Developer synchronizes the database date with the RDBMS. The value of this variable must be specified in the following format:

MI:SS

Because frequent RDBMS queries can degrade performance, it is best to keep this value reasonably high. However, keep in mind that if the value is not synchronized often enough, some time discrepancy can occur.

Default

01:00 (Synchronization occurs after one minute of elapsed time.)

This does not mean that Forms Developer polls the RDBMS once every minute. It means that whenever Forms Developer needs to generate the value for the system variables $$DBDATE$$, $$DBDATETIME$$, $$DBTIME$$, or SYSTEM.EFFECTIVE_DATE, it updates the effective date by adding the amount of elapsed time (as measured by the local operating system) to the most previously queried RDBMS value.

If the amount of elapsed time exceeds the date threshold, then a new query is executed to retrieve the RDBMS time and the elapsed counter is reset.

Usage Notes

If a form never references the database date, Forms Developer never executes a query to retrieve the RDBMS date, regardless of the value of SYSTEM.DATE_THRESHOLD.

The operating system clock and the RDBMS clock rarely drift by more than one or two seconds, even after hours of elapsed time. However, since your database administrator can reset the RDBMS clock at any time, it is safest to set the threshold no higher than a few minutes.

Often, a Forms Developer block may contain multiple references to these $$DBDATE$$, $$DBDATETIME$$, or $$DBTIME$$ defaults. By setting SYSTEM.DATE_THRESHOLD to the default of one minute, nearly all such references in a form can be satisfied with a single query of the RDBMS.

SYSTEM.EFFECTIVE_DATE System Variable

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

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.

SYSTEM.EFFECTIVE_DATE Restrictions

This variable is only valid when the database contains a definition of the DUAL table.

SYSTEM.EFFECTIVE_DATE Examples

Assume you have set a DATE or TIME text item to one of the three system variables $$DBDATE$$, $$DBDATETIME$$, or $$DBTIME$$. To override that date or time, set the SYSTEM.EFFECTIVE_DATE system variable to a specific date and/or time.

```
:System.Effective_Date := '31-DEC-1997 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:

31-DEC-1997 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. Forms Developer polls the RDBMS whenever a reference to the effective date is required by the application.

SYSTEM.EVENT_WINDOW System Variable

Syntax

SYSTEM.EVENT_WINDOW

Description

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, assign the value of the variable to any of the following:

- global variable
- parameter
- variable
- item, including a null canvas item

SYSTEM.EVENT_WINDOW Examples

The following example sets the input focus into a particular item, depending on the window affected:

```
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 System Variable

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 Forms Developer. If a form contains a large number of blocks and many records, using SYSTEM.FORM_STATUS could affect performance.

SYSTEM.FORM_STATUS Examples

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;
```

Related topics

SYSTEM.BLOCK_STATUS examples

SYSTEM.RECORD_STATUS examples

SYSTEM.LAST_FORM System Variable

Syntax

SYSTEM.LAST_FORM

Description

SYSTEM.LAST_FORM represents the form document 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 document ID or NULL.

Usage Notes

SYSTEM.LAST_FORM is not valid with CALL_FORM.

SYSTEM.LAST_QUERY System Variable

Syntax

SYSTEM.LAST_QUERY

Description

SYSTEM.LAST_QUERY represents the query SELECT statement that Forms Developer most recently used to populate a block during the current Runform session. The value is always a character string.

SYSTEM.LAST_RECORD System Variable

Syntax

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.

SYSTEM.LAST_RECORD Examples

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.

```
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 System Variable

Syntax

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.

Please see the reference topic for SYSTEM.COORDINATION_OPERATION for more information.

SYSTEM.MESSAGE_LEVEL System Variable

Syntax

SYSTEM.MESSAGE_LEVEL

Description

SYSTEM.MESSAGE_LEVEL stores one of the following message severity levels: 0, 5, 10, 15, 20, or 25. The default value is 0.

SYSTEM.MESSAGE_LEVEL can be set to either a character string or a number. The values assigned can be any value between 0 and 25, but values lower than 0 or higher than 25 will generate an error.

During a Runform session, Forms Developer 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. Forms Developer does not suppress prompts or vital error messages, no matter what severity level you select.

SYSTEM.MESSAGE_LEVEL Examples

Assume that you want Forms Developer 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';
```

SYSTEM.MODE System Variable

Syntax

SYSTEM.MODE

Description

SYSTEM.MODE indicates whether the form is in Normal, Enter Query, or Fetch Processing mode. The value is always a character string.

NORMAL	Indicates that the form is currently in normal processing mode.
ENTER-QUERY	Indicates that the form is currently in Enter Query mode.
QUERY	Indicates that the form is currently in fetch processing mode, meaning that a query is currently being processed.

Usage Notes

When using SYSTEM.MODE to check whether the current block is in Enter Query mode, be aware that if testing from a When-Button-Pressed trigger in a control block, Enter Query mode will never be entered, because the control block is not the current block.

SYSTEM.MODE Examples

Assume that you want Forms Developer 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;
```

SYSTEM.MOUSE_BUTTON_MODIFIERS System Variable

Syntax

SYSTEM.MOUSE_BUTTON_MODIFIERS

Description

SYSTEM.MOUSE_BUTTON_MODIFIERS indicates the keys that were pressed during the click, such as SHIFT, ALT, or CONTROL. The value is always a character string.

For example, if the operator holds down the control and shift keys while pressing the mouse button, SYSTEM.MOUSE_BUTTON_MODIFIERS contains the value "Shift+Control+".

The values returned by this variable will be invariant across all platforms, and will not change across languages. SYSTEM.MOUSE_BUTTON_MODIFIERS should be used in place of SYSTEM.MOUSE_BUTTON_SHIFT_STATE.

Possible values are: "Shift+", "Caps Lock+", "Control+", "Alt+", "Command+", "Super+", and "Hyper+".

SYSTEM.MOUSE_BUTTON_PRESSED System Variable

Syntax

SYSTEM.MOUSE_BUTTON_PRESSED

Description

SYSTEM.MOUSE_BUTTON_PRESSED indicates the number of the button that was clicked, either 1, 2, or 3 (left, middle, or right). The value is always a character string.

Notes:

On Motif platforms pressing the right mouse button will not set the SYSTEM.MOUSE_BUTTON_PRESSED value.

SYSTEM.MOUSE_BUTTON_PRESSED Examples

```
/*

** 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;
```

Related topics

SYSTEM.MOUSE_BUTTON_SHIFT_STATE examples

SYSTEM.MOUSE_CANVAS examples

SYSTEM.MOUSE_FORM system variable

SYSTEM.MOUSE_ITEM examples

SYSTEM.MOUSE_RECORD examples

SYSTEM.MOUSE_RECORD_OFFSET system variable

SYSTEM.MOUSE_X_POS examples

SYSTEM.MOUSE_Y_POS examples

SYSTEM.MOUSE_BUTTON_SHIFT_STATE System Variable

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. The string itself may depend on the user's platform. For example, in Microsoft Windows, the strings returned are in the language of the operating system.

