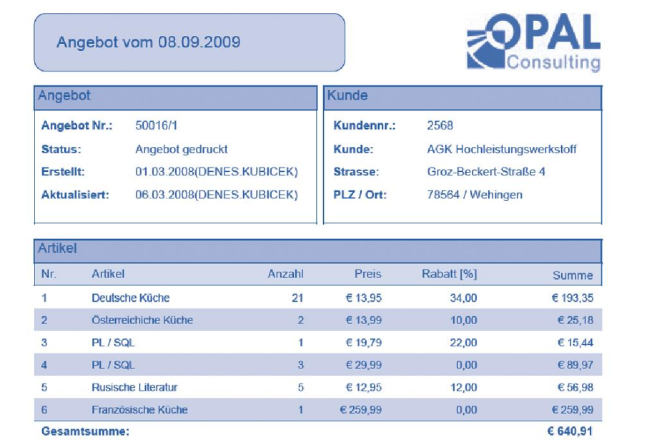
PDF Printing With APEX – A Cost-Free Alternative

By Dietmar Aust, Opal-Consulting

THE PROBLEM

Producing print ready reports (in PDF or MS Word format) is a common requirement in most applications. More often than not you have to deal with somewhat more complex requirements, formatting of data, grouping, calculating totals and subtotals, conditional showing and hiding of information, and integration of images or matrix (pivot) queries. Typical use cases are financial reports, invoices, sales orders, or tax forms. See Figure 1 as an example.



Angebot vom 08.09.2009

Angebot		Kunde	
Angebot Nr.:	50016/1	**Kundennr.:**	2568
Status:	Angebot gedruckt	**Kunde:**	AGK Hochleistungswerkstoff
Erstellt:	01.03.2008(DENES.KUBICEK)	**Strasse:**	Groz-Beckert-Straße 4
Aktualisiert:	06.03.2008(DENES.KUBICEK)	**PLZ / Ort:**	78564 / Wehingen

Artikel

Nr.	Artikel	Anzahl	Preis	Rabatt [%]	Summe
1	Deutsche Küche	21	€ 13,95	34,00	€ 193,35
2	Österreichiche Küche	2	€ 13,99	10,00	€ 25,18
3	PL / SQL	1	€ 19,79	22,00	€ 15,44
4	PL / SQL	3	€ 29,99	0,00	€ 89,97
5	Rusische Literatur	5	€ 12,95	12,00	€ 56,98
6	Französische Küche	1	€ 259,99	0,00	€ 259,99
Gesamtsumme:					**€ 640,91**

Figure 1: Sample report

Print capabilities have been added to Oracle Application Express (APEX) in version 3.0. The basic functionality is provided by the integration with the cost-free Apache FOP library.

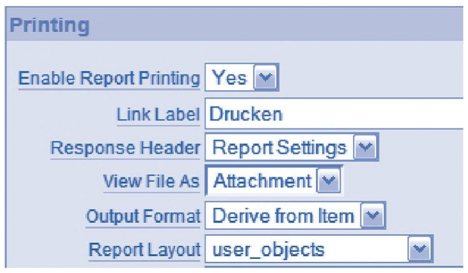
In this simple integration, the report definition has to be specified by writing a XSLT transformation without any GUI support. This is a quite cumbersome and error-prone process. In order to create more advanced and complex reports, you would have to buy a license of Oracle's BI Publisher. Since this is a very expensive option, many Oracle shops are looking for cheaper alternatives for their printing needs.

YOUR OPTIONS

Several different approaches to producing print quality reports with APEX are possible. First we will cover some of the most popular options, and then focus on a very specific integration with the popular Java reporting engine Jasper Reports.

Apache FOP

For the basic APEX printing support, you can use the Apache FOP library (or any other XSL-FO compatible library). The report definition has to be specified as an XSL transformation. The only available output format is PDF.
The basic printing support is really easy to use. You can enable it on any report region declaratively.



