



Le **BIRT** Expert's Guide to ...

BIRT Best Practices

How to Develop a Reporting Framework,
Implement Best Practices and
Manage Your BIRT Reporting Project

By David Mehi

Copyright Notice

Author: David Mehi

Title: Le BIRT Expert's Guide to BIRT Best Practices

© Copyright 2010 by David Mehi

Self publishing

Detroit, MI, USA

Version: Final 1.0

Le BIRT Expert Name and Logo are trademarks of David Mehi and InfoLight Solutions LLC

www.lebirtexpert.com

www.infolightsolutions.com

www.davidmehi.com

ALL RIGHTS RESERVED. This book contains material protected under International and Federal Copyright Laws and Treaties. Any unauthorized reprint or use of this material is prohibited. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system without express written permission from the author / publisher. For permission, contact sales@lebirtexpert.com.

All product and company names are trademarks or registered trademarks of their respective holders. The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by the author. The author assumes no responsibility or liability for any errors or inaccuracies that may appear in this guide.

Book Contents

Chapter 1 Introduction	15
1.1 Who this Book is for.....	16
1.2 What This Book is Not.....	16
1.3 Assumed Prior Knowledge	16
1.4 Version of BIRT	16
1.5 About the Author.....	17
1.6 Contents of the Book	17
1.7 Additional Resources	19
Book Examples	19
Base Reporting Framework.....	19
Classic Models Example Project	20
1.8 Continuing the discussion	21
Chapter 2 Plan Ahead.....	23
2.1 Questions to Ask.....	23
2.2 Design Decisions	25
Part I Developing a Reporting Framework	27
Chapter 3 Preparing the Framework	29
3.1 Create a Folder Structure.....	29
3.1.1 Alternative Folder Structure for Use in Multiple Projects	32
3.2 Creating Report Libraries	33
3.3 Creating a Java Project.....	34
3.3.1 Create the Java Project.....	35
3.3.2 Referencing the Java Project	37
3.4 Common Naming Conventions	37
3.4.1 Naming Report Items	40

Chapter 4 Create Re-Usable Report Themes and Styles	41
4.1 What is a Theme?	41
4.2 Create a New Theme	42
4.3 Predefined Styles.....	44
4.3.1 Adding predefined styles to the Theme	44
4.3.2 Testing the Theme	47
4.3.3 Link the reportThemes.rptlibrary to the report design.....	48
4.3.4 Using the Theme in the Report Design	49
4.3.5 Switching Themes in a report	51
4.4 Using Custom Styles in Your Theme	54
4.4.1 Creating custom styles in your theme.....	54
4.4.2 Using custom styles in your report	58
4.4.3 Switching Themes using Custom Styles.....	59
4.5 Importing a CSS file into your theme	62
4.6 Linking to an external CSS file	63
4.6.1 A warning about using external CSS files	65
4.6.2 Why does BIRT handle Styles this way?	66
4.6.3 Why does BIRT internalize the Styles, even when the CSS is external?.....	66
4.6.4 Does BIRT maintain the names of the original style names in the generated HTML?	66
4.6.5 What happens if we enable BIRT to include the CSS file at view time?.....	68
4.7 How to Proceed with Styles?	68
4.8 Creating styles directly in your report design	69
4.9 Le BIRT Expert's Recommended Approach.....	70
Chapter 5 Creating and Using Report Items for Common Use in Libraries	71
5.1 Adding a report item to a library.....	71
5.1.1 Modifying an Item From a Library.....	74
5.2 Establishing your initial Common Report Items Library – What every library should start with	74
5.2.1 Setting the Library to use the Styles Library.....	75
5.2.2 Setting the Default Theme	75
5.2.3 Common Master Pages	75

5.2.4 Common Report Header	76
5.2.5 Common Report Footer	79
5.3 Limitations of BIRT Libraries	81
5.3.1 Designer Quirks when using Libraries	81
5.3.2 Unable to re-use report items in the same library	81
5.3.3 Can inherit but not extend	83
5.3.4 Can change the grid properties	85
5.3.5 Can change the label properties	85
5.3.6 Can only inherit once	86
5.4 Recommendations when using BIRT Libraries	86
Chapter 6 Establishing a Secure and Consistent Way to Access Data.....	87
6.1 Create the Data Source	88
6.2 Flat File Data Source	89
6.3 Web Services Data Source	92
6.3.1 Custom Driver Class	95
6.3.2 Recommended Approach.....	96
6.4 XML Data Source	97
6.4.1 Creating an XML Data Source	97
6.4.2 Recommendation.....	99
6.5 JDBC Data Source	100
6.5.1 JDBC Drivers.....	100
6.5.2 Alternative Method to register JDBC driver with BIRT Designer	103
6.5.3 SQL vs Stored Procedures.....	104
6.5.4 Stored Procedures.....	105
6.5.4.1 Stored Procedure Options	109
6.5.5 Testing Stored Procedures	110
6.5.6 Using Stored Functions.....	110
6.5.7 Using SQL.....	110
6.5.8 Designing the Query.....	111
6.5.9 Using DTP to design your Query	112
6.6 Dataset Best Practices	114

6.6.1 Computed Columns.....	114
6.6.2 Parameters	115
6.6.3 Linked Parameters Versus Default Value	115
6.6.4 Filters.....	116
6.6.5 Property Binding	117
6.6.6 Settings.....	118
6.7 Tips and Tricks	118
6.7.1 Don't Use Database Schema Name or Namespace	118
6.7.2 Using Views.....	119
6.7.3 Changing SQL at Runtime	119
6.7.4 Dynamic SQL Cautions.....	121
6.7.5 Using dynamic SQL to Secure Your Queries	122
6.7.6 Dynamic Filter and Sorting Example	122
6.7.7 Passing in a delimited list of values for filtering, using IN or NOT IN.....	123
6.7.8 Ad-hoc queries.....	126
6.8 Connection Pools.....	126
6.9 Scripted Data Source	128
6.9.1 Using JavaScript	128
6.9.1.1 Use Java Objects.....	129
6.9.1.2 Create a New Scripted Data Source	132
6.9.1.3 Scripted Data Source Example	133
6.9.1.4 Create a Scripted Data Set	134
6.9.1.5 Scripted Data Set Example	135
6.10 Using Java Event Handlers	137
6.10.1 Implementing the Scripted Data Set in Java.....	137
6.10.2 Referencing the Java Event Handler in the Report Design	139
6.10.3 Testing the Report.....	140
6.11 Using POJOs, Spring, Hibernate and other Java Frameworks with BIRT	140
6.12 Other forms of Open Data Access – ODA	141
6.13 Joint Data Sets	143
6.14 Other Data Access Using Actuate BIRT	145
6.15 Using Data Cubes.....	146

6.15.1 Creating a Data Cube.....	146
6.15.2 Formatting Cross Tab Items.....	149
6.15.3 Using Charts in the Cross Tab	151
6.15.4 Cross Tab Scripting	152
6.15.5 Additional Cross Tab Resources.....	152
6.16 So what is the best way?.....	152
6.17 Best Practice - Putting data sources and data sets into libraries.....	153
6.18 Best Practice - Externalizing connection properties.....	157
6.18.1 Externalize the Data Source in the Report Library – Best Practice	157
6.18.2 Using BIRT's Default Option.....	158
6.18.3 Using a JDBC Connection Pool	160
6.18.4 Use JNDI.....	160
6.18.5 Pass Data Source Connection Properties Through the Application Context....	161
6.18.6 Use a Scripted Data Source.....	161
6.18.7 Using a Connection Profile	161
6.18.7.1 Database Connection Profile	162
6.18.7.2 ODA Data Source Profile	166
6.18.8 Using a Custom Java Class Called by Scripting.....	172
6.18.9 Summary.....	175
6.19 Externalizing Connection Properties with Commercial BIRT and iServer	175
Chapter 7 Report Parameters.....	177
7.1 Re-Usable Report Parameters.....	177
7.1.1 Creating a Re-Usable Report Parameter	177
7.1.2 Using a Re-Usable Report Parameter.....	179
7.2 Linking a Report Parameter to a Data Set	179
7.3 Parameter Grouping	181
7.4 Cascading Parameters	182
7.5 Hidden Parameters	183
7.6 Parameter Validation.....	183
7.7 Should You Use the Default BIRT Parameter Page or Create Your Own?	185
7.8 Parameter validation considerations	187

7.9 Parameter Bad Practice	187
Chapter 8 Scripting using JavaScript.....	189
8.1 Using the Script Palette	190
8.2 Advantages and Disadvantages of JavaScript	190
8.3 Scripting Guidelines	191
8.4 Calling external Java Objects.....	192
8.4.1 Using the Java String Object	192
8.5 Creating and leveraging re-usable scripting libraries.....	192
8.5.1 Creating a Scripting Library.....	192
8.5.2 Using a Scripting library.....	195
8.6 Runtime Manipulation of the Report Design.....	197
8.7 Don't want to use JavaScript? A Simple Alternative	197
8.7.1 Creating a Report Variable	198
8.7.2 Using a Report Variable - Example.....	199
Chapter 9 Create and Leverage Re-usable Java Components.....	201
9.1 Using Custom Java Classes.....	201
9.2 Java Event Handlers.....	204
9.3 BIRT Events.....	205
9.4 Creating a BIRT Event Handler Java Project.....	205
9.4.1 JAR Files Needed.....	206
9.4.2 Creating Base Classes	208
9.4.3 Referencing Java Project in BIRT Project.....	211
9.5 Using Java Event Handlers – Best Practices.....	212
9.5.1 Logging	212
9.5.2 Class and Event Implementation	212
9.5.3 Referencing BIRT Related Objects.....	213
9.5.4 Setting in Report Design.....	215
9.6 Java Event Handler Example	216
9.7 Additional Resources	218
9.8 Deploying external Java classes	219

Chapter 10 Logging	221
10.1 Java Script.....	221
10.2 Use custom Java classes to log messages.....	222
10.3 Java Event handlers	222
10.4 Recommendation	222
10.5 Logging Example	222
10.6 Another Logging Example	224
10.7 Le BIRT Expert Debug Popup Window.....	224
10.8 Eclipse Log File.....	224
Chapter 11 Creating and Using Templates – A Starting Point for Future Reports	225
11.1 Creating a template	226
11.2 Adding Report Items to a Template	228
11.3 Registering a template.....	229
11.4 “New Report Wizard” Template Changes	230
11.5 Using a template.....	231
11.6 Template drawbacks.....	231
Part II Continuing Best Practices in Your Report Design	233
Chapter 12 BIRT Tables.....	235
12.1 Understanding Column Bindings.....	235
12.2 Dataset Bindings	237
12.3 BIRT Table Bad Practices	238
12.3.1 Filtering.....	238
12.3.2 Sorting	238
Chapter 13 Using Charts	241
13.1 Chart Output Formats.....	242
13.2 Which Format is Recommended?	242
13.3 Using Styles in Charts.....	243
13.4 Using i18n Messages in Charts	245
13.5 Chart Scripting	246

13.6 Chart Interactivity	246
13.7 Using Images in a Chart	247
Chapter 14 Planning Ahead for Internationalization (i18n) and Localization (l10n) .	249
14.1 Using external resource files	249
14.2 Use Scripting to display formatted values	250
14.3 Use Database Values for Localized Values.....	251
14.4 Use existing Java framework for handling Localization.....	252
14.5 Recommendations.....	252
14.6 Using External Messages in Reports.....	253
Chapter 15 Separating Business Logic from Display Logic	255
Chapter 16 Exception and Error Handling	257
16.1 No Data Messages	257
16.2 Catching an Exception in JavaScript	259
16.3 Handling Exceptions in Java Event Handlers.....	260
16.4 Forcing Exceptions to Fail a Report	260
16.5 Display Error Messages with Scripted Data Sets.....	260
16.6 Error and Exception Messages in the BIRT Viewer	261
16.7 Error and Exception Message using a Custom Viewer	262
Chapter 17 Debugging Reports.....	263
17.1 Using Logging	263
17.2 Using the Le BIRT Expert Debug Popup Window	263
17.3 Using the JavaScript Debugger.....	265
17.4 Using the Java Debugger for External Java classes	269
17.5 Using the Process of Elimination	272
Chapter 18 Improving the performance of BIRT Reports	273
18.1 Using Filters and Sorting in the Design.....	273
18.2 Simplify the Report Design.....	274
18.3 Measuring Performance	274

Chapter 19 Preparing for different Report Output Types.....	275
19.1 PDF	276
19.2 CSV	278
19.2.1 BIRT Engine	278
19.2.2 CSV Emitter	279
19.2.3 BIRT Viewer.....	279
19.2.4 BIRT iServer IDAPI	281
19.2.5 Custom Code.....	281
19.3 Excel Output	281
19.4 XML Output	282
19.5 Other Formats	282
Part III Following Conventional Software Engineering Methods for your Reporting Project	283
Chapter 20 Using Source Control.....	285
20.1 Using a Source Control Tool.....	285
20.2 Using CVS	286
20.3 Using Subversion	286
Chapter 21 Using BIRT in a Team Environment	289
21.1 Using Source Control to collaborate	289
21.2 Using a Network Drive	290
21.3 Using the Resource and Template Folders	291
21.4 Configuring Eclipse in a Team Environment	293
21.5 Using iServer for Source Control	293
Chapter 22 Comments.....	295
22.1 Comments in JavaScript and Java.....	295
22.2 Comments in the Report Design	297
22.3 Comments Best Practices.....	298
Chapter 23 Documentation	299

23.1 Report Documentation Contents	299
23.2 Using the Description Field in BIRT Designer	300
Chapter 24 Solid Testing	301
24.1 Developing a Test Plan.....	301
24.2 Testing the Data	302
24.3 Testing the Report Parameter Screens.....	302
24.4 Testing the Report Output	303
24.5 Testing Report Security.....	304
24.6 Testing Performance.....	304
24.7 User Testing.....	305
24.8 Improving Performance	305
24.9 Unit Testing	307
24.10 Automated Testing	307
Chapter 25 Bug Tracking	309
25.1 Bug Tacking	309
25.2 Tools to Track Defects.....	309
Appendix I Check List.....	311
Appendix II Deployment.....	312
Deploying to the BIRT Report Runtime Engine.....	312
Using the BIRT Engine to Execute Reports	314
Deploying to the BIRT Viewer	314
Deploying to the BIRT Server	316

Chapter 1

Introduction

In this Chapter:

- Who This Book is For
 - What This Book is Not
 - Assumed Prior Knowledge
 - Version of BIRT Covered
 - About the Author
 - Contents of the Book
 - Additional Resources
 - Continuing the Discussion
-

This guide is meant to provide best practice advice for developing BIRT Reports. BIRT is a powerful reporting tool with many different options and settings. There are also several different ways to do the same thing.

