



Advanced Functionality in Portal Applications

By Peter Brassington

Oracle Portal has rapidly made inroads in the Oracle community as a wonderful tool to build and deploy Web solutions. Developers see Portal's infrastructure as a great way for rapid application development and a significant advancement over the PL/SQL Web toolkit of earlier Oracle application servers.

The application component wizards in Oracle Portal provide developers with sufficient functionality to build Web applications. However, fully functional Web applications sometimes require richer functionality beyond what is generated from the component wizards. Many Web applications require functionality that users have come to expect from working with client server applications. However, when building thin client HTML interfaces, as in the case of Oracle Portal, the problem is twofold in that we must overcome the limitations of the wizards as well as the limitations of Web browsers. The problems introduced with HTML interfaces are due to the stateless nature of the Web, where it is difficult to maintain variables and object states that are always available. To overcome these limitations we have to exploit PL/SQL, JavaScript, and Oracle Portal's PDK. This article will address three problems that were encountered in building a fully functional Portal application and explain the solution for each.

Problem 1

Users have a form where a simple List of Values (LOV) is not enough since the LOV is very large and it is difficult to find desired values. The user wants to have an LOV that can be generated based on specific search criteria that will display additional fields in the list other the field being returned. The user will then select from this much shorter list and populate the form field. The LOV must also be able to populate multiple form fields based on the selected row.

Before discussing the solution, it is important to understand the concept of session storage and how it is used within Oracle Portal. Due to the stateless nature of the Web it is difficult to store variables that can easily be referenced at any point in time. To overcome this in Oracle Portal it is necessary to make use of built in session storage functionality. Session storage allows an application to store variables for the current session in the database and to reference these variables at any point after they are defined in the current session. A session store is only visible to the current user in the current session and its values are lost once the browser is closed or the user logs off.

There are two types of session storage that are used in the solution:

1. Session storage objects that are explicitly defined and loaded for storing your variables.
2. Module session storage that is created each time a module component is run. The module session stores information such as the state of a form and the current values of any of the form's fields.

For more information about session storage, refer to the PDK documentation.

Problem 1 Solution

Create an LOV Button in the form requiring the list. This button will call a Portal report with searchable parameters. The report will return rows from which the user can select the desired row. Each row in the report will have a button that, when pressed, will populate the form field(s) with the desired value(s) and return control to the calling form.

The above solution involves the following steps.

1. Add a button next to the form field requiring the LOV.
2. Define a SQL report that will generate the LOV. Include any parameters that may be useful in filtering the list generated by the report.
3. Define Custom logic on the form's button to do the following.
 - a. Create a session and store the form's name and module session ID.
 - b. Call the parameter form of the report that will generate the LOV.
4. Define a function that will generate a row level button in the report and include the function in the report's SQL.
5. Define a procedure to process the report buttons. This procedure should set form values in the calling form based on the selected record and return control to the calling form.

Step 1

The following form requires a user to enter a valid department number or search for it. This form has the LOV button already added next to the department number field. This button may have any user-preferred label as evidenced by the 'Search' label on the button.

Figure 1: Example Form

Step 2

Once the form is built, a report must be defined to return records that can be selected and returned to the calling form. The report shown in Figure 2 is the result set for the LOV.

	Dept No	Name	Location
Select	10	ACCOUNTING	NEW YORK
Select	20	RESEARCH	DALLAS
Select	30	SALES	CHICAGO
Select	40	OPERATIONS	BOSTON

Figure 2: Sample LOV Result Set

The report shown in Figure 2 is generated using the following query:

```
SELECT  PORTAL30_DEMO.DISPLAY_RPT_DEPT_BUTTON (A.DEPTNO) Sel, A.DEPTNO,
        A.DNAME, A.LOC, '</FORM>' fc
FROM    PORTAL30_DEMO.DEPT A
Where A.DEPTNO like: department
/* PORTAL30_DEMO.DISPLAY_RPT_DEPT_BUTTON(A.DEPTNO) generates the select button
on the report and captures the deptno to be used for additional processing.
'</FORM>' generates the closes form tag corresponding to the open form tag
for each Select button.*/
```

The following column formatting shown in Figure 3 must also be set in the report. The significant setting here is “Display As HTML.” This allows the report to interpret certain strings as HTML and not plain text.

