



An Oracle White Paper
February 2013

Oracle Fusion Middleware 11gR1 & 11gR2 - Integrating Oracle Reports with Oracle Forms

Introduction	1
Oracle Forms	2
Oracle Reports	2
The Oracle Forms RUN_REPORT_OBJECT Built-in	3
How to use RUN_REPORT_OBJECT	4
RUN_REPORT_OBJECT Examples	4
Using a Parameter List with RUN_REPORT_OBJECT	7
Calling Reports that Display a Parameter Form.....	8
Building a Procedure to Call Reports with Parameter Forms.....	10
The Oracle Forms WEB.SHOW_DOCUMENT Built-in	14
WEB.SHOW_DOCUMENT Syntax	15
Calling Reports using WEB.SHOW_DOCUMENT	15
Summary	17

Introduction

This paper discusses how to integrate Oracle Reports with Oracle Forms. After reading this whitepaper you will:

- Understand how to use the Oracle Forms RUN_REPORT_OBJECT built-in.
- Understand how a report is requested from an Oracle Forms application.
- Understand how to retrieve a completed report from Oracle Forms.
- Understand how to call reports which include a Reports parameter form.
- Understand how to call Oracle Reports from Oracle Forms when single sign-on (SSO) is enabled.

This document is intended for individuals with a working knowledge of how to write Oracle Forms application code and understand the basic functionality in Oracle Forms and Reports on the middle tier. Some understanding of HTTP Server and WebLogic Server functionality will also be helpful. The sample code found within this document is provided for illustration and educational purposes only. This code should not be used in production applications without first performing thorough and complete testing prior to use.

Oracle Forms

Oracle Forms consists of two high level components: the Oracle Forms Developer design-time component (aka the Forms Builder) and Oracle Fusion Middleware – Forms Services deployment (aka Forms Runtime) component. For the purpose of this document, we will only be discussing those features and built-ins that are necessary to call a report from an Oracle Forms application.

There are two Oracle Forms built-ins which are supported for calling Oracle Reports from Oracle Forms:

- RUN_REPORT_OBJECT
- WEB.SHOW_DOCUMENT

These built-ins are explained in more detail within the Oracle Forms Builder Online help. An explanation of how these will be used to call Oracle Reports will be explained later in this paper. More information about deploying Oracle Forms can be found in the “*Oracle Forms Deployment Guide*”, which is included in the Fusion Middleware 11g and 11gR2 documentation library on the [Oracle Technology Network](#) (OTN).

If your application is being migrated from an earlier version of Oracle Forms, specifically version 6.x or older and the built-in RUN_PRODUCT was used for Oracle Forms and Oracle Reports integration and you are not able or willing to rewrite your code to use RUN_REPORT_OBJECT, please refer to the documentation which discusses how to use the Forms Migration Assistant (FMA). This information can be found in the Fusion Middleware documentation library in the document titled, “*Oracle Forms Upgrading Oracle Forms 6i to Oracle Forms 11g*”.

Oracle Reports

Like Oracle Forms, Oracle Reports consists of a primary design-time tool commonly referred to as the Oracle Reports Builder and the Oracle Fusion Middleware – Reports Server component for deployment. The deployment component within Fusion Middleware is referred to as Oracle Reports Services or Server. Throughout this paper, the terms Reports Services and Reports Server are used interchangeably for the same component(s).

More information about deploying Oracle Reports can be found in the Oracle Reports deployment guide, titled “*Publishing Reports with Oracle Reports Services*” which is included in the Fusion Middleware 11g and 11gR2 documentation library on OTN.

The Oracle Forms RUN_REPORT_OBJECT Built-in

The most secure approach for calling Oracle Reports from Oracle Forms is to use the RUN_REPORT_OBJECT built-in. Because the user's database connection is implicitly passed from Oracle Forms to Oracle Reports on the middle tier server, there is no risk of interception as when passed such information in a URL.

In Oracle Forms Builder, to use the RUN_REPORT_OBJECT built-in, you will need to create a new Reports object under the "Reports" node in the Object Navigator. Each Reports object has a logical name, which is used within Forms to call the report from PL/SQL. You can optionally create a new Reports object for each physical Reports file. One Reports object can also be used with many physical Reports files. The attributes of this object can be set in the Builder's Property Palette at design-time or can be set programmatically at runtime.