As report requirements change, designs get more complex and more developers get involved – it becomes extremely important that developers follow a set of Best Practices. This is necessary to ensure a consistent look and feel of the reports, to promote code re-use, and to maintain the reports going forward. As projects continue through their lifecycle – developers may come and go, users may request design changes and the database undergoes modifications. However, having a framework for Best Practices will help ease maintenance and reduce time of development over the long term.

This guide will cover

- The implementation decisions every project needs to make
- Developing a Reporting Framework to encourage Best Practices
- Using Best Practices in your Report Design
- Following Conventional Software Engineering Methods for your Reporting Project

The techniques described in this book have been used successfully on a number of BIRT related projects over the past couple years. This guide offers *one point of view* and a starting point for others to create their own framework, best practices and coding standards. It is by no means

meant to be an “end of discussion” way of doing things, but a starting point for your organization or team to take forward.

1.1 Who this Book is for

This Book is for any Developer or Architect who is designing or implementing a BIRT solution. It provides valuable advice on the best way to use the designer tool. The book covers all aspects of designing a set of reports and focuses on delivering an effective overall BIRT reporting solution from the designer tool’s point of view.

1.2 What This Book is Not

This book is not a beginner’s BIRT tutorial. There are other books and web sites that have beginner tutorials that will be helpful to someone brand new to BIRT.

This book only covers BIRT as is in the BIRT Designer. It does not cover how to deploy BIRT. Appendix II does touch on deploying the framework to the BIRT Engine, but it does not go into depth on the topic.

1.3 Assumed Prior Knowledge

This book assumes basic knowledge of BIRT. The reader should understand how to connect to a data source and create a basic report. The book will give step-by-step instructions on certain topics, but will assume basic knowledge of the tool. This book is meant to help you start your BIRT team project off right by walking through the design decisions that need to be made, creating your own reporting framework that encourages best practices and following software engineering principles to ease development and future maintenance.

1.4 Version of BIRT

This book focuses on using the open source version of BIRT. Some specific details of the Actuate commercial option will be discussed, but all screenshots and techniques described will be using the open source version.

BIRT version 2.5.1 is used in the majority of the examples in this book. Features of BIRT 2.5.2 and 2.6 are also covered. The latest version of BIRT can be found at www.eclipse.org/birt.

1.5 About the Author

David Mehi, “Le BIRT Expert”, has worked as a JEE consultant for the past 10 years. After spending some time at Ford Motor Company and IBM Global Services, he worked at Actuate Professional Services as a Senior and Principal Consultant. Based out of New York City and London, he has worked at many name brand financial institutions, government and military offices and other small and large corporations. He has been involved in many high profile BIRT related projects over the past few years.

David now does independent consulting related to BIRT, Actuate and JEE through his company, InfoLight Solutions LLC (www.infolightsolutions.com). David is currently based out of Detroit, MI, USA. You can contact David Mehi at dmehi@lebirtexpert.com or visit his web site at www.davidmehi.com.

1.6 Contents of the Book

This book is divided into the following parts and chapters.

Chapter 2, Plan Ahead. Before jumping into that first report, it is helpful to pause and think about the best ways to approach the project.

Part I, Developing a Reporting Framework

Part I introduces the concept of a reporting framework and how to create one for your project.

Chapter 3, Preparing the Framework. This chapter covers the first steps to creating a framework – creating a folder structure, reporting libraries and using common naming conventions.

Chapter 4, Create Re-Usable Report Themes and Styles. This chapter covers the different ways to create themes and styles and offers recommendations on best approaches.

Chapter 5, Creating and Using Report Items for Common Use in Libraries. This chapter discusses how to use libraries in an efficient way.

Chapter 6, Establishing a Secure and Consistent Way to Access Data. This is the largest chapter in the book and discusses the most important part of report design. The many different ways to access data and best approaches for securing your connection details are covered.

Chapter 7, Report Parameters. Covers best practices for using report parameters in your framework and report designs.

Chapter 8, Scripting Using JavaScript. This chapter covers how to add scripting with JavaScript to your reports. Discusses advantages and best approaches.

Chapter 9, Create and Leverage Re-Usable Java Components. This chapter covers how to use existing Java objects and implement Java event handlers in your report designs.

Chapter 10, Logging. This chapter covers the different approaches to logging in the Reports.

Chapter 11, Creating and Using Templates – A Starting Point for Future Reports. All of the components of a framework can be combined into a template so the developer can start with everything in place. This chapter covers different approaches to using templates.

Part II, Implementing Best Practices

Part II introduces ways to continue Best Practices in other parts of your report design.

Chapter 12, BIRT Tables. This chapter offers tips on how to make the most of the table item.

Chapter 13, Using Charts. This chapter offers pointers on how to make the most of the chart item.

Chapter 14, Planning Ahead for Internationalization (i18n) and Localization (l10n). This chapter covers the different ways that BIRT supports different languages and formatting.

Chapter 15, Separating Business Logic from Display Logic. This chapter offers some suggestions on how to keep business logic separated.

Chapter 16, Exception and Error Handling. This chapter covers different ways exceptions and errors can be handled in BIRT.

Chapter 17, Debugging Reports. This chapter offers valuable tips on ways to debug your reports.

Chapter 18, Improving the Performance of BIRT Reports. This chapter offers tips on how to improve the performance of your reports.

Chapter 19, Preparing for Different Report Output Types. BIRT is capable of outputting reports in different formats. This chapter offers pointers on how to best prepare your design for these different formats.

Part III, Following Conventional Software Engineering Methods for your Reporting Project

Part III introduces ways to continue the typical software development process in your reporting project.

Chapter 20, Using Source Control. This chapter offers suggestions on how to leverage source control

Chapter 21, Using BIRT in a Team Environment. This chapter covers the different ways that multiple developers can work together to develop a set of reports.

Chapter 22, Comments. This chapter offers approaches for leaving comments for future developers.

Chapter 23, Documentation. This chapter provides suggestions on the type of documentation to leave behind for future maintenance.

Chapter 24, Solid Testing. This chapter covers approaches for developing a test plan to test all aspects of your reports.

Chapter 25, Bug Tracking. This chapter offers ways to track bugs and defects that arise from testing.

Appendix I, Checklist

Think you're ready to start your reporting project? Go through this checklist to confirm you have everything in place to begin.

Appendix II, Deployment

This book does not cover all aspects of how to deploy BIRT. However, there are a few things to keep in mind when deploying the framework and BIRT reports to the BIRT engine, BIRT Viewer or the Actuate Viewer and iServer.

1.7 Additional Resources

There are several report and code examples shown in this Book. The reports and code are relatively straightforward examples, and do not show very complex situations. However, they provide a good example of the topics discussed in the book and should help your project move forward. Also included is a document describing how to setup the MySQL Database.

All code examples can be found at:

<http://www.lebirtexpert.com/code/bestPractices/>

<http://code.google.com/p/lebirtexpert/>

Book Examples

These examples are BIRT projects that are shown in screenshots and code examples in the eBook. The projects are labeled according to Chapter.

Base Reporting Framework

This is the base reporting framework that your team can use as a starting point going forward. This framework will be maintained and continually improved upon as time goes on.

- ReportingFramework
- ReportingFramework_Java

Classic Models Example Project

This is a sample project that uses the base reporting framework as a starting point and contains some of the example reports and code from the book and other items.

- ClassicModelsExample
- ClassicModelsExample_Java

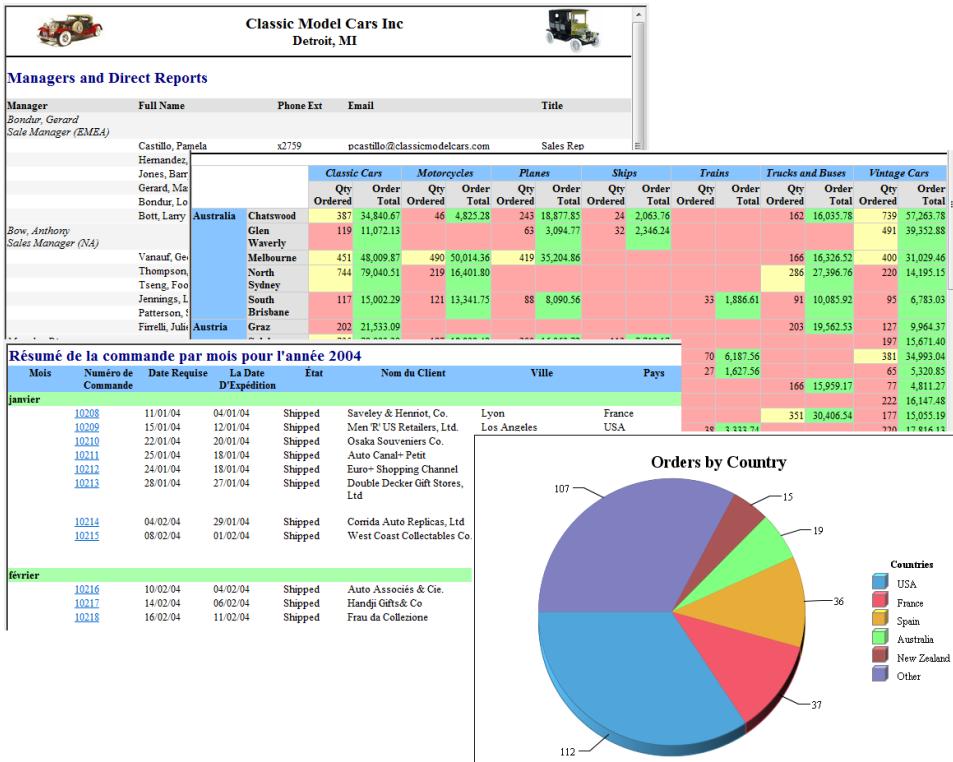
Here are a few screenshots of some of the example reports. These reports provide working examples of some of the concepts discussed in the book.

A screenshot of a file explorer window showing the 'Reports' folder. It contains several sub-folders and files:

- HR**: Contains files like 'employeeList.rptdesign', 'managerList.rptdesign', and 'orderDetail_dd.rptconfig'.
- Sales**: Contains files like 'charttest.rptdesign', 'ordersByMonth.rptconfig', and 'salesCrossTab.rptdesign'.
- TestReports**: Contains files like 'customerListReport_JEH.rptdesign', 'customerScriptedDS_JEH.rptdesign', and 'loggerTestReport.rptdesign'.
- TestThemeReports**: Contains files like 'customerList.rptdesign', 'customerListThemeSwitcher.rptdesign', and 'testConnection.rptdesign'.

A screenshot showing five separate report preview windows side-by-side, each displaying a different report layout:

- Report 1: A table for Sales Rep 'Jennings, Leslie' showing customer details like Customer Name, Customer Full Name, City, and Credit Limit.
- Report 2: A table for Sales Rep 'Jennings, Leslie' showing the same information.
- Report 3: A table for Sales Rep 'Jennings, Leslie' showing the same information.
- Report 4: A table for Sales Rep 'Jennings, Leslie' showing the same information.
- Report 5: A table for Sales Rep 'Jennings, Leslie' showing the same information.



1.8 Continuing the discussion

Finding ways to implement best practices in your BIRT project is an on-going discussion. Did the techniques in this book work for you? Do you have any questions on the content of the book? Do you have your own best practices to share? Do you have any suggestions for how to make this book better?

We would love to hear from you! Visit our website www.lebirtexpert.com to ask questions, submit feedback, join the email list and contribute to the discussion.

Updated versions of the book, new example code and other resources will be available to those who have purchased this book.

Chapter 2

Plan Ahead

In this Chapter:

- Questions to Ask
 - Design Decisions
-

A very common scenario is for a developer to start “playing around” with BIRT, and before they know it, they have created a wonderful report that the end user and upper management loves! They continue to create another one, and then another one. Soon enough, 10 reports are created and ready to be deployed. Another developer joins, “experiments” with BIRT, and they create five reports by the end of the week. The client is so excited!

With minimal testing, the reports are thrown into production. Within a week, the user community, emboldened by the recent progress, comes back with a few “suggestions.” They want to change the headings on all the reports. They want to add page numbers to the bottom and add alternating shading to the table rows. Oh yeah, and something about “all the reports looking a little different from each other.” The database administrator mentions a few table and field name changes that are going to be in the next release. And management says a third report developer has been hired to work on more reports based on “new requirements.”

As you can imagine, the changes required to all 15 reports can become messy very quickly! How can this be avoided? What processes can be put in place to ease future development and keep all of the developers on the same page?

2.1 Questions to Ask

Before several developers jump in and start developing reports independently of each other, it is important to pause and take some time to think about the best way to go forth. Here are a few questions to ask yourself before writing that first report.

What type of reports will be developed?

Be sure to gather the appropriate requirements for your reports. If you have not done so already, please refer to *Le Birt Expert's Guide to Gathering Bullet-Proof Report Requirements* ebook for guidance on how to develop rock solid report requirements.

Where is the data coming from?

Will the reports access the database directly? Will they access another form of data source such as a flat file or web service? Will your application pass in pre-populated POJO's? What type of special calculations will be needed? A consistent way of accessing data can be used across all report designs, and will make future reports that much easier to develop.

How will the reports look?

Will the reports have a common look and feel? Common header? Common footer? Common color scheme? By using a combination of techniques, the BIRT Designer allows developers to maintain a consistent look and feel among the reports.

How will the users interact with the report?

Will the users want to click parts of the report for more detail? What kind of parameters should be exposed to the user? Additional interactivity requirements? BIRT has ways to accommodate these needs.

How will the reports be viewed?

Determine the different ways the users will be viewing the reports. Will they be viewed over the web? Will they be sent directly to the printer? Converted to Excel for editing? The answers to these questions will determine what type of libraries need to be developed to accommodate common controls across report designs.

