Altering Oracle Forms Functionality Without Altering Oracle Forms Code (Using the CUSTOM Library)

REVISED AND EXTENDED FOR OAUG SPRING 2000

Copyright © 1999,2000 by Brad Goodwin
Oracle Corporation

Why We Have The CUSTOM Library

In the past when organizations wanted to create or replace internal business application packages, their options were primarily:

- Buy an 'off-the-shelf' application and hope that it meets most of the needs of the organization
- Custom build the required application from start to finish

These options have usually been considered mutually exclusive, and both have their own relative advantages and disadvantages. Off-the-shelf applications are ideal if the all of the companies requirements can be satisfied. On the other hand, custom-built applications give you everything that you wanted but at the cost of internal ownership of the application, which includes not just the code, but the support and maintenance of the custom application over time.

From Production 13 of the Oracle Applications Smart Client release, a new feature was introduced to the applications called the 'CUSTOM library'. The CUSTOM library is a facility that enables you to augment and extend the Oracle Applications without modification of Oracle Applications code. The CUSTOM library can be used for defining 'zooms' (a mechanism to 'jump' to forms from other forms) and for enforcing business rules.

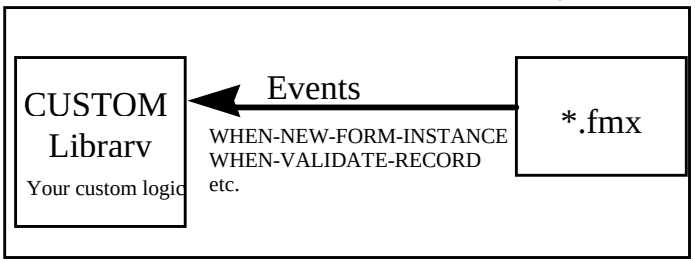
The CUSTOM library is:

- A mechanism that allows extension of Oracle Applications without modification of Oracle Applications code
- An Oracle Forms PL/SQL library that allows you to take full advantage of all of the capabilities of the Developer 2000 suite of products and integrate your code directly with Oracle Applications
- A supported, upgradeable, user definable, partitioned code repository for custom code

In basic terms, the CUSTOM library lets you change properties, functionality and operation of a form as if you were actually changing the source code of the form.

How the CUSTOM Library Works

The CUSTOM library works by sending 'events' from each Oracle Applications form to the CUSTOM library, which is automatically attached at runtime. User defined custom code, which is in the CUSTOM library, can then take effect based on these events.



WARNING: Do not modify any component of an Oracle Applications module (either directly or through the CUSTOM library) without a thorough understanding of the processing logic and underlying database structure for the component.

CUSTOM Library Details

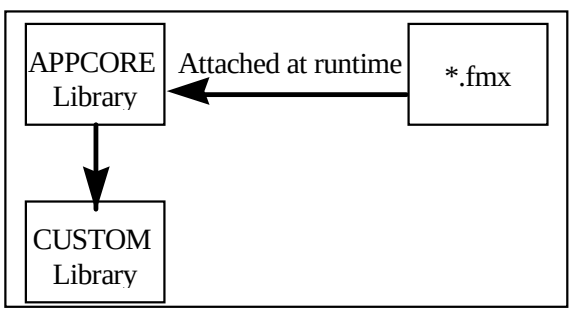
The CUSTOM library is available from within any of the Oracle Applications that are developed in Oracle Forms. It first appeared in the Production 13 Smart Client release of the Oracle Applications, and continues essentially unchanged in all subsequent versions of 10SC, 10.7 NCA and Release 11.

The CUSTOM library is actually a pair of files called CUSTOM.pll and CUSTOM.plx. The '.plx' extension is a compiled version of the '.pll' file. In release 10SC and 10.7NCA, these files reside in the $AU_TOP/res/plsql directory. The CUSTOM library you modify must replace the default CUSTOM library in this directory in order for your code to take effect. Be aware that if both the '.pll' and '.plx' versions of the file exist in the same directory then Oracle Forms will use the '.plx' version of the file. A '.plx' is only created when you generate a library using the Oracle Forms generator (using the parameter COMPILE_ALL set to Yes), not when you compile and save using the Oracle Forms Designer.