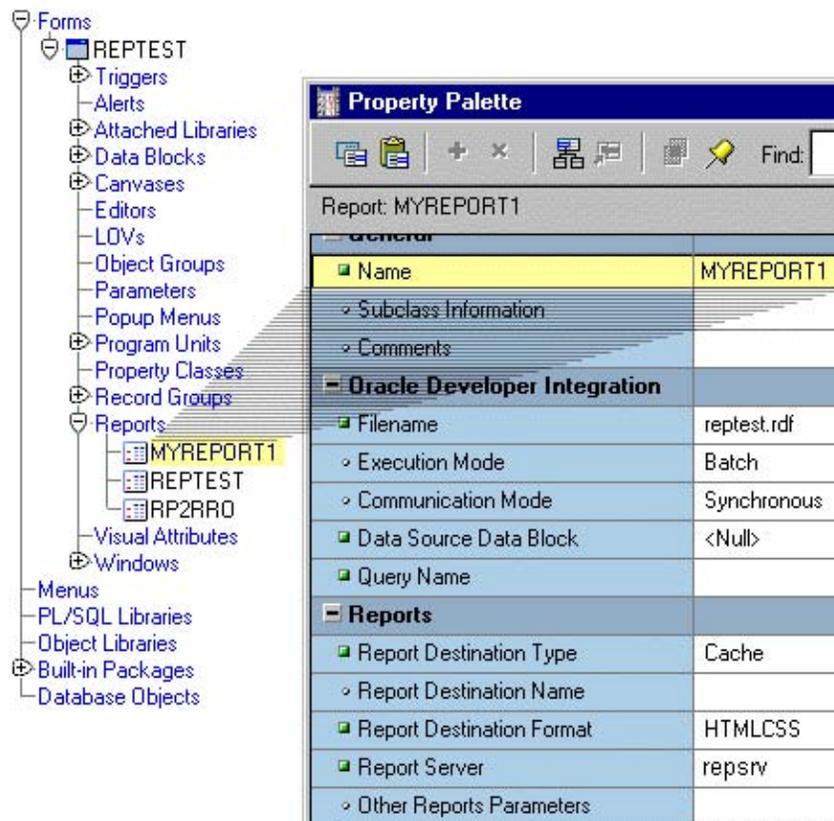


Figure 1: Oracle Forms Object Navigator and Property Palette. Note that the "Reports" node includes the objects "MYREPORT1", "REPEST", and "RP2RRO". The physical Oracle Reports file referenced by the "MYREPORT1" object is defined as "reptest.rdf". The Oracle Reports runtime settings below the "Reports" node in the Property Palette can be overwritten at runtime using SET_REPORT_OBJECT_PROPERTY.

How to use RUN_REPORT_OBJECT

To access a remote Reports Server using RUN_REPORT_OBJECT, Oracle Reports Services must be accessible for the Report object in Oracle Forms. You can do this dynamically, using the SET_REPORT_OBJECT_PROPERTY built-in, or statically, by entering the Oracle Reports Server name string into the Property Palette of the Report object.

It is also important to note that Oracle Forms Services and Oracle Reports Services must reside within the same network subnet in order to work properly. If they are not, either the Oracle Reports Naming Service or Oracle Reports Bridge can be used to overcome this particular configuration limitation. Refer to the *“Publishing Reports with Oracle Reports Services”* document previously mentioned for more information about using the Reports Naming Service or a Bridge.

RUN_REPORT_OBJECT Examples

Example 1

The following example runs a report using the Oracle Forms built-in RUN_REPORT_OBJECT. Note that at this point we are only requesting that a report be run. The data retrieved (i.e. report output) will not be returned to the end-user at this point. This may be desirable in some cases. If so, set the DESTYPE to “FILE” in order to permanently store the file on the server for later use.

In this example, the Reports object name is “MyReport1”. A user defined Reports parameter, “p_deptno”, is passed using the value of the “dept.deptno” field. The parameter form is suppressed using “paramform=no”.

```

DECLARE
    report_id Report_Object;
    ReportServerJob VARCHAR2(254);
BEGIN
    report_id := find_report_object('MyReport1');
    SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_COMM_MODE,SYNCHRONOUS);
    SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_DESTYPE,CACHE);
    SET_REPORT_OBJECT_PROPERTY(report_id, REPORT_DESFORMAT, 'PDF');
    SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_SERVER,'Repsrv');
    SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_OTHER,'p_deptno=||:Dept.Deptno||' paramform=no');
    ReportServerJob := run_report_object(report_id);
END;
```