Any other special requirements?

Are there any special requirements that you will need to plan ahead for? Making unusual web service calls, doing complex calculations, wanting to leverage existing Java classes? Plan ahead for any special requirements and confirm they can be achieved the way you anticipated. Do a "proof of concept" to make sure that the special requirement can be implemented and packaged to allow re-use.

2.2 Design Decisions

There are many design decisions that will need to be made when implementing a BIRT Reporting Solution. This section lists those decisions so the reader can keep them in mind. The rest of the book will cover all of the options, offer recommendations and help you make the right choice.

- 1) What should our project folder structure look like?
- 2) What control and variable naming conventions should we use?
- 3) How should we implement styles and themes?
- 4) How can we use libraries efficiently to create re-usable components?
- 5) What is the best way to create a secure and consistent way to access our data?
- 6) How can we efficiently use logging?
- 7) How can we leverage templates effectively?
- 8) How can our existing external Java classes be used?
- 9) How can we separate business logic from display logic?
- 10) How can we make our reports more dynamic using scripting?
- 11) What are Java Event Handlers? And how can they be useful to us?
- 12) What is Table Binding and how does that affect our Reports?
- 13) What is the best way to do Exception and Error Handling?
- 14) What are methods for handling i18n and l10n?
- 15) What source control methods are compatible with BIRT?
- 16) What debugging approaches work best?
- 17) What's the best way of having a team of BIRT developers work together efficiently?
- 18) How can we develop a test plan for BIRT Reports?
- 19) How can we keep track of bugs and enhancements?
- 20) What kind of documentation should we leave behind for future developers?

By the end of this book, you should be able to make the right decision for your project.

Part I

Developing a Reporting Framework

In this Section:

- *Chapter 3 – Preparing the Framework*
 - *Chapter 4 – Create Re-usable Report Themes and Styles*
 - *Chapter 5 – Creating and Using Reporting Items for Common Use in Libraries*
 - *Chapter 6 – Establishing a Secure and Consistent Way to Access Data*
 - *Chapter 7 – Use Common Report Parameters and Report Parameter Messages*
 - *Chapter 8 – Scripting using JavaScript*
 - *Chapter 9 – Create and Leverage Re-Usable Java Components*
 - *Chapter 10 – Logging*
 - *Chapter 11 – Creating and Using Templates – A Starting Point for Future Reports*
-

At the core of developing Best Practices for your development team is coming up with a reporting framework that can be used for all of your report designs. Using a framework will help keep all of the developers on the same page, allow quicker development time and ease future maintenance. Having a good framework in place will improve the efficiency of creating new reports.

The components of this framework are:

- Creating a folder and library structure for your project to grow
- Establish a common naming convention across all report designs, items, variables and parameters
- Create re-usable themes and styles
- Establish a common “look and feel” with re-usable report components and libraries
- Establish a secure and consistent way to access your data
- Use common report parameters and report parameter messages
- Create and leverage re-usable scripting libraries
- Create and leverage re-usable Java Components
- Methods for logging
- Create a common “starting point” for all future reports

Using a combination of libraries, templates, style sheets, script files, Java code and other tools, this guide will explain how to implement such a framework in the BIRT Report Designer. The framework is specific in the goals it is trying to achieve. This guide offers some examples on how a developer can achieve results. However, each project is unique and it would be impossible to offer examples of all possible combinations of implementation. Our hope is that you can leverage the ideas and examples in this guide to help you formulate your own framework to achieve Best Practices in your environment.

Chapter 3

Preparing the Framework

In this Chapter:

- Create a Folder Structure
 - Creating Report Libraries
 - Creating a Related Java Project
 - Common Naming Conventions
-

Each framework needs a structure and a place to grow. Without structure, your project will quickly become disorganized and unwieldy. You must prepare your environment to contain the framework.

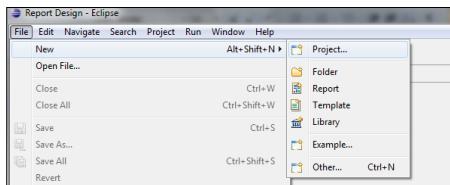
3.1 Create a Folder Structure

To prepare the framework, first create a folder structure to hold your report designs, style sheets, libraries and templates. A common mistake among first time report developers is to put everything into the root folder. Even though this is the easiest way (and the default), your root folder will quickly become cluttered.

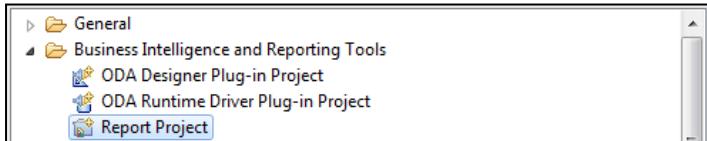
***Tip**

Le Birt Expert's recommendation is to create a folder structure that allows your team to add reports designs, library files, templates, script files and other components in a scalable, organized way. The following folder structure provides an example of such.

Create a new project in the Birt Designer.

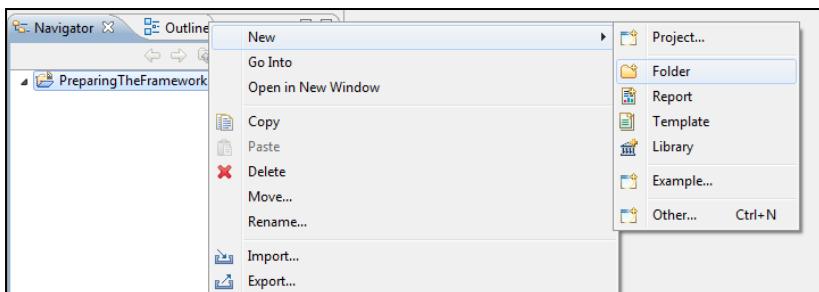


Choose Report Project.

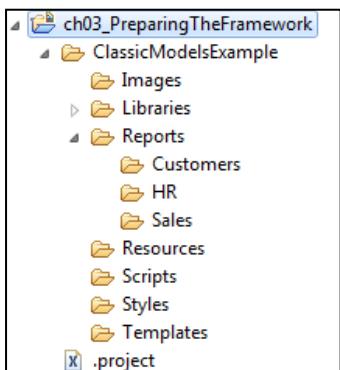


Click Next. Name the project

Right click on the project name, choose New → Folder



Add the following folder structure



<Project Name> - (*ClassicModelsExample*) – Project name as the parent folder

Images – provides a holding spot for images used in the reports.

Libraries – provides a place for all libraries developed.

Reports – provides a place for report designs to be stored. It is recommended to create additional subfolders to group reports relating to the same business function.

Resources – provides a place for property files, i18n resource files and other shared resources.

Scripts – provides a place for JavaScript files containing custom functions that can be called from anywhere in the report.

Styles – provides a place for external style sheets.

Templates – provides a place for report templates.

You may only have a couple reports now, but having a folder structure in place will allow you to keep your files organized as your project grows. It will be a major pain to have to re-arrange your files and folders at a later date.

The folder structure you pick will impact how you deploy your project. A folder structure that works well in the Report Designer may not work as well when deploying. It's best to consider how the solution will be deployed when you are creating your initial folder structure. Having a parent folder with your project name will help keep your folders and files separate when it comes time to deploy to the BIRT Engine. See Appendix II for more information.

***Tip**

Unfortunately, BIRT does not make it easy to re-arrange the folders and report designs later on. If a library, template or other resource is moved, the BIRT Designer does not maintain the link in the report design. If this happens, you will see an error message similar to this one.

Content of this report item is corrupted. If you wish to continue to work on this report, you should delete this item first. Failure to do so may further damage the report design file, and/or render the report designer unresponsive.

Your report design is not actually corrupt, but the link to the library or other items has changed. Click the “XML Source” tab to manually fix the path to the library or missing item.

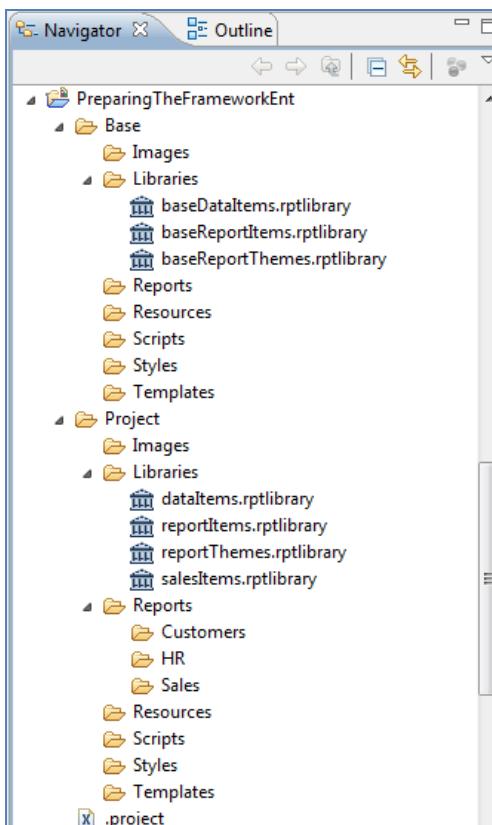
Your team will have to decide on the folder structure that is best for your needs.

3.1.1 Alternative Folder Structure for Use in Multiple Projects

For companies with multiple BIRT Projects, a base reporting framework can be established for use across multiple projects. The contents of this book focus on creating a framework for an individual project, however, the principles can be applied to creating a base framework used for all reporting projects in an organization. A base set of folders, libraries, themes and templates can be established for use across all reports, while still providing a way for project-specific items.

Use a separate folder to hold your base framework and all common items to be shared across all projects. Create another folder to hold your project specific report items. Using this separation, your base framework can remain consistent through your projects, while still providing room for your individual projects to grow.

An example of this would be the following.



It is up to each team to establish the best folder structure that meets their requirements. The rest of the book focuses on individual projects, but if you are designing a framework to be used on multiple projects, keep this approach in mind.

3.2 Creating Report Libraries

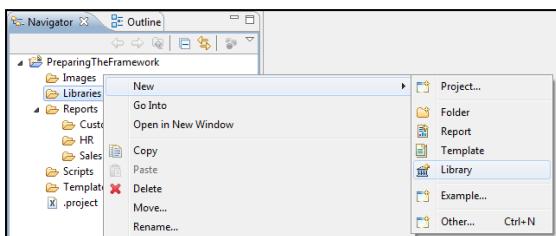
The BIRT Report Designer allows us to create a library to hold common report components. These libraries can then be used among all reports to share and re-use those components. This is a powerful capability that, if used correctly, allows for expansive growth of your report designs and re-usable components.

The libraries can hold several different types of components that can be used among the reports. A common mistake among first time report developers is to create one big library that holds everything. This library file will quickly become cluttered and it will be hard to find what you're looking for. Also, having only one library file means only a single developer can work on it at one time, limiting developer productivity.

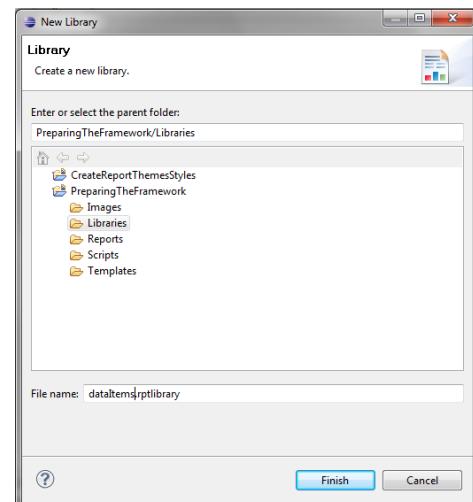
***Tip**

Le BIRT Expert recommends starting with at least three report libraries to hold the different types of re-usable report components. You can add additional libraries as necessary.

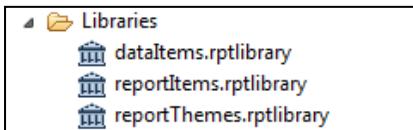
To create a report library, right click on the “Libraries” folder, go to New → Library



Name the library and click Finish



Create the following three libraries.

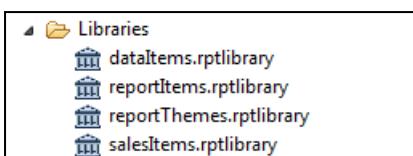


dataItems.rptlibrary – contain data related report items

reportItems.rptlibrary – contain report related report items

reportThemes.rptlibrary – contain report related themes and styles

For a small number of reports, these three libraries should be sufficient. For a larger number of reports, consider a finer-grain approach. In this example, I'm going to add another library that will contain items specifically used in the Sales Reports. If I had several libraries for the Sales Reports, I could create a subfolder to hold them.



***Tip**

If you have a need for many different types of re-usable data items, report themes, or other report items according to business function (or other type of grouping), create libraries as necessary. For example, if you have a web service data source that some of your reports use, it might make sense to create a "dataItemsWS.rptlibrary" file. A report design can use multiple libraries.

Creating the folder structure and the library files give us room to grow while we develop the rest of the framework and the reports. This insures us that our project will not become cluttered quickly and offers us placeholders to put our components and designs.

3.3 Creating a Java Project

Along with the BIRT Reporting Project, it is useful to create an accompanying Java Project. This Java Project will hold any Java classes or Java event handlers created specifically for your BIRT reports. Using a separate project for this is useful when it comes time to package and deploy

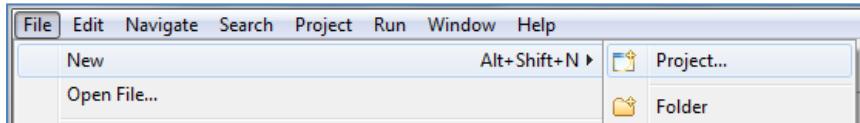
the Java Classes. (The Report Designs and the JAR files are typically deployed to different folders.) It also allows different developers to work on the Java classes and allows the Java-specific features of Eclipse to be utilized.

Once the Java specific project is created, it must be added to the BIRT Project's class path so the reports can reference the files.

