Form Personalization (Release 12)

This document describes the use of the Form Personalization feature, which allows you to declaratively alter the behavior of Forms-based screens.

The following topics are covered:

Overview

The Form Personalization feature allows you to declaratively alter the behavior of Forms-based screens, including changing properties, executing builtins, displaying messages, and adding menu entries.

For each function (a form running in a particular context based on parameters passed to it), you can specify one or more Rules. Each Rule consists of an Event, an optional Condition, the Scope for which it applies, and one or more Actions to perform.

Rules can be specified as acting either at the Function level (the default) or at the Form level. In this latter case they are applied for all functions that run the form. When creating rules at the Form level, pay close attention to the Sequence number and how it will blend with other rules at the function level for the same form.

An Event is a trigger point within a form, such as startup (WHEN-NEW-FORM-INSTANCE), or when focus moves to a new record (WHEN-NEW-RECORD-INSTANCE). There are standard events that almost every form sends, and certain forms send additional product-specific events.

The Scope is evaluated based on the current runtime context to determine if a Rule should be processed or not. The Scope can be at the Site, Responsibility, User, or Industry level. Each Rule can have one or more Scopes associated with it.

NOTE: the scope of 'Industry' is reserved for future use.

The Condition is an optional SQL code fragment that is evaluated when the Event occurs; if it evaluates to TRUE then the Actions are processed.

Each Action consists of one of the following:

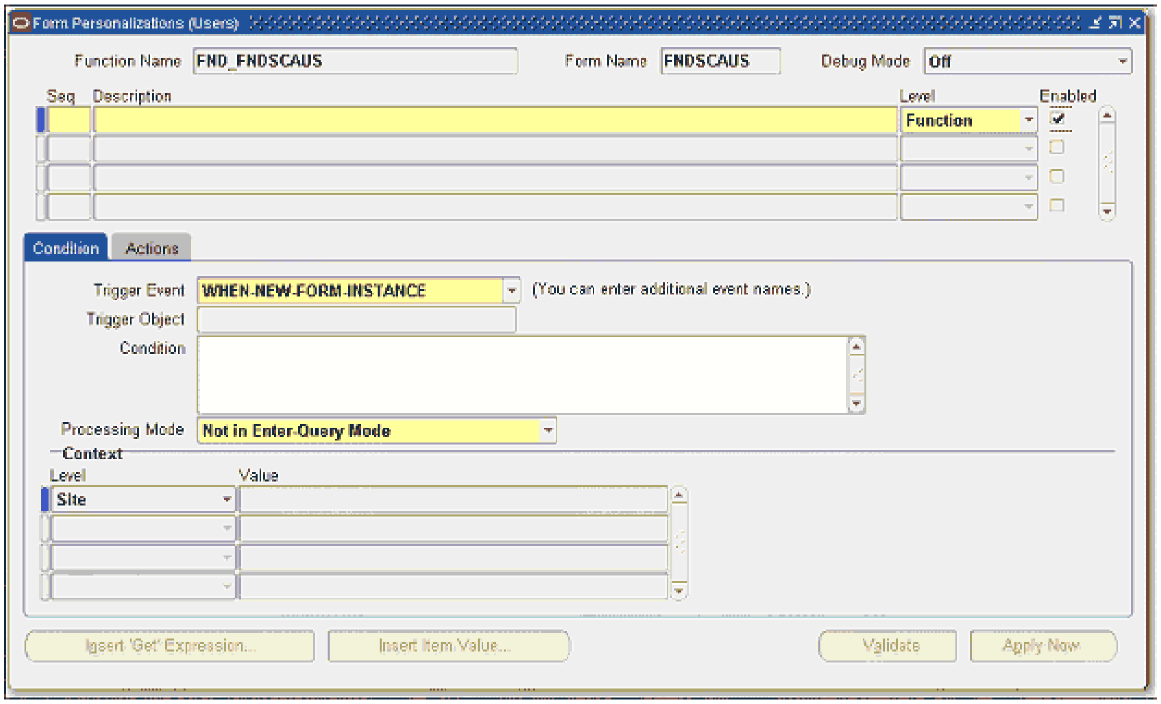
- setting a Property, such as making a field Required or hiding a Tab page
- executing a Builtin, such as GO_BLOCK, DO_KEY or FND_FUNCTION.EXECUTE
- displaying a Message
- enabling a menu entry

Once Rules are defined, when the target function is run then the Rules are automatically applied as events occur within that form.

Although the Form Personalization feature is declarative, the intended audience is a person familiar with Oracle Forms including the PL/SQL programming language, and the *Oracle E-Business Suite Developer's Guide*. Additionally, any change made could interfere with the base code of a form (the code that Oracle ships), thus the Support statements discussed later in this chapter must be followed diligently.

<u>Using the Personalization Form</u>
To create personalizations for a particular function, first invoke that function from the Navigation menu. While in the form, choose Help->Diagnostics->Custom Code-> Personalize from the pulldown menu. This menu entry is secured by the FND_HIDE_DIAGNOSTICS (Hide Diagnostics menu entry) and DIAGNOSTICS (Utilities:Diagnostics) profiles, as are most other entries on the Diagnostics menu.

The Personalization form will open and automatically query existing Rules for that function. After making changes, Save them then close and re-run the function to have them take effect. You can also Validate or Apply certain changes immediately to test them without having to re-run the target form by pressing the 'Validate' or 'Apply Now' buttons.


Form Personalizations (Users)

Function Name: FND_FNDSCAUS
Form Name: FNDSCAUS
Debug Mode: Off

Seq	Description	Level	Enabled
		Function	✓

Condition | Actions

Trigger Event: WHEN-NEW-FORM-INSTANCE (You can enter additional event names.)
Trigger Object:
Condition:
Processing Mode: Not in Enter-Query Mode

Context

Level	Value
Site	

Figure 1: The Personalization screen, when opened from the Users form (Function Name FND_FNDSCAUS)

Each Rule consists of the following fields:

Seq: The sequence in which rules will be processed. This is a value between 1 and 100, with 1 being processed first. The sequence of rules does not have to be unique. Note that there is an interaction with the Trigger Event field, described below.

Description: Use this field to document the personalization you are making.