Figure 2: General use of RUN_REPORT_OBJECT

Example 2

The following example uses a synchronous call to `RUN_REPORT_OBJECT` to run a report. It expects the Reports object name, the Reports Server name, and the desired output format (PDF, HTML, HTMLCSS, etc) to be passed as parameters. It will also attempt to verify that the report was successfully generated, and then display the results to the end user in a browser. The use of a procedure such as this is recommended in cases where the application is likely to call out to Reports from various places within the application.

```

PROCEDURE RUN_REPORT_OBJECT_PROC (vc_reportoj Varchar2, vc_reportserver varchar2, vc_runformat varchar2) IS
    v_report_id Report_Object;
    vc_ReportServerJob VARCHAR2(100); /* unique id for each Report request */
    vc_rep_status VARCHAR2(100); /* status of the Report job */
    vjob_id VARCHAR2(100); /* job_id as number only string*/
BEGIN
    /* Get a handle to the Report Object */
    v_report_id:= FIND_REPORT_OBJECT(vc_reportoj);

    /* Define the report output format and the name of the Reports Server as well as a user-defined parameter.
    Pass the department number from Forms to Reports. There is no need for a parameter form to be displayed,
    so paramform is set to "no".*/

    SET_REPORT_OBJECT_PROPERTY(v_report_id,REPORT_DESFORMAT,vc_runformat);
    SET_REPORT_OBJECT_PROPERTY(v_report_id,REPORT_DESTTYPE,CACHE);
    SET_REPORT_OBJECT_PROPERTY(v_report_id,REPORT_COMM_MODE,SYNCHRONOUS);
    SET_REPORT_OBJECT_PROPERTY(v_report_id,REPORT_SERVER,vc_reportserver);
    SET_REPORT_OBJECT_PROPERTY(v_report_id,REPORT_OTHER,'p_deptno=||:dept.deptno||'paramform=no');

    vc_ReportServerJob:=RUN_REPORT_OBJECT(v_report_id);
    vjob_id := substr(vc_ReportServerJob,instr(vc_ReportServerJob,'_',-1)+1);

    /* Check the report status. Because this was a synchronous call ( REPORT_COMM_MODE),
    the status check will only return FINISHED or an error. If COMM_MODE is set to "asynchronous", a timer
    should be used to periodically change the status of the running report before attempting to display it. */
    vc_rep_status := REPORT_OBJECT_STATUS(vc_ReportServerJob);
    IF vc_rep_status = 'FINISHED' THEN

    /* Call the Reports output to be displayed in the browser. The URL for relative addressing is valid
    only when the Reports Server resides on the same host as the Forms Server and is accessed via the same port.
    For accessing a remote Reports environment, you must use a fully qualified URL (i.e. http://hostname:port ) */

        WEB.SHOW_DOCUMENT ('/reports/rwservlet/getjobid||' vjob_id ||'?server=||vc_reportserver,'_blank');
    ELSE
        message ('Report failed with error message '||vc_rep_status);
    END IF;
END;
```

Figure 3: Using `RUN_REPORT_OBJECT` for integrating calls to Oracle Reports

If you are upgrading from Oracle Forms or Oracle Reports 6i, when calling `WEB.SHOW_DOCUMENT` you will need to modify the Reports `job_ID` that is retrieved by the `RUN_REPORT_OBJECT` built-in so that the Reports Server name is not included.

To use the procedure described above, you would pass the following information in a “When-Button-Pressed” trigger or other appropriate trigger:

```
RUN_REPORT_OBJECT_PROC (<'REPORT_OBJECT'>,<'REPORT_SERVER_NAME'>,<'FORMAT'>)
```

REPORT_OBJECT	Forms Report object name containing the rdf filename for the Report
REPORT_SERVER_NAME	Name of the Reports Server
FORMAT	Any of these formats: html html css pdf xml delimited rtf

Figure 4: Parameters needed to use `RUN_REPORT_OBJECT_PROC`

A synchronous call to Reports will cause the user to wait while the report is processed on the server.

For long-running Reports, it is best that the report be run asynchronously by setting the `REPORT_COMM_MODE` property to *asynchronous* and the `REPORT_EXECUTION_MODE` to *batch*. For example:

```
SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_EXECUTION_MODE,BATCH);
SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_COMM_MODE,ASYNCHRONOUS);
```

After calling `RUN_REPORT_OBJECT`, you must create a timer to run periodic checks on the current `REPORT_OBJECT_STATUS` in a When-Timer-Expired trigger. After the report is generated, the “When-Timer-Expired” trigger calls the `WEB.SHOW_DOCUMENT` built-in to display the Reports output file, identified by its unique `job_ID`, to the client’s browser.