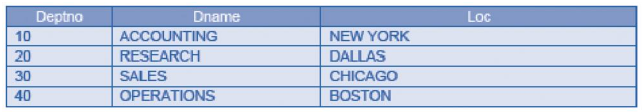
Printing

Enable Report Printing	Yes
Link Label	Drucken
Response Header	Report Settings
View File As	Attachment
Output Format	Derive from Item
Report Layout	user_objects

Figure 2: Enable printing on a report region

You can choose a generic layout or a custom layout (with named columns), and also specify some basic print attributes declaratively like a custom header or footer, the columns to be included, as well as their corresponding widths, fonts, colors, etc.

Using the generic layout, which is easy to use out of the box, you can only create really simplistic reports, see Figure 3.



Deptno	Dname	Loc
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Figure 3: Sample report output

Once your report gets just slightly more complex, you would have to create a custom layout and program it using the XSL syntax without any support through a graphical user interface. This is really hard to do.

Apache Cocoon

Carl Backstrom from the APEX development team published another option based on the Apache Cocoon framework. It does the same as the Apache FOP solution but adds RTF support in addition to PDF:

http://carlback.blogspot.com/2007/03/apex-cocoon-pdf-and-more.html

Robert Stefanov extended Carl's solution even further by adding support for XLS, XML, and HTML as well:

http://rste.blogspot.com/2008/07/excel-export-from-apex-cocoon.html

Oracle BI Publisher

For your more advanced printing needs, Oracle suggests its Oracle BI Publisher.

The most notable distinction is its capability to design the report layouts within MS Word. It ships with plug-ins to integrate APEX session variables and report queries into your MS Word document via drag-&-drop. Once you are done with the report design, you can upload the report definition file to the APEX repository. This uploaded report layout can then be selected when defining the print properties on a report region.

Also, you can call the report via a special URL syntax anywhere in your application.

Code Solutions, e.g. PL/PDF

Another well known strategy for producing PDF reports is to program it with lower level functions, PL/PDF being the most popular option.

This framework is written completely in PL/SQL. It is fast and works really well. Nevertheless I shy away from programming reports manually. The complexity is usually growing pretty fast. You need many lines of code for a typical layout, and you intermingle the business logic with the layout. This quickly becomes hard to maintain.

URL-based Approaches

Also, you can easily integrate any kind of reporting engine   which provides a URL interface for invoking the report.

For example, for this approach you could use Oracle Reports, Crystal Reports, the Eclipse BIRT project, or Jasper Reports. The URL to call the report, including the required parameters, can easily be included or generated on any page in your APEX application. The report will be invoked with a simple hyperlink having syntax similar to this example:

http://<server>/<gateway>?r=<reportName>&p=<parameters>&f=<reportFormat>&ds=<data source or connect information>

Unfortunately, this approach will quickly raise security concerns. When the user communicates directly to the report server, you will have to make sure that the report server does some sort of authentication (who is the user invoking the report) and authorization (is this user allowed to access this report) checking. This would require you to build non-trivial security integration between the APEX application and the report server. Also, you would need to keep the authentication and authorization possibly in two separate locations.

Java / extproc

Another option is to load Java-based reporting engines directly into the database, and provide a PL/SQL-based interface to your applications.

It is also possible to integrate with other reporting libraries on your operating system with the EXTPROC mechanism provided by the Oracle listener.

JASPER REPORTS

JasperReports (http://www.jasperforge.org/jasperreports) is one of the world's most popular open source reporting engines. It is entirely written in Java, and it is able to use data coming from any kind of data source and produce pixel-perfect documents that can be viewed, printed, or exported in a variety of document formats including HTML, PDF, Excel, OpenOffice, and Word.

Ultimately, it is only a Java library that can be embedded into your own Java applications. Thus you can easily embed the library into your SWT or SWING client applications, any applet or J2EE applications, or you call it from the command line directly or via the ANT build tool. It is really flexible to use.