Enabled: Uncheck this checkbox to temporarily disable processing of a Rule.

The following fields appear on the Condition tab:

Trigger Event: Select the event at which you want the Rule to be processed. You can pick from the list of standard events, or type in a specific event unique to the form. Note that this field is a Combobox, which is a hybrid of a Poplist and Text Item. Rules are processed first by matching the Event, then by their Sequence number.

Trigger Object: Depending on the Trigger Event, this field may be Disabled, or Enabled and Required in which case it will validate against a List of Values. For example, if Trigger Event WHEN-NEW-ITEM-INSTANCE is selected, then you must enter a specific *block.field* for that trigger to be processed.

Condition: This is an optional SQL code fragment that is evaluated when the Event occurs; if it evaluates to TRUE then the Actions are processed. The condition can contain any of the following:

- SQL functions and operators, such as AND, OR, TO_CHAR, DECODE, and NVL
- References to bind variables (:block.field), including :system, :global and :parameter values. Use the 'Add Item...' button to assist with item names.
- Calls to server-side functions that do not have OUT parameters

The entire fragment must be syntactically correct, and can be tested with the 'Validate' button, which will evaluate it in the current context of the target form. If the evaluation fails, the processing engine will return an ORA error as if the condition had been part of a SQL expression.

Some examples:

Condition	**Comments**
:users.user_name is not null and :users.description is null	The rule will be processed if the user_name field has a value and the description field does not
sysdate >= to_date('1-1-2005', 'DD-MM-RRRR')	The rule will be processed if the current date is equal to or after January 1, 2005.

<u>You can "Get" a property in your expression of your condition.</u>

Conditions can refer to properties of objects using a SPEL syntax (Simplest Possible Expression Language). For example, this enables you to build a Condition that tests if a field is displayed or not. These expressions take the following general format:

${objectType.objectName.Property}

Internally, the SPEL expression is a cover for Oracle Forms builtins like GET_ITEM_PROPERTY, GET_BLOCK_PROPERTY, etc. Additionally, the SPEL expressions support retrieving Profile values, Message Dictionary text, and Local Variables (described later).

A new window, activated by the "Insert 'Get' Expression…" button, is provided to construct the expression automatically. When complete, the expression is inserted into the Condition field from where the window was launched. You can manually type the expression too; however, we strongly recommend using the window to build it to reduce mistakes.

All processing drives off the existence of the string '${', so that string is no longer valid in any Condition except for this purpose.

An example:

Type	**String typed in Personalization form**	**Result at runtime**
Condition	${item.block.field.required} = 'TRUE'	{True or False, depending on the current Required property of field block.field}

For more information on using the 'Get' expression, see: Evaluation of Strings, below.

Fire in Enter-Query Mode: Specify whether the Rule(s) should be applied while not in Enter-

Query mode (the default), only in Enter-Query mode, or in Both modes.

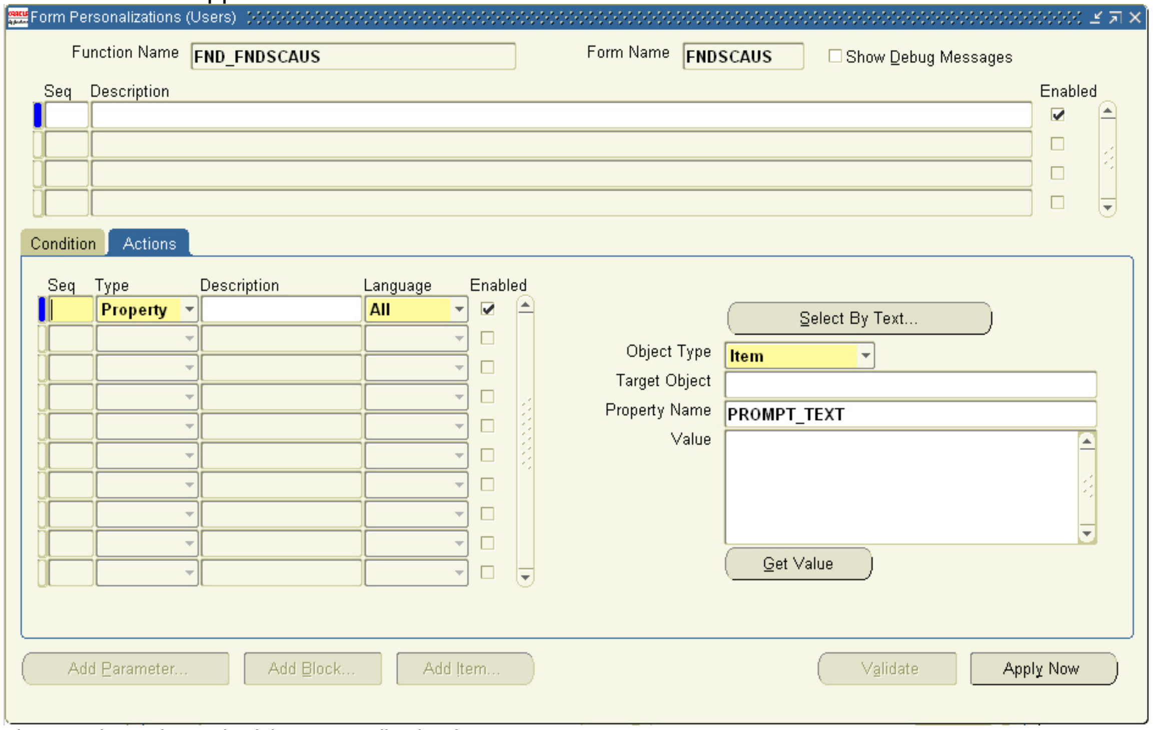
Each Rule consists of one or more Scope rows, and one or more Actions. If a Rule has no Scope rows or Action rows, it is not processed. Note that upon saving a Rule, if no Scope rows have been entered the form will automatically create a row at the Site level. If any scope matches the current runtime context then the Rule will be processed.

The following Scope fields appear in the Context region of the Condition tab:

Level: Select the level at which you want the rule to be applied, either Site, Responsibility, User, or Industry.

Value: Based on the Level, either Disabled, or Enabled and Required in which case it will validate against a List of Values.