Here is an example of how the report status can be checked from the When-Timer-Expired trigger:

```
(...)
/* :global.vc_ReportServerJob needs to be global because the information about the Report job_id is shared
between the trigger code that starts the report and the trigger code When-Timer-Expired that checks the Report status. */

vc_rep_status:= REPORT_OBJECT_STATUS(:global.vc_ReportServerJob);
IF vc_rep_status='FINISHED' THEN
    vjob_id :=
        substr(:global.vc_ReportServerJob,length(reportserver)+2,length(:global.vc_ReportServerJob));
    WEB.SHOW_DOCUMENT ('/reports/rwservlet/getjobid'||vjob_id||'?server=||vc_reportserver,'_blank');
    DELETE_TIMER (timer_id); -- Report done. No need to check any more.
ELSIF vc_rep_status not in ('RUNNING','OPENING_REPORT','ENQUEUED') THEN
    message (vc_rep_status||' Report output aborted');
    DELETE_TIMER (timer_id); -- Report failed. No need to check any more.
END IF;
(...)
```

Figure 5: Performing asynchronous call to Reports and checking its status from When-Timer-Expired

Using a Parameter List with RUN_REPORT_OBJECT

With the RUN_PRODUCT¹ built-in (*no longer supported for use in Oracle Forms*), Reports system parameters and user-defined parameters are passed in a parameter list. The same parameter lists can be used with RUN_REPORT_OBJECT, with the exception of the system parameters, which need to be set with the SET_REPORT_OBJECT_PROPERTY built-in.

REPORT_EXECUTION_MODE	BATCH or RUNTIME ²
REPORT_COMM_MODE	SYNCHRONOUS ASYNCHRONOUS
REPORT_DESTYPE	FILE PRINTER MAIL CACHE ³
REPORT_DESFORMAT	HTML HTMLCSS PDF RTF XML DELIMITED SPREADSHEET ²
REPORT_FILENAME	The report filename
REPORT_DESNAME	The report destination name
REPORT_SERVER	The Report Server name

Figure 6: List of system parameters used by RUN_REPORT_OBJECT.

If your existing parameter list already contains definitions for system parameters, you may experience errors. To prevent such problems from occurring, modify the parameter list itself, either by removing the entries for DESNAME and DESTYPE, or by adding

```
DELETE_PARAMETER (<parameter list>,'<name>');
```

to your code before using SET_REPORT_OBJECT_PROPERTY. The syntax for using parameter lists in RUN_REPORT_OBJECT is as follows:

```
ReportServerJob := RUN_REPORT_OBJECT (report_id,paramlist_id);
```

Note that having DESTYPE defined both in the parameter list and in SET_REPORT_OBJECT_PROPERTIES should not prevent the module from compiling, but may prevent it from running.

-
1. Using RUN_PRODUCT to generate Reports output is not supported in Oracle Forms 9.0.4 and greater. Forms module containing integrated calls to Reports using RUN_PRODUCT will not compile. Refer to the Forms Upgrade Guide for information on how to use the Forms Migration Assistance (FMA) or consider updating your code as discussed in this paper.
 2. Report_Execution_Mode is a client/ server feature and no longer used in Oracle Forms. However, setting the value to either BATCH or RUNTIME is required.
 3. Additional DESTYPE and DESFORMAT values can be found in the Oracle Reports deployment guide, titled "Publishing Reports with Oracle Reports Services"

Calling Reports that Display a Parameter Form

Using the previous examples, a report's parameter form will not work when called from RUN_REPORT_OBJECT because the <ACTION> attribute in the generated report HTML parameter form is empty. RUN_REPORT_OBJECT calls are sent directly to Oracle Reports on the server. Therefore, Oracle Reports cannot access the web environment to obtain the information required to populate the action attribute when generating the HTML parameter form.

The <ACTION> attribute is part of the standard HTML <FORM> tag that defines what to do when a user presses the submit button. The <ACTION> attribute in the Oracle Reports parameter form should contain hidden runtime parameters that are required to process the request after the user presses the submit button. Additional code is required to overcome this condition.

