Integrating Oracle9*i*AS Reports in Oracle9*i*AS Forms

An Oracle White Paper

Updated version for Forms 6i patch 5and higher, covering Oracle9i Application Server integration and the new Forms Listener Servlet Architecture

September 2001

ORACLE

Forms Services 6i Reports Integration

EXECUTIVE OVERVIEW

Forms Services 6*i* is a component of Oracle9*i* Application Server. The official component name is Oracle9*i*AS Forms Services. Therefore, Forms Services 6*i* and Oracle9*i*AS Forms Services are synonymous and may be used interchangeably throughout this paper.

Oracle9*i*AS Forms Services 6*i* can be used to Web deploy Oracle Forms applications built in a client-server architecture. More and more companies are moving from client-server computing to a pure Web architecture while trying to leverage existing programming skills.

Feedback from Oracle customers, user groups, trade shows, and the customer advisory board indicate that companies want to leverage Forms-built applications when moving towards a 100 % Web architecture rather than redevelop their business applications for the Web from scratch.

This paper focuses on deploying client-server Forms applications with integrated Oracle9*i*AS Reports to the Web. The target audience for this paper includes technical software managers, consultants, and project leads who want to learn about the benefits of such a configuration.

After reading this paper you'll be able to identify areas in your application that can better scale on the Web. You'll also be able to incrementally move applications with embedded report calls to the Web for a 100 % three-tier architecture solution.

Experienced Forms developers can use the coding examples and deployment tips in this paper to help guide them towards Web deployment. If you are new to Oracle9iAS Forms Services and Oracle9iAS Reports Services, but are familiar with running client-server Forms applications from previous or current releases, you'll learn about the basic Web architectures provided by these products and how to use them in your own applications.

Furthermore, this paper covers changes in your programming calls to integrated Oracle9iAS Reports in preparation for future upgrades to Oracle9iAS Forms Services.

HOW THIS PAPER IS STRUCTURED

This paper is structured into three main sections depending on your level of experience.

Start here if you are new to running Forms applications on the Web

Section 1: Introduction to Forms Services and the Basics of Calling Embedded Reports in Forms on the Web

This section introduces the Forms and Reports Services architecture, explaining the differences between the client-server and the Web architecture. It covers installation and configuration of Forms and Reports Services and explains the configuration files involved. Although the installer configures Forms and Reports services out-of-the-box, including all settings required to run embedded reports on the Web, a basic understanding of these settings is considered helpful. The RUN_REPORT_OBJECT built-in also is re-introduced in this section.

Start here if you already know how to run Forms on the Web but still run client-server applications. Use these recommendations to run embedded calls to Reports without modifying Forms application modules.

Section 2: Forms Services Applications With Integrated Calls to Oracle Reports Using the Reports Runtime Engine

This section explains how to use integrated calls to Oracle Reports in Forms Web applications, using the reports runtime engine in the same way it is used in client-server. You'll learn that the reports runtime engine is no longer located on the local client but resides on the middle-tier server.

The Forms built-ins RUN_PRODUCT and RUN_REPORT_OBJECT are explained using the reports runtime engine on the Web. This section helps users who want to move their Forms client-server applications to the Web but don't want to change embedded calls to Oracle Reports. Not changing embedded calls to Oracle Reports is a considerable but intermediate step when migrating large Forms client-server applications to the Web.

Start here if you already run your Forms applications on the Web but don't use Reports Services for integrated calls to Oracle Reports as recommended by Oracle.

Section 3: Forms Services applications With Integrated calls to Oracle Reports Using Reports Services

This final section helps you become an expert. You learn how to use the RUN_REPORT_OBJECT and the WEB.SHOW_DOCUMENT built-in to access Oracle9*i*AS Reports Services for embedded calls to Oracle Reports in Forms. This implementation is recommended by Oracle when running Web-deployed Forms applications. You learn how to handle Reports parameter forms in the Web when called from Forms applications and how to gain access to local printers from the Web. Hints, tips, and examples are provided to make reporting on the Web as comfortable as reporting on the client-server architecture.

SECTION 1: INTRODUCTION TO FORMS SERVICES AND THE BASICS OF CALLING EMBEDDED REPORTS IN FORMS ON THE WEB

Oracle9i Application Server Enterprise Edition integrates the Forms Server and the Reports Server. Their official names as part of Oracle9*i*AS product branding are Oracle9*i*AS Forms Services and Oracle9*i*AS Reports Services.

In this document, both the old product names and the new product names are used interchangeably.

When you use Oracle9iAS Forms Services to migrate your existing Forms applications from a client-server architecture to the Web, you may wonder how to include calls to Oracle Reports that are currently handled by a client-side report runtime environment. Once the Web architecture is in place, the client-side runtime environment no longer exists. As a result, reporting must be done on the server side.

There are several ways to start a report from a Forms application running on the Web. Coming from the client-server paradigm, most Forms applications use one of the following built-ins to create a report using the Reports client runtime engine:

- RUN_PRODUCT built-in
- RUN_REPORT_OBJECT built-in

Technically, the same runtime engine can still perform reporting on the Web. However, a more enhanced solution would be to have a Forms Server application call Oracle Reports Services to print reports on the Web using the following built-ins:

- RUN_REPORT_OBJECT built-in
- WEB.SHOW_DOCUMENT built-in

This white paper explores various ways to implement calls to Oracle Reports from Forms applications running on the Web.

Understanding Forms 6i Services in Oracle9i Application Server

Before discussing the integration of Oracle9*i*AS Reports in Oracle9*i*AS Forms, when transitioning from a Forms client-server application to the Web, you must understand the technical differences of both architectures.

Firstly, the Web is a request model; nothing is sent to the client without the client first requesting it. This is also true for the output from integrated Reports in Forms. In a client-server architecture, when the report call is handled by the client runtime engine, the preview screen is automatically displayed once the report has finished running. This is possible because Forms Services and Reports Services are installed on the same machine. The Web is server centric; there are no local installations other than Oracle JInitiator, a certified client-side Java Virtual Machine to execute Forms Web application interfaces, if needed.

More information about client platform support and JInitiator can be found at http://otn.oracle.com

Client Server Architecture vs. Web Architecture

In a client-server architecture, the locally installed Forms runtime engine interprets the application code stored in Forms application modules. The user interface is rendered with native operating system interface classes such as Motif on UNIX or Windows on the Microsoft platform. The Forms runtime engine handles the

business logic and communicates with the generic user interface to display the application's graphical elements. Access to the file system is handled locally.

On the Web, the business logic is separate from the user interface. The business logic and the Forms runtime engine operate on the server, while the user interface gets downloaded to the Web client rendered through generic Java classes.

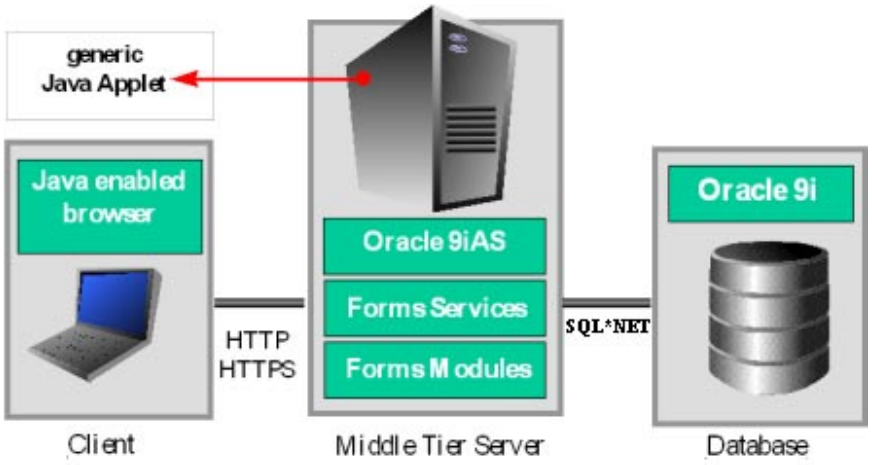

Figure 1: Forms Web architecture

More information about the Forms Services Web architecture can be found at http://otn.oracle.com/products/forms

The goal of this paper is to understand the impact that the Web architecture has on Oracle Reports integrated in Forms applications. A report called by a Forms application on the Web is executed on the server and the output is sent to the server first. In addition, Oracle9iAS Reports configurations are set on the server, not on the client.