All Action fields appear on the Actions tab.



Form Personalizations (Users)

Function Name FND_FNDSCAUS Form Name FNDSCAUS Show Debug Messages

Seq	Description	Enabled
		✓

Condition | Actions

Seq	Type	Description	Language	Enabled
	Property		All	✓

Object Type: Item
Target Object:
Property Name: PROMPT_TEXT
Value:

Figure 2: the Actions tab of the Personalization form

Seq: The sequence in which actions will be processed within that Rule. This is a value between 1 and 100, with 1 being processed first. The sequence does not have to be unique. All of the actions associated with a particular rule are processed as a group, then the next rule (if any) is processed. This feature is particularly useful when moving items, in which case a canvas may have to be resized first before an X Position can be altered.

Type: the type of action to take:

- Property: allows you to select a specific object, a property of that object, and specify a new value for that property
- Builtin: allows execution of a standard Forms Builtin, such as GO_BLOCK or DO_KEY
- Message: displays a message in one of several styles
- Menu: enables a special menu entry, defining its label, icon name and which blocks it applies to.

Description: Use this field to document the personalization action you are making.

Language: Specify 'All' to have the action processed for any language, or select a specific language. Typically text-related personalizations would be applied for a specific language.

Enabled: Uncheck this checkbox to temporarily disable processing of the action.

Apply Now: For several Types, this button will be enabled. It allows you to apply the change immediately to the target form to test its effect. Note that the actual effect that will occur during normal runtime execution of rules may be different, due to timing of triggers and other considerations.

The following buttons are enabled conditionally based on the Type field:

Add Parameter…: List of Values that displays currently used parameters. Applies to the builtin FND_FUNCTION.EXECUTE only.

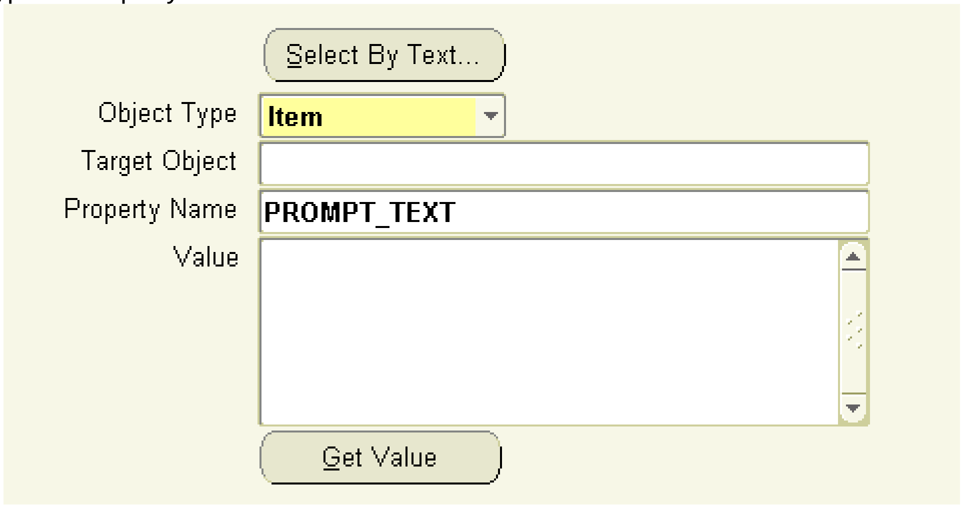
Add Block…: List of Values that displays block names.

Add Item…: List of Values that displays item names.

Validate: Used to test if the syntax of your string is valid. If the evaluation fails, the processing engine will return an ORA error as if the string had been part of a SQL expression. Otherwise, it will display the text exactly as it would appear at runtime in the current context.

The following fields appear conditionally based on the Type field:

For a Type of 'Property':



Object Type	Item
Target Object	
Property Name	PROMPT_TEXT
Value	

Figure 3: The fields associated with an action of 'Property'

Select By Text: This button allows you to select an object based on text appearing on the screen at the point in time that you invoke the Personalization form, including any changes that current rules might have performed. For example, if you want to change a field with the current prompt of 'Item Number', you should see 'Item Number' in this list, and selecting it will automatically fill in the Object Type and Target Object fields.

> **Note:** As the visible text for objects may be identical across multiple windows of the same form, it is sometimes difficult to determine exactly which object you wish to personalize.

The “Select By Text” list includes the window title in parentheses following the normal object text.

Object Type: the type of object, including Item, Window, Block, Tab, Canvas, Radio button, List of Value (LOV), View, Global variable, Local variable, or Parameter. A Local variable is similar to a Global variable, except that it creates a variable local to the current form, with a maximum length of 4000 bytes (Global variables are limited to 255 bytes).

Target Object: based on the Object Type, the internal name of the object. For Object Types of GLOBAL and PARAMETER, the Target Object name must not include those keywords. For example, to refer to GLOBAL.XX_MY_VARIABLE, only enter XX_MY_VARIABLE.

Note: If a Tab page does not appear in the list of objects its name can be entered manually. This is meant to address the known limitation that some tab names cannot be automatically detected at runtime. You will need to open the form in the Oracle Forms Builder to determine these tab names, and then you can type it into this Combobox.

Property Name: based on the Object Type, the properties that can be personalized. The Object Type of Item supports a vast array of properties including:

- Item-level properties, which set the property for all instances of that object.
- Item-instance properties, which set the property for the current record of that block using set_item_instance_property()
- Applications cover properties, which are a hybrid of multiple item and item-instance level properties. These are fully documented in the *Oracle E-Business Suite Developer’s Guide*.

For the LOV object type, simple LOV properties can be modified using Form Personalization, including GROUP_NAME, HEIGHT, TITLE, WIDTH, X_POS and Y_POS. At this point in time, the property RECORD_GROUP_QUERY, which would allow you to dynamically change the underlying query of an LOV, cannot be changed; this is being considered for a future release.