The report definition files are stored in an XML file (e.g. report.jrxml). A shortened sample file will look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport xmlns="http://jasperreports.sourceforge.net/
jasperreports" pageWidth="595" pageHeight="842" colum-
nWidth="555" leftMargin="20" rightMargin="20" topMar-
gin="20" bottomMargin="20">
   <queryString language="SQL">
      <![CDATA[select * from user_objects]]>
   </queryString>
   <field name="OBJECT_NAME" class="java.lang.String">
      <fieldDescription><![CDATA[]]></fieldDescription>
   </field>
   <field name="OBJECT_TYPE" class="java.lang.String">
     <fieldDescription><![CDATA[]]></fieldDescription>
   </field>
   <title>
    <band height="79" splitType="Stretch">
      <staticText>
        <reportElement x="216" y="29" width="100"
height="20"/>
       <textElement/>
         <text><![CDATA[Der erste Bericht]]></text>
      </staticText>
    </band>
   </title>
   <detail>
    <band height="20" splitType="Stretch">
      <textField>
        <reportElement x="0" y="0" width="100"
height="20"/>
       <textElement/>
         <textFieldExpression class="java.lang.
String"><![CDATA[$F{OBJECT_NAME}]]></textFieldExpres-
sion>
        </textField>
    </band>
   </detail>
</jasperReport>
```

Once the report definition is specified in this XML file, it needs to be compiled into a binary file (e.g. report.jasper) in order to be run by the reporting engine.

When you download JasperReports, you can also download lots of samples of how to use the different features and the different integration techniques to embed the library into your own Java applications.

JasperReports supports many different data sources, e.g. SQL over JDBC, XML files, CSV files, and Java Beans, and can be extended to implement your specific needs, too.

With one single report definition, you can export the report into multiple formats, e.g. PDF, RTF, or DOCX (MS Word), XLS (MS Excel), ODF (OpenOffice), or plain text.
JasperReports supports many concepts that are found in professional reporting environments:

- Report parameters
- Multiple data sources per report
- Local variables, calculations (sum, min, max, …)
- Matrix reports
- Subreports / modularization
- Conditional formatting (styles)
- Conditional rendering (expressions in Java, Groovy, JavaScript)
- Fonts, Unicode support
- Sorting, filtering, and grouping
- Diagrams (integration with JFreeCharts)
- National Language Support
- Report triggers

iREPORT

iReport *(http://www.jasperforge.org/ireport)* is another open source tool to design even complex report layouts for JasperReports graphically in a WYSIWYG environment. It basically creates the XML definition file for you.

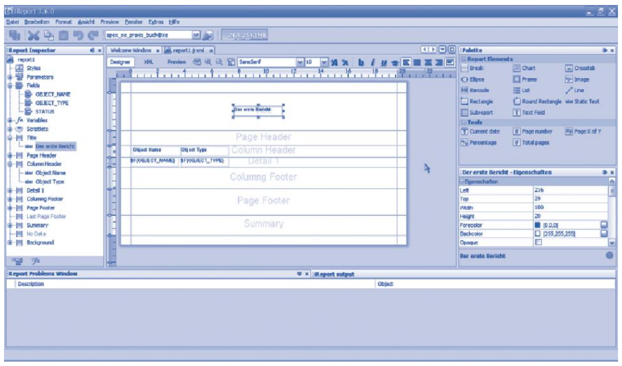


Figure 4: iReport (Version 3.6.0)

It provides an integrated support for different query languages like SQL, HQL, xPatch, EJBQL, and MDX. Furthermore, user-defined extensions are supported so that even PL/SQL can be used as a data source.

iReport offers a full IDE (integrated development environment) to visually design reports, connect live to the different data sources, and display a report preview in the different output formats like PDF, HTML, XML, RTF, or Excel.
It is a modern GUI and provides many features to manipulate the layout (alignment, sizing) of individual elements or of a group of elements.

You can download the iReport application here:
http://www.jasperforge.org/ireport.

INTEGRATION AND ARCHITECTURE

How can we integrate these reports into our APEX applications?

The most notable distinction to the default integration with Apache FOP or the BI Publisher is the configuration in the J2EE Server (e.g. Apache Tomcat). In the default integration you have a J2EE application running in the JEE container which accepts both the report definition and the data (in XML) in order to produce the desired output and return the result to the database and back to the client. No additional setup or configuration is needed in the J2EE server.

In our use case, we first create a complete report including all SQL Statements with iReport (e.g. *report.jrxml*). Then we compile the file to a binary (e.g. *report.jasper*).

This file is then copied to the J2EE server where it can be accessed by the J2EE application for the integration. This application is called JasperReportsIntegration, and you can download it from *http://www.opal-consulting.de/tools*.