You can migrate an Oracle report integrated in a Forms application running in a client-server architecture to the Web without any modifications. The same report definition file (rdf or rep file) operates on the Web using either the reports background engine or the Oracle9*i*AS Reports Services (Reports Server).

The main difference is that the report output needs to be downloaded to the client browser in a Web enabled format like HTML, HTMLCSS, XML, RTF, CSV, or PDF. The download can be performed automatically using a synchronous call like in RUN_PRODUCT built-in or programmatically when using Oracle9*i*AS Reports Services, which is the recommended method. Using Reports Services, the Report definition files can be on the same or different machine from the Forms Services.

In a Web architecture, the business logic for both applications, Forms and Reports, is located on the middle-tier server and the database connection is established using SQL*Net.

The Forms Services application calls integrated Oracle Reports with a server-side call and downloads the Reports output to the client, using the WEB.SHOW_DOCUMENT() built-in.

In the next section, a description of the two available Forms Services architectures on the Web are presented:

- Forms Listener architecture
- Forms Listener Servlet architecture (as of Forms 6*i* patch 4 and above)

Depending on the architecture you choose, configuring embedded Oracle Reports for Web deployment in Forms applications differs.

Forms Listener Architecture

The Forms Listener architecture was first introduced in Oracle Forms release 4.5. The Forms Listener passes a user request for a Forms application to the server-side Forms Web runtime engine. The Forms Listener and the Web runtime are standalone HTTP servers handling all HTTP traffic between the Forms client and the associated user process on the server.

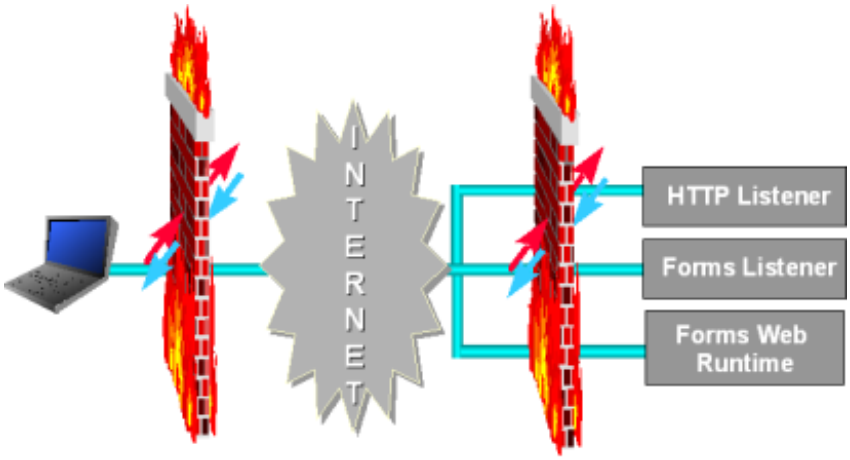

Figure 2: Forms Listener architecture

The Oracle Forms environment variables including Forms60_Path, Report60_Path, and ORACLE_HOME are set either in the Windows registry or in the environment on UNIX.

The environment in which the Forms Listener is started defines the configurations for the user's Web processes. All configurations for Reports integrated in Forms on the Web are set with the Forms Listener unless Reports Services are used. If your application supports different languages, then one Forms Listener process is required for each individual language.

The Forms Listener is automatically configured when Oracle9*i* Application Server Enterprise Edition Release 1.02.*x* is installed and will be supported throughout the lifecycle of Forms 6*i*. However, starting with Oracle9*i*AS Forms Services Release 2, the Forms Listener architecture will be desupported.

For more information about the Forms Listener architecture, see the architecture White Paper available on the Oracle Technology Network at http://otn.oracle.com/products/forms

Oracle therefore recommends that you use the Forms Listener Servlet architecture described below when migrating client-server applications with integrated reports to the Web.

Forms Listener Servlet Architecture

The difference between the Forms Listener and the Forms Listener Servlet architecture is that the latter uses the same port as the Oracle HTTP Server *powered by Apache* when communicating with the browser client. With the Forms Listener Servlet, you can run an application with different language settings by configuring the Forms Listener Servlet instead of starting a Forms Listener process for each language.

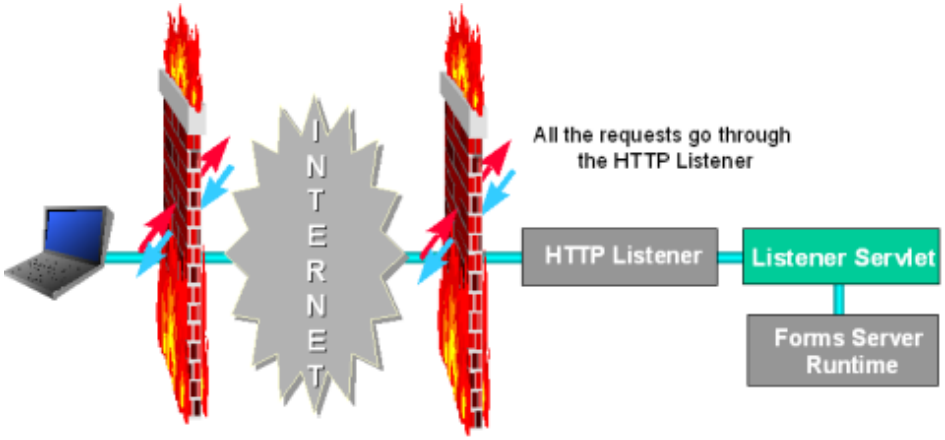

Figure 3: Forms Listener Servlet architecture

The environment settings for the Forms Listener Servlet, and the Forms Web processes, are set in either the HTTP Server configuration files or the new user defined Forms 6i environment files introduced with patch 5. On the Microsoft platform the registry is searched for all variables not set in the configuration files. This is not the case for Unix system. If you are using Forms applications Web deployed on Unix, then environment settings have to be created either in a Forms environment file referenced by the Servlet or in the Apache Jserv.properties configuration file.

For more information about the Forms Listener Servlet architecture, see the Forms Listener Servlet architecture White Paper available at http://otn.oracle.com/products/forms.

The Forms Listener Servlet is automatically configured when Oracle9*i* Application Server Enterprise Edition Release 1.02.*x* is installed.

Section Summary

In this section, you learned that it is possible to migrate existing Forms applications to the Web while leveraging integrated calls to Oracle Reports. The configuration of environment variables depends on the choice of architecture for deploying your Forms applications: Forms Listener or the Forms Listener Servlet. Oracle recommends using the Forms Listener Servlet architecture.

Understanding Reports 6i Services in Oracle 9i Application Server

Oracle9iAS Reports Services is a three-tier Web architecture enabling deployment of Oracle Reports applications in Oracle9*i* Application Server.

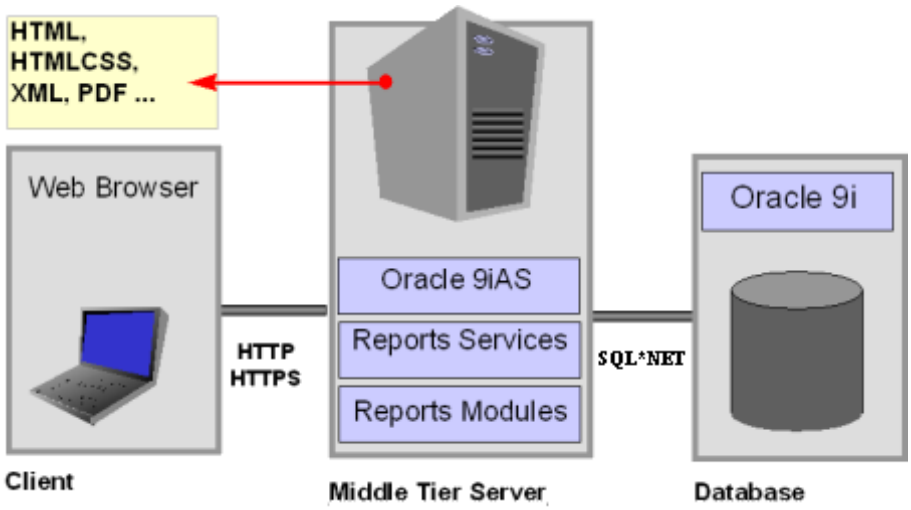

Figure 4: Reports Services Web architecture

Oracle Reports, integrated in Forms Services applications, can also be generated using the Report client runtime environment located on the server. However, the goal of this paper is to encourage customers to use Orarcle9iAS Reports Services instead as this will be supported in the upcoming Oracle9iAS Release 2.0 which includes Oracle9iAS Forms Services.