Some CUSTOM Library Rules

Since there is only a single library that is shared by all forms in your installation, be careful about what code you put in the library. All restrictions and limitations that exist in any standard Forms library also apply to the CUSTOM library. In addition, these rules apply:

- You cannot use SQL in the CUSTOM library (call server side packages instead, or use a record group to issue the SQL for you)
- You cannot change the specification of the CUSTOM package
- You cannot attach the APPCORE library to the CUSTOM library (because CUSTOM is attached to APPCORE), and you cannot call APPCORE routines (which usually start with "APP"). For example, function `APP_ITEM_PROPERTY` is not available within the CUSTOM library. You should use Oracle Forms `SET_ITEM_PROPERTY` instead.
- Use FNDSQF library for function security, flexfields and message procedures.



- Do not use `CALL_FORM` or `OPEN_FORM` within the CUSTOM library, use `FND_FUNCTION` package instead

All of these rules (and a few more) are documented in *Release 10 Oracle Applications Coding Standards*, chapter 12 (part #A42530) and the Release 11 *Oracle Applications Developers Guide*, chapter 25 (part # A58187).

The CUSTOM library can be switched on and off dynamically by the user by selecting the Help->Tools->Custom Code->Off option from the menu bar. You can prevent users from being able to switch off the CUSTOM library by setting the profile 'Diagnostics' to 'No'(this profile also controls access to most of the other functions on the 'Tools' menu).

When to Use the CUSTOM Library

There are four main ways to use the CUSTOM library. Each of these must be coded differently.

- **Zoom**: A Zoom opens another form and can pass parameters to the opened form.
- **Logic for generic events**: Augment Oracle Applications logic for certain generic form events such as WHEN–NEW–FORM–INSTANCE or WHEN–VALIDATE–RECORD.
- **Logic for product-specific events**: Augment or replace Oracle Applications logic for certain product–specific events that enforce business rules.
- **Custom entries for the Special menu**: Add entries to the Special menu for Oracle Applications forms, such as an entry that opens a custom form.

Coding Zoom

Zoom allows the user to "jump" from one screen to another via a single press of a button on the toolbar. For example, you may want to allow access to the Vendors form from within the Enter Purchase Order form. You can enable Zoom so that when the user's cursor is in the PO header block the Zoom button on the toolbar lights up, and when pressed, "jumps" to the Vendor definition form. If you don't put Zoom code in the CUSTOM library, the Zoom button will not be enabled.

Whenever the cursor changes blocks in the form, the form calls the ZOOM_AVAILABLE function in the CUSTOM library (via APPCORE). If this function returns TRUE, then the Zoom entries on the menu and toolbar are enabled; if it returns FALSE, then they are disabled. If the Zoom entries are enabled, then when the user invokes Zoom the form calls the ZOOM event code in the CUSTOM library. You write code for this event that branches based on the current form and block.

To code Zooms into the CUSTOM library:

1. Using Oracle Forms Designer, open $AU_TOP/res/plsql/CUSTOM.pll (or its platform equivalent) and open the procedure body for program unit CUSTOM. Add a branch to the CUSTOM.ZOOM_AVAILABLE function that specifies the form and block where you want a user to be able to invoke Zoom.

```
FUNCTION zoom_available RETURN BOOLEAN IS
  form_name  VARCHAR2(30) := NAME_IN('system.current_form');
  block_name VARCHAR2(30) := NAME_IN('system.cursor_block');
BEGIN
  --Zoom button enabled for Enter PO form:
  IF (form_name = 'POXPOEPO'
      AND block_name = 'PO_HEADERS') THEN
    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
END zoom_available;
```

2. Add a branch to the CUSTOM.EVENT procedure for the ZOOM event. Inside that branch, specify the form and block where you want a user to be able to invoke Zoom. Add the logic you want to occur when the user invokes Zoom.