Column	Column Heading Text	Sum Type	Align	Display as	Format Mask	Link	Edit Link
SEL	 		A	Left	HTML	%	
DEPTNO	Dept No		%	Left	Text	%	
DNAME	Name		A	Left	Text	%	
LOC	Location		A	Left	Text	%	
FC			A	Left	HTML	%	

Figure 3: Column formatting selection

Step 3

The following code is placed under the Custom event for the ‘Search’ button’s PL/SQL event handler.

```
DECLARE
/*Define and load a session that you will reference */
v_session portal30.wwsto_api_session :=
portal30.wwsto_api_session.load_session('CONTEXT', 'MY_SESSION');
BEGIN
/*Define and set variables (attributes) in the session */
v_session.set_attribute('FORM_SESSION_ID', p_session."_id");
v_session.set_attribute('FORM_NAME', p_session."_module".name);
/*Save the referenced session */
v_session.save_session;
/*Call the parameter form of the report that will generate the list */
portal30.wwa_app_module.set_target('PORTAL30_DEMO.RPT_DEPT.show_parms', 'CALL')
;
END;
```

Step 4

The following function generates the report buttons.

```
Function PORTAL30_DEMO.DISPLAY_RPT_DEPT_BUTTON (dept_no IN number) RETURN
varchar2 IS
tag
varchar2(4000);
BEGIN
tag := '<!-- Custom code for Select dept button -->'
|| '<FORM ACTION="/pls/' || portal30.wwctx_api.get_dad_name ||
'/portal30_demo.ADD_DEPT_NO">'
|| '<INPUT TYPE=HIDDEN NAME=p_dept_no VALUE="">'
|| '<INPUT TYPE=Submit NAME=p_button VALUE="Select"'
onClick="this.form.p_dept_no.value='
|| dept_no || ';'>";
return tag;
END; -- Function DISPLAY_RPT_DEPT_BUTTON
```

Step 5

Once a department is selected by pressing the button, the following procedure is called to set the department number in the calling form and return control to the form.

```
Procedure PORTAL30_DEMO.ADD_DEPT_NO (p_dept_no number, p_button varchar2
default null) IS
/*Define a module session that will reference the calling form module. */
p_session portal30.wwa_api_module_session;
/*Define a user session that will reference user-defined sessions */
v_session portal30.wwsto_api_session;
id number;
begin
/* Load the user defined session and retrieves the stored module name and
session id */
v_session := portal30.wwsto_api_session.load_session('CONTEXT', 'MY_SESSION');
id := v_session.get_attribute_as_number('FORM_SESSION_ID');
/* Load the module session and set the deptno field in the form */
p_session := PORTAL30.wwa_api_module_session.load_session(id);
if (p_session."_module".name = 'EXAMPLE_FORM') then
p_session.set_value('DEFAULT', 'A_DEPTNO', p_dept_no);
p_session.save_session;
/*Return control back to the calling form */
owa_util.redirect_url('/pls/' || portal30.wwctx_api.get_dad_name ||
'/PORTAL30.wwa_app_module.show?p_sessionid=' || p_session."_id");
end if;
exception when others then
null;
end;
```

Problem 2

Users want the ability to enter code fields and have the description fields populated automatically. Description fields should be populated as soon as the user navigates from the entered field. The description fields are display only fields and are not fields in the form’s base table.

Problem 2 Solution

The form must be refreshed whenever the code field changes value. When the form refreshes, the description must be looked up in the database and populate the description field in the form. This solution is not as straightforward in Oracle Portal since built-in Portal API’s cannot be used to populate non-base table fields.

continued on page 12

The steps for this solution are as follows.

1. Add a display field to the form. Set the field as updateable so that it shows up on the form as in Figure 1 (Dept Name).
2. Add the following code to the onChange JavaScript event handler for the field driving the description:

```
do_event(this.form,this.name,1,'ON_CLICK','');
```

This code will force the form to submit itself without any processing such as Insert or Update. The outcome of this is that the form refreshes itself from the database.

Add the following code to the form's "after displaying the page" PL/SQL block. This code will populate the Department Description field.

```
declare
v_dept_name varchar2(200);
v_dept_no number;
begin
/* Use built in Portal API's to grab the dept_no value */
v_dept_no := p_session.get_value_as_number(p_block_name => 'DEFAULT',
p_attribute_name => 'A_DEPTNO');
/*If the dept_no is valid fetch the description from the database otherwise
set Dept. description to 'INVALID DEPT'*/
if v_dept_no is not null then
begin
select dname into v_dept_name from portal30_demo.dept
where deptno = v_dept_no;
exception when no_data_found then
v_dept_name := 'INVALID DEPT.';
end;
end if;
/*Use JavaScript to set the description field value */
http.p('<SCRIPT LANGUAGE="Javascript1.1"> var deptname = new
String("EXAMPLE_FORM.DEFAULT.DEPT_DESC.01");
document.forms[0].elements[eval(deptname)].value="'||v_dept_name||'";</SCRIPT>
');
```

To find out the name of the form field, look at the HTML source of the displayed form.

Problem 3

A user requires a form to have additional buttons at the bottom of the form for navigation to other forms based on the data retrieved in the form.

Problem 3 Solution

To solve this problem, it is necessary to write code to generate buttons at the bottom of the form as well as code to support the buttons submitted by the procedure.

The form shown in Figure 4 will be used as the example for this solution. Notice the buttons at the bottom of the form. These buttons are only displayed after data is queried. The buttons displayed depend upon the form's data.

Figure 4: Example of buttons added to a form

The steps for the solution are as follows.

1. Create a procedure to generate a button. The procedure must include the HTML for displaying the buttons.
2. Create a procedure that will take form values when a button is pressed and use them in generating links to other components.
3. Add code to the form's "after displaying the page" PL/SQL block to call the procedure created in Step 1.