Solving the Problem with "PFACTION"

"PFACTION" is a command line parameter in Oracle Reports that can be used to add the hidden runtime parameter to a report's parameter form when calling Oracle Reports from Oracle Forms, using RUN_REPORT_OBJECT. The syntax for the value "pfaction" parameter looks like this:

```
<request URL_to_rwservlet?_hidden_<encoded_original_url_query_string>
```

The "request URL_to_rwservlet" portion contains the protocol, the host name, the port, the Oracle Reports web context path and the Oracle Reports Servlet name. For example:

```
http://someDomain.com:8888/reports/rwservlet
```

The "encoded_original_url_query_string" portion is a duplicate of all Oracle Reports system and application parameters passed from Oracle Forms to Oracle Reports using SET_REPORT_OBJECT_PROPERTY, encoded in a URL. For example, these Reports command line parameters,

```
destype=cache desformat=htmlcss userid=scott/tiger@orcl
```

would be added as a value to the "pfaction" parameter like this,

```
destype=cache%20desformat=htmlcss%20userid=scott%2Ftiger%40orcl
```

In order to call a report that contains a parameter form from Oracle Forms using RUN_REPORT_OBJECT, do the following:

1. Provide the protocol, the host name of the server, and the server port
2. Provide the virtual path and name of the Oracle Reports Servlet
3. URL encode the parameter value of the "pfaction" command parameter

The following Forms built-in will be used to pass the "pfaction" command from Oracle Forms Services to Reports:

```
SET_REPORT_OBJECT_PROPERTY (rep_id,REPORT_OTHER, 'pfaction ...');
```

Note that if you are using the REPORT_OTHER built-in to pass application parameters to Oracle Reports, the application parameters must also be contained in the pfaction parameter.

Using PL/SQL Functions to Encode URL Parameters

Because the “pfaction” parameter is added as an ACTION parameter to the Oracle Reports HTML parameter form, it is important to ensure that no un-encoded characters are included or errors may result. One example that may cause problems when embedded in a URL is a blank (space). Therefore, most developers URL-encode blank as “%20”. The PL/SQL function “ENCODE”, as listed below, is an example that encodes the following characters: “;”, “/”, “?”, “:”, “@”, “+”, “\$”, “,” and “ ” (semicolon, forward slash, question mark, colon, at, plus sign, dollar sign, comma, and blank space).

```

FUNCTION ENCODE (URL_PARAMS_IN Varchar2) RETURN VARCHAR2 IS
    v_url          VARCHAR2(2000) := URL_PARAMS_IN; -- Url string
    v_url_temp     VARCHAR2(4000) := ""; -- Temp URL string
    v_a            VARCHAR2(10); -- conversion variable
    v_b            VARCHAR2(10); -- conversion variable
    c              CHAR;
    i              NUMBER(10);
BEGIN
    FOR i IN 1..LENGTH(v_url) LOOP
        c:= substr(v_url,i,1);
        IF c in (':','/','?',':','@','+','$',' ','') THEN
            v_a := ltrim(to_char(trunc(ascii(substr(v_url,i,1))/16)));
            IF v_a = '10' THEN v_a := 'A';
                ELSIF v_a = '11' THEN v_a := 'B';
                ELSIF v_a = '12' THEN v_a := 'C';
                ELSIF v_a = '13' THEN v_a := 'D';
                ELSIF v_a = '14' THEN v_a := 'E';
                ELSIF v_a = '15' THEN v_a := 'F';
            END IF;
            v_b := ltrim(to_char(mod(ascii(substr(v_url,i,1)),16)));
            IF v_b = '10' THEN v_b := 'A';
                ELSIF v_b = '11' THEN v_b := 'B';
                ELSIF v_b = '12' THEN v_b := 'C';
                ELSIF v_b = '13' THEN v_b := 'D';
                ELSIF v_b = '14' THEN v_b := 'E';
                ELSIF v_b = '15' THEN v_b := 'F';
            END IF;
            v_url_temp := v_url_temp||v_a||v_b;
        ELSE
            v_url_temp :=v_url_temp||c;
        END IF;
    END LOOP;
    return v_url_temp;
END;
```

Figure 7: PL/SQL function to URL-encode strings

Building a Procedure to Call Reports with Parameter Forms

Calling Oracle Reports using RUN_REPORT_OBJECT is best handled by a generic PL/SQL procedure in Forms. To successfully call a report from Oracle Forms, the following minimum information needs to be passed to the Reports pfaction command:

- Desformat, to determine the Reports output format
- Destype, to determine the output device, (printer or cache)
- Userid, to connect Reports to the database
- Reports file name, to specify the Reports module to be executed
- Paramform = yes, to enable the Reports parameter form to be shown in the client browser

The generic PL/SQL procedure handling calls to Oracle Reports using RUN_REPORT_OBJECT requires a Reports Object node to be created in Forms as mentioned earlier in this paper. One Reports Object node can be used to run any Report.