When a user requests a report's output via a Web URL or from a call issued by an Oracle Forms Services application, this request is passed on to Oracle9*i*AS Reports Services (Reports 6*i* Services). Reports Services is a management instance for multiple reports runtime environments maintaining its own file cache.

Similar to the client-server model, the Reports modules, .rdf or .rep files, are interpreted by the Reports engine producing the formatted output and passing a reference back to the Reports 6*i* Services. The user request gets redirected to the Reports output file on the middle tier server. Output formats can be generated in HTML, HTMLCSS, XML, CSV, and PDF.

When using Reports Services, reports are generated on the server instead of on the user desktop. Thus, all printed output is sent to a network printer configured on the middle-tier server. Oracle provides, although does not officially support, a free utility which enables using local printers with Reports Services calls integrated in Forms. More information about this solution is provided later in this document.

Oracle9iAS Forms Services and Oracle9iAS Reports Services - Installation and Configuration

As part of Oracle9*i* Application Server Enterprise Edition, you can choose to install Oracle9*i*AS Forms Services and Oracle9*i*AS Reports Services. Both services are automatically configured out-of-the-box if you choose to install them. This section presents the settings that are automatically configured for you.

Environment Variables

A Forms application deployed to the Web can use the Oracle Reports client runtime engine to generate reports output, similar to the client-server method. Since the Web does not support preview screens, the reports' output is rendered directly in the client browser in HTML or PDF format or sent to another device such as printers. However, you must first set environment variables including FORMS60_MAPPING, FORMS60_REPFORMAT, and FORMS60_OUTPUT.

If you are using Reports Services, then the Forms application must read the network configuration file, `tnsnames.ora`, which contains connect information for Reports Services. The variables and locations required to integrate reports in Oracle9*i*AS Forms on the Web are explained below.

Net8 Configuration (tnsnames.ora)

If you are working with multiple Oracle Homes and have set a tns_admin entry in the registry or environment, make sure that access to the reports is defined in the shared tnsnames.ora file.

Reports Services are referenced in Forms Services via an entry in the network configuration file, `tnsnames.ora`. The Oracle9*i*AS installation creates a TNS entry named Rep60_<hostname> for the Reports Services. The location of the network file is as follows:

```
<Oracle_9iAS_Reports_Services_Home>\net80\tnsnames.ora
```

Example

Rep60_<host>,Rep60_<host>.world=(ADDRESS=(PROTOCOL=tcp)(HOST=<host or IP address>)(PORT=1949))

Web Server settings

Web server settings are configured in the Oracle HTTP Server *powered by Apache* which is installed as part of Oracle9*i* Application Server. Oracle9*i*AS Forms Services and Oracle9*i*AS Reports Services uses the following configuration files:

- httpd.conf
- jserv.properties
- zone.properties
- 6iServer.conf
- Forms Services .env file

Oracle HTTP Configuration (httpd.conf)

The `httpd.conf` file is where general Web settings such as virtual path mappings, CGI mappings, proxy configurations, listening ports, server names, and MIME Type definitions are configured. In addition, you can link to external configuration files from this file.

The term "Alias" is used to define a virtual directory mapped to a physical directory, while "ScriptAlias" is used to define a CGI directory.

The `httpd.conf` file is located in the following directory:

`<9iAS_Home>\Apache\Apache\conf`

In Forms Services and Reports Services, the following entries are automatically set:

Example

These settings can be included directly in the httpd.conf file. Default Forms and Reports Services environment settings are linked in the httpd.conf file using the 6iServer.conf file located in <Forms_Services_Home>\conf

```
Alias /dev60html/ "D:\Oracle\6iServices\tools\web60\html/"
Alias /jinitiator/ "D:\Oracle\6iServices \JINIT/"
Alias /forms60java/ "D:\Oracle\6iServices \FORMS60\java/"
Alias /dev60temp/ "D:\Oracle\6iServices \tools\web60\temp/"
# cgi settings
ScriptAlias /dev60cgi/ "D:\Oracle\6iServices \tools\web60\cgi/"
```

The Forms and Reports CGI Interface environment variables are set in the httpd.conf file because the HTTP listener is not installed in the same Oracle Home as Forms 6i Services.

```
# Forms and Reports CGI needs to have ORACLE_HOME set
SetEnvIf Request_URI  "rwcgi60"   ORACLE_HOME=D:\Oracle\6iServices
SetEnvIf Request_URI  "rwcgi60"   RW60=D:\Oracle\6iServices\report60
SetEnvIf Request_URI  "ifcgi60"   ORACLE_HOME=D:\Oracle\6iServices
SetEnvIf Request_URI  "ifcgi60"   FORMS60=D:\Oracle\6iServices\forms60

# Forms CGI needs location of formsweb.cfg file
SetEnv   FORMS60_WEB_CONFIG_FILE
D:\Oracle\6iServices\forms60\server\formsweb.cfg
# Mime types for Forms and Reports
AddType video/avi avi
AddType application/x-orarrp  rrpa rrpp rrpt rrpr
```

jserv.properties

The `jserv.properties` file defines environment variables for the Forms Servlet and the Forms Listener Servlet architectures.

To define environment variables, path and classpath settings, `wrapper.env`, `wrapper.path`, and `wrapper.classpath` are used. All variables that are defined in the `jserv.properties` file are set globally for all Forms Servlet instances.

The jserv.properties file is located in the following directory:

`<9iAS_Home>\Apache\Jserv\conf`

Example

Oracle Forms and Reports Servers

wrapper.path=d:\oracle\6iServices\BIN;D:\iSuites

wrapper.classpath=D:\oracle\6iServices\FORMS60\java\f60srv.jar

wrapper.classpath=D:\oracle\6iServices\FORMS60\java\servlet.jar

wrapper.classpath=D:\oracle\6iServices\FORMS60\java

wrapper.env=ORACLE_HOME=D:\oracle\6iServices

wrapper.env=FORMS60=D:\oracle\6iServices\FORMS60

wrapper.env=configFileName=D:\oracle\6iServices\FORMS60\server\formsweb.cfg

(…)

When running a Forms Services application with the Forms Listener Servlet architecture, the environment variables for integrated reports which include FORMS60_REPFORMAT, FORMS60_MAPPING, FORMS60_OUTPUT, RW60, and REPORTS60_PATH, are set in the `jserv.properties` file.

Apache JServ (zone.properties)

The `zone.properties` file defines alias names for the Forms Listener Servlet and sets initial arguments. For example, the `envFile` parameter passed to the Forms Listener Servlet as an initial argument sets the configuration filename for Forms applications run by this architecture.

Oracle recommends that you read the White Paper on the Forms Listener Servlet architecture published at http://otn.oracle.com/products/forms.

The Forms Listener Servlet architecture lets you define different environment settings for the same application. For example, you can set different language settings by defining an alias for the Listener Servlet. For each aliased servlet, the Forms Services environment file overwrites the settings in the `jserv.properties` file.

If integrated reports are executed by the background engine in the Forms Listener Servlet architecture with aliased Servlets, then the Forms Services environment file sets the necessary FORMS60_REPFORMAT, FORMS60_OUTPUT, and FORMS60_MAPPING variables.

The Apache JServ configuration file is located in the following directory:

`<9iAS_Home>\Apache\Jserv\servlets`

Forms Applications Configuration File (6iServer.conf)

The `6iserver.conf` file contains the web-related settings in Forms Services and Reports Services that are linked in the Apache `httpd.conf` file.