Key Pressed	Value
SHIFT	Shift+
CONTROL	Ctrl+
ALT	Alt+
SHIFT+CONTROL	Shift+Ctrl+

SYSTEM.MOUSE_BUTTON_SHIFT_STATE Examples

```
/*

** 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(30) := 'FALSE';
x_position_clicked NUMBER(30);
y_position_clicked NUMBER(30);
```

```
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;
```

Related topics

SYSTEM.MOUSE_BUTTON_PRESSED examples

SYSTEM.MOUSE_CANVAS examples

SYSTEM.MOUSE_FORM system variable

SYSTEM.MOUSE_ITEM examples

SYSTEM.MOUSE_RECORD examples

SYSTEM.MOUSE_RECORD_OFFSET system variable

SYSTEM.MOUSE_X_POS examples

SYSTEM.MOUSE_Y_POS examples

SYSTEM.MOUSE_CANVAS System Variable

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 operator presses the left mouse button, then moves the mouse
- the platform is non-GUI

SYSTEM.MOUSE_CANVAS Examples

```
/*

** Trigger: When-Mouse-Move
** Example: When the mouse enters any one of several overlapping
** canvases, Forms Developer 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;
```

Related topics

SYSTEM.MOUSE_BUTTON_PRESSED examples

SYSTEM.MOUSE_BUTTON_SHIFT_STATE examples

SYSTEM.MOUSE_ITEM examples

SYSTEM.MOUSE_RECORD examples

SYSTEM.MOUSE_RECORD_OFFSET system variable

SYSTEM.MOUSE_X_POS examples

SYSTEM.MOUSE_Y_POS examples

SYSTEM.MOUSE_FORM System Variable

Syntax

SYSTEM.MOUSE_FORM

Description

If the mouse is in a form document, SYSTEM.MOUSE_FORM represents the name of that form document 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.

Related topics

SYSTEM.MOUSE_BUTTON_PRESSED examples

SYSTEM.MOUSE_BUTTON_SHIFT_STATE examples

SYSTEM.MOUSE_CANVAS examples

SYSTEM.MOUSE_ITEM examples

SYSTEM.MOUSE_RECORD examples

SYSTEM.MOUSE_RECORD_OFFSET system variable

SYSTEM.MOUSE_X_POS examples

SYSTEM.MOUSE_X_POS examples

SYSTEM.MOUSE_ITEM System Variable

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 operator presses the left mouse button, then moves the mouse
- the platform is not a GUI platform

If you use SYSTEM.MOUSE_ITEM in a WHEN-BUTTON-PRESSED-Trigger, it doesn't work (SYSTEM.MOUSE_ITEM has got a null value) as long, as there is no MOUSE-Trigger defined in the FORM.

SYSTEM.MOUSE_ITEM Examples

```
/* 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;
```

Related topics

SYSTEM.MOUSE_BUTTON_PRESSED examples

SYSTEM.MOUSE_BUTTON_SHIFT_STATE examples

SYSTEM.MOUSE_CANVAS examples

SYSTEM.MOUSE_FORM system variable

SYSTEM.MOUSE_RECORD examples

SYSTEM.MOUSE_RECORD_OFFSET system variable

SYSTEM.MOUSE_X_POS examples

SYSTEM.MOUSE_Y_POS examples

SYSTEM.MOUSE_RECORD System Variable

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).

SYSTEM.MOUSE_RECORD Examples

```
/*

** 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;
```

Related topics

SYSTEM.MOUSE_BUTTON_PRESSED examples

SYSTEM.MOUSE_BUTTON_SHIFT_STATE examples

SYSTEM.MOUSE_CANVAS examples

SYSTEM.MOUSE_FORM system variable

SYSTEM.MOUSE_ITEM examples

SYSTEM.MOUSE_RECORD_OFFSET system variable

SYSTEM.MOUSE_X_POS examples

SYSTEM.MOUSE_Y_POS examples

SYSTEM.MOUSE_RECORD_OFFSET System Variable

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. SYSTEM.MOUSE_RECORD_OFFSET is only valid within mouse triggers. Its value represents which row within the visible rows the mouse has clicked.

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).

Related topics

SYSTEM.MOUSE_BUTTON_PRESSED examples

SYSTEM.MOUSE_BUTTON_SHIFT_STATE examples

SYSTEM.MOUSE_CANVAS examples

SYSTEM.MOUSE_FORM system variable

SYSTEM.MOUSE_ITEM examples

SYSTEM.MOUSE_RECORD examples

SYSTEM.MOUSE_X_POS examples

SYSTEM.MOUSE_Y_POS examples

SYSTEM.MOUSE_X_POS System Variable

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.

SYSTEM.MOUSE_X_POS Examples

```
/*

** Example: See SYSTEM.MOUSE_ITEM and
** SYSTEM.MOUSE_BUTTON_SHIFT_STATE.
*/
```

Related topics

SYSTEM.MOUSE_BUTTON_PRESSED examples

SYSTEM.MOUSE_BUTTON_SHIFT_STATE examples

SYSTEM.MOUSE_CANVAS examples

SYSTEM.MOUSE_FORM system variable

SYSTEM.MOUSE_ITEM examples

SYSTEM.MOUSE_RECORD examples

SYSTEM.MOUSE_RECORD_OFFSET system variable

SYSTEM.MOUSE_Y_POS examples

SYSTEM.MOUSE_Y_POS System Variable

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.

SYSTEM.MOUSE_Y_POS Examples

```
/*

** Example: See SYSTEM.MOUSE_ITEM and
** SYSTEM.MOUSE_BUTTON_SHIFT_STATE.
*/
```

Related topics

SYSTEM.MOUSE_BUTTON_PRESSED examples

SYSTEM.MOUSE_BUTTON_SHIFT_STATE examples

SYSTEM.MOUSE_CANVAS examples

SYSTEM.MOUSE_FORM system variable

SYSTEM.MOUSE_ITEM examples

SYSTEM.MOUSE_RECORD examples

SYSTEM.MOUSE_RECORD_OFFSET system variable

SYSTEM.MOUSE_X_POS examples

SYSTEM.RECORD_STATUS System Variable

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, Forms Developer 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.

SYSTEM.RECORD_STATUS Examples

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.

```
IF :System.Record_Status IN ('CHANGED', 'INSERT') THEN
```

```
Commit_Form;
END IF;
Clear_Record;
```

Related topics

GET_RECORD_PROPERTY built-in

SET_RECORD_PROPERTY built-in

SYSTEM.BLOCK_STATUS examples

SYSTEM.SUPPRESS_WORKING System Variable

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 Forms Developer from issuing the "Working..." message.
FALSE	Allows Forms Developer to continue to issue the "Working..." message.

SYSTEM.SUPPRESS_WORKING Examples

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.TAB_NEW_PAGE System Variable

Syntax

SYSTEM.TAB_NEW_PAGE

Description

The system variable SYSTEM.TAB_NEW_PAGE specifies the name of the tab page to which navigation occurred. Use it inside a When-Tab-Page-Changed trigger.

SYSTEM.TAB_NEW_PAGE Examples

/* Use system variable SYSTEM.TAB_NEW_PAGE inside a

```
** When-Tab-Page-Changed trigger to change the label of
** the tab page to UPPERCASE when an end user navigates
** into the tab page:
*/
DECLARE
tp_nm VARCHAR2(30);
tp_id TAB_PAGE;
tp_lbl VARCHAR2(30);
BEGIN
tp_nm := :SYSTEM.TAB_NEW_PAGE;
tp_id := FIND_TAB_PAGE(tp_nm);
tp_lbl := GET_TAB_PAGE_PROPERTY(tp_id, label);
IF tp_nm LIKE 'ORD%' THEN
SET_TAB_PAGE_PROPERTY(tp_id, label, 'ORDERS');
END IF;
END;
```

Related topics

SYSTEM.TAB_PREVIOUS_PAGE examples

When-Tab-Page-Changed

FIND_TAB_PAGE built-in

GET_TAB_PAGE_PROPERTY built-in

SET_TAB_PAGE_PROPERTY built-in

SYSTEM.TAB_PREVIOUS_PAGE System Variable

Syntax

SYSTEM.TAB_PREVIOUS_PAGE

Description

The system variable SYSTEM.TAB_PREVIOUS_PAGE specifies the name of the tab page from which navigation occurred. Use it inside a When-Tab-Page-Changed trigger.