The following arguments are expected by this procedure:

report_id	The Report Object as obtained by a call to FIND_REPORT_OBJECT('<name>');
report_server_name	The name of the Reports Server, for example: "repserv11g"
report_format	HTML, HTMLCSS, PDF, RTF, XML ⁴
report_destype_name	CACHE, FILE, MAIL, PRINTER ⁴
report_file_name	The Reports source module with or without extension
report_other_param	Any other parameter like paramform or custom application parameters. If printing to FILE or MAIL, desname needs to be provided as 'otherparam'
report_servlet	The virtual path for the Reports Servlet. The virtual path, if not specified as relative, must contain the protocol (http or https), the host name, the port, the Reports context root (i.e. /reports) and the name of the Servlet (i.e. rwservlet).

Figure 8: Parameters passed in by the generic procedure example

4. Additional DESTYPE and DESFORMAT values can be found in the Oracle Reports deployment guide, titled "*Publishing Reports with Oracle Reports Services*"

```

PROCEDURE RUN_REPORT_OBJECT_PROC (
    report_id          REPORT_OBJECT,
    report_server_name VARCHAR2,
    report_format      VARCHAR2,
    report_destype_name NUMBER,
    report_file_name   VARCHAR2,
    report_otherparam  VARCHAR2,
    reports_servlet    VARCHAR2) IS
    report_message     VARCHAR2(100) := "";
    rep_status         VARCHAR2(100) := "";
    vjob_id            VARCHAR2(4000) := "";
    hidden_action      VARCHAR2(2000) := "";
    v_report_other     VARCHAR2(4000) := "";
    i                  number (5);
    c                  char;
    c_old              char;
    c_new              char;
BEGIN
    -- Set up Reports runtime parameters
    SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_COMM_MODE,SYNCHRONOUS);
    SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_FILENAME,report_file_name);
    SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_SERVER,report_server_name);
    SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_DESTYPE,report_destype_name);
    SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_DESFORMAT,report_format);

    -- Set up string for pfaction parameter
    hidden_action := hidden_action || '&report=' ||
        GET_REPORT_OBJECT_PROPERTY(report_id,REPORT_FILENAME);
    hidden_action := hidden_action || '&destype=' ||
        GET_REPORT_OBJECT_PROPERTY(report_id,REPORT_DESTYPE);
    hidden_action := hidden_action || '&desformat=' ||
        GET_REPORT_OBJECT_PROPERTY (report_id,REPORT_DESFORMAT);

    -- Note that the username and password information will be visible within the html source
    -- of the parameter form. This can be prevented by enabling single sign-on.
    IF get_application_property (sso_userid) IS NULL Then
        -- No SSO user
        hidden_action := hidden_action || '&userid=' ||
            get_application_property(username) || '/' ||
            get_application_property(password) || '@' ||
            get_application_property(connect_string);
    ELSE
        -- SSO user identified
        hidden_action := hidden_action || '&ssoconn=' || get_application_property(config);
    End if;
    -- The Reports "other" parameters are passed as key value pairs. For example: "key1=value1 key2=value2..."

```

Figure 9: Generic procedure to call reports which use parameter forms. (part 1 of 2)

```

-- The following loop replaces the delimited blank with an "&" used on the web. This replacement only works for
-- values that don't include blanks themselves. If this is a requirement, then you need to customize this code accordingly.

-- c_old is initialized with a dummy value
c_old := '@';
FOR i IN 1..LENGTH(report_otherparam) LOOP
    c_new:= substr(report_otherparam,i,1);
    IF (c_new = ' ') THEN
        c:='&';
    ELSE
        c:= c_new;
    END IF;
-- eliminate multiple blanks
    IF (c_old = ' ' and c_new = ' ') THEN
        null;
    ELSE
        v_report_other := v_report_other||c;
    END IF;
-- save current value as old value
    c_old := c_new;
END LOOP;

hidden_action := hidden_action || '&' || v_report_other;

--"reports_servlet" contains the path to the Reports Servlet
-- Example 1:
--      Forms and Reports are on the same host and accessed using the same port (e.g. 8888)
--      reports_servlet := '/reports/rwservlet'
-- Example 2:
--      Forms and Reports are on separate hosts and/or are accessed using a different port (e.g. 9001 & 9002)
--      reports_servlet := 'http://host:port/reports/rwservlet'

hidden_action := reports_servlet||'?_hidden_server='||report_server_name|| encode(hidden_action);
SET_REPORT_OBJECT_PROPERTY (report_id,REPORT_OTHER,'pfaction='||
                            hidden_action||' '||report_otherparam);

-- call Reports
report_message := run_report_object(report_id);
rep_status := report_object_status(report_message);
IF rep_status='FINISHED' THEN
    vjob_id :=substr(report_message,length(report_server_name)+2,length(report_message));
    WEB.SHOW_DOCUMENT(reports_servlet||'/getjobid'||vjob_id||'?server='||report_server_name,'_blank');
ELSE
-- handle errors here
    message(rep_status);
END IF;
END;
```