Note: The character-mode version of Oracle Applications has always had a Zoom capability, which simplified movement from one screen to another by allowing users to avoid cumbersome menu keystrokes. Unlike the character mode version, releases 10SC, 10.7NCA and 11 do not ship with any predefined Zoom logic. Most Zooms that were predefined in the character-mode have been incorporated into 10SC, 10.7NCA and 11 as buttons or new windows.

```
PROCEDURE event(event_name VARCHAR2) IS
  form_name  VARCHAR2(30) := NAME_IN('system.current_form');
  block_name VARCHAR2(30) := NAME_IN('system.cursor_block');
BEGIN
 IF (event_name = 'ZOOM') THEN
  IF (form_name = ' POXPOEPO'
      AND block_name = 'PO_HEADERS') THEN
    FND_FUNCTION.EXECUTE(FUNCTION_NAME=>'AP_APXVDMVD',
                         OPEN_FLAG=>'Y',
                         SESSION_FLAG=>'Y',
                         OTHER_PARAMS=>'');
  END IF;
 END IF;
END event;
```

Coding For Generic Events

The CUSTOM library receives two different kinds of event calls: generic and product specific. Generic events are common for all Forms in Oracle Applications. Product specific events are reserved for future use by the applications and are specific to individual applications.

Generic CUSTOM library events in Release 10 Smart Client are:

- WHEN-NEW-FORM-INSTANCE
- WHEN-NEW-BLOCK-INSTANCE
- EXPORT
- WHEN-NEW-RECORD-INSTANCE
- WHEN-NEW-ITEM-INSTANCE
- WHEN-VALIDATE-RECORD
- ZOOM

New to Release 11:

- WHEN-FORM-NAVIGATE
- SPECIAL*n* (where n is a number)
  SPECIAL allows you to specify custom code to execute when the user selects a menu item from the Special menu.

You really have almost limitless possibilities to alter the way Oracle Applications screens behave using this method. Some examples of the types of things that can be done with the CUSTOM library include:

- Changing form field properties, e.g. hiding form fields, changing their colors, etc.
- Adding additional field validation and cross field validation
- Changing a default LOV or Default Where clause
- Automatically populate fields based on some form of server side computation
- Write audit logs directly to file
- Call new forms that interact with core forms
- Manipulating conditionally required fields
- Manipulating mutually exclusive fields
- Setting of default values
- Setting of inter field calculation between added fields
- General processing at event points
- Update non-enterable fields
- Call a server side procedure that then executes a PL/SQL program

For example, let's say that your company insists that vendor names always be entered in uppercase. The CUSTOM library allows you to add code to do this without altering any Oracle source code:

```
PROCEDURE event(event_name VARCHAR2) IS
form_name VARCHAR2(30) := NAME_IN('system.current_form');
BEGIN
 IF (event_name = 'WHEN-NEW-FORM-INSTANCE') THEN
  IF (form_name = 'APXVDMVD') THEN
/* Force the value of Vendor Name field to uppercase */
    set_item_property('VNDR.VENDOR_NAME_MIR',
                      CASE_RESTRICTION, UPPERCASE);
  END IF;
 END IF;
END event;
```