This application provides a URL-based interface to invoke the report against a preconfigured data source (e.g. the Oracle user HR in the local XE instance). It connects via JDBC to the database (the connection uses the internal Tomcat connection pool), generates the report in the desired output format, and passes the result back to the database.

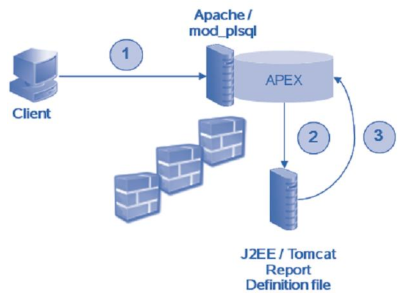


Figure 5: Architecture of the integration

Thus we have the following flow of events:

1. First the user clicks on a link or a button to generate the PDF report.
2. Within your APEX application you know the user (since he is authenticated to the application), and you can validate whether he is authorized to start the report with the supplied parameters.
   Subsequently, the URL for the *JasperReportsIntegration* application is constructed by the PL/SQL package XLIB_JASPERREPORTS which defines a simple interface to call the reports which are stored on the server. Via UTL_HTTP the report is invoked, and the result is returned to the user.
3. Within the J2EE application *JasperReportsIntegration*, a new database session is generated from a JNDI data source which is configured in the J2EE server and optimized for performance. Usually it is taken from the internal connection pool.
   Then the report is run by calling the embedded Jasper Reports library internally, and returning the result back to the database and then to the client.

Using the database as a proxy in our architecture has some advantages:

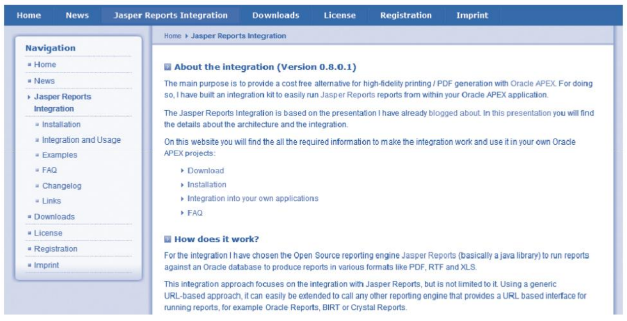
- Only a single tcp/ip port is needed for your APEX application and the reporting server.

- The authorization checking has not been duplicated, it is sufficient to only check it within your APEX application.
- If the local J2EE server is only accessible by the database, and all external access is denied, then no additional setup for SSL or other security measures are needed. You can use a firewall to protect your J2EE server.
- The complexity of the overall solution is reduced to a single URL-based interface.

DOWNLOAD AND INSTALLATION

The package can be downloaded here: *http://www.opal-consulting.de/tools/jasper_integration.*

Follow the instructions on the site to install the integration toolkit.



About the integration (Version 0.8.0.1)

The main purpose is to provide a cost free alternative for high-fidelity printing / PDF generation with Oracle APEX. For doing so, I have built an integration kit to easily run Jasper Reports reports from within your Oracle APEX application.

The Jasper Reports Integration is based on the presentation I have already blogged about. In this presentation you will find the details about the architecture and the integration.

On this website you will find the all the required information to make the integration work and use it in your own Oracle APEX projects:

- Download
- Installation
- Integration into your own applications
- FAQ

How does it work?

For the integration I have chosen the Open Source reporting engine Jasper Reports (basically a java library) to run reports against an Oracle database to produce reports in various formats like PDF, RTF and XLS.

This integration approach focuses on the integration with Jasper Reports, but is not limited to it. Using a generic URL-based approach, it can easily be extended to call any other reporting engine that provides a URL based interface for running reports, for example Oracle Reports, BIRT or Crystal Reports.

Figure 6: Download of the Package

There you will also find additional examples on how to use the integration.

THE FIRST REPORT

Let's start with our first report. We will go through a complete example from start to finish, and even integrate the report into the default sample application that is shipped with APEX. The sample application can be installed in any APEX workspace through the wizards, and is already preinstalled in most workspaces (unless you manually suppress it).

Creating the Report

First we start the iReport designer to create the first report. In this example, I will refer to version 3.6.0.

Configure the Data Source

In order to design a report, we need to configure a data source to base the report on.

In our example, we will connect as the Oracle user HR to our local XE instance.

First we click on the symbol in Figure 7 to manage the data sources.

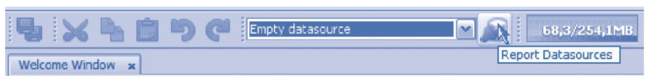
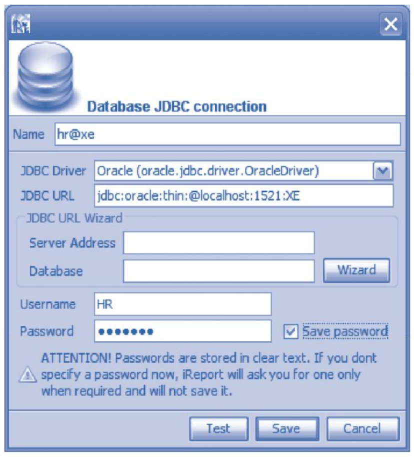


Figure 7: Manage the data sources.

Then we click on New to create a new data source. We choose JDBC as the data source type.

Then we enter the connect information in the appropriate fields, in our demo the Oracle user HR in our local XE instance.