SYSTEM.TAB_PREVIOUS_PAGE Examples

```
/* Use system variable SYSTEM.TAB_PREVIOUS_PAGE inside a

** When-Tab-Page-Changed trigger to change the label of the
** tab page to initial-cap after an end user navigates out
** of the tab page:
*/
DECLARE
tp_nm VARCHAR2(30);
tp_id TAB_PAGE;
tp_lbl VARCHAR2(30);
BEGIN
tp_nm := :SYSTEM.TAB_PREVIOUS_PAGE;
tp_id := FIND_TAB_PAGE(tp_nm);
tp_lbl := GET_TAB_PAGE_PROPERTY(tp_id, label);
IF tp_nm LIKE 'ORD%' THEN
SET_TAB_PAGE_PROPERTY(tp_id, label, 'Orders');
END IF;
END;
```

Related topics

SYSTEM.TAB_NEW_PAGE examples

When-Tab-Page-Changed

FIND_TAB_PAGE built-in

GET_TAB_PAGE_PROPERTY built-in

SET_TAB_PAGE_PROPERTY built-in

SYSTEM.TRIGGER_BLOCK System Variable

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.

SYSTEM.TRIGGER_BLOCK Examples

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'));
```

Related topics

Post-Form Trigger

Pre-Form Trigger

SYSTEM.TRIGGER_ITEM System Variable

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.

Usage Notes

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.

SYSTEM.TRIGGER_ITEM Restrictions

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.

SYSTEM.TRIGGER_ITEM Examples

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_MENUOPTION System Variable

Syntax

SYSTEM.TRIGGER_MENUOPTION

Description

SYSTEM.TRIGGER_MENUOPTION is provided for backward compatibility with existing applications. It replaces the Menu parameter :SO and provides
exactly the same behavior as the :SO built-in substitution parameter.

The value of the :SO built-in substitution parameter is predefined and always initialized by Forms at runtime. The value of the built-in parameter :SO is the current menu item (Selected Option).

:SO returns a string value with the format <menu_name>-<index number> where index number is the number of the item in the menu, for example:

FILE_MENU-1
EDIT_MENU-3

Note that the index number is based on all the items in the menu, not just those that are displayed.

SYSTEM.TRIGGER_NODE System Variable

Syntax

SYSTEM.TRIGGER_NODE

Description

SYSTEM.TRIGGER_NODE represents the node the user selected and it returns a value of type `NODE`.

SYSTEM.TRIGGER_NODE_SELECTED System Variable

Syntax

SYSTEM.TRIGGER_NODE_SELECTED

Description

SYSTEM.TRIGGER_NODE_SELECTED contains a valid value only during the WHEN-TREE-NODE-SELECTED trigger, indicating whether the trigger is firing for a selection or a deselection. The values are either TRUE or FALSE.

SYSTEM.TRIGGER_RECORD System Variable

Syntax

SYSTEM.TRIGGER_RECORD

Description

SYSTEM.TRIGGER_RECORD represents the number of the record that Forms Developer 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.

SYSTEM.TRIGGER_RECORD Examples

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;
```