3.3.1 Create the Java Project

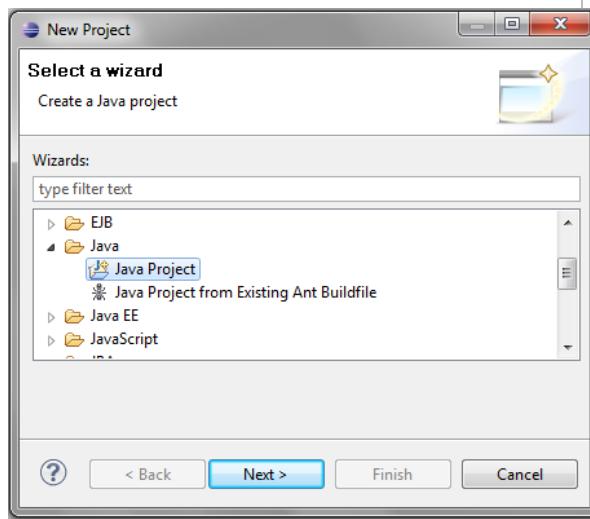
To create a Java specific project, do the following:

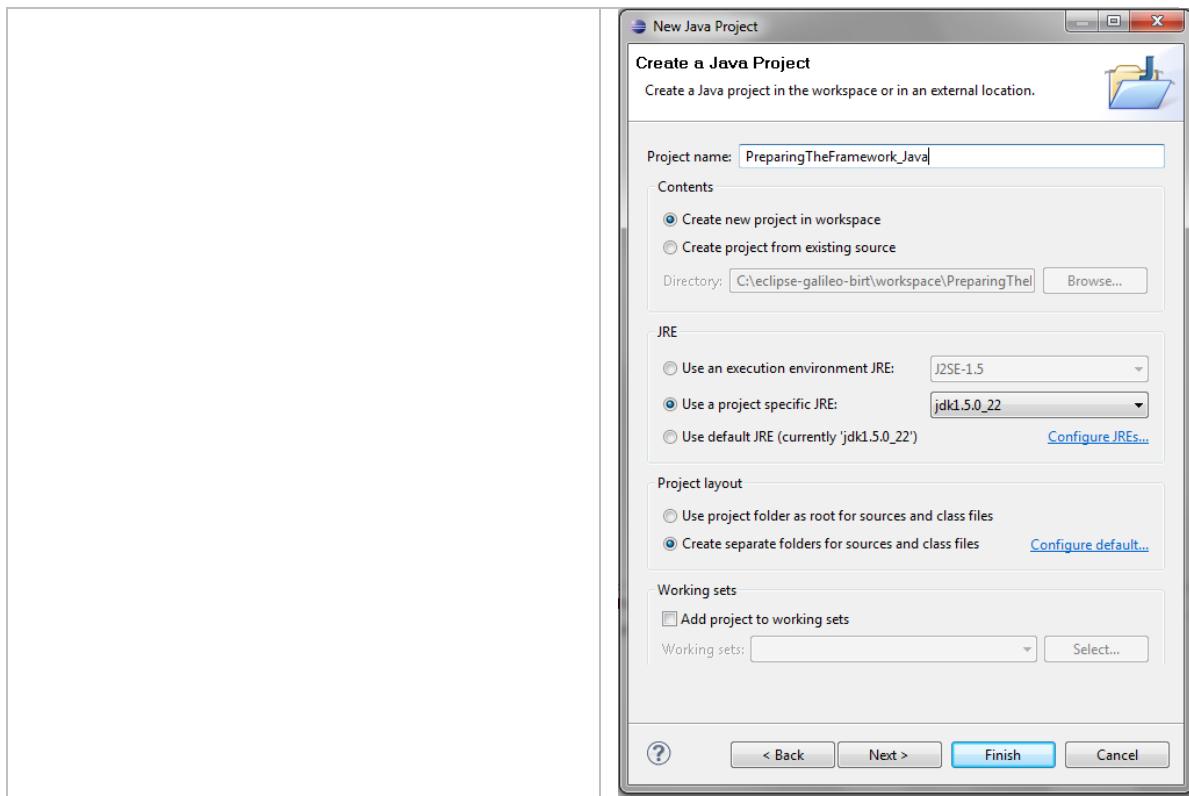
Go to File → New → Project ...



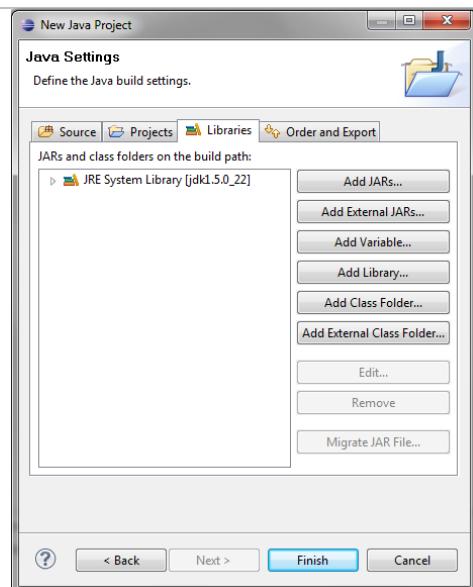
Choose Java Project and click Next.

Enter the name of the project. A good naming convention is to use the same project name with either “_Java” or “_JEH” at the end. (JEH stands for Java Event Handlers).

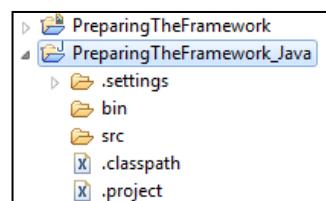




Enter additional properties or add JAR files to the classpath. Click Finish



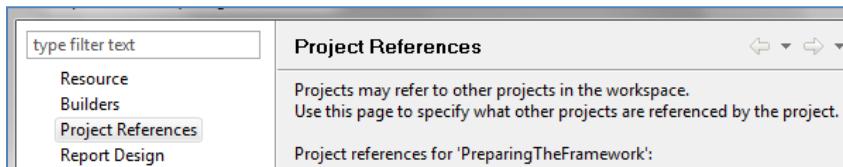
Another project will be created to hold your related Java classes.



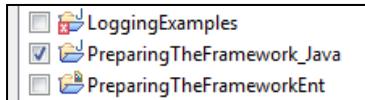
3.3.2 Referencing the Java Project

Even though the Java project has been created, the BIRT project must know about it in order to reference it.

Right-click on your BIRT project and choose “Properties”. Click on the Project References Link.



Select the Java Project that was created in the previous step



Click OK

The BIRT Project is now enabled to reference any Java classes created in this Java Project.

3.4 Common Naming Conventions

It is important that the team follows a common naming convention across all report files, controls, data fields, variables and parameter names, especially when there are multiple developers creating multiple reports with multiple libraries and multiple code functions! Code conventions improve the readability of the code, allowing developers to understand new code more quickly and thoroughly.

A common mistake is to forget your naming convention, and have the same variable or data field spelled differently in each report. (One of *Le Birt Expert's* pet peeves is when developers' variable names are spelled wrong. Example: dblAmuontRecieved)

***Tip**

Le Birt Expert offers the following recommendations for naming conventions. Some developers can be religious when it comes to naming conventions. The goal is to have a common convention that your team can use and understand. Whatever convention you choose, be consistent.

The name assigned to a file, control, variable, function, style or anything else should be as descriptive as possible, without being too long. If your name has over 25 characters, try to abbreviate the words.

Keep the names consistent. Some developers will use “pStartDate” for one report, and then “pBeginDate” for another report. If these names serve two different business purposes, then fair enough. But if they both mean the same thing, then call them the same thing.

A convention Le BIRT Expert recommends is “camelCase”. The first “word” should be lower case. Any additional “word” should have its first letter capitalized and the rest of the letters in lower case. Any exception to this would be an acronym that is better understood in all CAPS. This applies to file names (report, library, template, script, image), function names, and style names.

Examples:

customerSalesDetail.rptdesign, headerTitle, salesReport.rptlibrary, convert.ToDecimal,
customerPNL.rptdesign, ACHdetail.rptdesign

Variable names have type abbreviated in front of name. Since Javascript is not type-cast by default, it helps to put the type in front of the name.

JavaScript Recommended Naming Conventions

Variable Type	Prefix	Example
String	str	strCompanyName
Integer	int	intTotalCount
Double	dbl	dblAmountDue
Float	flt	fltTotalBalance
Array	arr	arrDataFields
Undefined Type	var	varDefineMe
Boolean	bln	blnOpenWindow
Object	obj	objContainer

Date	dte	dteStartDate
Long	lng	lngStartTime
Report Variable	rpt + type-prefix	rptBlnShowHeader
Page Variable	pge + type-prefix	pgeFltTotalAmount
Global Variable	gbl + type-prefix	gblIntTotalCount

Parameters have “p” as a prefix. It is not necessary to define the type, because in BIRT you can define the type of the parameter. Often, the name is enough to determine the type. Using parameters defined in the library will also bring a common way to handle the same kind of parameters (this is discussed later in the book). However, if you feel that including the type in the name will help, then go for it.

Report Items have the type abbreviated in front of the name

Report Items Recommended Naming Convention		
Report Items	Prefix	Example
Label	lbl	lblReportTitle
Text	txt	txtProductDesc
Dynamic Text	dt	dtUserComment
Data	dat	datColumnName *Best to use name that reflects the piece of data that is being displayed
Image	img	imgEmployee
Grid	grd	grdAlignment
List	lst	lstPageLayout
Table	tbl	tblDetailResults
Chart	cht	chtSalesNum
Cross Tab	crt	crtProducts
Data Sources	db	dbClassicModels * Best to use name of data source. Even though it may not be a database, good to use “db”. Can also use “file”,

		“ws” or other abbreviation that applies to your data source.
Data Sets	ds	dsCustomerOrders
Data Set Parameters	dsp	dspStartDate
Data Cube	dc	dcCustomerCube
Report Parameters	p	pEndDate
Aggregation	agg	aggTotalSum
Master Page	mp	mpMainPage

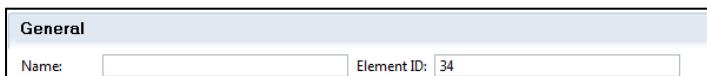
Any Java classes should follow conventional Java naming standards.

***Tip**

The names chosen in the examples for the rest of the book are names that I see fit to describe my report items. They are offered as an example of how to name items. Each team will need to decide on a set of naming standards that fit their environment.

3.4.1 Naming Report Items

BIRT provides a “name” property for each report item.



It is not necessary to name every Report Item that is created. However, there are two situations where you will want to provide a name.

- 1) When adding a Report Item to a library, be sure to give it a name that represents the function of the Report Item. This way, when other developers see it, they will have an idea what it is for.
- 2) When using the report through Actuate’s BIRT Studio or Interactive Viewer, the name of the Report Item is often displayed to the user when they are manipulating the report design. Make sure the Report Items that are exposed to the user have user-friendly names.

Chapter 4

Create Re-Usable Report Themes and Styles

In this Chapter:

- What is a Theme?
 - Create a New Theme
 - Using Pre-defined Styles in your Theme
 - Creating Custom Styles in Your Theme
 - Importing a CSS File Into Your Theme
 - How to Proceed With Styles?
 - Creating Styles Directly in the Report Design
 - Le BIRT Expert's Recommended Approach
-

Companies typically want their reports to have a consistent look and feel to them. Part of this is establishing a common font style, font size, specific formatting and color scheme. Defining themes and styles in your library allow them to be used by other reports.

4.1 What is a Theme?

A theme can best be described as a set of styles. A report can apply only one theme at a time. However, styles in other themes can be applied programmatically as long as the report references the library they are located in. A report can also have individual styles defined in its design. More on this later.

All themes and styles should be created in the “reportThemes.rptlibrary”. This allows the reports to share the common themes and styles, thus promoting re-use and maintaining a consistent look and feel across reports.

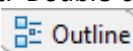
The BIRT Report Designer allows five different ways to implement themes and styles. The best way depends on how the reports will be used and if you have existing styles that can be

leveraged. This section will show you the five different ways, talk about the pro's and con's of each, and offer suggestions on which one to choose for your project.

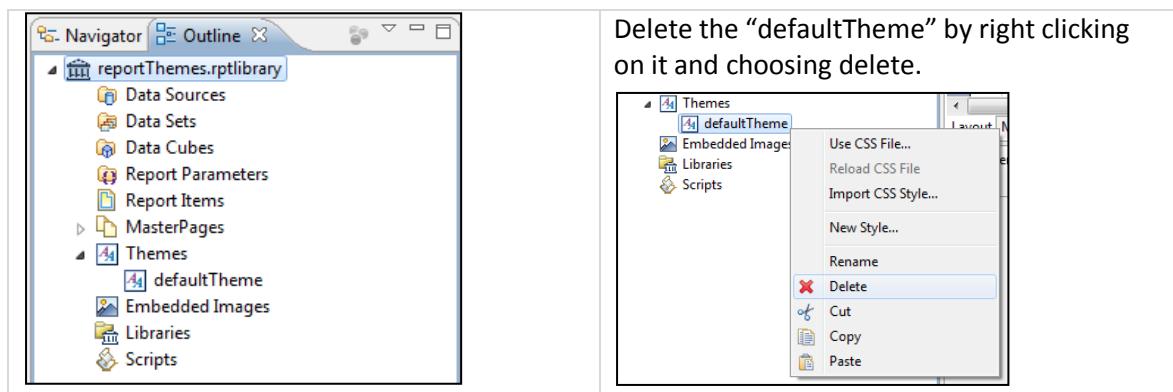
The five ways are:

- Creating predefined styles in your theme
- Creating custom styles in your theme
- Importing a CSS file into your theme
- Linking to an external CSS file in your theme
- Creating custom styles in your report design

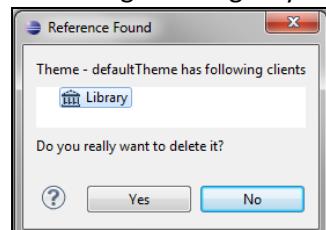
4.2 Create a New Theme

When you create a new library in the BIRT Designer, a default empty theme is automatically assigned to it. Double-click to open the “/Libraries/reportThemes.rptlibrary”. Click on the outline tab.  Expand the “Themes” tab. This will give you a view of all the items defined in the library. The “defaultTheme” is empty for now.

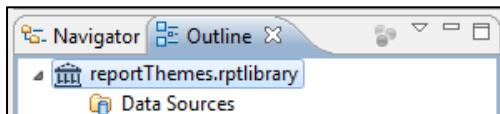
You can choose to rename the theme, add a new theme and delete this one, or delete the theme and add a new one. In this example, we'll delete the “defaultTheme” and create a new one.