The `6iserver.conf` file is located in the following directory:

`<9iAS_Forms_Services_Home>\conf`

Forms Services .env file

Environment files were introduced in Forms 6i patch 5

Although not a Web server configuration file, it is worth to mention the Forms Services configuration file in this context. This file contains a user-defined name with a .env file extension. For the Forms Listener Servlet or one of its defined servlet alias names to use the environment file, the following settings are required in the `zone.properties` file:

#defining an Alias "reptest" for the Servlet

servlet.reptest.code=oracle.forms.servlet.ListenerServlet

setting the environment file to "reptest.env"

servlet.reptest.initArgs=EnvFile=d:\oracle\6iServices\forms60\server\reptest.env

For a successful integrated Report to work, the entries in `reptest.env` should be similar to the following:

FORMS60_PATH=d:\www\forms\fmx

FORMS60_OUTPUT=D:\Oracle\806\tools\web60\temp

FORMS60_MAPPING=/dev60temp/

FORMS60_REPFORMAT=pdf

REPORTS60_PATH=d:\www\reports\rdf

The search order of environment files is `envFile`, `jserv.properties`, and registry settings (on Windows systems only).

Introducing the RUN_REPORT_OBJECT Built-in

The RUN_REPORT_OBJECT built-in was first introduced in Forms Developer Release 2.1. It extends the functionality of RUN_PRODUCT through the addition of Reports Server access and some new procedures to read and write the Reports object properties.

RUN_REPORT_OBJECT was created to be used with Web deployed Forms applications.

In Forms Developer 6*i*, reports are included as Report Objects located below the new Reports node in the Forms Developer object navigator. Each Report Object gets a logical name in addition to the Report's physical filename. The logical name is used to call the Report from PL/SQL.

Example

/* The following example runs a report using the RUN_REPORT_OBJECT built-in. The report object node defined in Forms Developer is named "report_node1". A user-defined Reports parameter "p_deptno" is passed by Forms using the value in the "dept.deptno" field. The parameter form is suppressed. */

```
report_id             Report_Object;
report_job_id         VARCHAR2(100);

BEGIN
report_id:= find_report_object('report_node1');
SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_EXECUTION_MODE,RUNTIME);
SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_COMM_MODE,ASYNCHRONOUS);
SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_DESTYPE,PREVIEW);
SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_SERVER,' ');
SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_OTHER,'p_deptno='||:Dept.De
ptno||' paramform=no');
report_job_id:=run_report_object(report_id);
END;
```

After creating a Report_Object node in Forms Developer, the properties can also be set by editing the Forms object property sheet itself. It is important to have a dummy value set for the "Reports Server" name attribute in the properties, even if the Reports Server is not used.

The RUN_REPORT_OBJECT built-in is well documented in the Oracle Forms Services reference manuals and online help.

Using Parameter Lists in RUN_REPORT_OBJECT

With the RUN_PRODUCT built-in Reports system parameters and user defined parameters are passed in a parameter list. The same parameter lists can be used with RUN_REPORT_OBJECT, except for the system parameters that need to be set by SET_REPORT_OBJECT_PROPERTY ( ). The following is a list of Reports System parameters that need to be set when needed.

List of System Parameters in RUN_REPORT_OBJECT

REPORT_EXECUTION_MODE	BATCH or RUNTIME
REPORT_COMM_MODE	SYNCHRONOUS ASYNCHRONOUS
REPORT_DESTYPE	PREVIEW, FILE, PRINTER, MAIL, CACHE, or SCREEN

REPORT_FILENAME	The report filename
REPORT_DESNAME	The report destination name
REPORT_DESFORMAT	The report destination format
REPORT_SERVER	The report server name

Note that having DESTYPE defined in the parameter list and in SET_REPORT_OBJECT _PROPERTIES does compile but doesn't run

If your existing parameter list already contains definitions for system parameters then it will overwrite the configuration in SET_REPORT_OBJECT_PROPERTY( ). To avoid confusion or unwanted behavior we recommend to modify the parameter list itself, removing the entries for DESNAME and DESTYPE, or add

delete_parameter(<parameter list>,'<name>');

to your code before using SET_REPORT_OBJECT_PROPERTIES( ).

Important: When migrating, Oracle recommends using RUN_REPORT_OBJECT instead of RUN_PRODUCT particularly if you plan on using Oracle9iAS Forms Services. In Oracle9*i*AS Release 2.0, RUN_PRODUCT will not be supported when calling an integrated Oracle Reports in Forms.

Section Summary

When you are planning to use Forms 6i Services for Web deployment, but want to use Forms 9i in the future, Oracle recommends using the Forms Listener Servlet architecture. In this case, the environment settings required to integrate Reports are defined in the jserv.properties file and the environment file, referenced by the Servlet definition in the zone.properties file.

In this section, you learned that the environment settings for Reports integrated in Forms are either configured in the registry (the environment on UNIX systems) or set in the Apache HTTP Server configuration files, `jserv.properties` and `zone.properties`. If running the Oracle9*i*AS installer, default settings are applied. Also, the `jserv.properties` and `zone.properties` are the configuration files used by the Forms Servlet and the Forms Listener Servlet architecture. The `envFile` parameter defined in the `zone.properties` file takes precedence over the `jserv.properties` and the registry settings.

If you use the Forms Listener architecture, then the Forms environment variables are defined solely in the registry or the environment in which the Forms Listener is started (on UNIX Systems).

We also introduced the RUN_REPORT_OBJECT built-in as the recommended call to integrated Oracle Reports in Forms.

SECTION 2: FORMS SERVICES APPLICATIONS WITH INTEGRATED CALLS TO ORACLE REPORTS USING THE REPORTS RUNTIME ENGINE

In a client-server architecture, the Reports runtime engine (also referred to as *background engine*) is a single-user environment that resides on the user's desktop and handles report requests. To start a report from a form, the Forms application calls the Reports runtime engine (rwrun60.exe), which prints report output rendered on the screen, to a file, to a printer, or to the mail system.

Once you move a Forms application to the Web, the Reports runtime engine can reside on the middle-tier which sends reports output to the client's Web browser. However, this is not best practice since the reports runtime engine can service only one user at a time.

Middle-tier Web deployment caters to a multi-user environment. With the Reports runtime engine handling one request at a time, the system is not scaleable enough and performance is not satisfactory to serve a Web architecture. For this reason, Oracle recommends that you limit the use of the runtime engine solution for application migration to the Web.

Common configurations when using the Reports client Runtime Engine

When using the Reports client runtime engine for executing integrated Reports in Forms on the Web, the configuration for both built-ins, RUN_PRODUCT and RUN_REPORT_OBJECT, is the same.

Configuration with Forms Listener

The Forms Listener reads the following Reports settings: REPORTS60, REPORTS60_ PATH, FORMS60_MAPPING, FORMS_60_OUTPUT, FORMS_REP-FORMAT, and RW60 from the registry on Windows systems or the Listener environment on UNIX.

Configuration with Forms Listener Servlet

The Forms Listener Servlet reads the following Reports settings: REPORTS60, REPORTS60_ PATH, FORMS60_MAPPING, FORMS_60_OUTPUT, FORMS_REPFORMAT, and `RW60` from the `jserv.properties` and `zone.properties` files, where the `zone.properties` file references the Forms Services environment files located in the following directory:

```
<9iAS_Forms_Services_Home>\Forms60\Server
```

Using the RUN_PRODUCT Built-in

Note that all registry settings mentioned here have to be set in the jserv.properies or the .env file if using the Forms Listener Servlet architecture.

If you are already using RUN_PRODUCT to call reports from an existing client-server Forms application, you can use Forms Server to run the application on the Web without changing existing application code. Use the Forms 6*i* installer to

install the Reports runtime engine on the middle-tier server rather than on the client. The installer sets the following registry entries to run a report on the Web using the RUN_PRODUCT built-in.

Registry Settings

Under `HKEY_LOCAL_MACHINE\ SOFTWARE\Oracle`

Forms60_output

This variable defines the physical address where temporary Web files are stored. For example:

```
<ORACLE HOME>\tools\Web60\temp
```

Forms60_mapping

This variable defines the virtual mapping of a Web directory that points to the physical address defined in Forms60_output, where temporary reports output files are located. The default value is `/dev60temp/`.

Forms60_repformat

This variable determines the format of the report's output, either HTML or PDF. For HTML output, set the value to `html`. For PDF output, set the value to `pdf`.

In order for RUN_PRODUCT to display the report output in a Web browser, you must set the `Report Communication mode` to `synchronous` and the `destype` to `screen` or `preview`.