Another example: In Oracle Applications, the System Administrator can grant/revoke responsibilities to him/herself. In many organizations, it is imperative that not any one individual have such sweeping, all-inclusive power. A simple and effective way to address this solution is to use the CUSTOM library. The code below works against the Application Object Library user security definition screen. It checks to see if the user has queried up his/her own record, and if so, disables the entire responsibilities block.

```
PROCEDURE event(event_name VARCHAR2) IS
  form_name   VARCHAR2(30) := name_in('system.current_form');
  block_name  VARCHAR2(30) := name_in('system.current_block');
  new_state   number;
BEGIN
  IF form_name = 'FNDSCAUS' THEN
    IF event_name = 'WHEN-NEW-BLOCK-INSTANCE' THEN
      IF block_name = 'USER_RESP' THEN
        IF (fnd_profile.value('USERNAME') = name_in('USER.user_name')) THEN
          new_state := property_false;
        ELSE
          new_state := property_true;
        END IF;
        set_block_property(block_name, insert_allowed, new_state);
        set_block_property(block_name, update_allowed, new_state);
      END IF;
    END IF;
  END IF;
END event;
```

Another example: Some organizations have many instances of Oracle Applications: test, development, production—even multiple production databases. For users who switch between instances, it can be difficult to know which database instance they are connected to. The following code, placed in the CUSTOM library, will change the title bar to include the connect string of the instance you are connected to. (Sorry, this code only works with 10SC)

```
PROCEDURE event (event_name varchar2) IS
  BEGIN
    IF (event_name = 'WHEN-NEW-FORM-INSTANCE') then
       set_window_property(FORMS_MDI_WINDOW, TITLE,
           'Oracle Applications Instance: (' ||
            upper(get_application_property(CONNECT_STRING)) || ')');
    END IF;
  END;
END event;
```

Another example: Ever wish that mandatory fields were highlighted so that users could quickly identify which fields were required? The code fragment below shows how to accomplish this on any form. Place this code in the EVENT procedure within the CUSTOM library. NOTE: Highlighting required fields will be a standard feature of Release 11.5.

```
set_item_property('person.last_name',VISUAL_ATTRIBUTE,'SELECTED_DATA');
set_item_property('person.d_person_type_id',VISUAL_ATTRIBUTE,'SELECTED_DATA');
set_item_property('person.sex',VISUAL_ATTRIBUTE,'SELECTED_DATA');
set_item_property('person.employee_number',VISUAL_ATTRIBUTE,'SELECTED_DATA');
set_item_property('person.national_identifier',VISUAL_ATTRIBUTE,'SELECTED_DATA');
set_item_property('person.date_of_birth',VISUAL_ATTRIBUTE,'SELECTED_DATA');
```

Taking this example one step further, you could add a dialog with the user explaining which fields are mandatory, and even restricting them from saving their work unless all mandatory fields are entered:

```
IF name_in('person.last_name') IS NULL
        OR name_in('person.d_person_type_id') IS NULL
            OR name_in('person.sex') is null
              OR name_in('person.national_identifier') IS NULL
                 OR name_in('person.date_of_birth') IS NULL THEN
                  fnd_message.set_string('You have not entered all required
fields: Last Name, Gender, Person Type, Social Security Number, Birth Date');
                  fnd_message.show;
END IF;
-- Prevent the form from commiting
RAISE form_trigger_failure;
```

Another example: Ever want to hide a field?

```
set_item_property('person.d_home_office',DISPLAYED,PROPERTY_FALSE);
```

(But please note that the boilerplate label of the field will still be visible. ...Unless, of course, you put an entry into the CUSTOM library that covers it with a graphical object...)

Coding For Product Specific Events

Product specific events are rare prior to Release 11. Oracle HR has been the first module to embrace them, and other modules have begun to follow. In certain windows Oracle HR has created additional event points that are passed to the CUSTOM library. These event points are:

- WHEN–CREATE–RECORD
- POST–QUERY
- KEY–DELREC
- PRE–UPDATE
- PRE–INSERT
- PRE–DELETE
- POST–INSERT
- POST–UPDATE
- POST–DELETE
- POST–FORMS–COMMIT
- WHEN–BUTTON–PRESSED
- NAVIGATE (The NAVIGATE event is passed to the CUSTOM library whenever a user selects a task flow navigation option and before opening the new window.

Oracle HR has 21 forms and Oracle Training Administration has 12 forms that have been altered for these additional events. They are documented in the Release 11 *Oracle Applications Product Update Notes* (part #A57984).

