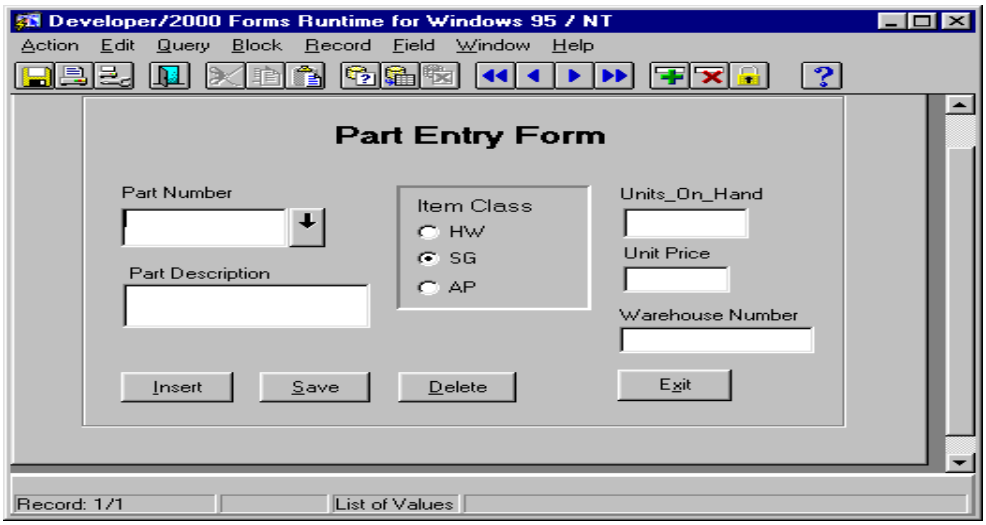
Creating Customized Oracle Forms

Custom forms do not contain data blocks that are associated with specific database tables. Instead, they use SQL statements to manipulate data. These SQL statements are embedded in triggers (*form triggers* that are different from the *database triggers*) that run when a user performs an action such as clicking a button, e.g,. **event**. In a data block form, the triggers were generated automatically when you created the data block. In a custom form, the person who develops the form must write a trigger (an event procedure) for each button.

Creating a Customized Form Module

(1) Create a Form Module in Object Navigator.
(2) Create a Data Block not associated with a specific table. Change the Window, Canvas, Data block names if necessary.
(3) Open the Form layout editor.
(4) Insert necessary item objects (e.g., text item, button, etc.) in the form.
(5) Insert necessary form triggers (event procedures) for buttons in the form.

Example Custom Form for Part Table



In this form, the INSERT button is used to insert a new record, the DELETE button is used to delete an existing record, the SAVE button is used to modify an existing record, and the EXIT button is used to exit the form.

The **access key**, Insert, Save, etc., can be created using **&**, e.g., **&Insert** in the property palette (right-click the button).

Let's **create a sequence object** to insert the part number assuming the part number start with 10 incremented by 1 each time. Type this command at SQL*Plus prompt:

CREATE SEQUENCE partseq START WITH 10;

Insert button trigger
Right-click the INSERT button → Select the PL/SQL editor → Select a proper trigger routine (e.g., 'WHEN-BUTTON-PRESSED') → Enter the following code for this event:

```
CLEAR_FORM;       -- will clear the form

:GLOBAL.mode := 'INSERT';     -- GLOBAL.mode is a global variable

SELECT partseq.nextval
       INTO :part.text_partnum
FROM DUAL;
```

Compile the code (click the compile menu) → Close if there is no error → Save the form

Global Variables
This form has two states: INSERT and SAVE. We are going to use a **global variable** that represents the current status of this form. Global variables provide a way to share information among different triggers. They are referenced using the following general format:
:GLOBAL.variable_name, e.g., :GLOBAL.my_var := TO_CHAR(:order.total*.085);