```
NAME: hr@xe
JDBC DRIVER: Oracle (oracle.jdbc.driver.OracleDriver)
JDBC URL: jdbc:oracle:thin:@localhost:1521:XE
USERNAME: hr
PASSWORD: <password for hr>
```

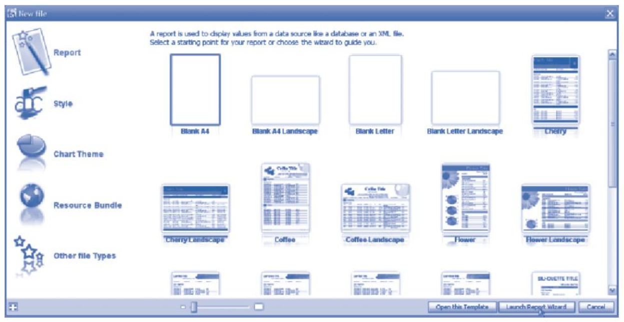


Name: hr@xe
JDBC Driver: Oracle (oracle.jdbc.driver.OracleDriver)
JDBC URL: jdbc:oracle:thin:@localhost:1521:XE
Username: HR
Password: •••••••
[✓ Save password]
ATTENTION! Passwords are stored in clear text. If you dont specify a password now, iReport will ask you for one only when required and will not save it.

Figure 8: Configure the data source

Create the Report

Once the data source is configured, we can start by creating the first report. After invoking File > New you will see the dialog to create a new report. There you can choose to use a predefined report template; you can even create your own templates for your company. In this demo, we start with the blank one by clicking on *"Launch Report Wizard."*



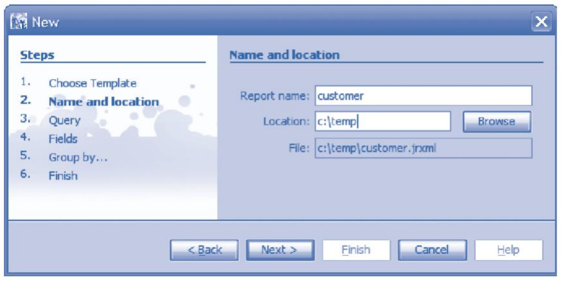
A report is used to display values from a data source like a database or an XML file.
Select a starting point for your report or choose the wizard to guide you.

Figure 9: Choose a report template

After that we specify the name and location. The report name will be *customer* so that the resulting file will be named *customer.jrxml.* (See Figure 10, next page.)

Next we choose the newly created data source *hr@xe* and enter the sql query: *select * from demo_customers.* (See Figure 11, next page.)

Finally, we select the columns we want to use in the report, and accept the default values on the remaining dialogues.



New

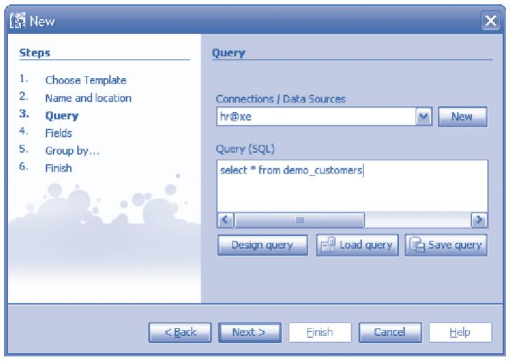
Steps

1. Choose Template
2. **Name and location**
3. Query
4. Fields
5. Group by…
6. Finish

Name and location

Report name: customer
Location: c:\temp
File: c:\temp\customer.jrxml

Figure 10: Specify name and location



New

Steps

1. Choose Template
2. Name and location
3. **Query**
4. Fields
5. Group by…
6. Finish

Query

Connections / Data Sources: hr@xe