Syntax

Below is an example of the syntax for RUN_PRODUCT and a brief description of its associated arguments:

```
Run_Product(Reports,Report_Name,Communication_Mode,Execution_Mo
de, Report_Module_Location, List_id)
```

Reports	Defines the called application to be Reports
Report_Name	Name of the Report definition file (*.rdf or *.rep)
Communication_Mode	Synchronous or asynchronous
Execution_Mode	Allowable values include `Batch` and `Runtime`
Report_Module_Location	Location of the Report module (filesystem)
List_id	Internal ID associated with the Forms parameter list

RUN_PRODUCT is well documented in the Forms Services reference manuals and online help.

Example

```
/* Assumes a parameter list is available that contains all user-defined parameters. */
ADD_PARAMETER(paramlist_id,'DESTYPE',TEXT_PARAMETER,'SCREEN');
ADD_PARAMETER(paramlist_id,'PARAMFORM',TEXT_PARAMETER,'NO');
...
RUN_PRODUCT(REPORTS,'report_name.rdf',synchronous,runtime,
filesystem,paramlist_id);
```

Using the RUN_REPORT_OBJECT Built-in

In client-server computing, the behavior of RUN_REPORT_OBJECT is similar to RUN_PRODUCT. Both built-ins use a client-side runtime engine for report execution. Although RUN_PRODUCT is still available in Forms 6*i*, Oracle recommends that you use RUN_REPORT_OBJECT for new applications, and in some cases re-code existing applications before moving them to the Web. The RUN_REPORT_OBJECT built-in is better at running reports on the Web than RUN_PRODUCT and the former is expected to have a longer lifecycle in Forms.

Set the following properties, using the SET_REPORT_OBJECT_PROPERTY( ) command, to run a report successfully from the Web:

REPORT_EXECUTION_MODE	RUNTIME
REPORT_COMM_MODE	SYNCHRONOUS
REPORT_DESTYPE	SCREEN (or PREVIEW)
REPORT_OTHER	Additional Report parameters such as user-defined variables

RUN_REPORT_OBJECT is called in Forms using the logical name of it's Reports node in the Forms object navigator.

```
report_id:= find_report_object('report_node1');
(…)
report_job_id:=run_report_object(report_id);
```

Passing Forms Parameter Lists With RUN_REPORT_OBJECT

With RUN_PRODUCT in client-server Forms, application parameters and values are passed in parameter lists to integrated Oracle Reports. Although all system parameters in RUN_REPORT_OBJECT must be set using the SET_REPORT_OBJECT_PROPERTY built-in, parameter lists can still be used for all other parameters.

The syntax for using parameter lists in RUN_REPORT_OBJECT is

```
report_job_id:=run_report_object(report_id,paramlist_id);
```

where paramlist_id is the same id used with RUN_PRODUCT.

Please see Appendix B for a coding example.

Printing Reports on the Web

The reports client runtime engine has moved from the client desktop to the server in a Web architecture. The reports-formatted output cannot be sent to a local printer because it is not accessible. Instead, the report can be sent to a network printer addressed with the `desname` parameter in the call to reports.

Section Summary

In this section, you learned how to configure the environment parameters for the Forms Listener or the Forms Listener Servlet to address the Reports client runtime on the Server from a Forms application running on the Web, using either the RUN_PRODUCT built-in or the RUN_REPORT_OBJECT built-in.

SECTION 3: FORMS SERVICES APPLICATIONS WITH INTEGRATED CALLS TO ORACLE REPORTS USING REPORTS SERVICES

Oracle recommends using the Reports Multi-tier Server for integrating calls to Oracle Reports in Forms on the Web. The Oracle Reports Multi-tier Server is a logical unit that manages multiple reports runtime engines in parallel on the middle-tier server. Parallel runtime engines allow many reports to process at the same time. Performance is increased if you install the Reports Server on a different physical machine from the Forms Server (remote server).

The Reports Server can be accessed via an alias added to the `tnsnames.ora` file in the `<Oracle Home> \net80\admin` directory. If no other name is specified during installation, the Oracle9*i*AS Installer defines this entry automatically as "Rep60_ <hostname>".

```
Rep60_<hostname>,Rep60_<hostname>.world =
(ADDRESS =
(PROTOCOL = TCP)
(HOST = <hostname>)
(PORT = 1949)
)
```

To use Reports Server in combination with the Forms RUN_REPORT_OBJECT() built-in, the Reports Server alias should not contain a number or an underscore in its name. Instead, add a *second* alias for the same Reports Server to the `tnsnames.ora`. For example:

```
RepSRV.world =
        (ADDRESS =
                (PROTOCOL = TCP)
                (HOST <hostname>)
                (PORT = 1949)
        )
```

RUN_REPORT_OBJECT with Reports 6i Services

The most secure approach for calling Reports from Forms on the Web is to use the Reports Multi-tier Server in combination with RUN_REPORT_OBJECT. Because the user's database connection is implicitly passed from Forms to Reports on the Server, there is no risk of interception as there would be if it were passed in the URL.

To access a remote Reports Server using RUN_REPORT_OBJECT, the `tnsnames.ora` entry for this server must be recognized by the Report Object. You can do this dynamically, using the Set_Report_Object_Property built-in, or statically, by entering the Reports Server tnsnames string into the property sheet of the Report Object.

Run_Report_Object Example

/* This example uses a synchronous call to RUN_REPORT_OBJECT to run a Report. It expects the name of the Report node, the tnsname.ora entry of the Reports Server, and the desired output format (PDF, HTML, HTMLCSS) to be passed as a parameter. */

```
PROCEDURE      RUN_REPORT_OBJECT_PROC      (vc_reportname      Varchar2,
vc_reportserver varchar2, vc_runformat varchar2) IS


v_report_id           Report_Object;
vc_report_job_id      VARCHAR2(100); /* unique id for each Report
                                        request */
vc_rep_status         VARCHAR2(100); /* status of the Report job */


BEGIN
        /* Get a handle to the Report Object itself. */
        v_report_id:= FIND_REPORT_OBJECT(vc_reportname);
        SET_REPORT_OBJECT_PROPERTY(v_report_id,REPORT_COMM_MODE,
        SYNCHRONOUS);
        SET_REPORT_OBJECT_PROPERTY(v_report_id,REPORT_DESTYPE,CACHE);


/* Define the Report output format and the name of the Reports Server as well as a user-defined parameter, passing the department number from Forms to the Report. We don't need a parameter form to be displayed, and therefore paramform is set to "no". */


SET_REPORT_OBJECT_PROPERTY(v_report_id,REPORT_DESFORMAT, vc_runformat);
        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_report_job_id:=RUN_REPORT_OBJECT(report_id);
```

/* Check the report status if finished. */

```
        vc_rep_status := REPORT_OBJECT_STATUS(vc_report_job_id);
        IF vc_rep_status='FINISHED' THEN
```

/* Call the Report output to be displayed in a separate browser window. The URL for relative addressing is only valid when the Reports Server is on the same host as the Forms Server. For accessing a Remote Reports Server on a different machine, you must use the prefix http://hostname:port/ */

```
        Web.show_document          ('/dev60cgi/rwcgi60.exe/getjobid='||
        vc_report_job_id ||'?server='vc_reportserver,'_blank');

        ELSE
        message ('Report failed with error message '||rep_status);
        END IF;
END;
```

To use the procedure described above, you would pass the following information in the case of a "When-Button-Pressed Trigger."

RUN_REPORT_OBJECT_PROC(<'REPORT_NODE'>,<'REPORT_ SERVER_TNS'>',<'FORMAT'>)

Report_Node	Forms Report node name containing the `rdf` filename for the Report
Report_Server_tns	Name of the Reports Server in the `tnsnames.ora` file
Format	Any of these formats: html|htmlcss|pdf|xml|delimited|rtf

Calling a report synchronously makes the user wait while the report gets processed on the server. For long-running Reports, we recommend that you start the Report asynchronously, by setting the REPORT_COMM_MODE property to `asynchronous` and the REPORT_EXECUTION_ MODE to `batch`.

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

After calling the RUN_REPORT_OBJECT built-in, a timer must be created to run frequent checks on the current REPORT_OBJECT_STATUS in a When-Timer-Expired trigger. The timer should not fire more than four times a minute. After the report is generated, the When-Timer-Expired trigger calls the WEB.SHOW_DOCUMENT built-in to load the Reports output file, identified by a unique job_id, to the client's browser.

The following describes the When-Timer-Expired trigger that checks for the Report_Object_Status.