Customizing the Order Entry Screen

Although not directly related to the CUSTOM *library*, Oracle Order Entry has a special feature on the Sales Orders form, which is referred to as the CUSTOM *form.* The CUSTOM form allows customers to customize the appearance of the Sales Orders form, but because these visual aspects are stored in a different fmx file from the form's functional logic, the customizations are preserved through most upgrades.

The CUSTOM form contains the Sales Orders form's presentation layer, which you can modify using Developer/2000 Forms Designer. The MAIN form (OEXOEMOE), which references customizable objects in CUSTOM (OEXOECOE), is executed in the application's normal runtime environment.

When you edit the CUSTOM form, you can choose from the following possible customizations:

- Hide an item (text item, button, check box, poplist, option group) Caution: Do not hide any item required for entry or booking that is not defaulted. For example, do not hide the Order Type.
- Display hidden items
- Resize an item
- Resequence an item
- Move an item from one region to another
- Move an item from a multi-row block to the overflow area, or vice versa
- Edit boilerplate labels
- Increase or decrease the number of rows displayed in a multi-row block
- Resize a window
- Set a button as default

These capabilities are only available in the Enter Sales Orders screen of Oracle Order Entry.

Advanced Custom Coding

Notice that not all Oracle Forms events are sent to the CUSTOM library (such as WHEN-CHECKBOX-CHANGED, and many others). This means that you must always make sure that your custom code can be set into motion by one of the supported triggering events mentioned above.

But what if the triggering event needed for your requirements is not on the official list? Or you must add a bunch of custom code to a form? The CUSTOM library can (and should) still be used when base code simply has to be changed, as it puts the majority of your customizations into the CUSTOM library and minimizes code changes that must be re-applied when the base form is patched or upgraded.

One of the companies in my area had a problem with Receivables users doing blind queries on the Account Details screen, which would initiate a full table scan of a 6 million row table (which was further joined to 8 other tables…). You don't have to be a DBA to understand that this brought the database to its knees even if only a few users did it simultaneously. This company wanted to ensure that any queries done against the Account Details screen always included some sort of search criteria, so the result set would be smaller and so that indexes would be used by the optimizer for fastest performance. The event that had to be caught in order to accomplish this was the WHEN-BUTTON-PRESSED event from the Find window.

The CUSTOM library was the perfect repository for the required code, but the form definition had to be altered to pass a customer-defined event 'CUSTOM_FIND' to the CUSTOM library. The code is below.

```
PROCEDURE event(event_name VARCHAR2) IS
   query_fields   VARCHAR2(100);
BEGIN
-- Normally the CUSTOM library doesn't get the
-- WHEN-BUTTON-PRESSED event. We modified the
-- AR Account Details Find screen's "FIND" button to pass
-- this event.  See C:\apps10\au10\res\plsql\ARXCOQIT.PLL,
-- in the ARXCOQIT_FIND procedure, which is executed when
-- the FIND button is pressed on that screen.
-- We added a call to custom.event('CUSTOM_FIND')

    IF (event_name = 'CUSTOM_FIND') THEN
      -- Has the user entered criteria into any fields?
      query_fields := NAME_IN('CQIT_FIND.trx_number')||
                      NAME_IN('CQIT_FIND.name')||
                      NAME_IN('CQIT_FIND.number')||
                      NAME_IN('CQIT_FIND.sales_order')||
                      NAME_IN('CQIT_FIND.purchase_order');

      -- If not, then inform user of error
      IF query_fields IS NULL THEN
        fnd_message.set_string('To avoid serious performance
                               degradation, you must supply
                               at least one criteria before
                               querying.');
        fnd_message.error;
        -- Disable query
        COPY( 'FALSE' ,'parameter.g_query_find' );
      END IF;
    END IF;
  END event;
```

New to Release 11i

The CUSTOM library stays mostly unchanged in release 11i, except for some new functionality that makes it easier to deploy. The three new features are:

- **Extra list of values (LOV) in every form.** Each Applications form now has an "extra" LOV available to the CUSTOM library for your use: APPCORE_ZOOM LOV. You would use this if you have multiple zooms defined from a form. This LOV can be used to display a pop-up window that lists all of the destination forms, and let the user choose the one they want to go to.
- **Custom visual attributes.** The CUSTOM library now has the ability to change visual attributes of form objects (such as buttons, fields, etc.) at runtime.
- **Some previous coding restrictions have been removed.** You can now call any Applications stored procedure, because package variables are now local within a form database session. (There is still the remaining restriction that you cannot have SQL or DML in the CUSTOM library.)

  Some APPCORE routines have been exposed for use by the CUSTOM library, in a new library called APPCORE2. The APPCORE2 library duplicates most APPCORE routines:
  - APP_ITEM_PROPERTY2
  - APP_DATE2
  - APP_SPECIAL2

  To use these routines, attach APPCORE2 to your CUSTOM library. Only the library name has changed; the procedures and functions within the APPCORE2 library have the same names as their APPCORE counterparts.

Conclusion

The CUSTOM library provides a non-invasive mechanism that allows customers to extend the application in a manageable and controlled environment while preserving support for the core application. The ability to turn CUSTOM library code on or off ensures application support even when problems are encountered because it enables you to identify whether the problem is custom code related or core application related.

Acknowledgements

Special thanks to Oracle employees:

- **Andrew McGhee** for his excellent white paper on the CUSTOM library, some of which, with his kind permission, forms the basis for this paper.
- **Scott Spendolini** for his security example.
- **Peter Wallack** who reviewed for accuracy, and improved on my code.
- **Sara Woodhull** who reviewed for accuracy, and pointed me towards the HR enhancements in Release 11.

About the Author

Brad Goodwin (bgoodwin@us.oracle.com) is a eBusiness technical specialist for Oracle in Portland, Oregon. Ten of his thirteen years with Oracle have been focused on the technical architecture and components of Oracle Applications. He has presented at five prior OAUG conferences on Oracle Alert, Oracle's reporting strategy, and the CUSTOM library.

Bibliography

White Paper: "*GUIDELINES FOR THE ADDITION OF USER DEFINED BUSINESS RULES WITHIN THE ORACLE APPLICATIONS*," Andrew McGhee, HRMS Product Support Manager, Oracle Corporation, Bracknell, UK, April 1998

Release 11 *Oracle Applications Developers Guide*, chapter 25 (part # A58187)

Release 10SC *Oracle Applications Coding Standards*, chapter 12 (part #A42530)

Release 11 *Oracle Applications Product Update Notes* (part #A57984)

Appendix - CUSTOM.pll

Package Header

```
PACKAGE custom IS

    function  zoom_available              return BOOLEAN;
    function  style(event_name varchar2)  return integer;
    procedure event(event_name varchar2);
    before    constant integer := 1;
    after     constant integer := 2;
    override  constant integer := 3;
    standard  constant integer := 4;

END custom;
```