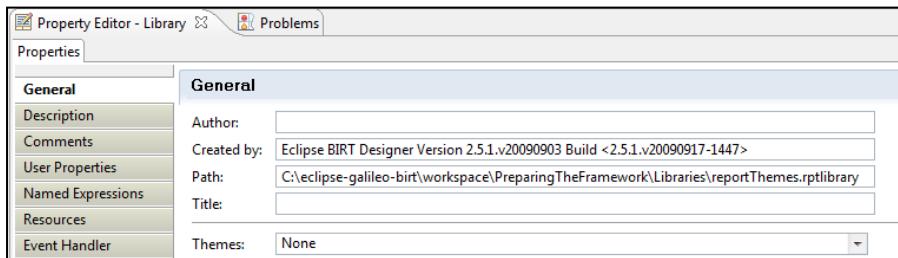
BIRT Designer will give you a warning. Click “Yes” to continue.



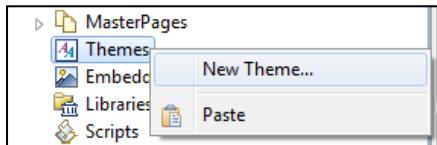
This means that this Theme was the default Theme for the library. In the outline tab, click the “reportThemes.rptlibrary” library.



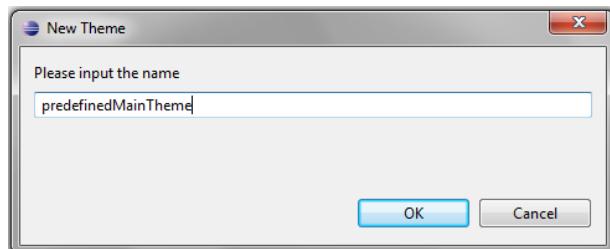
In the Properties window, you will see the General Properties associated with this library. Notice the “Theme” now is set to “None”. Initially, it was set to “defaultTheme”. That is why the Report Designer warned that the library was using it. We will create a new theme next.



Create a new theme by right clicking on the “Themes” icon and choose “New Theme ...”.

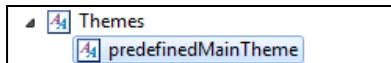


Enter the name of your new Theme. This theme will be named “predefinedMainTheme”. Why this name? This name reflects that it is the main theme that will be used in the reports. It also tells us that it uses “predefined styles”.



Click “Ok”.

The BIRT Report Designer has now added a new Theme to the library.

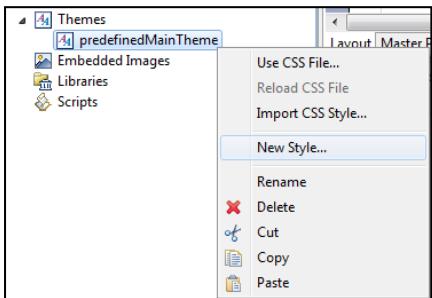


4.3 Predefined Styles

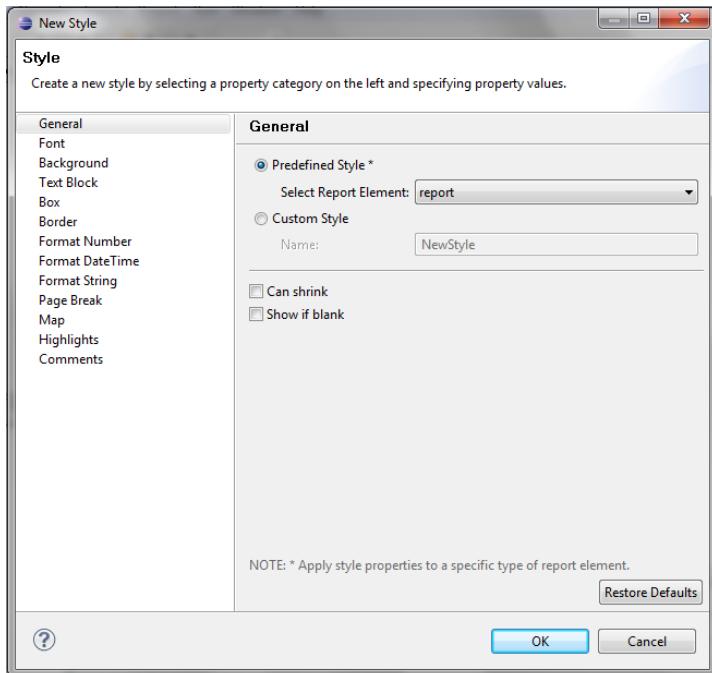
This section covers using Predefined Styles in the Theme.

4.3.1 Adding predefined styles to the Theme

To add a predefined style to the Theme, right click on the Theme and choose “New Style”.

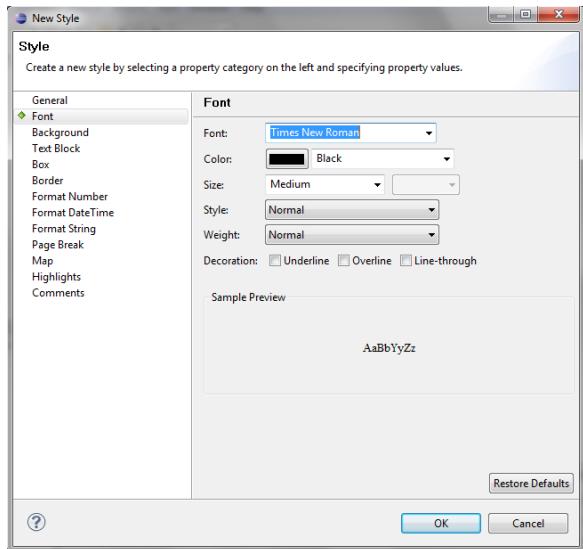


The New Style window allows you to create a new style.



Choose “Predefined Style” option. Select “report” from the element option list. This allows the developer to set the properties for the whole report by default. This allows a global setting for fonts, background colors, borders, page breaks, etc.

Click the “Font” property on the left side. Choose “Times New Roman”.

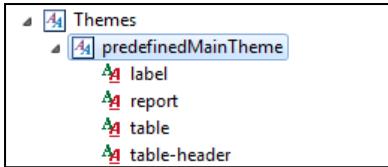


You will notice a “report” style has been added to your Theme.



Add the following styles with properties

Predefined Styles	
Predefined Style	Settings
table	Background color: RGB(240,240,240)
table-header	Font: Times New Roman Weight: Bold Background color: RGB(225,225,225)
label	Font: Times New Roman Size: Larger Weight: Bolder Decoration: Underline

***NOTE**

Be sure to save the report library after your changes! An unsaved report library will have an “*” next to it.



Using predefined styles allows you to control the look and feel of all the report items. Labels, Table Headers, Grouping Footer and many other report elements can be formatted in a generic way.

***NOTE**

BIRT Styles are similar to CSS (Cascading Style Sheets) in some ways, but different in many other ways. For example, using predefined styles, you can set a style for *all* tables in the report. There is no predefined “table” style that can be created, like a blue “table” or a green “table.”

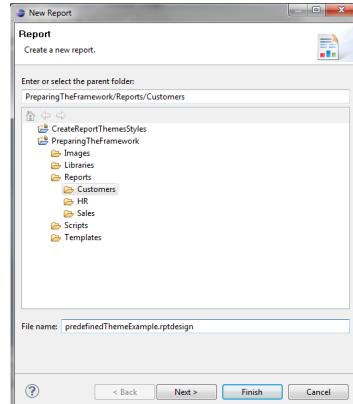
4.3.2 Testing the Theme

Let's create a new report and test out the Theme.

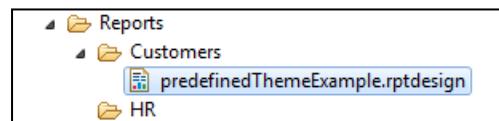
Right click on the “/Reports/Customers” folder and choose New → Report



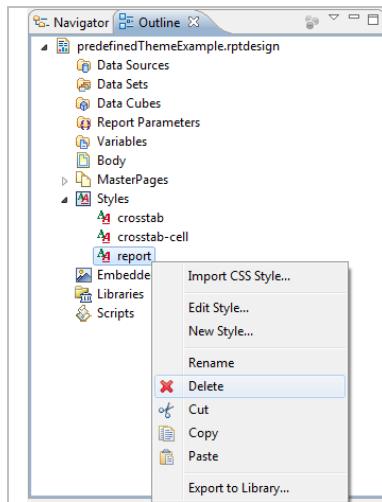
Name the report
“predefinedStyleExample.rptdesign”



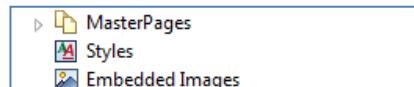
Click Finish



In the report, click on the “Outline” tab. Expand the “Styles” component. Delete the three default styles. Right click on the style and choose “Delete”.

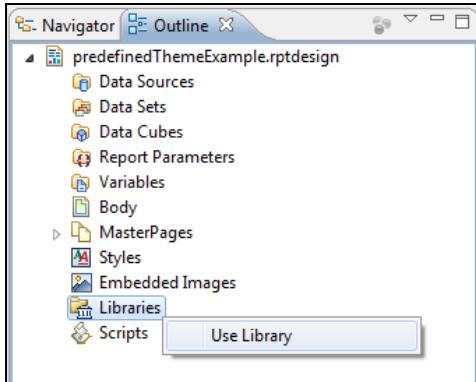


Result

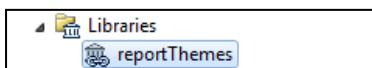
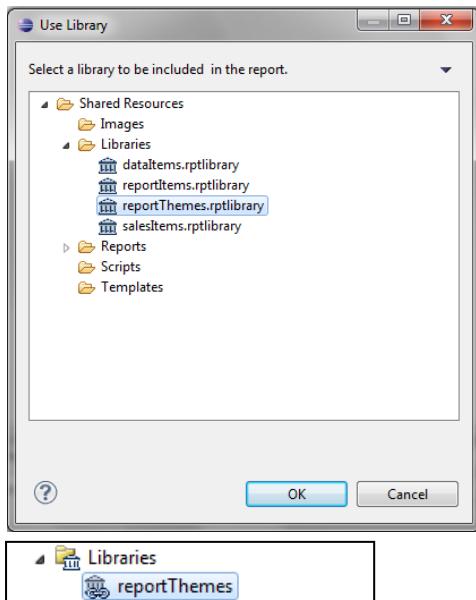


4.3.3 Link the reportThemes.rptlibrary to the report design

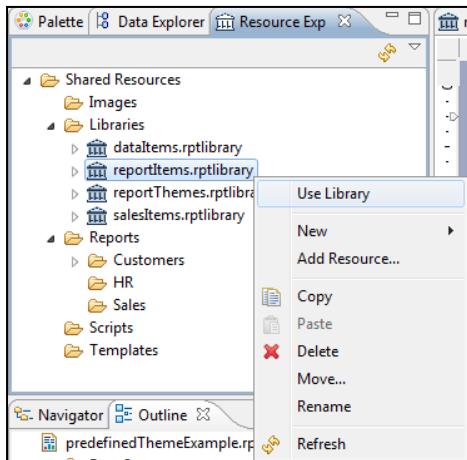
To use the “reportThemes” library in the report design, in the “Outline” tab, right click on the “Libraries” section. Choose “Use Library”.



Select “/Libraries/reportThemes.rptlibrary” from the list. Click “Ok”.

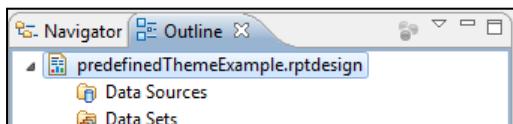


There is an alternative way to use a library in a report in the Report Designer. From the “Resource Explorer”, right click on the library and choose “Use Library”. This will link that library with the report that is currently open.

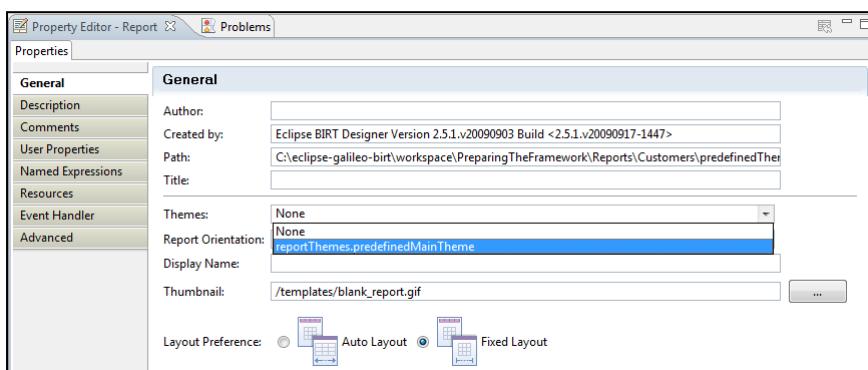


4.3.4 Using the Theme in the Report Design

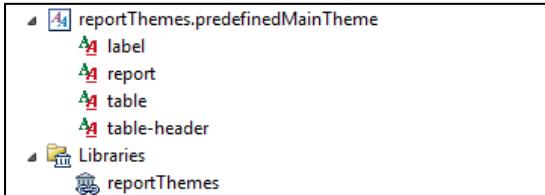
In the “Outline” tab, click on the report design top level section.



The “Property Editor” will display the “General” settings for the report. In the “Themes” drop down list, choose the “reportThemes.predefinedMainTheme” option.

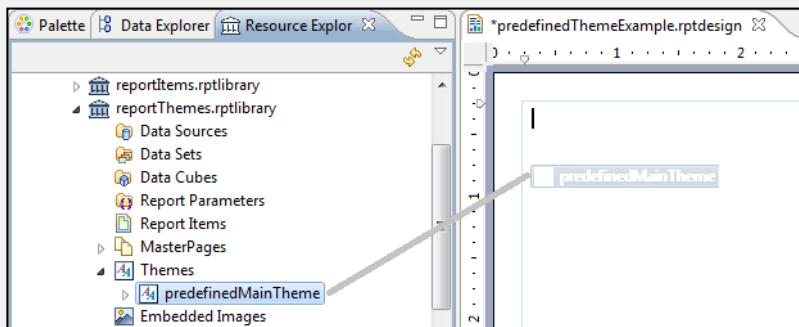


This will add the Theme to the Outline of the Report.