- A global variable is an Oracle Forms variable whose value is accessible to triggers and subprograms in any module that is active during the current session. A global variable stores a character string of up to 255 characters in length.
- Global variables are not formally declared the way PL/SQL local variables are. Rather, you initialize a global variable that the first time you assign a value to it:
- To reference a global variable, prefix the variable name with the word GLOBAL and a colon, e.g., calculate_discount(TO_NUMBER(:GLOBAL.my_var));
- Referencing a global variable that has not been initialized through assignment causes a runtime error.
- To destroy a global variable and release its memory, use the ERASE built-in procedure: ERASE('GLOBAL.my_var');

Item objects in a form
The general format for referencing the contents of a form text item is:
:block_name.item_name, -- always precede the block name with a colon **:**
e.g., `:part.text_partnum` -- part is a block name and text_partnum is a item name.

Creating the DELETE button trigger:
In the 'WHEN-BUTTON-PRESSED' event, type the following code:

```
DELETE FROM part WHERE part_number = :part.text_partnum;
COMMIT;
CLEAR_FORM;
```

Creating the EXIT button trigger:
In the 'WHEN-BUTTON-PRESSED' event, type the following code:

```
EXIT_FORM;
```

Save button trigger in the WHEN-BUTTON-PRESSED event

```
IF :GLOBAL.mode = 'INSERT' THEN
   INSERT INTO part VALUES(:part.text_partnum,:part.text_partdesc,
                           :part.radio_itemclass,:part.text_unitonhand,
                           :part.text_unitprice, :part.text_warehouse);
   :GLOBAL.mode := 'SAVE'; -- state is changed to SAVE
ELSE
    UPDATE part
    SET part_description = :part.text_partdesc,
        Units_on_hand = :part.text_unitonhand,
        Item_class = :part.radio_itemclass,
        Unit_price = :part.text_unitprice,
        Warehouse_number = :part.text_warehouse
    WHERE part_number = :part.text_partnum;
END IF;
COMMIT;
CLEAR_FORM;
```

Creating the List of Values (LOV)
The ⇓ button right beside the *Part Number* text item is the LOV button. We can use this LOV button to help data entry person enter data easily using descriptive information such as *Part Description* (more informative than number) instead of *Part Number*.

In the Form Layout Editor, add a Push-button to the Part Number text item and change the property of the button, 'Iconic' property to 'Yes', 'Iconfilename' to 'down'.

To Create a LOV object in the Object Navigator: Select LOVs and click add button (+) → change the name of LOV.

Enter a query: this will create a Record Group

```
SELECT part_number, part_description
       INTO :part.text_partnum, :part.text_partdesc
FROM part
ORDER BY part_description;
```

Formatting the LOV Display
Open the Properties sheet for the LOV object → Click Column Mapping properties, and then click the More button → change the properties → Click OK → Close → Save the Form.

Associating the LOV with a Text field on the form
Open the Text item properties sheet → click 'List of Values' (or double click the LOV property so the LOV item appears) → select the LOVE object → close

Creating the LOV item button trigger:
In the 'WHEN-BUTTON-PRESSED' event, type the code shown below:

```
GO_ITEM('text_partnum'); -- put the insertion point in the text
                         -- item associated with the LOV
DO_KEY('LIST_VALUES');   -- show the LOV
```

GO_ITEM(item_name); -- give focus to the item

GO_FORM(form_name); -- give focus to the form

DO_KEY(built-in-program);
DO_KEY executes the key trigger that corresponds to the specified built-in subprogram. If no such key trigger exists, then the specified subprogram executes.

e.g., DO_KEY('LIST_VALUES');

LIST_VALUES(kwd);
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 value remains displayed until the user dismisses the LOV or selects a value.