(...)

```
/* :global.vc_report_job_id needs to be global because the information about the Report job_id is shared between the trigger code that starts the Report and the When-Trigger-Expired trigger that checks the current Report status. */

vc_rep_status:= REPORT_OBJECT_STATUS(:global.vc_report_job_id);

IF vc_rep_status='FINISHED' THEN

        web.show_document
        ('/dev60cgi/rwcgi60.exe/getjobid='||:global.vc_report_job_id

        ||'?server='vc_reportserver,'_blank');

ELSIF vc_rep_status not in ('RUNNING','OPENING_REPORT','ENQUEUED') THEN

        message (vc_rep_status||'  Report output aborted');

END IF;
```

Note: Do not forget to delete the timer when it is no longer needed.

Passing Forms parameter lists in Run_Reports_Object

See Appendix B for a coding example.

Parameter lists used with RUN_PRODUCT in client-server can also be used with RUN_REPORT_OBJECT calling Reports Services to perform integrated Reports in Forms on the Web. Note that system parameters must be set by Set_Report_Object_Property. The syntax for using parameter lists in RUN_REPORT_OBJECT is as follows:

report_job_id:=run_report_object(report_id,paramlist_id);

where paramlist_id is the same id used with RUN_PRODUCT.

Using Run_Reports_Object When Calling a Report Containing a Parameter Form

When you use the RUN_REPORT_OBJECT built-in to create a Report output in Forms, the Reports Server is called directly on the server side, rather than from the Web. The Reports Server is unaware of the Web access path to the machine that

hosts the Reports Server Web interface because this information is not passed with the RUN_REPORT_OBJECT call.

When you use a server side call to run a Report that contains a parameter form, the Reports parameter form in the Web is displayed but is not functional when the user clicks the Submit button. The reason for this can be identified by analyzing the parameter form HTML source code that is generated by Oracle Reports Server.

(…) <form method=post action=""> (…)

Notice that the HTML form "action" tag contains an empty string. For the form to properly generate the final Report output, a valid action entry is needed, for example:

<form method=post action="http://<hostname>/cgi /rwcgi60.exe?">

Similarly, the hidden_run_parameters string value which is used to store parameter values getting passed implicitly with each subsequent Report call, also has an empty value:

```
<input name="hidden_run_parameters" type=hidden
value="">
```

A valid string for the hidden_run_parameters is as follows:

```
<input name="hidden_run_parameters" type=hidden
value="report
%3Dreptest+destype%3Dcache+desformat%3Dhtmlcss+
userid%3Dscott%2      Ftiger%40fnimphiu+server%
3DRepSRV">
```

Once you identify the source of the problem, you can fix it with Forms and Reports Developer. Unfortunately, you cannot resolve this without modifying the Reports module.

Solution

The solution to the above-mentioned issue includes some programming in both Reports and Forms.

In Forms

For a parameter form to work, three pieces of information are required that cannot be retrieved in Reports when called by the RUN_REPORT_OBJECT using Reports Server:

username/password@connect_string

Report Server name

Web access path to the Reports Server used.

Consequently, this information must be passed in addition to all other Reports runtime parameters using the "other" parameter in the RUN_REPORT_OBJECT built-in.

In this example, let's use a procedure in Forms to call RUN_REPORT_OBJECT. The procedure gets a value passed for the report_id, the Reports Server name, and the output format: runformat.

Parameters	Values
report_id	Identifier of the Report Object
reportserver	TNS name of the Reports Server
runformat	html, htmlcss, pdf, xml …

Example

```
PROCEDURE RUN_REPORT_OBJECT_PROC(report_id REPORT_OBJECT, reportserver
varchar2,runformat varchar2) IS

report_job_id          VARCHAR2(100);

rep_status             VARCHAR2(100);

vc_user_name           VARCHAR2(100);    /* used for creating parameter form */

vc_user_password       VARCHAR2(100);    /* used for creating parameter form */

vc_user_connect        VARCHAR2(100);    /* used for creating parameter form */

vc_connect             VARCHAR2(300);    /* used for creating parameter form */


BEGIN

        /* get user connect string */

        vc_user_name:=get_application_property(username);

        vc_user_password:=get_application_property(password);

        vc_user_connect:=get_application_property(connect_string);

        /* creating complete connect string  */

        vc_connect:=vc_user_name||'/'||vc_user_password||'@'
        ||vc_user_connect;

        /*set Reports properties for run_report_object*/
```

```
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,
runformat);
SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_SERVER,
reportserver);

/* P_USER_CONNECT and P_SERVERNAME are custom parameters in the
Reports module */
 P_ACTION is also a user defined variable in Reports and takes the Web access path to the
Report (don't need to be hard coded as in this sample ) */

SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_OTHER,'p_deptno='||
:Dept.Deptno||' paramform=yes P_USER_CONNECT='||vc_connect||'
P_SERVERNAME='||reportserver||'P_ACTION=http://fnimphiu-lap'
||'.de.oracle.com/cgi/rwcgi60.exe?');
report_job_id:=run_report_object(report_id);
rep_status := report_object_status(report_job_id);
IF rep_status='FINISHED' then
        Web.show_document('/cgi/rwcgi60.exe/getjobid='||report_job_id||'?
        server=RepSRV',' _blank');
 ELSE
        message (rep_status||'  Report output aborted');
END IF;

END;
```

In Reports

In Reports, the following user-defined variables must be declared:

P_ACTIONS	Value for the empty “action” parameter
P_USER_CONNECT	username/password@database used as hidden parameter
P_SERVER_NAME	Reports Server Name

All user parameters are of type character. Don't forget to clear the "Restrict list to predetermined values" check box.

P_ACTIONS has an initial value of _action_ . Omitting this breaks the Report in standalone mode when called with a URL

Use the following code in a Report "before form" trigger to substitute the default form values (as presented in the Reports module attributes) with your own string.

Example

```
function BeforePForm return boolean is

vc_parameter_form              varchar2(4000);

vc_hidden_runtime_values       varchar2(1000);

vc_report_name                 varchar2(100);


begin

/* If Reports is called from the URL and not from Forms then p_action is set to it's default value. In this case the     hidden_value has to keep the default value too */

        If (:p_action='_action_') then

                vc_hidden_runtime_values:='_hidden_';

        else

        /* The Report is started from Run_Report_Object and the hidden     parameter has to be set */

        /* 1. get the report module name */

                srw.get_report_name(vc_report_name);

        /* 2. the name needs to be cut off blanks up to the length that it has in characters */

                vc_report_name:=substr(vc_report_name,1,instr

                (vc_report_name,' ')-1);

        /* Note that I'm not using any custom defined parameters except for :p_action,:p_user_connect, :p_servername. If you have additional user defined parameters in your Report output then     this parameter needs to be added to the "vc_hidden_runtime _values" string */

                vc_hidden_runtime_values:='report='||vc_report_name||

                '&destype='||:destype||'&desformat='||:desformat||'

                &userid='||:p_user_connect||'&server='||:p_servername;

        end if;

        /* build the parameter forms HTML code */

        vc_parameter_form:='<html><body bgcolor="#ffffff"><form     method=post
        action="'||:P_ACTION||'">'||'<input     name="hidden_run_parameters"     type=hidden
        value="'||vc_hidden_runtime_values||'">'||'<center><p><table border=0     cellspacing=0
        cellpadding=0><tr><td>'||'<input     type=submit></td><td>     width=15><td><input
        type=reset></td>'     ||'</tr></table><p><hr><p>';
```

```
    /* set the modified before form value, to overwrite the default */

        srw.set_before_form_html (srw.text_escape, vc_parameter_form)

  return (TRUE);

end;
```

The next time you run a Report with an integrated parameter form from Run_Report_Object, the empty values in the HTML source will get replaced. For example:

<form method=post
action="http://fnimphiu-lap.de.oracle.com/cgi/rwcgi60.exe?">

and

<input name="hidden_run_parameters" type=hidden
value="report=Reptest&destype=Cache&desformat=HTMLCSS&
userid=SCOTT/TIGER@fnimphiu&server=RepSRV">

The HTML template code for building the Report HTML form, as used in the above example, can be copied from the Reports "before Form value" property.