Triggers

Block Processing triggers

Interface Event triggers

Key triggers

Master-Detail triggers

Message-Handling triggers

Mouse Event triggers

Navigation triggers

On triggers

Post triggers

Pre triggers

Query-Time triggers

Stored Procedure triggers

Transactional triggers

Validation triggers

When triggers

Related topic

About calling built-in subprograms in triggers

Block Processing Triggers

When-Clear-Block

When-Create-Record

When-Database-Record

When-Remove-Record

Forms Developer Triggers

Triggers

A

B

C

D

Delete-Procedure

E

F

Function Key Triggers

G

H

I

Insert-Procedure

J

K

Key-Fn

Key-Others

L

Lock-Procedure

M

N

O

On-Check-Delete-Master

On-Check-Unique

On-Clear-Details

On-Close

On-Column-Security

On-Commit

On-Count

On-Delete

On-Error

On-Fetch

On-Insert

On-Lock

On-Logon

On-Logout

On-Message

On-Populate-Details

On-Rollback

On-Savepoint

On-Select

On-Sequence-Number

On-Update

P

Post-Block

Post-Change

Post-Database-Commit

Post-Delete

Post-Form

Post-Forms-Commit

Post-Insert

Post-Logon

Post-Logout

Post-Query

Post-Record

Post-Select

Post-Text-Item

Post-Update

Pre-Block

Pre-Commit

Pre-Delete

Pre-Form

Pre-Insert

Pre-Logon

Pre-Logout

Pre-Popup-Menu

Pre-Query

Pre-Record

Pre-Select

Pre-Text-Item

Pre-Update

Q

Query-Procedure

R

S

T

U

Update-Procedure

User-Named

V

W

When-Button-Pressed

When-Checkbox-Changed

When-Clear-Block

When-Create-Record

When-Database-Record

When-Form-Navigate

When-Image-Activated

When-Image-Pressed

When-List-Activated

When-List-Changed

When-Mouse-Click

When-Mouse-DoubleClick

When-Mouse-Down

When-Mouse-Enter

When-Mouse-Leave

When-Mouse-Move

When-Mouse-Up

When-New-Block-Instance

When-New-Form-Instance

When-New-Item-Instance

When-New-Record-Instance

When-Radio-Changed

When-Remove-Record

When-Tab-Page-Changed

When-Timer-Expired

When-Validate-Item

When-Validate-Record

When-Window-Activated

When-Window-Closed

When-Window-Deactivated

When-Window-Resized

X

Y

Z

Interface Event Triggers

When-Button-Pressed

When-Checkbox-Changed

When-Image-Activated

When-List-Activated

When-List-Changed

When-Mouse-Click

When-Mouse-DoubleClick

When-Mouse-Down

When-Mouse-Enter

When-Mouse-Leave

When-Mouse-Move

When-Mouse-Up

When-Radio-Changed

When-Timer-Expired

When-Window-Activated

When-Window-Closed

When-Window-Deactivated

When-Window-Resized

Key Triggers

Key-Fn

Key-Others

Master-Detail Triggers

On-Check-Delete-Master

On-Clear-Details

On-Populate-Details

Message-Handling Triggers

On-Error

On-Message

Mouse Event Triggers

When-Custom-Item-Event

When-Mouse-Click

When-Mouse-DoubleClick

When-Mouse-Down

When-Mouse-Enter

When-Mouse-Leave

When-Mouse-Move

When-Mouse-Up

Navigation Triggers

Post-Block

Post-Form

Post-Record

Post-Text-Item

Pre-Block

Pre-Form

Pre-Record

Pre-Text-Item

User-Named

When-New-Block-Instance

When-New-Form-Instance

When-New-Item-Instance

When-New-Record-Instance

On Triggers

On-Check-Delete-Master

On-Check-Unique

On-Clear-Details

On-Close

On-Column-Security

On-Commit

On-Count

On-Delete

On-Error

On-Fetch

On-Insert

On-Lock

On-Logon

On-Logout

On-Message

On-Populate-Details

On-Rollback

On-Savepoint

On-Select

On-Sequence-Number

On-Update

Pre Triggers

Pre-Block

Pre-Commit

Pre-Delete

Pre-Form

Pre-Insert

Pre-Logon

Pre-Logout

Pre-Popup-Menu

Pre-Query

Pre-Record

Pre-Select

Pre-Text-Item

Pre-Update

Post Triggers

Post-Block

Post-Change

Post-Database-Commit

Post-Delete

Post-Form

Post-Forms-Commit

Post-Insert

Post-Logon

Post-Logout

Post-Query

Post-Record

Post-Select

Post-Text-Item

Post-Update

Query-Time Triggers

Post-Query

Pre-Query

Stored Procedure Triggers

Delete-Procedure

Insert-Procedure

Lock-Procedure

Query-Procedure

Update-Procedure

Transactional Triggers

On-Check-Delete-Master

On-Check-Unique

On-Clear-Details

On-Close

On-Column-Security

On-Commit

On-Count

On-Delete

On-Error

On-Fetch

On-Insert

On-Lock

On-Logon

On-Logout

On-Message

On-Populate-Details

On-Rollback

On-Savepoint

On-Select

On-Sequence-Number

On-Update

Post-Block

Post-Change

Post-Database-Commit

Post-Delete

Post-Form

Post-Forms-Commit

Post-Insert

Post-Logon

Post-Logout

Post-Query

Post-Record

Post-Select

Post-Text-Item

Post-Update

Pre-Block

Pre-Commit

Pre-Delete

Pre-Form

Pre-Insert

Pre-Logon

Pre-Query

Pre-Select

Pre-Text-Item

Pre-Update

Validation Triggers

When-Validate-Item

When-Validate-Record

When Triggers

When-Button-Pressed

When-Checkbox-Changed

When-Clear-Block

When-Create-Record

When-Database-Record

When-Form-Navigate

When-Image-Activated

When-Image-Pressed

When-List-Activated

When-List-Changed

When-Mouse-Click

When-Mouse-DoubleClick

When-Mouse-Down

When-Mouse-Enter

When-Mouse-Leave

When-Mouse-Move

When-Mouse-Up

When-New-Block-Instance

When-New-Form-Instance

When-New-Item-Instance

When-New-Record-Instance

When-Radio-Changed

When-Remove-Record

When-Tab-Page-Changed

When-Timer-Expired

When-Validate-Item

When-Validate-Record

When-Window-Activated

When-Window-Closed

When-Window-Deactivated

When-Window-Resized

Delete-Procedure Trigger

Description

Automatically created by Forms Developer when the delete data source is a stored procedure. This trigger is called when a delete operation is necessary. Think of this as an ON-DELETE trigger that is called by the system instead of doing default delete operations.

Do not modify this trigger.

Enter Query Mode Not applicable.

On Failure

No effect

Function Key Triggers

Description

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 defined 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.

Key Trigger	**Associated Function Key**
Key-CLRBLK	[Clear Block]
Key-CLRFRM	[Clear Form]
Key-CLRREC	[Clear Record]
Key-COMMIT	[Accept]
Key-CQUERY	[Count Query Hits]
Key-CREREC	[Insert Record]
Key-DELREC	[Delete Record]
Key-DOWN	[Down]
Key-DUP-ITEM	[Duplicate Item]
Key-DUPREC	[Duplicate Record]
Key-EDIT	[Edit]
Key-ENTQRY	[Enter Query]

Key-EXEQRY	[Execute Query]
Key-EXIT	[Exit]
Key-HELP	[Help]
Key-LISTVAL	[List of Values]
Key-MENU	[Block Menu]
Key-NXTBLK	[Next Block]
Key-NXT-ITEM	[Next Item]
Key-NXTKEY	[Next Primary Key]
Key-NXTREC	[Next Record]
Key-NXTSET	[Next Set of Records]
Key-PRINT	[Print]
Key-PRVBLK	[Previous Block]
Key-PRV-ITEM	[Previous Item]
Key-PRVREC	[Previous Record]
Key-SCRDOWN	[Scroll Down]
Key-SCRUP	[Scroll Up]
Key-UP	[Up]
Key-UPDREC	Equivalent to Record, Lock command on the default menu

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 screen or user interface management system and not by Forms Developer.

[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

On Failure

no effect

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.

Function Key Triggers Restrictions

- Forms Developer ignores the Key-Commit trigger when an operator presses [Commit/Accept] in a dialog box.

Insert-Procedure Trigger

Description

Automatically created by Forms Developer when the insert data source is a stored procedure. This trigger is called when a insert operation is necessary. Think of this as an ON-INSERT trigger that is called by the system instead of doing default insert operations.

Definition Level

Do not modify this trigger.

Enter Query Mode Not applicable.

On Failure

No effect

Key-Fn Trigger

Description

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 Forms Developer operations. These keys are referred to as Key-F0 through Key-F9.

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.
- The key description shown in the default menu's **Help→Keys** menu will always be for the form-level trigger defined for that key. If there are any lower-level triggers (e.g., block-level triggers) that are also defined for the key, their descriptions will be shown when focus is in the lower level (e.g., the block) and [Show Keys] is pressed, but they will not be displayed in the default menu's **Help→Keys** menu.
- Not all keys can be remapped on certain operating systems. For example, the Microsoft Windows operating system always displays the Windows Help System when F1 is pressed, and attempts to close the application window when Alt-F4 is pressed.

Key-Fn Trigger Restrictions

Forms Developer ignores Key-Fn triggers in Edit mode.

Key-Others Trigger

Description

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 restrictions apply). When this occurs, however, Forms Developer still displays the function key's default entry in the 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.

Also note:

- The key description shown in the default menu's **Help→Keys** menu will always be for the form-level trigger defined for that key. If there are any lower-level triggers (e.g., block-level triggers) that are also defined for the key, their descriptions will be shown when focus is in the lower level (e.g., the block) and [Show Keys] is pressed, but they will not be displayed in the default menu's **Help→Keys** menu.

Key-Others Trigger Restrictions

Forms Developer 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 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.

Lock-Procedure Trigger

Description

Automatically created by Forms Developer when the lock data source is a stored procedure. This trigger is called when a lock operation is necessary. Think of this as an ON-LOCK trigger that is called by the system instead of doing default lock operations.

Do not modify this trigger.

Enter Query Mode Not applicable.

On Failure

No effect

On-Check-Delete-Master Trigger

Description

Forms Developer creates this trigger automatically when you define a master/detail relation and set the Delete Record Behavior 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

Any command, unrestricted built-ins, restricted built-ins

Enter Query Mode no

On Failure

Prevents the deletion of the current master record

Fires In

Master/Detail Coordination

On-Check-Delete-Master Trigger Examples

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.

```
DECLARE
the_sum NUMBER;
BEGIN
SELECT SUM(dollar_amt)
INTO the_sum
FROM po_distribution
```

```
WHERE po_number = :purchase_order.number;

/* Check for errors */
IF the_sum <> :purchase_order.total THEN
Message('PO Distributions do not reconcile.');
RAISE Form_Trigger_Failure;
ELSIF form_fatal OR form_failure THEN
raise form_trigger_failure;
end if;
END;
```

On-Check-Unique Trigger

Description

During a commit operation, the On-Check-Unique trigger fires when Forms Developer 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 PRIMKEYB property set to Yes, Forms Developer, 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, Forms Developer displays message FRM-40600: Record has already been inserted.