Figure 10: Generic procedure to call reports which use parameter forms. (part 2 of 2)

Examples of How to Call the Generic Procedure

Assuming that a Reports Object “reptest” exists in the Forms module, the following code can be used in a WHEN-BUTTON-PRESSED trigger to execute a Report that has a parameter form.

The generic PL/SQL procedure requires the Reports Object to be passed as an argument. The Reports Object of a given Reports Object name can be obtained using the FIND_REPORT_OBJECT built-in.

Because it is assumed that the server running Forms Services also hosts the Reports Services, no host name needs to be passed as an argument for the Reports Servlet path variable. Instead ‘/reports/rwservlet’ can be used as a relative path. Paramform=yes makes Reports first expose its parameter form. The parameter form will always display in HTML even if the Reports destination format is PDF.

```
(...)
-- Find the id of the Reports Object in Forms
      report_id:=FIND_REPORT_OBJECT('reptest');
-- Call the generic PL/SQL procedure to run the Reports
-- Remember that, in this example the value of reports_server can only be a
-- relative path if the same host and port are used to access both Forms and Reports.
      RUN_REPORT_OBJECT_PROC( report_id,
                              'repserv1-pc',
                              'HTMLCSS',
                              CACHE,
                              'REPTTEST',
                              'paramform=yes',
                              '/reports/rwservlet');
(...)
```

Figure 10: Generic procedure to call reports which use parameter forms.

In the example below, an additional parameter is passed to Reports. Note that the delimiter is a blank character between the first key-value pair and the second.

```
(...)
-- Find the id of the Reports Object in Forms
      report_id:=FIND_REPORT_OBJECT('reptest');
-- Call the generic PL/SQL procedure to run the Reports
-- Remember that, in this example the value of reports_server can only be a
-- relative path if the same host and port are used to access both Forms and Reports.
      RUN_REPORT_OBJECT_PROC( report_id,
                              'repserv1-pc',
                              'HTMLCSS',
                              CACHE,
                              'REPTTEST',
                              'paramform=no p_deptno=10',
                              '/reports/rwservlet');
(...)
```

Figure 11: Calling a report that has no parameter form, but requires p_deptno to be passed as a runtime parameter.

If the Reports Services runs on a different machine or with a different port number than Forms Services, you'll need to provide the fully qualified URL for the Reports Servlet. Note that these examples work with the PL/SQL procedure `RUN_REPORT_OBJECT_PROC`, which is explained earlier in this document. You are free to create your own PL/SQL code to handle the “pfactions” command in your applications.

```
(...)
-- Find the id of the Reports Object in Forms
      report_id:=FIND_REPORT_OBJECT('reptest');
-- Call the generic PL/SQL procedure to run the Reports
-- Remember that, in this example the value of reports_server can only be a
-- relative path if the same host and port are used to access both Forms and Reports.
      RUN_REPORT_OBJECT_PROC( report_id,
                              'rep serv1-pc',
                              'HTMLCSS',
                              CACHE,
                              'REPTTEST',
                              'paramform=yes',
                              'http://someServer:9002/reports/rwervlet');
(...)
```

Figure 12: Calling a report which has a parameter form, but whose Reports Servlet listens on a different port (or host).

The Oracle Forms `WEB.SHOW_DOCUMENT` Built-in

Use the `WEB.SHOW_DOCUMENT` built-in to access any web site (URL) from a Forms application. This built-in passes the provided URL to the client's default web browser. Oracle Forms has no control over the receiving browser. The built-in simply passes the URL to the browser. It is recommended that only web protocols be used with this built-in although it is possible to access some local content. For example, this built-in works best with protocols like HTTP, HTTPS, FTP, etc. You can use others such as MAIL, MAILTO, FILE, but these are not recommended. These others should be accessed using other means such as WebUtil. Refer to the Forms Builder Online Help for information about WebUtil.