Intercepting Reports error messages using RUN_REPORT_OBJECT with Reports Services

Running integrated Oracle Reports in Forms on the Web, using Run_Report _Object with Reports Services, does not return detailed error messages when a report fails. This is because error messages like:

> REP-0736: There exist uncompiled program unit(s).
>
> REP-1247: Report contains uncompiled PL/SQL.

are sent to Reports Services instead of to the client browser or the Forms application. Though Forms on the Web recognizes that an integrated reports failed, the error message displayed is "Frm-41214: unable to run report" which does not specify the exact problem.

The following explains how to retrieve detailed error messages from the server side Reports background engine at the same level as client-server, using the Reports Services Queue table.

After installing Oracle9*i*AS, the SQL file, `rw_server.sql`, used to install the Reports Services Queue table, is located in the <9iAS_Reports_Services_Home>\ Reports60\SQL directory. Using SQL*Plus, install this file into the database schema used by the application or into another schema where your application has read access.

The following example code expects the Queue table to be in the application schema. You have to modify the code if you are using another schema for the table.

Enter the following parameter in the Reports Services configuration file (Rep60_<hostname>.ora) located in the <9iAS_Reports_Services_Home>\Reports60\Server directory :

```
repositoryconn=<schema
name>/<password>@tns_name
```

The username/password pair should match the schema where you installed the Reports Queue table. After restarting the Reports Services, all actions performed by the Reports Server are now logged in the database.

The Queue table is cleared whenever the Reports Server is restarted.

Three columns in the Queue table are read by the following example code, when a Reports call integrated in Forms fails:

- status_code
- status_message
- job_id

The job_id is the unique identifier for each report requested to the Reports Services. The following function uses the job_id as an argument to retrieve error messages from the Queue table.

The previous section explained how to use Reports parameter forms with Run_Reports_Object and the Reports Services. Running an integrated Oracle Reports with a parameter form creates a job_id for both the parameter form and the subsequent Report run. In this case, only the parameter form is subject to error message retrieval.

You don't get a notification even if the subsequent Reports run failed. However, a message appears in the client browser if an error occurs. If your program is dependent on a report's success, then you can check for failure by incrementing the job_id retrieved for the parameter form (see code example below) and querying the Queue table for the status_code column. A successful Reports run inserts a status of '0 ' while a failed Reports run inserts an error number.

The following function checks the Reports Services Queue table for error messages associated with a passed job_id:

```
FUNCTION check_for_errors (unique_identifier varchar2) return varchar2 IS

error_message varchar2 (4000):= ' ';

CURSOR check_for_errors IS select status_code ||' - '||status_message from rw_server_queue where

job_id = to_number(unique_identifier);


BEGIN

        open check_for_errors;
```

```
        fetch check_for_errors into error_message;
        close check_for_errors;
        return error_message;
END;
```

The following sample code is an excerpt of the code sample used throughout this document. When a Reports call succeeds, then the message 'Finished' is returned by the built-in function report_object_status(), indicating a successful Reports run..

```
PROCEDURE RUN_REPORT_OBJECT_PROC(report_id REPORT_OBJECT, reportserver varchar2,
user_interface varchar2,runformat varchar2) IS

    report_job_id          VARCHAR2(100);
    rep_status             VARCHAR2(100);
    error_message          VARCHAR2(4000);
    vjob_id                VARCHAR2(4000);

BEGIN
   (…)
    rep_status := report_object_status(report_job_id);
   (…)
    /* call the Reports output if report run was successful
    IF rep_status='FINISHED' then
        Web.show_document('/cgi/rwcgi60.exe/getjobid='||report_job_id||'?server='
        ||reportserver,'_blank');
    ELSE
        synchronize;

        /* retrieve the Reports Services Job_id */

        vjob_id :=substr(report_job_id,length(reportserver)+2,length(report_job_id));
        /* calling the function to retrieve error message */
        error_message:=check_for_errors(vjob_id);
```

Synchronized is used instead of a timer waiting for the Reports Services to write the error messages to the Queue table. If you don't receive the error message in your code then you may want to check using a timer firing once instead.

The report_job_id variable contains the job_id added to the Reports Server name. For example, the Reports Server name is 'RepServ' and the job_id is '1' then the variable report_job_id has a value of 'RepSRV_1'.

Substr() is used to filter the job_id out of this string.

```
                    /* do something with the error message, e.g. print it to a text field
                    :control.error:=' Error: '||error_message;
        END IF;
        (…)
END;
```

WEB.SHOW_DOCUMENT with Reports 6i Services

Use the WEB.SHOW_DOCUMENT built-in procedure to access any Web site from a forms application on the Web.

SYNTAX

The following table provides the syntax for WEB.SHOW_DOCUMENT and a brief description of its associated arguments:

```
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 addressing could be used (/virtual_path/page.HTML)
DESTINATION	Definition of the target where the addressed Web page should be displayed. Values must be single quoted **_blank** Displays the Web page in a new browser window. **_parent** Displays the Web page in the parent frame of the current page. **<target_name>** Displays the Web page in a frame specified by the target_name.

A Report Server is accessible on the Web through the use of the Reports Server Web interface file, `rwcgi60.exe`, which is located in the `<oracle home>\tools\Web60\cgi` directory. During installation, the Oracle Installer maps this directory to the virtual name dev60cgi, which is also used as a component of the Reports Server URL:

http://<hostname>:<port>/dev60cgi/rwcgi60.exe?server=<reportserver_tns>&report=<report>.rdf&desformat=[htmlcss|pdf|xml|delimited|]&destype=cache&userid=<user/pw@database>&paramform=[no|yes]

The following example calls this Report from Forms on the Web. It assumes that the user parameter "p_deptno" is read from a Forms item "deptno" in the block "dept."

Example 1 - Web.show_document ()

```
/* WHEN-BUTTON-PRESSED */
DECLARE
vc_url  varchar2(100);

BEGIN
       vc_url:='http://<hostname><port>/dev60cgi/rwcgi60.exe?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;
```

Example 2 - Web.show_document () with relative addressing

Use relative addressing if the Reports Server is installed on the same host as the Forms Server.

```
/* WHEN-BUTTON-PRESSED */
DECLARE
vc_url  varchar2(100);

BEGIN
```

```
        vc_url:='/dev60cgi/rwcgi60.exe?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;
```

Example 3 - WEB.SHOW_DOCUMENT () with encrypted URL parameters

Passing the user's name and password to the URL in a human readable format might lead to a loss of security. When you call Reports from Forms, you can use a hexadecimal encryption of the connect string in the URL that is passed by WEB.SHOW_DOCUMENT(). This encryption is useful for hiding the connect string from human readability but it does not provide real protection.

/* This procedure expects the report output format [html, htmlcss, pdf, rtf, xml, delimited] and the name of the Reports definition file. */

```
PROCEDURE    WEB_SHOW_DOCUMENT_PROC    (runformat    varchar2,reportname
varchar2) IS
vc_user_name   VARCHAR2(30) := get_application_property(username);
vc_user_pw     VARCHAR2(30)   := get_application_property(password);
vc_url         VARCHAR2(200);
vc_url_temp    VARCHAR2(300);
v_a            VARCHAR2(10);
v_b            VARCHAR2(10);
i              NUMBER(10);
vc_user_connect
               VARCHAR2(30):=get_application_property(connect_string);
BEGIN
        /* Create the users database connect string. */
        vc_url := 'userid='||vc_user_name||'/'||vc_user_pw||'@'||
```

```
        vc_user_connect;
        /* Convert the connect string into a hexadecimal character string. */
        FOR i IN 1..LENGTH(vc_url) LOOP
        v_a := ltrim(to_char(trunc(ascii(substr(vc_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(vc_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;
        vc_url_temp := vc_url_temp||'%'||v_a||v_b;
        END LOOP;
        /* Create the Reports URL. */
        vc_url:='/dev60cgi/rwcgi60.exe?server=repSRV+report='||
        reportname
        || '+destype=Cache+desformat='||runformat||'+'||vc_url_temp
        ||'+p_deptno='||:dept.deptno;