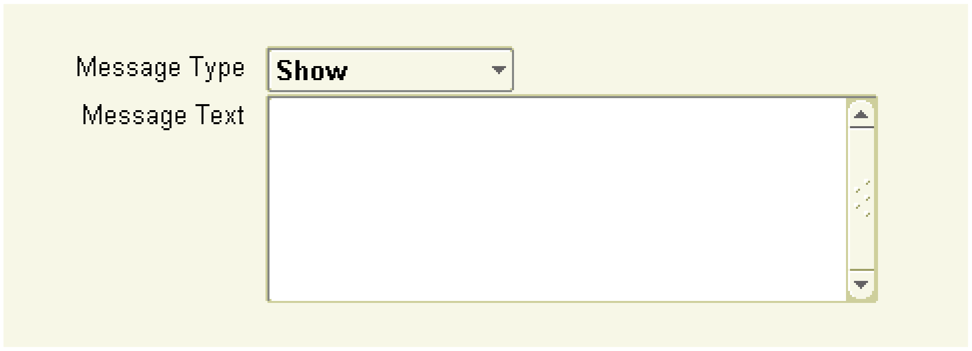
For the Block object type, the property ALLOW_NON_SELECTIVE_SEARCH is provided. For those blocks where a “Blind Query” has been prevented by development for performance reasons, this new property allows you to re-enable “Blind Query” on a block-by-block basis.

Also for the Block object type, the property EXPORT_HIDDEN_CANVASES allows for exporting block data that is on hidden canvases. See: Exporting Data on Hidden Canvases, below.

Value: the new value. The appearance and validation of this field changes based on whether the property accepts Boolean values (True/False), numbers, a restricted set of values, or a string (See Evaluation of Strings below)

Get Value: This button gets the current property value of the object.

For a Type of 'Message':


Message Type: Show

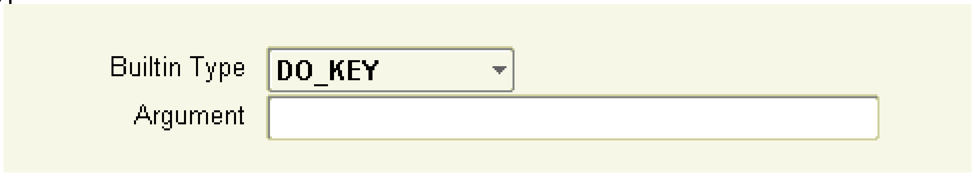
Message Text:

Figure 4: The fields associated with an action of 'Message'

Message Type: either 'Show', 'Hint', 'Warn', 'Error', or 'Debug'. 'Error' and 'Warn' if the user selects the 'Cancel' button will raise a form_trigger_failure after executing, and stop all further processing. Messages of type 'Debug' will only be displayed if the 'Show Debug Messages' checkbox is checked.

Message Text: The text you want to display in the message. (See Evaluation of Strings below)

For a Type of 'Builtin':


Builtin Type: DO_KEY

Argument:

Figure 5: The fields associated with an action of 'Builtin'

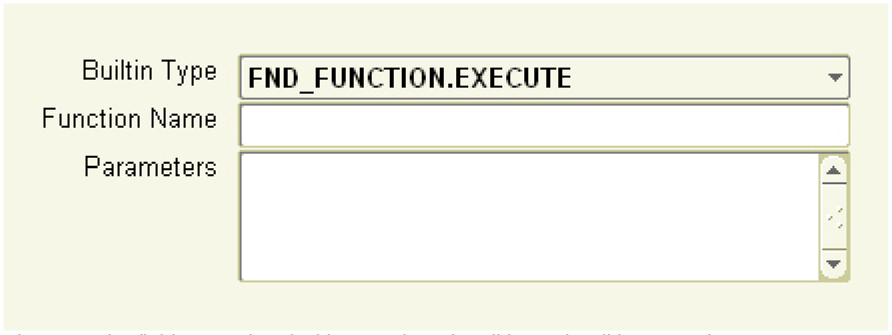
Builtin Type: The name of the builtin. Examples include:

- GO_ITEM
- DO_KEY
- GO_BLOCK
- RAISE FORM_TRIGGER_FAILURE
- FORMS_DDL
- 'Launch a URL' (formerly called FND_UTILITIES.OPEN_URL). This builtin allows an HTML target window to be set. By default the target is '_BLANK', which will open a new browser window. You can select from 4 pre-defined targets, or enter your own target name.
- 'Launch a Function' (formerly called FND_FUNCTION.EXECUTE). The function can be selected by either its internal (developer) name or its user-friendly name.
- 'Launch SRS Form' allows the Submit Concurrent Request form (SRS) to be run in the context of a specific program. This takes a single argument of the program name. Note that no additional security is applied to this program name – whatever you supply will be available to the user, even if a standard invocation of the SRS screen does not allow access to that program.
- 'Execute Procedure' allows a stored procedure to be executed. Type the procedure name and arguments exactly as you would in PL/SQL code. The entire string will be evaluated before being sent to the server for processing. See Evaluation of Strings below. Internally, this uses the FORMS_DDL builtin, but automatically supplies the additional PL/SQL wrapper text.
- SYNCHRONIZE is intended for rare circumstances when the screen is not synchronized between the client and middle tiers, typically at form startup.

- EXECUTE_TRIGGER allows you to execute a trigger within the form. It takes an argument of the trigger name. This builtin will test form_success after completion, and if that is False then it will raise form_trigger_failure.
- ‘Call Custom Event’ allows the CUSTOM library to be called. Despite the declarative interface of Forms Personalization it is still sometimes necessary to write PL/SQL code within the CUSTOM library. To facilitate this we have introduced a new builtin type of ‘Call Custom Event’ which takes a single argument of the event name. Within the CUSTOM library, code can then be written to respond to this event.
- ‘Create Record Group from Query’ allows you to create a record group dynamically. This takes two arguments, the SQL statement to create the group and the record group name which must not already exist within the form. This builtin can be used to declaratively create a record group that can then be used to populate an LOV. Note that using this builtin requires knowledge of the structure of the existing record group in order to achieve success. The structure of the current record group assigned to the LOV can only be determined by opening the form in Forms Builder, determining the correct record group, and then looking at the SQL statement. Also note that all this routine does is create a record group – it does not populate the group

Argument: The argument for the currently selected builtin, if applicable.

Depending on the specific builtin, other argument fields may appear.