Save the report

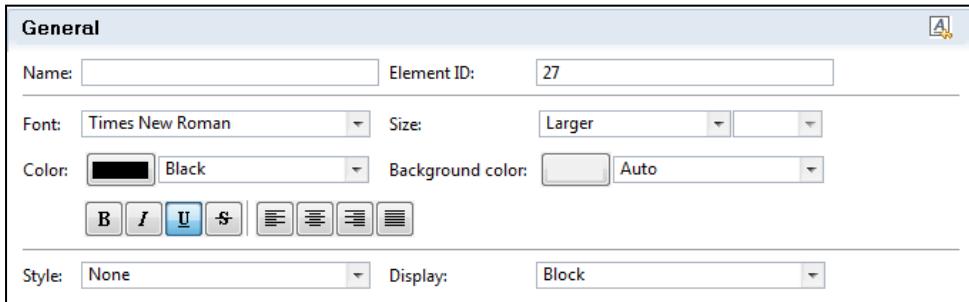
* An alternative to this method is from the “Resource Explorer”, expand the “reportThemes.rptlibrary”, choose the “predefinedMainTheme”, and drag and drop onto the open report design. This will automatically add the library to the report and set the report property to use the theme.



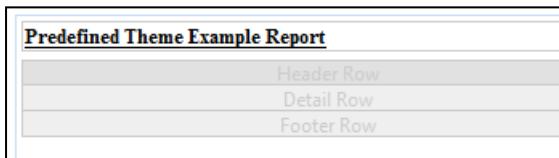
Add a label to the Report Design. Type the words “Predefined Theme Example Report”. Notice how the label retains the formatting of the “label” settings in the library. Any label used in the report will follow this setting.



Click on the label and look at the General Properties. They have been set to the settings for a “label” in the library. A developer can override these settings. However, the label no longer follows the library settings. If the library changes, the properties changed in the control will remain the same.

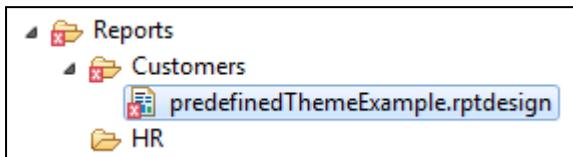


Add a table to the Report Design. Choose 2 columns and 1 detail row.



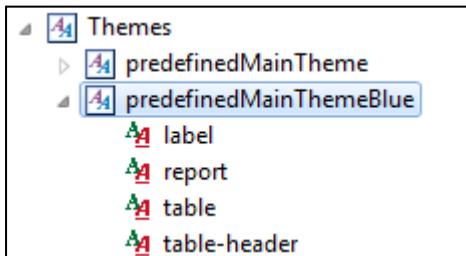
Notice how the table will automatically be formatted using the table style settings in the Theme.

*NOTE: You may notice a small red X in the Navigator next to the report design. This is because we have not created a data source or linked the table to a data set. This error can be ignored for now.



4.3.5 Switching Themes in a report

Open the reportThemes.rptlibrary file. In the “Outline” tab, create a copy of the “predefinedMainTheme”. Name it “predefinedMainThemeBlue”.

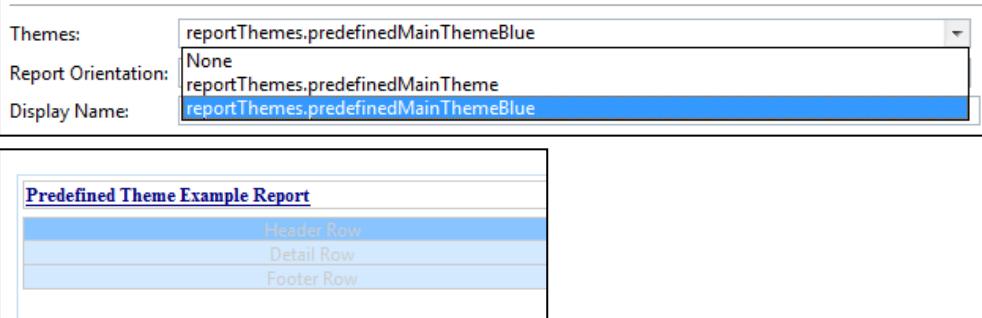


Edit each style in the theme to use the following settings:

Predefined Style	Settings
table	Background color: RGB(210,233,255)
table-header	Background color: RGB(136,196,255)
label	Font color: Navy
report	Background color: RGB(236,245,255)

Save the library

Open the “predefinedThemeExample.rptdesign”. Change the “Theme” property in the general properties of the report to use “predefinedMainThemeBlue”. If you do not see the option, confirm that your library was saved. Close your report design and reopen to refresh the library link.



The screenshot shows the "Report Properties" dialog box. In the "Themes" section, the dropdown menu is open, showing "reportThemes.predefinedMainThemeBlue" as the selected option. Below it, the "Display Name" field also contains "reportThemes.predefinedMainThemeBlue". At the bottom, there is a preview window titled "Predefined Theme Example Report" showing a table with three rows: "Header Row", "Detail Row", and "Footer Row", all colored according to the blue theme.

Notice that the colors have changed. The only change was to point the report to a different theme.

***TIP:** This change can also be done dynamically using scripting. Using the following Javascript, one can dynamically switch the Theme of the report at runtime. Scripting will be covered later in the book.

```
// Get Parameter Value  
var blnBlueTheme = reportContext.getParameterValue("pBlueTheme");  
  
logToDebugWindow("blnBlueTheme: " + blnBlueTheme);  
  
// If blue theme, then set theme name  
// If you are seeing an exception message in the Web Viewer,  
// add a try/catch statement around the setThemeName call  
if(blnBlueTheme == true) {  
    reportContext.getDesignHandle().setThemeName("reportThemes.predefinedMainThemeBlue");  
} else {  
    reportContext.getDesignHandle().setThemeName("reportThemes.predefinedMainTheme");  
}
```

***TIP:** In this example, I created a copy of a Theme. This can be useful when you have a completed “Main” theme and you want to create a variation, such as putting the colors in blue. However, once a copy is created, they are two separate items. When you make a change to one Theme, it is **not** reflected in the other one. The change will need to be made in both places. Be aware of this when creating many copies and variations. BIRT does not offer true inheritance in its libraries, especially when it comes to Themes and Styles. Currently, it is not possible to create “parent” themes/styles, and have “children” themes/styles inherit from them.

The problem with using Predefined Styles in your Theme is that is applies to every instance of that item in the report. If you have a complex report with multiple tables, labels and controls, it's impossible to cleanly assign different styles to each item. A developer can override the Theme with custom settings, however, that defeats the purpose of using a style library. Those changes are isolated in the report and cannot be re-used.

Predefined styles can be useful in very basic reports or reports designed for the web. If your reports are only going to have one table and a few labels, and similar formatting across all reports, it is okay to use predefined styles. These small reports can be useful for displaying in portals, dashboards and embedding on web pages. Dynamically switching between themes at runtime can be very useful for report personalization or report branding. However, for more complex reports, using only predefined styles can lock you in a corner and it will be messy to get out.

If you are creating report templates that will be used in Actuate's BIRT Report Studio (a web based report designer for business users), using predefined styles is acceptable and encouraged to make use of the full product. There is a feature to apply a theme to a whole report dynamically. This uses different themes to change the color scheme and formatting of the report. Business users like the easy way to change the look and feel of the report. However, if your BIRT Report Studio report design has more than one table, your theme will be applied to the whole report, not just individual tables.

For more information on creating styles specifically for BIRT Report Studio, refer to:

<http://www.birt-exchange.org/devshare/designing-birt-reports/1160-birt-studio-style-guide/>

4.4 Using Custom Styles in Your Theme

Creating custom styles in your theme will allow greater control on how to keep a consistent look and feel to your reports. Unfortunately, the BIRT Designer does not allow cascading styles or any form of style inheritance. It also does not provide the capability to create custom predefined styles. For example, if you want to create multiple “table” styles in your theme, like a green table and a blue table, you would not be able to. Using custom styles, you will have to flatten your style structure and create several different combinations of styles.

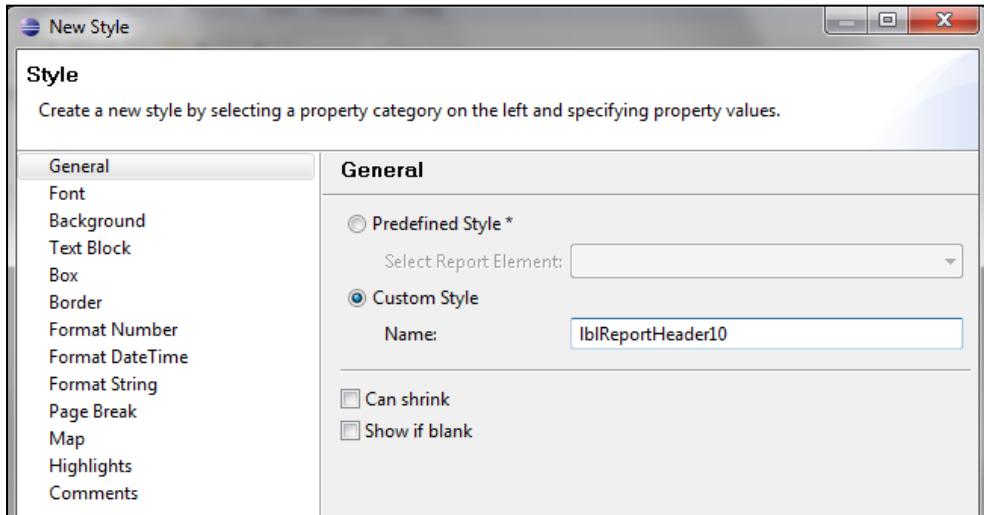
4.4.1 Creating custom styles in your theme

The following is an example using custom styles.

Open the “reportThemes.rptlibrary”

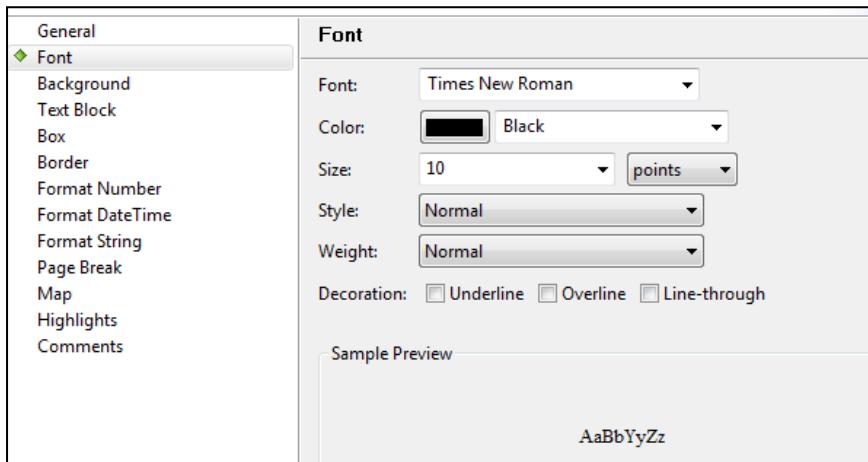
In the “Outline” tab, add a new theme named “customMainTheme”.

Right click on the theme and add a new style.



Name the style “lblTableHeader10”.

Click on the Font properties. Set the font size to 10 points. Set font to “Times New Roman”.



This will be used for any report header labels that should have font size 10. Notice the naming convention of the style. The type, purpose and property details of the style are in the name.

Using custom styles allows us to create a wider variety of styles to use. Instead of just one style setting for all labels, we can create several different kinds of styles to use.

Create the following custom styles in the Theme

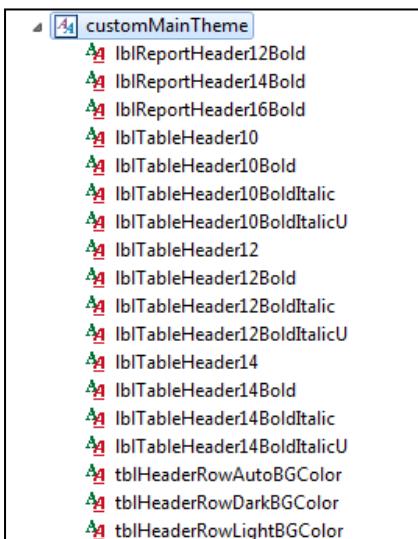
Custom Style Names	
Style Name	Settings
IblTableHeader10	Font size: 10 points
IblTableHeader12	Font size: 12 points
IblTableHeader14	Font size: 14 points
IblTableHeader10Bold	Font size: 10 points Font weight: bold
IblTableHeader12Bold	Font size: 12 points Font weight: bold
IblTableHeader14Bold	Font size: 14 points Font weight: bold
IblTableHeader10BoldItalic	Font size: 10 points Font weight: bold Font style: italic
IblTableHeader12BoldItalic	Font size: 12 points Font weight: bold Font style: italic
IblTableHeader14BoldItalic	Font size: 14 points Font weight: bold Font style: italic
IblTableHeader10BoldItalicU	Font size: 10 points Font weight: bold Font style: italic Font decoration: underline
IblTableHeader12BoldItalicU	Font size: 12 points Font weight: bold Font style: italic Font decoration: underline
IblTableHeader14BoldItalicU	Font size: 14 points Font weight: bold Font style: italic Font decoration: underline
tblHeaderRowAutoBGColor	Background Color: Auto
tblHeaderRowLightBGColor	Background Color: RGB(240,240,240)

tblHeaderRowDarkBGColor	Background Color: RGB(225,225,225)
tblDetailRowAutoBGColor	Background Color: Auto
tblDetailRowLightBGColor	Background Color: RGB(240,240,240)
tblDetailRowDarkBGColor	Background Color: RGB(225,225,225)
IblReportHeader12Bold	Font size: 12 points Font weight: bold
IblReportHeader14Bold	Font size: 14 points Font weight: bold
IblReportHeader16Bold	Font size: 16 points Font weight: bold

***Tip**

Some Developers use a different naming convention for styles that uses dashes. Ex: “table-detail-row” As mentioned earlier, which style is chosen doesn’t matter, as long as it’s consistent and all developers follow it.