For a record that has been marked for insert, Forms Developer always checks for unique primary key values. In the case of an update, Forms Developer 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

Check Record Uniqueness

Post and Commit Transactions

On-Check-Unique Trigger 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;
ELSIF form_fatal OR form_failure THEN
raise form_trigger_failure;
END IF;
END;
```

Related topics

CHECK_RECORD_UNIQUENESS built-in

On-Clear-Details Trigger

Description

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, unrestricted built-ins, restricted built-ins

Enter Query Mode no

Usage Notes

Forms Developer creates the On-Clear-Details trigger automatically when a Master/Detail block relation is defined.

On Failure

Causes the coordination-causing operation and any scheduled coordination triggers to abort.

Fires In

Master Detail Coordination

On-Close Trigger

Description

Fires when an operator or the application causes a query to *close*. By default, Forms Developer 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 Forms Developer "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

ABORT_QUERY

Close The Query

On-Close Trigger Examples

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_datasource_close('DX110_DEPT');
ELSIF form_fatal OR form_failure THEN
raise form_trigger_failure;
END IF;
END;
```

Related topics

ABORT_QUERY built-in

On-Column-Security Trigger

Description

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

By default, Forms Developer 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, Forms Developer makes the corresponding base table items in the form non-updateable by setting the Update Allowed item property Off dynamically. Forms Developer 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

On-Column-Security Trigger Examples

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');
Set_Item_Property(itm_id,ENABLED,on_or_off);
Set_Item_Property(itm_id,UPDATEABLE,on_or_off);

IF form_fatal OR form_failure THEN
raise form_trigger_failure;
END IF;
END;
```

Related topic

ENFORCE_COLUMN_SECURITY built-in

On-Commit Trigger

Description

Fires whenever Forms Developer 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.

The default COMMIT statement that Forms Developer 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 Forms Developer commit processing to fit the particular requirements of a commit to a non-ORACLE database.

On Failure

Aborts Post and Commit processing

Fires In

Post and Commit Transactions

On-Commit Trigger 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.

```
BEGIN
IF Get_Application_Property(DATA_SOURCE) = 'ORACLE' THEN
Commit_Form;
ELSIF form_fatal OR form_failure THEN
raise form_trigger_failure;
END IF;

/*
** Otherwise, no action is performed
*/
END;
```

Related topic

COMMIT_FORM built-in

On-Count Trigger

Description

Fires when Forms Developer 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, Forms Developer issues the standard query hits message: FRM-40355: Query will retrieve <n> records.

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.
- 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.
- Forms Developer 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

On-Count Trigger 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.

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

Related topic

COUNT_QUERY built-in

On-Delete Trigger

Description

Fires during the Post and Commit Transactions process and replaces the default Forms Developer processing for handling deleted records during transaction posting. 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.

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 Forms Developer processing for handling deleted records during transaction posting.
- To perform the default Forms Developer 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

Forms Developer rolls back to the most recent savepoint

Fires In

Post and Commit Transactions

On-Delete Trigger 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;
IF form_fatal OR form_failure THEN
raise form_trigger_failure;
END IF;
END;
```

Related topic

DELETE_RECORD built-in

On-Error Trigger

Description

An On-Error trigger fires whenever Forms Developer 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 Forms Developer is performing internal navigation, such as during a Commit process.

On Failure

no effect

On-Error Trigger 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
/*
** More tasks here
*/
ELSE
Message(lv_errtyp||'-'||to_char(lv_errcod)||': '||lv_errtxt);
RAISE Form_Trigger_Failure;
END IF;
END;
```

On-Fetch Trigger

Description

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.

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:
    1. Retrieve the appropriate number of records from the non-ORACLE data source, as indicated by the setting of the Records_To_Fetch property.
    2. Create that number of queried records in the current block.
    3. Populate the records with the retrieved data.
    4. 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 signals an end-of-fetch to Forms Developer, 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.
    - until the trigger raises the built-in exception FORM_TRIGGER_FAILURE.

- To perform the default Forms Developer 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

Fetch Records

On-Fetch Trigger Examples

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(blk_name, 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;
IF form_fatal OR form_failure THEN
raise form_trigger_failure;
END IF;
```

```
END;
```

Related topic

CREATE_QUERIED_RECORD built-in

On-Insert Trigger

Description

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

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 Forms Developer processing for handling inserted records during transaction posting.
- To perform the default Forms Developer processing from this trigger, include a call to the INSERT_RECORD built-in.

On Failure

Forms Developer performs the following steps when the On-Insert trigger fails:

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

Fires In

Post and Commit Transactions

Related topic

INSERT_RECORD built-in

On-Lock Trigger

Description

Fires whenever Forms Developer 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.

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 Forms Developer processing for locking rows. For example, for an application intended for use on a single-user system, use the On-Lock trigger to speed processing by bypassing all lock processing. Also, use On-Lock when 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 Forms Developer 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 Forms Developer attempts to put the input focus on the current item. If the current item cannot be entered for some reason, Forms Developer attempts to put the input focus on the previous navigable item.

Fires In

Lock the Row

Related topic

LOCK_RECORD built-in

On-Logon Trigger

Description

Fires once for each logon when Forms Developer normally initiates the logon sequence.

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.
- To create an application that does not require a data source, supply a NULL command to this trigger to bypass the connection to a data source.
- To perform the default Forms Developer processing from this trigger, include a call to the LOGON built-in.

On Failure

Forms Developer attempts to exit the form gracefully, and does not fire the Post-Logon trigger.

Fires In

LOGON

Related topic

Logon (Form Compiler)

On-Logout Trigger

Description

Fires when Forms Developer normally initiates a logout procedure from Forms Developer 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 Forms Developer 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, Forms Developer will not fire other triggers, such as Post-Logout .

Fires In

LOGOUT

Related topic

BRW-15411: Logout failed

On-Message Trigger

Description

Fires whenever Forms Developer would normally cause a message to display and pre-empts the message.

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 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 Forms Developer is performing internal navigation, such as during a Commit process.

On Failure

no effect

On-Message Trigger Examples

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;
IF form_fatal OR form_failure THEN
raise form_trigger_failure;
END IF;

END;
```

On-Populate-Details Trigger

Description

Forms Developer creates this trigger automatically when a Master/Detail relation is defined. It fires when Forms Developer 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.

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, Forms Developer 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.

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, Forms Developer 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.

On Failure

Can cause an inconsistent state in the form.

Fires In

Master/Detail Coordination

On-Rollback Trigger

Description

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

Definition Level form

Legal Commands

SELECT statements, PL/SQL, unrestricted built-ins

Enter Query Mode no

Usage Notes

Use an On-Rollback trigger to replace standard Forms Developer rollback processing.

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

Fires In

CLEAR_FORM

Post and Commit Transactions

Related topics

CLEAR_FORM built-in

ISSUE_ROLLBACK built-in

On-Savepoint Trigger

Description

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

Definition Level form

Legal Commands

SELECT statements, PL/SQL, unrestricted built-ins

Enter Query Mode no

Usage Notes

To perform default Forms Developer 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 Forms Developer 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 Forms Developer would roll back.

Suppress default savepoint processing by setting the Savepoint Mode form document property to Off. When Savepoint Mode is Off, Forms Developer does not issue savepoints and, consequently, the On-Savepoint trigger never fires.

On Failure

no effect

Fires In

CALL_FORM

Post and Commit Transactions

SAVEPOINT

Related topic

Savepoint Mode property

On-Select Trigger

Description

Fires when Forms Developer would normally execute the open cursor, parse, and execute phases of a query, to identify the records in the database that match the current query criteria.

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 Forms Developer processing from this trigger, include a call to the SELECT_RECORDS built-in.

On Failure

no effect

Fires In

EXECUTE_QUERY

Open The Query

On-Select Trigger 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;
END IF;
ELSE
/*
** Perform the default Forms Developer task of opening the query.
*/
Select_Records;
END IF;
```

Related topic

SELECT_RECORDS built-in

On-Sequence-Number Trigger

Description

Fires when Forms Developer would normally perform the default processing for generating sequence numbers for default item values. Replaces the default series of events that occurs when Forms Developer 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, Forms Developer queries the database to get the next value from the SEQUENCE whenever the Create Record event occurs. Suppress or override this functionality with an On-Sequence-Number trigger.

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

On Failure

no effect

Fires In

GENERATE_SEQUENCE_NUMBER

On-Update Trigger

Description

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

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 Forms Developer processing for handling updated records during transaction posting.

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

On Failure

Forms Developer performs the following steps when the On-Update trigger fails:

- sets the error location
- rolls back to the most recently issued savepoint

Fires In

Post and Commit Transactions

Post-Block Trigger

Description

Fires during the Leave the Block process when the focus moves off the current block.