Query (SQL): select * from demo_customers

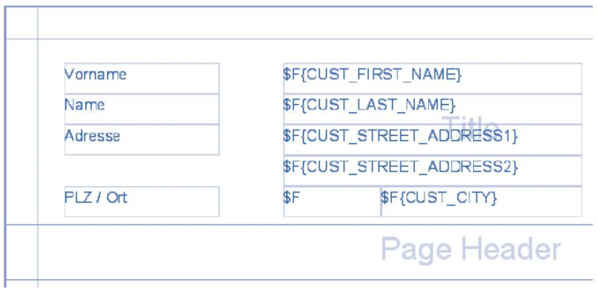
Figure 11: Specify the report query

PLACING THE FIELDS ON THE REPORT

Next we want to place the report fields on the report itself. In our demo, we want to show the customer details for a selected customer.

On the left side of the IDE we see the pane called *Report Inspector*. There we open the node called *Fields,* and then we drag the individual elements onto the report where we place them in the *Title* band; it is the topmost report band.

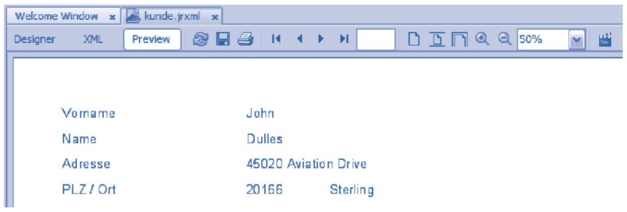
Then we create labels for each report field. On the right side in the IDE, you can see the pane *Palette*. We click on the element *Static Text* and drag it onto the report, too. Once we are done the report should look like Figure 12.



Vorname	$F{CUST_FIRST_NAME}	
Name	$F{CUST_LAST_NAME}	
Adresse	$F{CUST_STREET_ADDRESS1}	
	$F{CUST_STREET_ADDRESS2}	
PLZ / Ort	$F	$F{CUST_CITY}

Figure 12: Report design

We can see a preview of the report when we click on the *Preview* button.



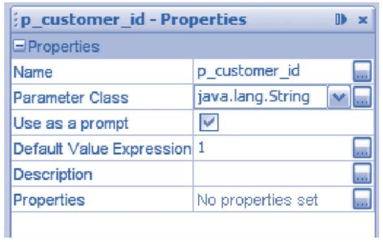
Vorname	John	
Name	Dulles	
Adresse	45020 Aviation Drive	
PLZ / Ort	20166	Sterling

Figure 13: Report preview

Filter for a Specific Customer

We don't want to display *ALL* customers; therefore we need to add a filter to the report. First we create a parameter called *p_customer_id* by right-clicking on the *Parameters* node in the *Report Inspector* pane.

We have to specify all parameters as java.lang.String due to the implementation of the J2EE application. We cannot use any other data type.



p_customer_id - Properties

Properties

Name	p_customer_id
Parameter Class	java.lang.String
Use as a prompt	✓
Default Value Expression	1
Description	
Properties	No properties set

Figure 14: Parameter p_customer_id

Next we modify the sql query to incorporate the parameter as a filter.

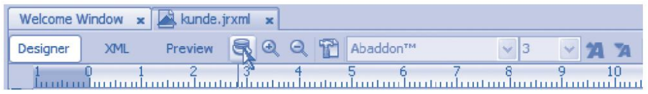


Figure 15: Modifying the report query

The modified query is:

```
select * from demo_customers where customer_id = $P{p_customer_id}
```

Deploying the Report

Now we have created our first report called *customer.jrxml*. This is the report definition in XML format. Next we need to compile the report to a binary by clicking on Compile Report as seen in Figure 16.

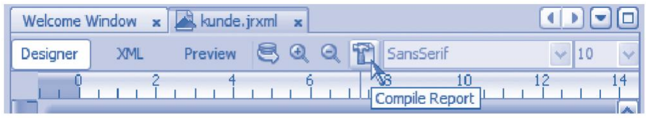


Figure 16: Compiling the report

This will create the file *customer.jasper* in the same directory as the file *customer.jrxml*.

This file *customer.jasper* needs to be copied into the directory *%CATALINA_HOME%\webapps\JasperReportsIntegration\reports*. There you will also find the file test.jasper.

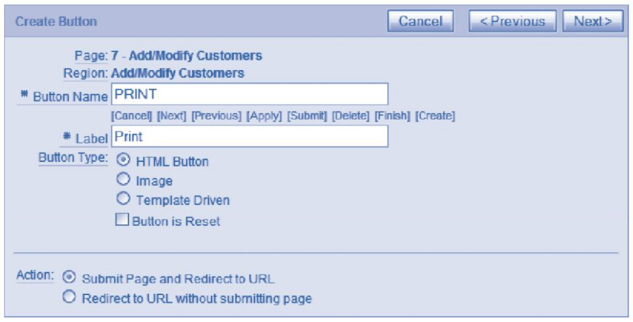
On the Web page *http://127.0.0.1:8080/JasperReports-Integration/* you can now test the report by entering customer as the value for the parameter *_repName*. The resulting URL will be similar to:

http://127.0.0.1:8080/JasperReportsIntegration/report?_repName=customer&_repFormat=pdf&_dataSource=default&_outFilename=&_repLocale=&_repEncoding=

Integration into Your APEX Application

In order to demonstrate the integration, we extend the demo application which is installed into all APEX workspaces by default.

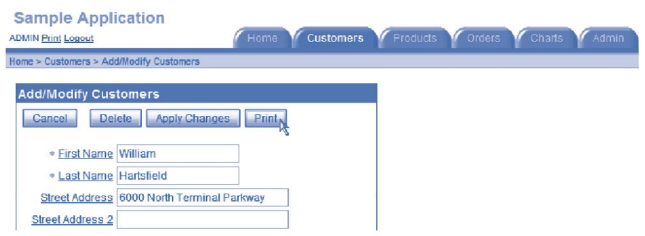
In the next section we create a new button called *PRINT* in the region *Add/Modify* Customers as seen in Figure 17.



Page: 7 - Add/Modify Customers
Region: Add/Modify Customers
* Button Name: PRINT
[Cancel] [Next] [Previous] [Apply] [Submit] [Delete] [Finish] [Create]
* Label: Print
Button Type: [✓ HTML Button] Image / Template Driven / Button is Reset
Action: [✓ Submit Page and Redirect to URL] / Redirect to URL without submitting page

Figure 17: Create a print button

We create a condition *Value of item in Expression 1 is not null* with the value *P7_CUSTOMER_ID* in *Expression 1*.



* First Name: William
* Last Name: Hartsfield
Street Address: 6000 North Terminal Parkway
Street Address 2:

Figure 18: Final button layout

Next we create a process to show the report. We need a page process of type *On Submit – After Computations and Validations*.

This page process should be linked to the button PRINT, thus we need to specify the *When Button Pressed* attribute on the process accordingly.

In order to show the PDF report in the current window, we use the following PL/SQL source code:

```
BEGIN
  xlib_jasperreports.set_report_url(
    p_report_url =>   G_REPORT_URL);

  xlib_jasperreports.show_report (
    p_rep_name     => 'customer',
    p_rep_format   => 'pdf',
    p_data_source  => 'default',
    p_out_filename => null,
    p_additional_params => 'p_customer_id='
                      || :p7_customer_id);

  -- stop rendering of the current APEX page
  apex_application.g_unrecoverable_error := TRUE;
END;
```

This approach implies that the variable *G_REPORT_URL* is set to the value *http://127.0.0.1:8080/JasperReportsIntegration/report* either in the global application attributes (see section *Substitutions* on the page *Shared Components > Defintion*) or in an application item. When using an application item please make sure to to set the attribute *Session State Protection to Restricted – May not be set from browser*. You can also store the value in a local configuration table in your schema; this is up to you.

By specifying a value for the parameter *p_out_filename* the browser's default save dialogue is invoked. The report is no longer displayed online on the current page, but the user can save or open the report with an external program.
Our PL/SQL block would look like this:

```
BEGIN
  xlib_jasperreports.set_report_url(
    p_report_url =>   G_REPORT_URL);

  xlib_jasperreports.show_report (
    p_rep_name     => 'customer',
    p_rep_format   => 'pdf',
    p_data_source  => 'default',
    p_out_filename => 'customer'
                   || :p7_customer_id
                   || '.pdf',
    p_additional_params => 'p_customer_id='
                      || :p7_customer_id);

  -- stop rendering of the current APEX page
  apex_application.g_unrecoverable_error := TRUE;
END;
```

Additional Examples on the Web site

You can find more examples on the Web site *http://www.opal-consulting.de/tools/jasper_integration*, especially how to store a generated report as a blob in a table, or how to send the report as an e-mail attachment using apex_mail.

CONCLUSION

There are many different approaches to produce print ready reports in APEX in addition to the out-of-the-box integrations—Apache FOP and Oracle BI Publisher. Basically we can integrate any kind of reporting engine that provides a URL-based interface to invoke the reports. By using the database as a proxy (via UTL_HTTP), we can easily create a secure solution (authentication and authorization) without much hassle.

Focusing on JasperReports, one of the world's most popular open source Java reporting engines, we have a really cheap and powerful alternative to satisfy even complex reporting needs.

For designing your reports, the iReport designer is a perfect fit. Both tools are open source.

In this article, we have shown a complete example from start to finish on how to design a new report, deploy it on the application server, and integrate it into the application. You can download the integration here: *http://www.opal-consulting.de/tools/jasper_integration.*

About The Author

Dietmar Aust is working as a freelance consultant in Germany, focusing on Oracle Application Express and Oracle XE. Starting in 1997, he worked for three years as a consultant for Oracle in Germany. Since then, he helped numerous leading companies in Germany to successfully deliver Web-based applications based on the Oracle product stack, especially the Internet application server - Oracle Portal and Reports. He is a regular presenter at various Oracle conferences (ODTUG, OOW, DOAG), conducts training classes on APEX, and recently co-authored a book on APEX best practices in German ("Oracle APEX und Oracle XE in der Praxis"). You can reach him at http://www.opal-consulting, http://daust.blogspot.com, or via e-mail at dietmar.aust@opal-consulting.de.

ODTUG KALEIDOSCOPE VOLUNTEERS GO BACK TO SCHOOL

Be a part of an ODTUG tradition of giving back to the community that hosts the ODTUG Kaleidoscope Conference. In 2008, we painted an elementary school in New Orleans. Last year we rehabilitated preserved coastal habitat in the Monterey, California area. This year in Washington, D.C., (Saturday, June 26), we are going back to school.

Under the guidance of Greater DC Cares (GDCC), ODTUGgers will give five hours of their day to beautify a school in Washington, D.C. When you register for ODTUG Kaleidoscope 2010, be sure to indicate that you are interested in participating in the Community Service Day.

ODTUG COMMUNITY SERVICE DAY SPONSORSHIP

Greater DC Cares depends on generous individuals and community-minded companies to fund its programs, including the ODTUG Kaleidoscope 2010 Community Service Day. Program leaders are needed to organize the project; food is needed to keep our volunteers energized; paint and other supplies must be purchased; and transportation costs need to be covered.

Companies or individuals donating more than $100 will have their names placed on the back of the official ODTUG Community Service T-Shirt.

How do I donate?

To become an ODTUG Community Service Sponsor, please send a check (memo line of your check should read - June 26 ODTUG Event) made out to Greater DC Cares to:

Greater DC Cares c/o Gregory Hill
1156 15th Street, NW Suite 840, Washington, DC 20005

Please know that Greater DC Cares and ODTUG value your financial support, as well as the donations of your time, energy, skills, and enthusiasm.

You are truly helping to impact our hosting community - Washington, D.C.!