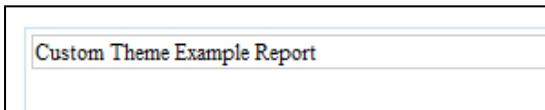
It's up to you how fine-grained you want to make your styles. Some developers create all possible styles up front. Others create basic ones, and then add styles as they need them. The goal is to define your styles so they can be re-used across report designs.



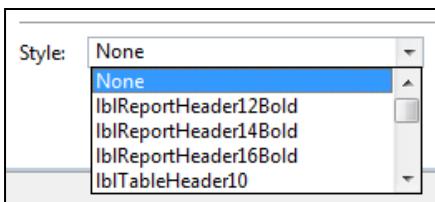
4.4.2 Using custom styles in your report

Do the following:

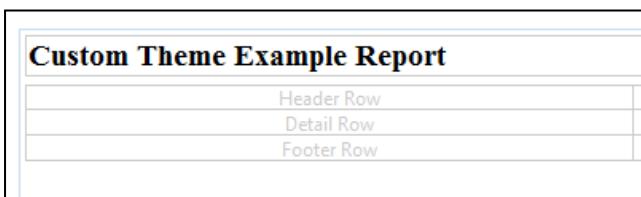
- Create a new report in the “/Reports/Customer” folder named “customThemeExample.rptdesign”.
- Delete the default styles in the report.
- Add a reference to the “reportThemes” library.
- In the general report properties window, under Themes, choose “reportThemes.customMainTheme”.
- Drag and drop a label onto the report. Enter the text “Custom Theme Example Report”. Notice that the control has no formatting.



Click on the label. In the General Properties window, click the arrow of the Styles options box.



Choose “IblReportHeader16Bold”. The label will automatically change to the format specified in the style.



Notice that the developer can override these formatting settings. Avoid doing this. It's better to rely on the defined styles in your theme. Any changes made in the library are automatically reflected in the report. However, if a report item has its own settings, those settings will override any style settings.

Drag and drop a table onto the report design. Choose 2 columns and 1 detail row. Notice the table has no formatting. Click the table, click the table row, and set the style to

“tblHeaderRowDarkBGColor”. Click the detail row and set the style to “tblDetailRowLightBGColor”. Do the same for the footer row.

The screenshot shows a report window with a title bar 'Custom Theme Example Report'. Below the title bar is a table with three rows. The first row is labeled 'Header Row', the second 'Detail Row', and the third 'Footer Row'. The 'Header Row' has a dark gray background color, while the 'Detail Row' and 'Footer Row' have a light gray background color. The table is enclosed in a black border.

Header Row
Detail Row
Footer Row

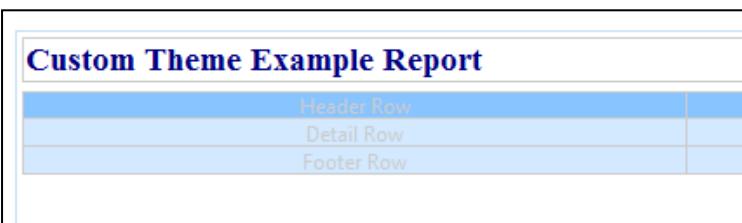
4.4.3 Switching Themes using Custom Styles

In the previous section with predefined styles, it was very easy to switch between themes for different color schemes. Depending how you implemented your themes and styles, this technique can also be used for custom styles.

If you have a requirement to switch themes, this is how you do it:

- In your “Main” Theme, limit your custom styles to generic styles such as bold, italic and font size. If you need to use colors, use a gray scale color scheme.
- In your report, all items should be assigned a style.
- Open the library containing your themes. Copy your Main theme and name it “customMainThemeBlue”.
- Open all existing styles in your theme and change as necessary. In this example, I will change all styles to incorporate the color blue. Save the library.
- Close and re-open the report design. In the General Report properties, choose the “reportThemes.customMainThemeBlue” Theme.
- All report items that have styles assigned to it will change to the new formatting defined in the new Theme.
- In order for this to work correctly, the libraries will need to have identical style names. If the new Theme is missing some style definitions, the report design will have errors and may not run correctly.

<u>Main Theme</u>	<u>Blue Theme</u>
<pre> A customMainTheme A IblReportHeader12Bold A IblReportHeader14Bold A IblReportHeader16Bold A IblTableHeader10 A IblTableHeader10Bold A IblTableHeader10BoldItalic A IblTableHeader10BoldItalicU A IblTableHeader12 A IblTableHeader12Bold A IblTableHeader12BoldItalic A IblTableHeader12BoldItalicU A IblTableHeader14 A IblTableHeader14Bold A IblTableHeader14BoldItalic A IblTableHeader14BoldItalicU A tblDetailRowAutoBGColor A tblDetailRowDarkBGColor A tblDetailRowLightBGColor A tblHeaderRowAutoBGColor A tblHeaderRowDarkBGColor A tblHeaderRowLightBGColor </pre>	<pre> A customMainThemeBlue A IblReportHeader12Bold A IblReportHeader14Bold A IblReportHeader16Bold A IblTableHeader10 A IblTableHeader10Bold A IblTableHeader10BoldItalic A IblTableHeader10BoldItalicU A IblTableHeader12 A IblTableHeader12Bold A IblTableHeader12BoldItalic A IblTableHeader12BoldItalicU A IblTableHeader14 A IblTableHeader14Bold A IblTableHeader14BoldItalic A IblTableHeader14BoldItalicU A tblDetailRowAutoBGColor A tblDetailRowDarkBGColor A tblDetailRowLightBGColor A tblHeaderRowAutoBGColor A tblHeaderRowDarkBGColor A tblHeaderRowLightBGColor </pre>



If you do not have a requirement to switch themes, and don't think you will in the future, then you can put all sorts of style combinations in your one "Main" theme, including color variations.

▲ Themes
▷ A4 predefinedMainTheme
▷ A4 predefinedMainThemeBlue
▷ A4 customMainTheme
▷ A4 customMainThemeBlue
▷ A4 customMainThemeAll
A4 IblReportHeader12Bold
A4 IblReportHeader14Bold
A4 IblReportHeader16Bold
A4 IblReportHeader16BoldNavy
A4 IblTableHeader10
A4 IblTableHeader10Bold
A4 IblTableHeader10BoldItalic
A4 IblTableHeader10BoldItalicU
A4 IblTableHeader12
A4 IblTableHeader12Bold
A4 IblTableHeader12BoldItalic
A4 IblTableHeader12BoldItalicU
A4 IblTableHeader14
A4 IblTableHeader14Bold
A4 IblTableHeader14BoldItalic
A4 IblTableHeader14BoldItalicU
A4 tblDetailRowAutoBGColor
A4 tblDetailRowDarkBGColor
A4 tblDetailRowLightBGColor
A4 tblDetailRowLightBGColorBlue
A4 tblHeaderRowAutoBGColor
A4 tblHeaderRowDarkBGColor
A4 tblHeaderRowDarkBGColorBlue
A4 tblHeaderRowLightBGColor

Custom Theme Example Report

Header Row
Detail Row
Footer Row

Custom Theme Example Blue Report

Header Row
Detail Row
Footer Row

Using generic named styles such as “tblHeaderRowBGLight” or “IblReportHeader16BoldItalic” allows you to keep the generic formatting in place, but gives you the freedom to switch themes to reflect colors and other formatting changes. How generic you make your styles depends on your reporting requirements. If you think theme-switching will be needed, you will want to keep generic names.

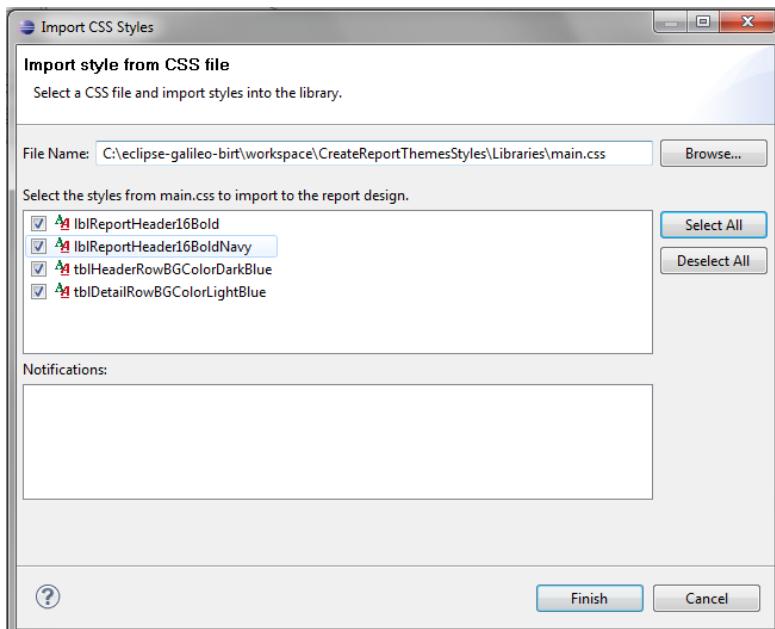
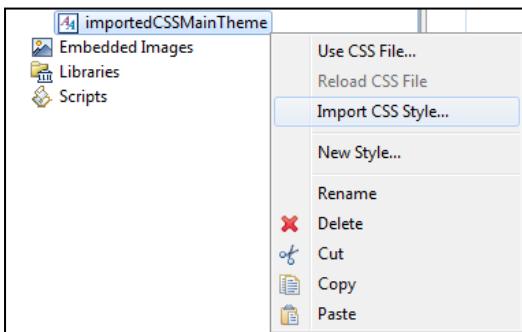
For example, if you have a need for a blue-colored table in the report to highlight monthly sales, instead of naming the associated styles “tblHeaderRowBGBLue”, you should name the style “tblHeaderRowBGSales” and assign it a blue color. Down the road, if the Sales table needs to change to green, or if your application allows the user to customize the formatting of the report and wants to change it to green, the report just needs to switch to a different Theme instead of the developer manually updating each report item to use the green version of the style.

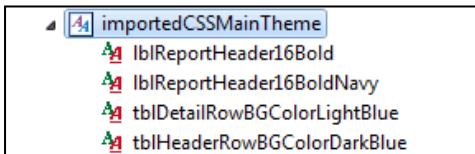
If you don't have a requirement for personalization or theme-switching, then creating your whole Theme to be generic might be overkill. In this case, you can define all necessary styles in one Theme and just use that.

4.5 Importing a CSS file into your theme

The BIRT Report Designer allows you to leverage existing CSS files. If you already have style sheets used in your web application, you can import them into a Theme for use in the report.

Beware! This does not maintain a live link to the css file. It will do a one-time import of the styles into the Theme.



***Tip**

You can import more than one CSS file. It will import the styles and add them to the Theme. If there are styles in the Theme that already have the same name, BIRT will still import them, and add a version number after the name.

This is best suited if you have an old CSS file that you don't use anymore, and want to convert to a Theme. Or if you want to use an existing CSS file as a starting point for a Theme. Once you have imported the styles, you can edit and manipulate them without regard for the external file. If you are going to use an external CSS file, it's best to link to it.

4.6 Linking to an external CSS file

If you already have style sheets used in your web application, you can link to them for use in the report design. The link is a live link, and any changes in the CSS file are picked up by the report.

Using an external CSS file

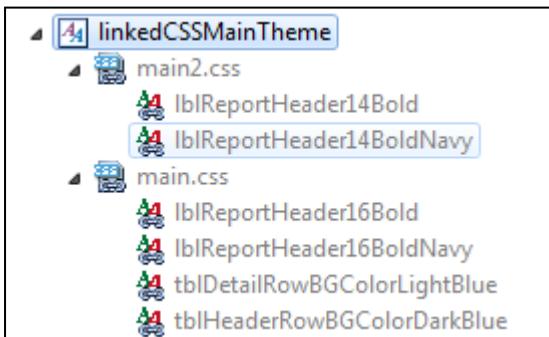
The screenshot shows the BIRT interface with a tree view on the left. A node named 'linkedCSSMainTheme' is selected and expanded, showing options like 'Embedded Images', 'Libraries', and 'Scripts'. A context menu is open over this node, with the 'Use CSS File...' option highlighted. To the right, a 'Use CSS' dialog box is displayed. The 'File Name' field contains 'Libraries/main.css'. The 'Theme' dropdown is set to 'linkedCSSMainTheme'. Below these fields, there is a checkbox for 'Include CSS file at view time (Apply only to HTML format)' and a 'View time location of the CSS file' button. A 'URI:' input field is present with the placeholder 'e.g. http://mydomain/test.css, myfolder/test/css'. A scrollable list titled 'Available styles from Libraries/main.css' shows four items: 'IblReportHeader16Bold', 'IblReportHeader16BoldNavy', 'tblHeaderRowBColorDarkBlue', and 'tblDetailRowBColorLightBlue'. At the bottom of the dialog are 'OK' and 'Cancel' buttons.

***Tip**

The “Include CSS option at view time” gives the developer the option of including an HTML link to the CSS file in the final HTML generated. The URL to the view time location must be supplied.

***Tip**

Multiple CSS files can be linked to the same Theme. BIRT does not allow the same CSS file to be included twice. However, style names with the same names, even in different CSS files, are accepted. BIRT will use whichever one it finds last, so be careful not to have style name interference. You can also add new styles to the theme without interfering with the linked CSS.



Notice the CSS file and styles are grayed out. This means that they are read-only, and cannot be changed through the BIRT Designer. Any styles changes must be made in the CSS file. Changes in the CSS file will be picked up by the Report.

The BIRT Report Design will parse the CSS file and internalize the styles in the CSS file, even though the CSS file is external. Even when you choose to provide a URL link at view time, it still will do this. The BIRT Engine needs to know what the styles are so it can reflect the styles in the GUI.