Definition Level form or block

Legal Commands

SELECT statements, unrestricted built-ins

Enter Query Mode no

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.

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, Forms Developer tries to set the target to a particular block, record or item. Failing that, Forms Developer 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, Forms Developer aborts the form.

Fires In

Leave the Block

Post-Block Trigger Restrictions

A Post-Block trigger does not fire when the Validation Unit form document property is set to Form.

Post-Change Trigger

Description

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.
- Forms Developer 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 Forms Developer. 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, Forms Developer 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.

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

Fires In

Validate the Item

Fetch Records

Post-Change Trigger Restrictions

Note that it is possible to write a Post-Change trigger that changes the value of an item that Forms Developer is validating. If validation succeeds, Forms Developer 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.

Related topic

Post-Query Trigger

Post-Database-Commit Trigger

Description

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 Forms Developer 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, Forms Developer has already moved past the point at which a successful rollback operation can be initiated as part of a failure response.

Fires In

Post and Commit Transactions

Post-Database-Commit Trigger 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;
```

Related topics

Post-Forms-Commit Trigger

Post-Delete Trigger

Description

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

Forms Developer performs the following steps when the Post-Delete trigger fails:

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

Fires In

Post and Commit Transactions

Post-Form Trigger

Description

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

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

Leave the Form

Post-Forms-Commit Trigger

Description

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 Forms Developer 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, Forms Developer 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: Forms Developer issues a ROLLBACK and decrements the internal Savepoint counter.

Fires In

Post and Commit Transactions

Post-Forms-Commit Trigger 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.

```
/*

** 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;
```

Related topic

Post-Database-Commit Trigger

Post-Insert Trigger

Description

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

Forms Developer performs the following steps when the Post-Insert trigger fails:

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

Fires In

Post and Commit Transactions

Post-Logon Trigger

Description

Fires after either of the following events:

- The successful completion of Forms Developer 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, Forms Developer exits the form, and returns to the operating system.
- If the trigger fails after a successful logon, Forms Developer raises the built-in exception FORM_TRIGGER_FAILURE .

Fires In

LOGON

Post-Logon Trigger 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.

```
BEGIN
User_Exit('LogCrypt '||
USER||' ' ||
TO_CHAR(SYSDATE,'YYYYMMDDHH24MISS'));
END;
```

Related topics

Logon (Form Compiler)

Post-Logout Trigger

Description

Fires after either of the following events:

- Forms Developer 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 Forms Developer 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

LOGOUT

Related topic

BRW-15411: Logout failed

Post-Query Trigger

Description

When a query is open in the block, the Post-Query trigger fires each time Forms Developer 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, Forms Developer 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 property of each record to QUERY in the Post-Query trigger. To set record status programmatically, use SET_RECORD_PROPERTY .

On Failure

Forms Developer 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, Forms Developer closes the query and waits for the next operator action.

Fires In

Post-Query Trigger Examples

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

Description

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 to perform an action whenever the operator or the application moves the input focus from one record to another. For example, to set a visual attribute for an item as the operator scrolls down through a set of records, put the code within this trigger.

On Failure

The input focus stays in the current record.

Fires In

Leave the Record

Post-Record 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.

Post-Select Trigger

Description

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 such as an action based on the number of records that match the query criteria.

On Failure

no effect

Fires In

Execute the Query

Open the Query

Post-Text-Item Trigger

Description

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.

On Failure

Navigation fails and focus remains in the text item.

Fires In

Leave the Item

Post-Text-Item Trigger Restrictions

- 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).
- If "Validation Unit" property is set to a value other than "Item" or "Default", then POST-TEXT-ITEM does not fire.

Post-Update Trigger

Description

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, unrestricted built-ins

Enter Query Mode no

Usage Notes

Use a Post-Update trigger to audit transactions.

On Failure

Forms Developer performs the following steps when the Post-Update trigger fails:

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

Fires In

Post and Commit Transactions

Pre-Block Trigger

Description

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

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

On Failure

Navigation fails and focus remains in the source item.

Fires In

Enter the Block

Pre-Block Trigger 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.

Pre-Commit Trigger

Description

Fires once during the Post and Commit Transactions process, before Forms Developer processes any records to change. Specifically, it fires after Forms Developer determines that there are inserts, updates, or deletes in the form to post or commit, but before it commits the changes. 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, at any time 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 Forms Developer 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

Post and Commit Transactions

Pre-Delete Trigger

Description

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: Forms Developer creates a Pre-Delete trigger automatically for any master-detail relation that has the Delete Record Behavior 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

Forms Developer performs the following steps when the Pre-Delete trigger fails:

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

Fires In

Post and Commit Transactions

Pre-Form Trigger

Description

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

Forms Developer leaves the current form and fires no other triggers.

Fires In

Enter the Form

Related topic

When-New-Form-Instance Trigger

Pre-Insert Trigger

Description

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

Forms Developer performs the following steps when the Pre-Insert trigger fails:

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

Fires In

Post and Commit Transactions

Pre-Insert Trigger Examples

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

Description

Fires just before Forms Developer 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 Forms Developer 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 Forms Developer is attempting to execute the LOGON built-in from within the trigger, Forms Developer raises the FORM_TRIGGER_FAILURE exception.

Fires In

LOGON

Pre-Logout Trigger

Description

Fires once before Forms Developer 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, the COPY built-in cannot be called from a Pre-Logout trigger because Pre-Logout fires after the Leave the Form event. Because the form is no longer accessible at this point, the COPY operation is not possible.

On Failure

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

- If Forms Developer is exiting the form and the trigger fails, the form is exited gracefully, but no other triggers are fired.
- If the trigger fails while Forms Developer is attempting to execute the LOGOUT built-in from within the trigger, Forms Developer raises the FORM_TRIGGER_FAILURE exception.

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

Fires In

LOGOUT

Pre-Popup-Menu Trigger

Description

This trigger is called when a user causes a pop-up menu to be displayed. (In a Microsoft Windows environment, this occurs when a user presses the right mouse button.) Actions defined for this trigger are performed before the pop-up menu is displayed.

Legal Commands

SELECT statements, restricted built-ins, unrestricted built-ins

Enter Query Mode yes

Usage Notes

Use this trigger to enable or disable menu items on a pop-up menu before it is displayed.

On Failure

No effect

Pre-Query Trigger

Description

Fires during Execute Query or Count Query processing, just before Forms Developer 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

COUNT_QUERY

EXECUTE_QUERY

Open the Query

Prepare the Query

Pre-Query Trigger Example

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

```
/*
** 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
*/

BEGIN

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
** 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 check box named 'Show_Term' in a control
** block named 'Switches'.
*/
IF Check box_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

Description

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.

On Failure

Navigation fails and focus remains in the current item.

Fires In

Enter the Record

Pre-Record Trigger 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.

Pre-Record Trigger 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.

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

Pre-Select Trigger

Description

Fires during Execute Query and Count Query processing, after Forms Developer 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. The current query fetched no records from the table. The table is empty, or it contains no records that meet the query's search criteria.

Fires In

EXECUTE_QUERY

Open the Query

Prepare the Query

Related topic

SYSTEM.LAST_QUERY examples

Pre-Text-Item Trigger

Description

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.

On Failure

Navigation fails and focus remains in the current item.

Fires In

Enter the Item

Pre-Text-Item Trigger 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.

Pre-Update Trigger

Description

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

Forms Developer performs the following steps when the Pre-Update trigger fails:

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

Fires In

Post and Commit Transactions

Pre-Update Trigger 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.

```
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;

/*
** 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;
```

Query-Procedure Trigger

Description

Automatically created by Forms Developer when the query data source is a stored procedure. This trigger is called when a query operation is necessary. Think of this as an On-Query trigger that is called by the system instead of doing default query operations.