Step 1.

The following procedure is used to generate the buttons at the bottom of the form.

```
/* This procedure displays the buttons that appear at the bottom of the form
MD_FORM_COMPOUNDS.
These buttons navigate out to other components.*/
procedure compound_buttons(p_compound_id IN out number) is
v_comp_count number := 0;
begin
if p_compound_id is not null then
/*Define a PL/SQL form. 'process_compound_buttons' is the procedure that's
called when the form is submitted. The compound_id and the button name are
submitted when the buttons are processed. */
http.formopen('process_compound_buttons');
/* Define hidden form field for form values used in processing*/
http.formhidden('p_compound_id',p_compound_id);
select count(*) into v_comp_count
from compound_structures c
where c.compound_id = p_compound_id;
/* This section will be used to call a form in either Insert or Update mode
depending on the results of the above query. */
if v_comp_count = 0 then
http.formsubmit('p_request','Add Structure');
else
http.formsubmit('p_request','View Structure');
end if;
/* http.formsubmit submits form buttons for processing */
http.formsubmit('p_request','Associate Person to Compound');
http.formsubmit('p_request','Associate Parent to Compound');
http.formclose;
end if;
end;
```

Step 2.

The following procedure is used to process any buttons submitted by the procedure above.

```
/* This procedure processes the buttons that appear at the bottom of the form
MD_FORM_COMPOUNDS.
The procedure takes the compound_id and Button Names as parameters*/
PROCEDURE PROCESS_COMPOUND_BUTTONS(p_request IN VARCHAR2 default
null,p_compound_id IN VARCHAR2 default null) IS
    p_request_upper varchar2(60);
    temp_vals varchar2(400);
BEGIN
    p_request_upper := upper(p_request);

/*The following code calls a portal form in insert mode setting the form's
compound_id field*/
If p_request_upper = 'ADD STRUCTURE' then
    PORTAL30.wwa_app_module.link(p_arg_names =>
portal30.wvw_standard_util.string_to_table2('_moduleid:compound_id'),
p_arg_values =>
portal30.wvw_standard_util.string_to_table2('4103399196:'||P_compound_id));

/*The following code calls other portal forms querying them based on the
compound_id*/
elseif p_request_upper = 'VIEW STRUCTURE' then
    temp_vals := '4103399196:YES:'||to_number(p_compound_id)||':=';
    PORTAL30.wwa_app_module.link(p_arg_names =>
portal30.wvw_standard_util.string_to_table2('_moduleid:show_header:compound_i
d:compound_id_cond'), p_arg_values =>
portal30.wvw_standard_util.string_to_table2(temp_vals));
elseif p_request_upper = 'ASSOCIATE PERSON TO COMPOUND' then
    temp_vals := '4639809911:YES:'||to_number(p_compound_id)||':=';
    PORTAL30.wwa_app_module.link(p_arg_names =>
portal30.wvw_standard_util.string_to_table2('_moduleid:show_header:compound_i
d:compound_id_cond'), p_arg_values =>
portal30.wvw_standard_util.string_to_table2(temp_vals));
elseif p_request_upper = 'ASSOCIATE PARENT TO COMPOUND' then
    temp_vals := '12143076441:YES:'||to_number(p_compound_id)||':=';
    PORTAL30.wwa_app_module.link(p_arg_names =>
portal30.wvw_standard_util.string_to_table2('_moduleid:show_header:compound_i
d:compound_id_cond'), p_arg_values =>
portal30.wvw_standard_util.string_to_table2(temp_vals));
end if;
END;
```

Step 3

The following code is added to the form's "after displaying the page" PL/SQL block to call the procedure created in Step 1.

```
Declare
    v_compound_id varchar2(10);
begin
    /* captures the vale of the compound_id in the form to pass it to the
    procedure displaying the buttons. The procedure will only display buttons if
    this is a valid compound_id. If no values are passed, no buttons are
    displayed. */
    v_compound_id := p_session.get_value_as_NUMBER(p_block_name =>
'MASTER_BLOCK',
    p_attribute_name => 'A_COMPOUND_ID');
    sms.inventory_custom_logic.compound_buttons(v_compound_id);
end;
```

Conclusion

By exploiting the optional PL/SQL block in Portal components, you can build fairly powerful Portal applications. Some very basic knowledge of JavaScript is also helpful in adding functionality to Portal applications. In summary, Oracle Portal component wizards combined with custom PL/SQL and JavaScript provide developers with a powerful tool for building fully functional applications.



About the Author

Peter Brassington is the Director of Emerging Technologies for MFG Systems Corporation, a New Jersey based consulting firm specializing in Oracle solutions. Mr. Brassington has over 8 years of Oracle experience, including Web Development, Data Warehousing, Systems Design and Analysis, and Project Management. Peter has lately been focused on Oracle 9i Application Server and Oracle Portal. Peter is a frequent presenter at various user groups including NYOUG, NJOUG and DVOUG.