Builtin Type: FND_FUNCTION.EXECUTE
Function Name
Parameters

Figure 6: The fields associated with an action of ‘Builtin’ and Builtin Type of FND_FUNCTION.EXECUTE

Function Name: The name of the function that should be executed.

Parameters: You can manually enter parameters or use the ‘Add Parameter…’ button. The ‘Add Parameter…’ button displays an LOV listing currently defined parameters for the Function Name, by querying other functions for that same form. It is possible that other parameters exist that will not be displayed in the LOV. The only way to see all parameters that a function has is to open the form in the Oracle Forms Builder. Oracle makes no warranties that any function provides the specific input parameters that you may desire, nor that any existing parameter and its behavior will remain unchanged after a patch.

For a Type of 'Menu':

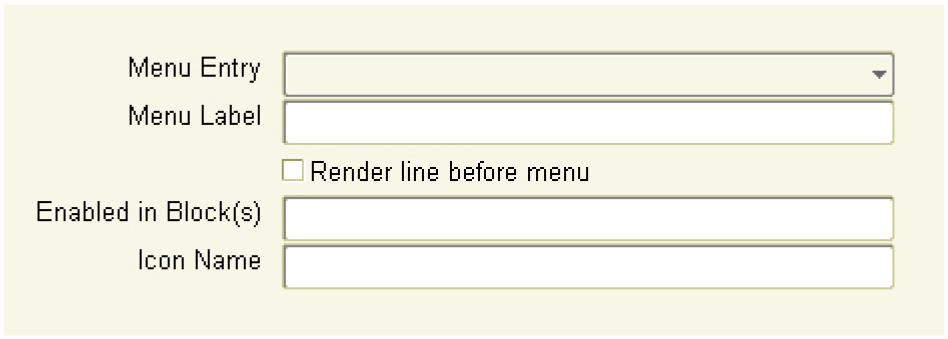


Figure 7: The fields associated with an action of 'Menu'

Menu Entry: A menu entry that you can activate. If you select a menu that the base form is already using, your functionality will override the form's functionality.

Menu Label: The textual label that you want on the menu entry. (See Evaluation of Strings, below).

Render line before menu: If checked, will render a line above the menu entry to visually separate it from prior entries.

Enabled in Block(s): Specify the blocks that you want the menu entry enabled in; specify more than one block by separating the names with a comma. If no blocks are specified, the entry will be enabled in all blocks. Use the 'Add Block...' button to add a blockname to the end of the field.

Icon Name: Specify an optional icon name that you want added to the Toolbar to achieve the same functionality as in the special pulldown menu entry.

Specifying an action of 'Menu' (formerly called 'Special') merely activates the appropriate menu entry. When the user selects the entry, it will fire the corresponding MENU# trigger. You must also create another rule that traps this Trigger Event and performs the desired functionality.

Note that the Actions block automatically defaults some values from the prior row when you create a new row. In the initial release of Form Personalization, the Actions block used to automatically default most values from the prior row when you created a new row. Now a smaller number of properties is defaulted, in order to reduce confusion. To copy all the values from the prior row, use the menu entry Edit > Duplicate > Record Above.

Evaluation of Strings

Every property that takes a string can either be processed literally or evaluated at runtime.

- If you type a string in that does not start with '=', then the exact value you type in will be used at runtime
- If the string you type starts with '=', then the text immediately after that character will be *evaluated* at runtime. This allows you to write complex logic that can include references such as:
  - SQL operators, such as ||, TO_CHAR, DECODE, and NVL
  - bind variables (:block.field), including :system, :global and :parameter values. Use the 'Add Item...' button to assist with item names.
  - Calls to server-side functions that do not have OUT parameters.
  - SELECT statements. To use this form, you must follow these rules:
    - The text must start with '=SELECT'
    - The column being selected must evaluate to a CHAR, with a length no longer than 2000 bytes.

- Your SELECT statement should only return one row, but if more than one is returned only the value of the first row will be used.
- In the initial release of Form Personalization, you needed to alias the column being selected to 'A'. That is no longer necessary, although pre-existing aliased text will continue to function as before. However, there is the additional limitation that the result set MUST return only a single row and only a single column. If this is not the case then an error will be shown during processing.

When using this method you must follow SQL conventions for string processing, including escaping of quotes. The following examples show how this can be used:

String typed in Personalization form	Result at runtime
='Your password will expire on '||(sysdate+7)	Your password will expire on 31-DEC-2004 {assuming that sysdate is currently 24-DEC-2004}
='Your password must have at least '||:global.password_length||' characters.'	Your password must have at least 8 characters. {assuming that global variable password_length exists and has a value of 8}
='Your password isn't valid.'	Your password isn't valid.
=:items.part_number	{whatever the current value of variable :items.part_number is}
=SELECT meaning from fnd_lookups where lookup_type = 'DAY_NAME' and lookup_code = 'SAT'	Saturday {assuming that the system is running in English}
=SELECT 'The office is closed on '||meaning from fnd_lookups where lookup_type = 'DAY_NAME' and lookup_code = 'SAT'	The office is closed on Saturday {assuming that the system is running in English}

Use the 'Validate' button to test if the syntax of your string is valid. If the evaluation fails, the processing engine will return an ORA error as if the string had been part of a SQL expression. Otherwise, it will display the text exactly as it would appear at runtime in the current context.

The ability to 'Get' a property in an expression
Strings, (like Conditions described above) can now refer to properties of objects using a SPEL syntax (Simplest Possible Expression Language). For example, this enables you to build a Condition that tests if a field is displayed or not. These expressions take the following general format:

```
${objectType.objectName.Property}
```

Internally, the SPEL expression is a cover for Oracle Forms builtins like GET_ITEM_PROPERTY, GET_BLOCK_PROPERTY, etc. Additionally, the SPEL expressions support retrieving Profile values, Message Dictionary text, and Local Variables (described later).

A new window, activated by the "Insert 'Get' Expression…" button, has been provided to automatically construct the expression. When complete, the expression is inserted into the Value or Condition field from where the window was launched. You can manually type the expression too; however, we strongly recommend using the window to build it to reduce mistakes.

For completeness, the SPEL expression supports the 'Value' property for an item; however, runtime performance is faster with the :block.field syntax.