There are many other built-in Oracle key trigger:

Built-in	**key trigger**	**associated function key**
CLEAR_FORM	key-CLRFRM	[Clear Form]
EXIT_FORM	key-EXIT	[Exit/Cancel]
EXECUTE_QUERY	key-EXEQRY	[Execute Query]
LIST_VALUES	key-LISTVAL	[List]
COMMIT_FORM	key-COMMIT	[Commit]
UP	key-UP	[Up]
DOWN	key-DOWN	[Down]

…

☞ DO_KEY accepts built-in names only, not key names. To accept a specific key name, use the EXECUTE_TRIGGER.

e.g.,
```
BEGIN
        DO_KEY('Execute_Query');  -- simulate pressing the [Execute Query] key
END;
```

Controlling Radio Groups
A Radio Group item represents one of radio groups. Each radio button has a value that will be passed to its radio group when the user clicks the button.

☞ When you initialize a Radio Group, set a radio button name to the radio group's value or radio button's value to the radio group's initial value.

Creating a PRE-FORM trigger (form-level event)
PRE-FORM trigger is attached to a form and executes when the form first displays (you can use this to initialize :GLOBAL.mode variable).

To add a PRE-FORM trigger, Open the Object Navigator window → right-click the Form Module, and then click PL/SQL Editor → Select the 'PRE-FORM' event (or Click the New button on the PL/SQL Editor button bar and Select the 'PRE-FORM' event) → click OK → type the following initialize statement and save the form:

```
:GLOBAL.mode := 'INSERT';
```

You can put other initialize statements in this form.
Creating a Program Unit:
You can define Procedures or Functions for a form. These procedures and functions are usually local procedures and functions that will be called for a form.

To create a Program Unit, Open the Object Navigator in Ownership view → Click Program Units → Create button (+) → type a procedure or function code.

✓ You can shorten the Trigger code through using the Program Units.

Creating Alerts
To Create Alert objects in the Object Navigator → Click Alerts → Create button (+) → Set the properties of the Alert (right-click the Alert and change the properties).

To show Alerts, use the SHOW_ALERT function. For example, in the program unit

```
PROCEDURE DISPLAY_ALERTS IS
        alert_button NUMBER;                                -- Alert number
BEGIN
        alert_button := SHOW_ALERT('UPDATE_ALERT');         -- UPDATE_ALERT is the
                                                            -- alert name
        IF alert_button = ALERT_BUTTON1 THEN                -- user clicked OK
           COMMIT;
           alert_button := SHOW_ALERT('CONFIRM_ALERT');
           CLEAR_FORM;
        ELSE
           ROLLBACK;
        END IF;
END;
```

Displaying error message
MESSAGE('message to be display');

Calling another form in a form
CALL_FORM('full path to the form's .**fmx** file');

To change the properties of Window
SET_WINDOW_PROPERTY('window_name', property_name, property_value)

To set Form Properties dynamically
SET_ITEM_PROPERTY('form_item_name', property_name, property_value);

Creating an Image item
Insert an Image item → load the image file (READ_IMAGE_FILE('file-name', 'file-type', 'item-name')
e.g., READ_IMAGE_FILE('C:\graphics\splash.gif', 'GIF', 'myblock.splash_image');

Creating a Timer
TIMER_ID := CREATE_TIMER('timer_name', milliseconds, iterate);

splash_timer TIMER;
splash_timer := CREATE_TIMER('splash_timer', 5000, NO_REPEAT);
SHOW_WINDOW('window_name'); -- to show a window

Text Item States
Text Item has three different states: New, Changed, Validated

As soon as user begins entering data in one of the items, the item is in *Changed* state, (initial state before the user changes its item, is *New*). When user requests to commit the record, each item and the record itself is set to *Validated* if validation is successful (no error).

Enter item → validate item → leave item: different state has corresponding triggers, e.g., Pre-text-item-trigger, when-new-item-trigger ….

SYSTEM.MODE System variable
This is a special Oracle variable that indicates whether the form is in ***Normal***, ***Enter Query***, or ***Fetch*** processing mode. This variable is similar to GLOBAL variable but used for checking current system status.

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.
FETCH indicates that the form is currently in fetch processing mode, meaning that a query is currently being processed.

For more detail information about Developer/2000 and Oracle, Refer to Oracle Online Help in the form window or Documents in http://www.ecs.fullerton.edu/oracle