WEB.SHOW_DOCUMENT Syntax

```
WEB.SHOW_DOCUMENT (URL, DESTINATION);
```

URL	The URL is passed as a string ('http://www.oracle.com'), in a variable, or as a combination of both. If the addressed web page is located on the same host as the Forms Server, a relative path could be used ('/virtual_path/').
DESTINATION (aka TARGET)	Definition of the target where the addressed web page should be displayed. Values must be single-quoted and lower case.
	<p><code>_blank</code> (default)</p> <p>Loads the document into a new, unnamed top-level window. 'windowName'. Displays the document in a window named windowName. This window is created if necessary.</p> <p><code>_self</code></p> <p>Loads the document into the same frame or window as the source</p> <p><code>_parent</code></p> <p>Load the document into the parent window or frameset containing the hypertext reference. If the reference is in a window or top-level frame, it is equivalent to the target <code>_self</code>.</p> <p><code>_top</code></p> <p>Loads the document into the window containing the hypertext link, replacing any frames currently displayed in the window.</p> <p><code><target name></code></p> <p>Displays the Web page in a frame specified by the target_name.</p>

Figure 13: WEB.SHOW_DOCUMENT syntax

Calling Reports using WEB.SHOW_DOCUMENT

In the previous examples, it was illustrated how `RUN_REPORT_OBJECT` could be used to request that a report be generated then have its output opened by calling the Reports `GETJOBID` command in a `WEB.SHOW_DOCUMENT` call. For example:

```
WEB.SHOW_DOCUMENT(reports_servlet||'/getjobid'||vjob_id||'?server='||report_server_name,' _blank');
```

Another option for calling Oracle Reports from Oracle Forms is to access the Reports Servlet directly using the form using `WEB.SHOW_DOCUMENT`. The necessary URL would use the following syntax example:

```
http://<hostname>:<port>/reports/rwservlet?server=<reportserver>&report=<report>.rdf&desformat=[htmlcss|pdf|xml|delimited]&destype=cache&userid=<user/pw@database>&paramform=[no|yes]
```

The following example calls a report from a form. It assumes that the user parameter “p_deptno” is read from a form’s item “deptno” in the block “dept.”

```

/* WHEN-BUTTON-PRESSED */
DECLARE
    vc_url varchar2(200);
BEGIN
    vc_url:='http://<hostname><port>/reports/rwservlet?server=||
'Repsrv&report=reptest.rdf&desformat=htmlcss&destype=cache'||
'&userid=user/pw@database&p_deptno=||:dept.deptno||&paramform=no';
    WEB.SHOW_DOCUMENT(vc_url,'_blank');
END;

```

Figure 14: General use of WEB.SHOW_DOCUMENT calling the Reports Servlet

Here is an example of how to use relative addressing if the Oracle Reports Server is installed on the same host as Oracle Forms.

```

/* WHEN-BUTTON-PRESSED */
DECLARE
    vc_url varchar2(100);
BEGIN
    vc_url:='/reports/rwservlet?server=Repsrv&report=reptest.rdf&desformat=htmlcss'||
'&destype=cache&userid=user/pw@database&p_deptno=||:dept.deptno||&paramform=no';
    WEB.SHOW_DOCUMENT(vc_url,'_blank');
END;

```

Figure 15: Using a relative path to access the Reports Servlet from Forms

Hiding the Username and Password

To execute a report in this manner, the database connect information must be passed as part of the request URL. Adding sensitive user information to any URL request is a serious security issue because all URLs requested by a user can be looked up in the Browser’s URL history or easily captured in network traffic. To avoid this, implement single sign-on and omit the username and password from the request.

For more information on how to implement a single sign-on solution, refer to the Fusion Middleware documentation library for your version.

Summary

Oracle Forms and Oracle Reports can be integrated using either the `RUN_REPORT_OBJECT` built-in or the Forms `WEB.SHOW_DOCUMENT` built-in. Use the `RUN_REPORT_OBJECT` built-in to securely pass reports parameters to Oracle Reports Services. Using `RUN_REPORT_OBJECT` is also a good way to seamlessly request reports which will not immediately provide output to the end-user. `WEB.SHOW_DOCUMENT` is preferred for simple requests and those which do not require the passing of any sensitive data or user information.

Using single sign-on, users are automatically authenticated to run Oracle Reports Services if they have already been authenticated to access the Oracle Forms application which called the report.

Additional information about this topic can be found in the Forms Builder Online Help and the Forms Deployment Guide.



Oracle Fusion Middleware 11gR1 and 11gR2 –
Integrating Oracle Reports with Oracle Forms

February 2013

Author: Michael Ferrante

Contributing Authors: Frank Nimphius

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com



Oracle is committed to developing practices and products that help protect the environment

Copyright © 2013, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group. 0612

Hardware and Software, Engineered to Work Together