Do not modify this trigger.

Enter Query Mode See Usage Notes

Usage Notes

When constructing a query, any of the items may be used, but the Query Data Source Columns property must be set so that those items can be passed to the query stored procedure. Then, the query stored procedure has to use those values to filter the data. This means that the enter query mode does not happen automatically unless you specify it.

On Failure

No effect

Update-Procedure Trigger

Description

Automatically created by Forms Developer when the update data source is a stored procedure. This trigger is called when a update operation is necessary. Think of this as an On-Update trigger that is called by the system instead of doing default update operations.

Do not modify this trigger.

Enter Query Mode Not applicable.

On Failure

No effect

User-Named Trigger

Description

A user-named trigger is a trigger defined in a form by the developer. User-Named triggers do not automatically fire in response to a Forms Developer event, and must be called 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');
```

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

Usage Notes

- User-named PL/SQL subprograms can be written to perform almost any task for which one might use a user-named trigger.
- 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 to execute user-named subprograms defined in a form document 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 document.) 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

Description

Fires when an operator selects a button, by clicking with a mouse, or using the keyboard.

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

When-Button-Pressed Trigger 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

Description

Fires when an operator changes the state of a check box, either by clicking with the mouse, or using the keyboard.

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.

On Failure

no effect

When-Clear-Block Trigger

Description

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

Note that the When-Clear-Block trigger does not fire when Forms Developer 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 Forms Developer 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 reference to a specific record, its value is always accurate.

On Failure

no effect on the clearing of the block

Fires In

CLEAR_BLOCK

COUNT_QUERY

ENTER_QUERY

Open the Query

Related topics

CLEAR_FORM built-in

GET_RECORD_PROPERTY built-in

When-Create-Record Trigger

Description

Fires when Forms Developer 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, Forms Developer 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 Forms Developer 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 at design-time.

On Failure

Prevents the new record from being created. Returns to the previous location, if possible.

Fires In

CREATE_RECORD

When-Create-Record Trigger Examples

Example

This example assigns data-driven or calculated default values without marking the record as changed.

```
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;
```

Related topic

CREATE_RECORD built-in

When-Custom-Item-Event Trigger

Description

Fires whenever a JavaBean custom component in the form causes the occurrence of an event.

Definition Level:

form, block, item

Legal Commands:

unrestricted built-ins, restricted built-ins

Enter Query Mode:

yes

Usage Notes

Use a When-Custom-Item-Event trigger to respond to a selection or change of value of a custom component. 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 custom control.

Control event names are case sensitive.

On Failure:

no effect

When-Custom-Item-Event Trigger JavaBeans Example

This is an example of a procedure called by a When-Custom-Item-Event trigger in a form that uses two JavaBeans.

The trigger is fired by the dispatching of a custom event in one of the JavaBeans, in response to a change in value.

```
CustomEvent ce = new CustomEvent(mHandler, VALUECHANGED);
dispatchCustomEvent(ce);
```

In the form, the When_Custom_Item_Event trigger that is attached to this JavaBean's Bean Area item is automatically fired in response to that custom event.

In the trigger code, it executes the following procedure. Note that this procedure picks up the new values set in the JavaBean container (a new animation rate, for example) by accessing the System.Custom_Item_Event_Parameters.

In this example, the procedure then uses the `Set_Custom_Item_Property` built-in to pass those values to the other JavaBean.

```
PROCEDURE Slider_Event_Trap IS
BeanHdl Item;
BeanValListHdl ParamList;
paramType Number;
EventName VarChar2(20);
CurrentValue Number(4);
NewAnimationRate Number(4);
Begin
-- Update data items and Display fields with current radius
information
BeanValListHdl :=
get_parameter_list(:SYSTEM.Custom_Item_Event_Parameters);
EventName := :SYSTEM.Custom_Item_Event;
:event_name := EventName;
if (EventName = 'ValueChanged') then
get_parameter_attr(BeanValListHdl,'Value',ParamType, CurrentValue);
NewAnimationRate := (300 - CurrentValue);
:Animation_Rate := NewAnimationRate;
set_custom_item_property('Juggler_Bean','SetAnimationRate',
NewAnimationRate);
elsif (EventName = 'mouseReleased') then
get_parameter_attr(BeanValListHdl,'Value',ParamType, CurrentValue);
set_custom_item_property('Juggler_Bean','SetAnimationRate',
CurrentValue);
end if;
End;
```

When-Database-Record Trigger

Description

Fires when Forms Developer first marks a record as an insert or an update. That is, the trigger fires as soon as Forms Developer 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 Trigger

Description

Fires when navigation between forms takes place, such as when the user changes the focus to another loaded form.

Definition Level form

Legal Commands:

unrestricted built-ins, 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

When-Form-Navigate Trigger Example

This is an example of a procedure that can be called when Forms Developer fires the When-Form-Navigate Trigger.

```
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

Description

Fires when an operator uses the mouse to:

- single-click on an image item
- double-click on an image item

  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

Description

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

Description

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 drop-down lists or combo box style list items. The display style of a list item is determined by the List Style property.

On Failure

no effect

Related topic

List Style property

When-List-Changed Trigger

Description

Fires when an end user 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 end user 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 is changed directly by the end user. The When-List-Changed trigger is not fired if the value of the list is changed programmatically such as by using the DUPLICATE_ITEM built-in, or if the end user causes a procedure to be invoked which changes the value. For example, the When-List-Changed trigger will not fire if an end user duplicates the item using a key mapped to the DUPLICATE_ITEM built-in.

On Failure

no effect

Related topic

List Style property

When-Mouse-Click Trigger

Description

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 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.

On Failure

no effect

When-Mouse-DoubleClick Trigger

Description

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 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.

On Failure

no effect

When-Mouse-DoubleClick Trigger Example

Assume that an 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.

```
/*
** 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

Description

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 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.

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

On Failure

no effect

When-Mouse-Down Trigger Restrictions

Depending on the window manager, navigational code within a When-Mouse-Down trigger may fail. For example on Microsoft 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.

When-Mouse-Enter Trigger

Description

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

- if attached to the form, when the mouse enters any canvas 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.
- Do not use the When-Mouse-Enter trigger on a canvas that is larger than the window. Iconic buttons and items on the canvas below the initial window cannot be selected. The user is able to scroll the canvas to see the items. However, as soon as the mouse enters that area, the trigger fires and returns focus to the previous target, so the user is never able to click on those items.
- Changing a tooltip's property in a When-Mouse-Enter trigger cancels the tooltip before it is ever shown.
- 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 message box, 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

Description

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

- if attached to the form, when the mouse leaves any canvas 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.

On Failure

no effect

When-Mouse-Move Trigger

Description

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 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

Description

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 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

Description

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

On Failure

no effect

Fires In

Return for Input

Related topics

When-New-Form-Instance Trigger

When-New-Item-Instance Trigger

When-New-Record-Instance Trigger

When-New-Form-Instance Trigger

Description

At form start-up, Forms Developer 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

On Failure

no effect

Fires In

Run the Form

When-New-Form-Instance Trigger Restrictions

- When a new form is called, it will appear in the default x-y position on the screen. If this is not the desired position, you can change the x-y coordinates. However, they cannot be changed in this When-New-Form-Instance trigger. (This trigger fires too late in the sequence.) To change the coordinates, use the Pre-Form trigger.

When-New-Form-Instance Trigger Example

This example calls routine to display dynamic images, starts a timer to refresh the on-screen clock, and queries the first block.

```
BEGIN
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

Description

Fires when the input focus moves to an item. Specifically, it fires after navigation to an item, when Forms Developer 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.