Package Body

```
package body custom is
  --
  -- Customize this package to provide specific responses to events
  -- within Oracle Applications forms.
  --
  -- Do not change the specification of the CUSTOM package in any way.
  -- You may, however, add additional packages to this library.
  --
  ---------------------------------------------------------------------
  function zoom_available return boolean is
  --
  -- This function allows you to specify if zooms exist for the current
  -- context. If zooms are available for this block, then return TRUE;
  -- else return FALSE.
  --
  -- This routine is called on a per-block basis within every Applications
  -- form from the WHEN-NEW-BLOCK-INSTANCE trigger. Therefore, any code
  -- that will enable Zoom must test the current form and block from
  -- which the call is being made.
  --
  -- By default this routine must return FALSE.
  --
  /* Sample code:
    form_name  varchar2(30) := name_in('system.current_form');
    block_name varchar2(30) := name_in('system.cursor_block');
  begin
    if (form_name = 'DEMXXEOR' and block_name = 'ORDERS') then
      return TRUE;
    else
```

```
    return FALSE;
  end if;
end zoom_available;
*/
--
-- Real code starts here
--
begin
  return FALSE;
end zoom_available;

---------------------------------------------------------------------
function style(event_name varchar2) return integer is
--
-- This function allows you to determine the execution style for some
-- product-specific events. You can choose to have your code execute
-- before, after, or in place of the code provided in Oracle
-- Applications. See the Applications Technical Reference manuals for a
-- list of events that are available through this interface.
--
-- Any event that returns a style other than custom.standard must have
-- corresponding code in custom.event which will be executed at the
-- time specified.
--
-- The following package variables should be used as return values:
--
--    custom.before
--    custom.after
--    custom.override
--    custom.standard
--
-- By default this routine must return custom.standard
--
-- Oracle Corporation reserves the right to change the events
-- available through this interface at any time.
--
/* Sample code:
begin
  if event_name = 'OE_LINES_PRICING' then
    return custom.override;
  else
    return custom.standard;
  end if;
end style;
*/
--
-- Real code starts here
--
begin
  return custom.standard;
end style;

---------------------------------------------------------------------

procedure event(event_name varchar2) is
--
-- This procedure allows you to execute your code at specific events
-- including:
--
--    ZOOM
--    WHEN-NEW-FORM-INSTANCE
--    WHEN-NEW-BLOCK-INSTANCE
--    WHEN-NEW-RECORD-INSTANCE
--    WHEN-NEW-ITEM-INSTANCE
```

```
  --    WHEN-VALIDATE-RECORD
  --
  -- Additionally, product-specific events will be passed via this
  -- interface (see the Applications Technical Reference manuals for
  -- a list of events that are available).
  --
  -- By default this routine must perform 'null;'.
  --
  -- Oracle Corporation reserves the right to change the events
  -- available through this interface at any time.
  --
/* Sample code:
    form_name      varchar2(30) := name_in('system.current_form');
    block_name     varchar2(30) := name_in('system.cursor_block');
    param_to_pass1 varchar2(255);
    param_to_pass2 varchar2(255);
  begin
    -- Zoom event opens a new session of a form and
    -- passes parameter values to the new session.  The parameters
    -- already exist in the form being opened.

    if (event_name = 'ZOOM') then
      if (form_name = 'DEMXXEOR' and block_name = 'ORDERS') then
        param_to_pass1 := name_in('ORDERS.order_id');
        param_to_pass2 := name_in('ORDERS.customer_name');
        fnd_function.execute(FUNCTION_NAME=>'DEM_DEMXXEOR',
                             OPEN_FLAG=>'Y',
                             SESSION_FLAG=>'Y',
                            OTHER_PARAMS=>'ORDER_ID="'||param_to_pass1||
                              '" CUSTOMER_NAME="'||param_to_pass2||'"');
                     -- all the extra single and double quotes account for
                     -- any spaces that might be in the passed values
      end if;

    -- This is an example of a product-specific event.  Note that as
    -- of Prod 15, this event doesn't exist.
    elsif (event_name = 'OE_LINES_PRICING') then
      get_custom_pricing('ORDERS.item_id', 'ORDERS.price');

    -- This is an example of enforcing a company-specific business
    -- rule, in this case, that all vendor names must be uppercase.
    elsif (event_name = 'WHEN-VALIDATE-RECORD') then
      if (form_name = 'APXVENDR') then
        if (block_name = 'VENDOR') then
          copy(upper(name_in('VENDOR.NAME')), 'VENDOR.NAME');
        end if;
      end if;
    else
      null;
    end if;
  end event;
  */
  --
  -- Real code starts here
  --
    form_name      varchar2(30) := name_in('system.current_form');
 begin
  null;
 end event;
end custom;
```