        /* Call the Report in a new browser window using Web.show_document(). */
        WEB.SHOW_DOCUMENT(vc_url, '_blank');
END;
```

Printing Reports on the Web

The Web is a request model and as such no Reports output is delivered to a printer without a request. The Web is also server centric, which means that reporting occurs on the middle-tier server rather the user desktop.

Printing on Network Printers

Network printers can be accessed by a Reports output when the network printer is configured on the server where Reports services are installed. Use desname= <network printer access name> and destype=printer to send the Reports output to a network printer.

Printing on a Local Printer With the orarrp Utility

orarrp is a free utility available at http://otn.oracle.com/products/reports. Get a White Paper about orarrp in the Reports product area on OTN.

Local printers that are not configured on the middle-tier server can't be accessed by Oracle Reports Services. However, configuring all available printers in an enterprise to one server can be cumbersome. Actually the Web doesn't yet offer a native solution for this so that a workaround, though not officially supported, has been invented by Oracle. To achieve the same printing behavior on the Web like in client-server you can use the Oracle Remote Printing Utility (orarrp) published on the Oracle Technology Network at http://otn.oracle.com.

Orarrp takes the output of a Report run on the middle tier and causes it to be printed "locally"— to a printer you choose. Orarrp can use any printer you can normally use, be it a workgroup printer or a printer directly attached to your machine. The way Orarrp handles printing depends on the output type provided from the middle tier.

If the Report creates Adobe Acrobat (.PDF) output, then Orarrp invokes the print mechanism for Acrobat, which will include all of the print options provided by that application. The same is true if the output is Rich Text Format. In this case, Orarrp uses your word processor print facility, which will include all of the print options it provides. If the middle tier output is plain text, Postscript, or PCL, then you choose the destination printer and Orarrp handles the rest.

A White Paper is available which further explains how it works and how you can set it up. Because of the idea behind orarrp is simple, you can easily build your own version of it tailoring this solution to your requirements. The MIME type used by orarrp is already set up in the Oracle HTTP Server *powered by Apache* after installing Oracle9*i* Application Server.

Section Summary

In this section, you learned to identify the different options of using Reports Services to perform calls to integrated Reports in Forms on the Web.

We demonstrated how to use RUN_REPORT_OBJECT with Reports Services and how to create Reports parameter forms being used with RUN_REPORT_OBJECT calling Reports Services. You also learned to intercept server side Reports runtime error messages on the Web.

You learned an alternative solution to call an Oracle Reports on the Web and how to use of WEB.SHOW_DOCUMENT built-in to access Reports Services.

Also, you learned that printing on the Web doesn't natively access local printers. As a workaround, Oracle provides the Oracle Remote Printing Utility (orarrp). You can also print on a local printer by configuring the local printers as network printers to the middle-tier server where Reports Services are installed.

SUMMARY

When you migrate a client-server Forms Developer application to the Web, it is possible to run an integrated report without rewriting the report execution code. Use the reports client runtime engine installed to the middle-tier server. Set the communication mode to synchronous and the destype parameter to `screen` or `preview`. However, this solution is limited since only one user can run a report at any given time.

When you use the Forms RUN_PRODUCT built-in to call a report in a client-server environment, Oracle recommends that you replace it with the equivalent RUN_REPORT_OBJECT built-in when you move to Web deployment and upgrade to Forms 6*i*.

Forms on the Web is a multi-user environment, and reporting should be multi-user enabled, too. For achieving the best Forms-Reports integration on the Web, Oracle strongly recommends that you use the Reports Multi-tier Server in combination with RUN_REPORT_OBJECT.

APPENDIX A: USABILITY MATRIX

This function matrix compares the options described in this paper for running Reports from Forms on the Web.

Web Deployment using...	Runtime Engine		Reports Services	
Functionality	Run_Product	Run_Report Object	Run_Report Object	Web.Show Document
Multi User enabled			☑	☑
Parallel Reports processing			☑	☑
Scalable			☑	☑
Asynchronous reporting			☑	
Reports parameter form support			☑*	☑
Using Forms parameter lists	☑	☑	☑	
Secure user name/ password	☑	☑	☑	
Report status notification		☑	☑	
User defined parameters	☑	☑	☑	☑
Running Reports on a host other than the Forms Services			☑	☑
Multiple output formats			☑	☑
Recommended			☑	☑

* Requires additional coding as explained in this paper

APPENDIX B: PASSING FORMS PARAMETER LISTS IN RUN_REPORT_OBJECT

The way a parameter is passed to a Report when you use RUN_REPORT_OBJECT has changed. However, it is still possible to use the parameter lists created to run with RUN_PRODUCT in combination with RUN_REPORT_OBJECT.

The following example code includes an example parameter list in a call to RUN_REPORT_OBJECT to submit a Report to the Reports Multi-tier Server.

Example

```
PROCEDURE RUN_REPORT_OBJECT_LIST(report_id REPORT_OBJECT, reportserver
varchar2,
runformat varchar2) IS
report_job_id         VARCHAR2(100);
vc_rep_status         VARCHAR2(20);
paramlist_id          ParamList;
paramlist_name        VARCHAR2(10):='tmplist';

BEGIN
        /* The Reports_Object Properties needs to be set for using the Reports Server .*/
        SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_COMM_MODE,
        SYNCHRONOUS);
        SET_REPORT_OBJECT_PROPERTY(report_id,REPORT_SERVER,
        reportserver);
        /* Check for and delete parameterlist. */
        paramlist_id:= get_parameter_list(paramlist_name);

         IF NOT id_null(paramlist_id) THEN
                destroy_parameter_list(paramlist_id);
         END IF;

        paramlist_id:=create_parameter_list(paramlist_name);

        /* The parameterlist determines the destype, the Reports output format, and the user defined
        variable read from a Forms file's :dept.deptno. */
```

```
        add_parameter(paramlist_id,'DESTYPE',TEXT_PARAMETER,'CACHE');
        add_parameter(paramlist_id,'PARAMFORM',TEXT_PARAMETER,'NO');
        add_parameter(paramlist_id,'p_DEPTNO',TEXT_PARAMETER,
        :DEPT.DEPTNO);
        add_parameter(paramlist_id,'desformat',TEXT_PARAMETER,
        runformat );
        report_job_id:=RUN_REPORT_OBJECT(report_id,paramlist_id);
        END;
```

The procedure is called from a WHEN-BUTTON-Pressed trigger.

```
/* WHEN_BUTTON_PRESSED trigger. */
DECLARE
        report_id Report_Object;
BEGIN

        /*  The Report Object  name is "reptest". */
        report_id:= find_report_object('reptest');
        /* The Reports Server is accessed by the name  "repSRV". */
        PROCEDURE RUN_REPORT_OBJECT_LIST(report_id , 'repSRV' , 'pdf');
END;
```

Important: If your existing parameter list already contains definitions for system parameters then it will overwrite the configuration in SET_REPORT_OBJECT _PROPERTY( ). To avoid confusion or unwanted behavior we recommend to modify the parameter list itself, removing the entries for DESNAME and DESTYPE, or add

Note that having DESTYPE defined in the parameter list and in SET_REPORT_OBJECT _PROPERTIES does compile but doesn't run

delete_parameter(<parameter list>,'<name>');

to your code before using SET_REPORT_OBJECT_PROPERTIES( ).

APPENDIX C: TROUBLESHOOTING

Refer to this appendix if you encounter problems using RUN_REPORT_OBJECT to call the Reports Server.

Problem	Suggested Solution
No access to the report output file using 'getjobid'	1) Have the Forms Server and Reports Server started as a Windows NT services using the same account. 2) In the Reports Server configuration file, set the SECURITY parameter to 0.
The Forms Server could not access the Reports Server	1) If the Forms and Reports Servers are on different machines, check that the full Reports Server `tnsname` entry is the same on both machines. 2) Check that the Reports Server `tnsnames` alias does not contain any underscores or numbers. **Note:** If you installed the default Reports Server, it *does* include underscores and numbers in the name. To fix this, create an additional alias for the Reports Server in the `tnsnames` file that does not contain any of those characters, for example: Rep60_foo.world**, myrepsrv.world**= (ADDRESS= (PROTOCOL=tcp) (HOST=<hostname>) (PORT=1949))

Problem	Suggested Solution
RUN_REPORT_OBJECT and RUN_PRODUCT don't work with the Forms Listener Servlet when using the Reports Client Server engine	Verify that the environment settings for Reports, REPORTS60_PATH, REPORTS60, FORMS60_MAPPING, FORMS60_OUTPUT, and FORMS60_REPFORMAT are set either in the `jserv.properties` or Forms Services environment file.

Integrating Oracle9*i*AS Reports in Oracle9*i*AS Forms
September 2001
Author: Frank Nimphius
Contributing Authors: Susan Leveille

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
www.oracle.com