All processing drives off the existence of the string '${', so that string is no longer valid in any Condition or Value except for this purpose. If a Value contains a SPEL expression, it is processed as if it started with '='.

Some examples:

Type	String typed in Personalization form	Result at runtime
Value	='The prompt is:' ||${item.block.field.prompt_text}\|\|', located at: '\|\|${item.block.field.x_pos}	The prompt is: User Name, located at: 2.0 {assuming that the current Prompt of field block.field is 'User Name', and it's X position is 2.0}
Condition	${item.block.field.required} = 'TRUE'	{True or False, depending on the current Required property of field block.field}

Exporting Data on Hidden Canvases

When exporting data from a forms block, only the data that is currently displayed for the block is exported. If you wish to allow exporting the block data that is on other hidden canvases, as is often with tab pages, you can create a form personalization rule to allow exporting the hidden canvases.

To create a personalization to allow exporting hidden canvases, create a new personalization as follows:

Condition
  Trigger Event: WHEN-NEW-FORM-INSTANCE level.
  Processing Mode: Not in Enter-Query-Mode.
  Set the appropriate Context Level.
Actions
  Type: Property
  Object Type: Block
  Target Object: <the block for the export personalization>
  Property Name: EXPORT_HIDDEN_CANVASES
  Value: TRUE

Limitations

This feature has several significant limitations due to the architecture of Oracle Forms and/or the E-Business Suite.

You can only change what Oracle Forms allows at runtime. For example, the following cannot be changed:

- You cannot create new items
- You cannot move items between canvases
- You cannot display an item which is not on a canvas (thus, individual flexfield segments cannot be displayed)
- You cannot set certain properties such as the Datatype of an Item.
- You cannot change frames, graphics, or boilerplate
- You cannot hide the item that currently has focus

Form Personalization can only respond to events that are centrally processed and dispatched by APPCORE. These are limited to:

- WHEN-NEW-FORM-INSTANCE, WHEN-NEW-BLOCK-INSTANCE, WHEN-NEW-RECORD-INSTANCE, WHEN-NEW-ITEM-INSTANCE. These events occur as the user moves focus within the form.

- WHEN-VALIDATE-RECORD (in many but not all forms). This event occurs whenever changes have been made to the current record in the current block.
- MENU1 through MENU15 menu entries, available in the Tools pulldown menu. These are guaranteed not to be used by Oracle and are exclusively for customer use, therefore we strongly encourage you to use these entries as opposed to the SPECIAL menu entries in order to avoid collisions of code. When the user selects the entry, it will fire the corresponding MENU# trigger. You must also create another rule that traps this Trigger Event and performs the desired functionality.
- Product-specific events. These are typically documented in implementation manuals, such as 'Configuring, Reporting and System Administration in Oracle HRMS'.

You can see events that are being passed by enabling the 'Show Events' option in the Custom Code menu.

Certain personalizations must be performed at specific events:

- To specify the Initial Value of an Item, you must perform that action in the WHEN-NEW-RECORD-INSTANCE event of the block that contains the item.
- MENU# menu entries can only be created at form startup (WHEN-NEW-FORM-INSTANCE)

Both the Personalization form and the runtime processing engine will report errors for these cases and skip processing of them.

Certain objects may not be available to you to change, or cannot be validated. Specifically, the object types GLOBAL and PARAMETER cannot be detected, thus these fields have no LOVs to restrict their input. Use the 'Validate' or 'Apply Now' buttons to determine if the values you have entered actually exist. Note that GLOBAL variables are dynamically created, so whether they exist or not can be a matter of timing.

> **Note:** If a Tab page does not appear in the list of objects its name can be entered manually. This is meant to address the known limitation that some tab names cannot be automatically detected at runtime. You will need to open the form in the Oracle Forms Builder to determine these tab names, and then you can type it into this Combobox.

Most significantly, any change you make might interfere with the normal operation of the form. This can manifest itself in several ways, such as:

- You may make a personalization but it doesn't take effect, because there is code in the form that overrides it. In some cases you may be able to perform your personalization by moving the Trigger Event to a 'lower' level, such as block- or item-level.
- Your personalization may simply produce the wrong result, because your change interacted with the base code in unexpected and untested ways. At best this error will occur immediately upon the personalization being applied; at worst it could affect some later processing which does not appear to be directly related to the object or event.
- In extreme cases, your changes may prevent the form from running at all, making it difficult to open the Personalization screen to remove the offending personalization unless you turn off Custom Code.

Because of this, it is critical that any change be thoroughly tested in a Test environment. See the 'Troubleshooting, Support, and Upgrade considerations' section later in this chapter for more information.

Examples and Tips

Changing the prompt of an item
This is a step-by-step example of changing a prompt. In this case, we will modify the 'Users' form, and change the prompt ‘User Name’ to ‘Logon Name’:

1. Open the Users form
2. Select Help->Diagnostics->Custom Code-> Personalize from the pulldown menu. If this menu entry is disabled, check the values of the FND_HIDE_DIAGNOSTICS and DIAGNOSTICS profiles.
3. Create a rule with the following values:
    - *Seq*: 1
    - *Description*: Change prompt of User Name

   Accept the defaults for all other values of the Rule and Context
4. Select the Actions Tab and enter the following values:
    - *Seq*: 1
    - Press the ‘Select By Text’ button and choose the ‘User Name’ row from the LOV
    - *Property Name*: PROMPT_TEXT
    - *Value*: Logon Name

   Accept the defaults for all other values of the Actions.
5. Save
6. Activate the Users form, then close it.
7. Re-open the Users form. You should see that the prompt is now ‘Logon Name’.
8. To disable this Rule, set Enabled to unchecked (at either the Rule or Action level), or just delete the Rule, then Save.

Disabling or Hiding a Tab Page
When you disable or hide a tab page, it does not change the state of the items on that page. Most significantly, if the items remain Navigable, and there is code elsewhere that attempts to navigate to them, it will succeed, causing the tab page to appear. To completely achieve the effect of hiding or disabling a tab page, you may need to account for the following:

- The ‘Next Navigation Item’ of the item that immediately precedes the first item on the tab page.
- The ‘Previous Navigation Item’ of the item(s) that immediately follow the last item on the tab page.
- The ‘Next Navigation Block’ of the block that immediately precedes the block that contains an item on the tab page.
- The ‘Previous Navigation Block’ of the block(s) that immediately follow the block that contains an item on the tab page.