On Failure

no effect

Fires In

Return for Input

When-New-Item-Instance Trigger Restrictions

The conditions for firing this trigger are *not* met under the following circumstances:

- Forms Developer 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 Forms Developer menu

When-New-Record-Instance Trigger

Description

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 Forms Developer is ready to accept input in a record that is different than the record that previously had input focus.

Fires whenever Forms Developer 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 Forms Developer instantiates a new record. For example, when an operator presses [Down] to scroll through a set of records, Forms Developer fires this trigger each time the input focus moves to the next record, in other words, each time Forms Developer instantiates a new record in the block.

On Failure

no effect

Fires In

Return for Input

Related topics

When-New-Block-Instance Trigger

When-New-Form-Instance Trigger

When-New-Item-Instance Trigger

When-Radio-Changed Trigger

Description

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 using the keyboard.

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 an item 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. 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.

On Failure

no effect

When-Remove-Record Trigger

Description

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

Definition Level form, block

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 Forms Developer clears or deletes a record.

On Failure

Forms Developer navigates to the block level with or without validation depending on the current operation, and puts the cursor at the target block.

Fires In

CLEAR_RECORD

DELETE_RECORD

When-Tab-Page-Changed

Description

Fires whenever there is explicit item or mouse navigation from one tab page to another in a tab canvas.

Definition Level form

Legal Commands

unrestricted built-ins, restricted built-ins

Enter Query Mode no

Usage Notes

- Use a When-Tab-Page-Changed trigger to perform actions when any tab page is changed during item or mouse navigation.
- When-Tab-Page-Changed fires only when tab page navigation is explicit; it does not respond to implicit navigation. For example, the trigger will fire when the mouse or keyboard is used to navigate between tab pages, but the trigger will not fire if an end user presses [Next Item] (Tab) to navigate from one field to another field in the same block, but on different tab pages.
- When-Tab-Page-Changed does not fire when the tab page is changed programmatically.

On Failure

no effect

When-Tab-Page-Changed Examples

```
/* Use a When-Tab-Page-Changed trigger to dynamically

** change a tab page's label from lower- to upper-case
** (to indicate to end users if they already have
** navigated to the tab page):
```

```
*/
DECLARE
tp_nm VARCHAR2(30);
tp_id TAB_PAGE;
tp_lb VARCHAR2(30);

BEGIN
tp_nm := GET_CANVAS_PROPERTY('emp_cvs', topmost_tab_page);
tp_id := FIND_TAB_PAGE(tp_nm);
tp_lb := GET_TAB_PAGE_PROPERTY(tp_id, label);

IF tp_lb LIKE 'Sa%' THEN
SET_TAB_PAGE_PROPERTY(tp_id, label, 'SALARY');
ELSIF tp_lb LIKE 'Va%' THEN
SET_TAB_PAGE_PROPERTY(tp_id, label, 'VACATION');
ELSE null;
END IF;
END;
```

When-Timer-Expired Trigger

Description

Fires when a timer expires.

Definition Level form

Legal Commands

SELECT statements, unrestricted built-ins, restricted built-ins

Enter Query Mode yes

Usage Notes

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

- 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.

On Failure

no effect

Fires In

Process Expired Timer

When-Timer-Expired Trigger Restrictions

A When-Timer-Expired trigger will not fire when the user is currently navigating a menu.

When-Timer-Expired Trigger Examples

Example

The following example displays a message box each time a repeating timer expires. The following example is from a telemarketing application, in which sales calls are timed, and message boxes are displayed to prompt the salesperson through each stage of the call. The message box is displayed each time a repeating timer expires.

```
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
Set_Alert_Property(alert_id, ALERT_MESSAGE_TEXT, 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
Set_Alert_Property(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
Set_Alert_Property(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;
:GLOBAL.timer_count = 2;
END;
END;
```

When-Tree-Node-Activated Trigger

Description

Fires when an operator double-clicks a node or presses Enter when a node is selected.

Definition Level form or block

Legal Commands

SELECT statements, unrestricted built-ins, restricted built-ins

Enter Query Mode yes

Usage Notes

- SYSTEM.TRIGGER_NODE is the node the user clicked on. SYSTEM.TRIGGER_NODE returns a value of type NODE.
- No programmatic action will cause the When-Tree-Node-Activated trigger to fire. Only end-user action will generate an event.

On Failure

no effect

When-Tree-Node-Expanded Trigger

Description

Fires when a node is expanded or collapsed.

Definition Level form or block

Legal Commands

SELECT statements, unrestricted built-ins, restricted built-ins

Enter Query Mode yes

Usage Notes

- SYSTEM.TRIGGER_NODE is the node the user clicked on. SYSTEM.TRIGGER_NODE returns a value of type NODE.
- No programmatic action will cause the When-Tree-Node-Espanded trigger to fire. Only end-user action will generate an event.

On Failure

no effect

When-Tree-Node-Selected Trigger

Description

Fires when a node is selected or deselected.

Definition Level form or block

Legal Commands

SELECT statements, unrestricted built-ins, restricted built-ins

Enter Query Mode yes

Usage Notes

- SYSTEM.TRIGGER_NODE is the node the user clicked on. SYSTEM.TRIGGER_NODE returns a value of type NODE.
- No programmatic action will cause the When-Tree-Node-Selected trigger to fire. Only end-user action will generate an event.

On Failure

no effect

When-Validate-Item Trigger

Description

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 Forms Developer default item validation processing.
- It is possible to write a When-Validate-Item trigger that changes the value of an item that Forms Developer is validating. If validation succeeds, Forms Developer 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 setting you choose for the Defer Required Enforcement property can affect the When-Validate-Item trigger. See Defer_Required_Enforcement for details.

On Failure

If fired as part of validation initiated by navigation, navigation fails, and the focus remains on the original item.

Fires In

Validate the Item

When-Validate-Item Trigger Example

The following example finds the commission plan in the COMMPLAN 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 COMMPLAN table is located, the description of the COMMPLAN 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;
```

Related topic

When-Validate-Record Trigger

When-Validate-Record Trigger

Description

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 Forms Developer default record validation processing.

Note that it is possible to write a When-Validate-Record trigger that changes the value of an item in the record that Forms Developer is validating. If validation succeeds, Forms Developer 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

If fired as part of validation initiated by navigation, navigation fails, and the focus remains on the original item.

Fires In

Validate the Record

When-Validate-Record Trigger 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.

```
/* 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

Description

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 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

Related topics

When-Window-Closed Trigger

When-Window-Deactivated Trigger

GET_WINDOW_PROPERTY built-in

SYSTEM.EVENT_WINDOW examples

When-Window-Closed Trigger

Description

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 hide the window that contains the current item.

On Failure

no effect

When-Window-Closed Trigger 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);
```

Related topic

SYSTEM.EVENT_WINDOW examples

When-Window-Deactivated Trigger

Description

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

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.
- Note that if this form opens another form, this deactivate trigger does not immediately fire. Instead, it will fire later when control returns to this form. (Assuming this window also has an activate trigger, then when control returns to this form, first the deactivate trigger fires followed immediately by the activate trigger.)

On Failure

no effect

Related topics

When-Window-Activated Trigger

When-Window-Closed Trigger

GET_WINDOW_PROPERTY built-in

SYSTEM.EVENT_WINDOW examples

When-Window-Resized Trigger

Description

Fires when a window is resized, either by the operator or programmatically through a call to RESIZE_WINDOW or SET_WINDOW_PROPERTY. (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 the input focus in an item on the target window.
- Maintain certain visual standards by resetting window size if the window was improperly resized.

Related topics

When-Window-Activated Trigger

RESIZE_WINDOW built-in

SET_WINDOW_PROPERTY built-in

SYSTEM.EVENT_WINDOW examples