Depending on how the form was coded, there may be additional properties or events that need to be changed.

Messages are a great debugging tool
Due to interactions of Personalization and the base code of each form, it is common to create rules that do not seem to be getting applied, or are applied incorrectly.
The simplest way to debug is to include Message actions of type ‘Debug’ either immediately before or after the action of interest.

Debug Mode supports two levels. 'Show Debug Messages' will show messages of type Debug. The ‘Step-By-Step' mode that will show messages when each condition is evaluated and each action is executed, to help you determine if your rules and actions are being applied as expected. 'Off' disables these modes

If you do not see your debug message at all, then the most likely reasons are:

- The Rule or Action is not enabled
- The Condition you entered for the Rule has evaluated to FALSE
- The Trigger Event and/or Trigger Object were not what you expected

- The scope of the rule only consists of Responsibility, Industry and/or User, and none is true for the current context.
- An action is executing the Builtin 'RAISE FORM_TRIGGER_FAILURE' . That will abort all further processing for that event.
- The Language of the Action is different than what you are currently running
- You have set Custom Code to ‘Off’ or ‘Core code only’ in the pulldown menu.

Interaction with the ‘Close Other Forms’ checkbox
While building personalizations, you may need to exit and re-open your form in order to see the effect of your personalizations. We recommend that you set the ‘Close Other Forms’ option in the Tools menu of the Navigator to unchecked, otherwise the Personalization form itself will close automatically before your form runs.

Using the same value multiple times
Often in code you want to determine a value once, then refer to it multiple times. You can do that with Form Personalization by using the 'Value' property of a GLOBAL variable. You can set such a value using any of the methods mentioned in the section Evaluation of Strings. By setting that property, the global variable will be created at that point in time, and then any future actions can refer to that variable. To minimize the risk of collision with GLOBAL variables that the base form may create, name your variable starting with 'XX'.

Zooming to a form
A common scenario is to open another function and have that form query a specific row of data associated with the source form. For example, consider the Users form which allows entry of Responsibilities for each user. A possible zoom would be to carry the current responsibility key to the Responsibilities form and have it query that record automatically. Some strategies to accomplish this type of functionality are:

- The target function may already accept the value as an input parameter. Simply passing the parameter name and value as the argument to the Function may accomplish the desired result.
- In forms that have Find windows, it may be possible to populate the appropriate field in the Find window, then issue DO_KEY('NEXT_BLOCK') which should simulate pressing the Find Window. Pass the value(s) between forms by using global variables.
- You could modify the DEFAULT_WHERE clause of the appropriate block, then query it using DO_KEY('EXECUTE_QUERY'), then reset the DEFAULT_WHERE clause back to its original value. Pass the value(s) between forms by using global variables.
- Put the form into enter-query mode using DO_KEY('ENTER_QUERY'), populate the desired fields, then issue DO_KEY('EXECUTE_QUERY'). Pass the value(s) between forms by using global variables. This is a complex technique though because invoking enter-query mode will suspend processing of that rule; populating the fields and issuing the DO_KEY('EXECUTE_QUERY') would need to be done in the WHEN-NEW-RECORD-INSTANCE event that fires as a result of entering query-mode.

With any technique that uses global variables, be aware that they may not exist yet. You should always initialize them before referring to them by setting the Initial Value to null, which will create the variable if it does not yet exist, otherwise it will leave it unchanged.

You should also code Conditions to account for situations where the value you want to pass between forms has not yet been entered, for example when the cursor is sitting on a brand new row.

Administration Window
The ‘Find Personalizations’ administration window can be invoked only from the Personalization Form. In the Tools pulldown menu, select ‘Administration’. This will allow you to get a list of all functions that currently have Personalizations (Rules) defined.

Form: The filename of a form.

If you press the Find button while Form Name is empty, all forms that have any personalizations (enabled or not) will be queried. This is particularly useful after applying a patch; knowing which forms the patch affects, you can quickly determine if any personalizations need to be re-validated.

Enabled Rules: The number of active rules that a function has defined.

Moving Personalizations between Instances

Once you create and test personalizations in your test instance, you can move them to production instances. Personalizations can be extracted by the loader on a per-function basis or per-form basis (that is, each loader file will contain all of the personalizations for a single function or form, respectively). Note that upon uploading, all prior personalizations for that function are first deleted, and then the contents of the loader file are inserted.

The loader syntax is as follows:

To download rules for a particular function:

```
FNDLOAD <userid>/<password> 0 Y DOWNLOAD $FND_TOP/patch/115/import/affrmcus.lct
<filename.ldt> FND_FORM_CUSTOM_RULES function_name=<function name>
```

> ***Note**: this style is not recommended, as the personalizations that affect a particular function can now be a mix of function- and form-level rules.*

To download rules for a particular form:

```
FNDLOAD <userid>/<password> 0 Y DOWNLOAD $FND_TOP/patch/115/import/affrmcus.lct
<filename.ldt> FND_FORM_CUSTOM_RULES form_name=<form name>
```

To download all personalizations (all forms and functions):

```
FNDLOAD <userid>/<password> 0 Y DOWNLOAD $FND_TOP/patch/115/import/affrmcus.lct
<filename.ldt> FND_FORM_CUSTOM_RULES
```

Upload:

```
FNDLOAD <userid>/<password> 0 Y UPLOAD $FND_TOP/patch/115/import/affrmcus.lct
<filename.ldt>
```

Relationship to CUSTOM library

Form Personalization allows personalizations that could be made in the CUSTOM library, but it does not require that you use the Oracle Forms Builder to edit and compile the CUSTOM file. Depending on the complexity of your personalizations, it may still require a degree of coding skill comparable to that needed to use the CUSTOM library. And the CUSTOM library is able to support more complex personalizations because it gives you access to all of the capabilities of the PL/SQL programming language, including calling client-side program units, all Oracle Forms builtins, and issuing any SQL.

Both Form Personalization and the CUSTOM library drive off the exact same events. The Form Personalization feature receives and processes them first, then passes them to the CUSTOM library, thus you can use both mechanisms simultaneously. (**Note**: SPECIALXX and MENUXX events are not passed to the CUSTOM library).

Both features also respond identically to the Custom Code events of 'Normal', 'Off' and 'Core Code Only'.

In general, Oracle recommends that you use the Form Personalization feature whenever possible, and only use the CUSTOM library when significantly more complex processing is required.

<u>Relationship to Folders</u>
Folders allow an end-user to 'customize' a screen by changing the fields and records displayed. For the most part, folder blocks are identifiable by an enabled ‘Folder’ menu entry, and an ‘Open Folder’ icon above the block. In a few cases, folder technology may be used by base code to dynamically alter a block, but no folder functionality is exposed to the end user.

Folder blocks are constructed differently than ‘regular’ Forms blocks – they include an extra block that renders the prompts for the fields, and many properties of the block are dynamically managed by the folder code as it receives events. As a result, when using the Form Personalization feature on a folder block, you must be aware of certain restrictions:

The following properties of a folder block can only be set at form startup (WHEN-NEW-FORM-INSTANCE). More specifically, they must be set before any folder code attempts to read the values otherwise unexpected results may occur:

- PROMPT_TEXT
- DISPLAYED
- WIDTH
- DEFAULT_WHERE
- ORDER_BY
- X_POSITION and Y_POSITION, in a single-row folder block

The following properties also have special considerations:

- ENABLED: within a folder block, it is invalid to set this property to FALSE. The cursor must be able to navigate to every item in a folder block. Consider setting ALTERABLE to FALSE instead.
- NEXT_ NAVIGATION_ITEM and PREVIOUS_NAVIGATION_ITEM: These properties have no effect in a Folder block. In a single-row folder block, the navigation sequence is computed based on X_POSITION and Y_POSITION. The navigation sequence of a multi-row folder block cannot be changed.

<u>Troubleshooting, Support, and Upgrade considerations</u>
Using the Form Personalization feature to alter Oracle code at runtime may bypass important validation logic and may jeopardize the integrity of your data. You should thoroughly test all changes you make in a Test instance before using it in a production environment.

Before contacting Oracle Support, you should always confirm that your personalizations are not the source of the problem. Oracle Support and Development cannot provide assistance on how you can make personalizations to a particular form using this mechanism, nor does Oracle warrant that any desired personalization can be made with this mechanism.

Troubleshooting
Any personalization you make may have unintentional consequences, to an extreme of preventing a form from running at all. Should this happen, you can disable all personalizations by invoking the pulldown menu and selecting Help->Diagnostics->Custom Code-> Off. This menu entry is secured by the FND_HIDE_DIAGNOSTICS and DIAGNOSTICS profiles. This will allow you to open the form and invoke the Personalization screen so you can correct the problem.

Upgrade Considerations
A form may change after an upgrade or patch to Oracle E-Business Suite. You should test any personalization logic that you have defined to confirm that it still operates as intended before using it in a production environment.

It is common for object names within a form to change after a applying a patch. To assist you with this, there is a function which will confirm the existence of objects that your personalizations reference. You should perform the following for each form that is changed during a patch and has personalizations:

- ❑ Run that form. If the form fails to run due to personalizations that are now in error, first turn Custom Code to 'Off' then re-run the form.
- ❑ Invoke the Personalization screen from that form
- ❑ In the Tools pulldown menu, select 'Validate All'. This will process every rule for that function, checking for the validity of references to objects in the form. Only rules that are Enabled are processed.
- ❑ For each Rule or Action identified as having an error, you can then quickly locate that row and make corrections.

Note that this function only checks for object existence; it still may be the case that certain personalizations that previously worked no longer do. Note that pressing the 'Validate All' button will first create any GLOBAL variables referred to in Property settings of either Value or Initial Value; this will reduce missing bind variable references to them if the code that creates them has not yet run.

<u>Changes to Form Personalization in R12.ATG_PF.H.Delta.6 (RUP6) and later</u>

Support for the creation of record group from query

APPCORE now supports using SQL statements with bound or literal values for creation of records groups. This is typically done to change the underlying WHERE clause for the query for an LOV. If one needs to change the columns returned by the query, care must be taken to ensure that the column names and data types match what was defined for the LOV that is being changed.

The SQL may reference values from form fields, form parameters, and global variables. All SQL statements used for this feature should be tested from SQL*Plus to ensure that the SQL is valid and will work as expected. If the SQL for the personalization has references to bound variables, substitute values that would normally be expected.

To create a personalization to change the record group used by an LOV, create a new personalization as follows:

Condition
  Trigger Event: WHEN-NEW-FORM-INSTANCE level.
  Processing Mode: Not in Enter-Query-Mode.
  Set the appropriate Context Level.
Actions
 #1
    Type: Builtin
    Builtin Type: Create Group from Query
    Argument: <the SQL statement to be used>
    Group Name: <the new record group name>

#2
    Type: Property
    Object Type: LOV
    Target Object: <the LOV to use the new record group>
    Property Name: GROUP_NAME
    Value: <the record group name created in action step 1>

Oracle-delivered Form Personalizations in Release 12.1.2

The Oracle E-Business Suite Form Personalizations feature has been extended in Release 12.1.2 to allow for Oracle-delivered Form Personalizations.

The personalizations form (FNDCUSTM.fmb) now has two modes. When the form is displayed, the FND: Enable Industry Editing profile value is checked.   If the profile is set to Yes, only personalizations that have a value for the KEY column will be displayed.   If the profile is set to No or not set at all, only personalizations that have a null value for the KEY column will be displayed.

Customers should be aware that when creating their own personalizations, they should NOT have the profile FND: Enable Industry Editing (FND_INDUSTRY_EDIT) set to Yes.

Creating personalizations with the FND: Enable Industry Editing with the profile set to Yes may introduce possible conflicts with Oracle-delivered Form Personalizations.   Customers are advised to only set the FND: Enable Industry Editing profile to Yes if they wish to examine the personalizations rules that have been delivered by Oracle E-Business Suite development.