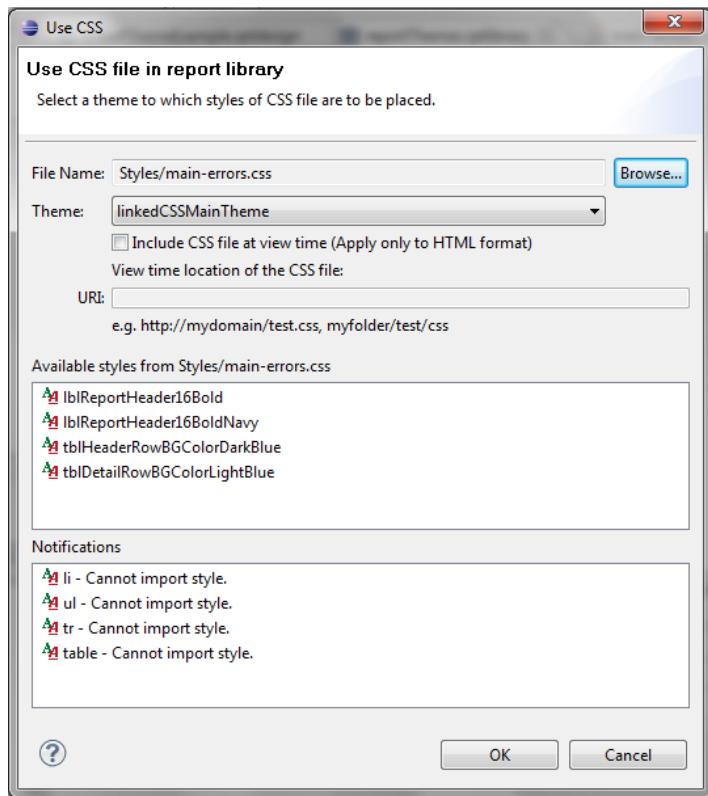
4.6.1 A warning about using external CSS files

BIRT supports basic CSS definitions, however, *it is not fully CSS 2.0 compliant*. When you import or link to a CSS file, BIRT will list what styles it can understand. It will also give error messages on the styles it could not understand. Only basic CSS classes will be accepted. Among the features of CSS 2.0 that BIRT does not support:

- Selectors or any kind of pattern matching
- Inheritance (I know this defeats the purpose of Cascading Style Sheets)
- Styles on HTML elements such as DIV, P, TABLE, TR, etc.

BODY, LI, TABLE or any other generic styles will be brought in as a new custom style with that "name", such as "A", "LI", and it won't be brought as expected.

The best thing to do is to try linking your CSS file to the Report Theme and see what happens. The Designer will tell you which styles it could read and which ones it couldn't understand.



4.6.2 Why does BIRT handle Styles this way?

Some developers may ask why BIRT does not support the full CSS 2.0 specification. There are several reasons for this.

- 1) It is complex and will take a lot of time and effort to do this. And is it worth spending all this time and effort to support the full spec?
- 2) The CSS 2.0 spec is geared towards viewing web pages in browsers. Many of the CSS 2.0 specifications are not necessary in other formats, such as PDF, Word, CSV and Excel.
- 3) Many of the advanced features of CSS 2.0 are not needed by most report developers.

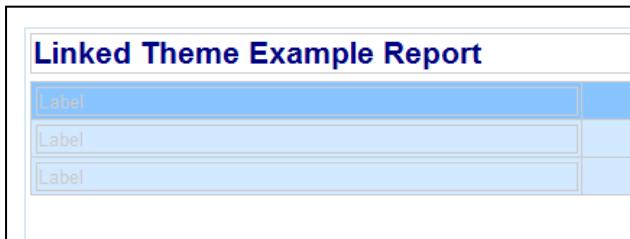
4.6.3 Why does BIRT internalize the Styles, even when the CSS is external?

There are a couple reasons why BIRT internalizes the Styles.

- 1) So the Designer can display to the developer how the style formatting is going to look. If it doesn't understand the style, then it cannot display to the user what the report item will look like.
- 2) BIRT supports many different output formats. It needs to internalize the styles so that it can keep a consistent look across output formats.
- 3) Styles are kept internally and stored in the "rptdocument" for archiving purposes. For example, say a user ran a report and your custom application saved it (in rptdocument format). Six months later, they wanted to view the report again and print it out. The report should look just as it did 6 months ago. New style sheets could potentially change the report and even some of the data. Internalizing the style prevents this. (See Chapter 19 *Preparing for Different Report Output Types* for more information about "rptdocument" format.)

4.6.4 Does BIRT maintain the names of the original style names in the generated HTML?

Unfortunately, BIRT does not. The styles are internalized and assigned random names in the generated HTML. In the "linkedThemeExample" report, we used three styles: "lblReportHeader14BoldNavy", "tblHeaderRowBGColorDarkBlue", and "tblDetailRowBGColorLightBlue". In the Designer, we associated the Report Header item with this style. When we run the report, what does the generated HTML look like? Does it create a style named "lblReportHeader14BoldNavy"?

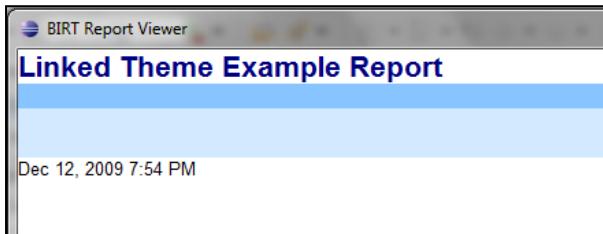


Style: **IblReportHeader14BoldNavy**

Style: **tblHeaderRowBGColorDarkBlue**

Style: **tblDetailRowBGColorLightBlue**

To see the generated HTML, select “View Report as HTML” option. Clicking Preview or Web Viewer does not display the HTML. It uses AJAX to pull the HTML and display it at runtime.



Right-click and choose “View Source”.

Notice the <style> tag. This defines the styles for the web page. Notice the style names. No sign of “IblReportHeader14BoldNavy”, “tblHeaderRowBGColorDarkBlue”, or “tblDetailRowBGColorLightBlue”.

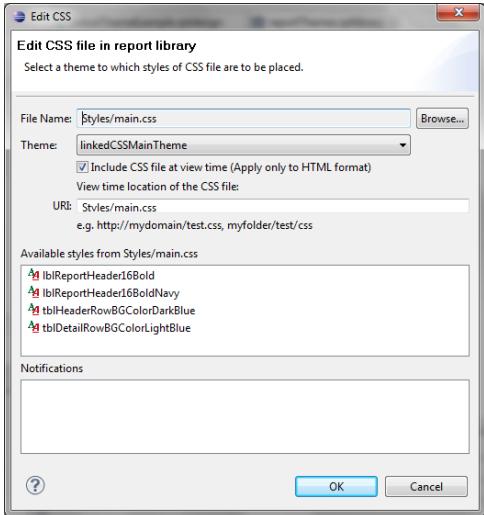
```
<style type="text/css">
    .style_2 { background-color: rgb(136, 196, 255);}
    .style_3 { background-color: rgb(210, 233, 255);}
    .style_0 { font-family: sans-serif; font-style: normal; font-variant: normal; font-weight: normal; font-size: 10pt; color: black; text-indent: 0em; letter-spacing: normal; word-spacing: normal; text-transform: none; white-space: normal; line-height: normal;}
        .style_1 { font-weight: bold; font-size: 16pt; color: navy;}
</style>
```

Look at the HTML code for the text “Linked Theme Example Report”:

```
<div class="style_1" id="AUTOGENBOOKMARK_1">Linked Theme Example Report</div>
```

The original style name is not carried down to the generated HTML.

4.6.5 What happens if we enable BIRT to include the CSS file at view time?



The generated HTML will include a reference to the internal style, plus the external style name.

```
<div class="style_1 lblReportHeader16BoldNavy" id="AUTOGENBOOKMARK_1">Linked Theme Example Report</div>
```

This option allows a little more control over how the report looks using additional style sheets. Most developers will not be concerned about the HTML being generated. Those that are will need to dig deeper in the source code to find solutions.

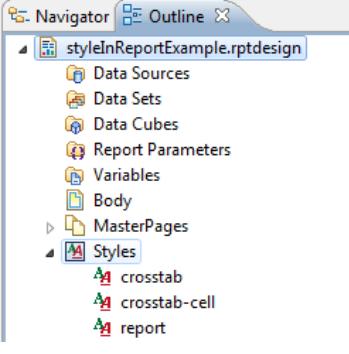
4.7 How to Proceed with Styles?

To successfully deal with Styles, you have to conform to how BIRT handles them internally. If you have a complex CSS file that BIRT won't accept, there are three solutions.

- 1) Create a subset of your CSS file that is “flattened” and readable by BIRT. Use this as your “reporting” CSS.
- 2) Use a “BIRT-friendly” CSS for the Report Designer. Use a more complex CSS file when displaying the report over the web. This complex CSS will override the styles set in the report. This will require the application to include the additional CSS files at view time. However, only the “BIRT-Friendly” styles will be used when viewing in other formats such as PDF.
- 3) Modify the Emitters to output the styles you want. The HTML, PDF and other emitters for BIRT are open source. You are free to modify the code, recompile it and use it as part of your application.

4.8 Creating styles directly in your report design

The BIRT Designer gives the developer the option of creating styles directly in the Report Design. When a new report is created, three styles are part of the report design by default.



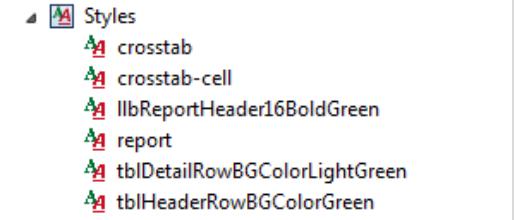
crosstab style – This style applies to all crosstabs used in a report. It specifies the dark-gray border line of the crosstab.

crosstab-cell – This style applies to all crosstab cells used in a report. It specifies the dark-gray border line of the crosstab cells.

report – This style sets the default font of the report to “Sans Serif, size 10 and black”.

It is recommended that these styles either be deleted or be moved into a library.

New styles can be added directly into the report design by right-clicking on the “Styles” section and choosing “New Style ...”. These styles are specific just for the report.



Style in Report Example	
Header Row	
Detail Row	
Footer Row	

*Tip

Embedding styles directly in the report design can be useful for individual test or demo reports. However, in the context of a BIRT project, it is discouraged. New Styles should be added to a Theme in a shared Library.

4.9 Le BIRT Expert's Recommended Approach

Le BIRT Expert's recommended approach is to use a combination of styles to create a mixed Theme. A few guidelines to follow:

- The mixed Theme should have limited use of the predefined Styles. Their use should be restricted to creating basic default settings for some of the report items.
 - Use the “report” style to define a font and other basic settings as a default for your reports.
 - Use the default “crosstab” and “crosstab-cell” styles, or create your own. This will act as default style for your crosstabs
 - Use other styles such as “page”, “table” and “grid” to define basic default settings.
 - An exception to this guideline would be if you are only creating small, simple reports used in Dashboards, Portals, embedding in web pages and for use in Actuate’s BIRT Studio, then the Theme should be heavy in predefined style use.
 - Another example would be if you know for sure that all tables are going to have the same formatting, then creating a predefined style would be useful.
- If you have a set of style sheets (CSS files) that you are already using for your web application or have a CSS guru as part of your team, link your Theme to an external CSS file. Just be aware of the limitations when using CSS files in BIRT. If you have a complex CSS structure, create a flattened version specifically used by BIRT. However, if you do not have an existing CSS file or a CSS guru available, it’s not worth the effort to maintain an external CSS “just because”.
- Make liberal use of custom styles in your Theme. Create a good set of style combinations up front, and then add to the Theme as you continue to do Report Development. There is nothing wrong with creating a set of BIRT Styles specifically used by the Report Designs. After all, that’s what it is there for. If you are using an external CSS file, you can still add custom styles to the Theme.
 - If you plan to do Theme switching, plan your styles and themes accordingly.
- Avoid the use of embedded styles in the Report Design.

Chapter 5

Creating and Using Report Items for Common Use in Libraries

In this Chapter:

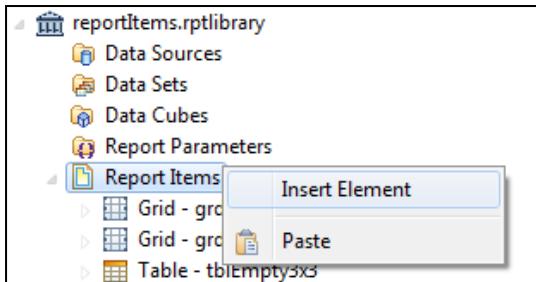
- Adding a Report Item to a Library
 - What Every Library Should Start With
 - Limitations of BIRT Libraries
-

Libraries provide a way to share common report items. This section offers recommendations on how to do this and covers the advantages and limitations of such an approach. BIRT Libraries do not offer true inheritance, and are not as extendable as some professional reporting tools. The developer may quickly come across the limitations of using libraries. This does not mean they should not be used. They do have their place and can be leveraged to create and share re-usable report items. Using libraries efficiently is a large part of developing a framework for your BIRT Reports. However, the capabilities may not meet the developer's initial expectations.

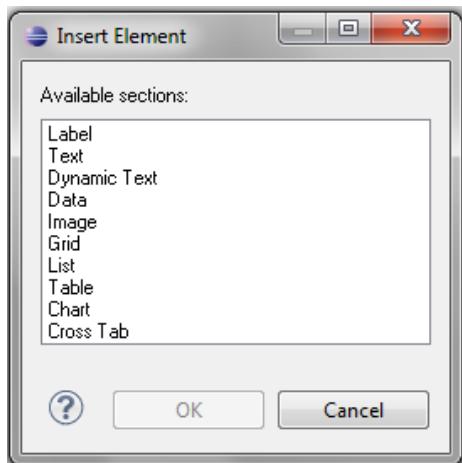
[5.1 Adding a report item to a library](#)

There are three ways to add a report item to a library.

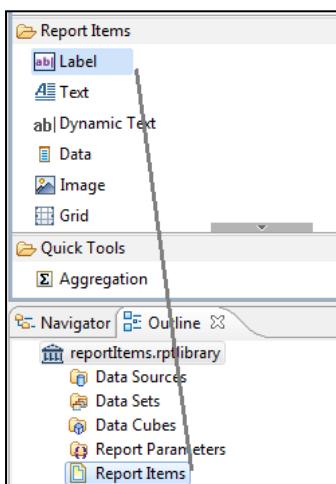
The first way is to open the library and right click on the “Report Items” section. Choose “Insert Element”



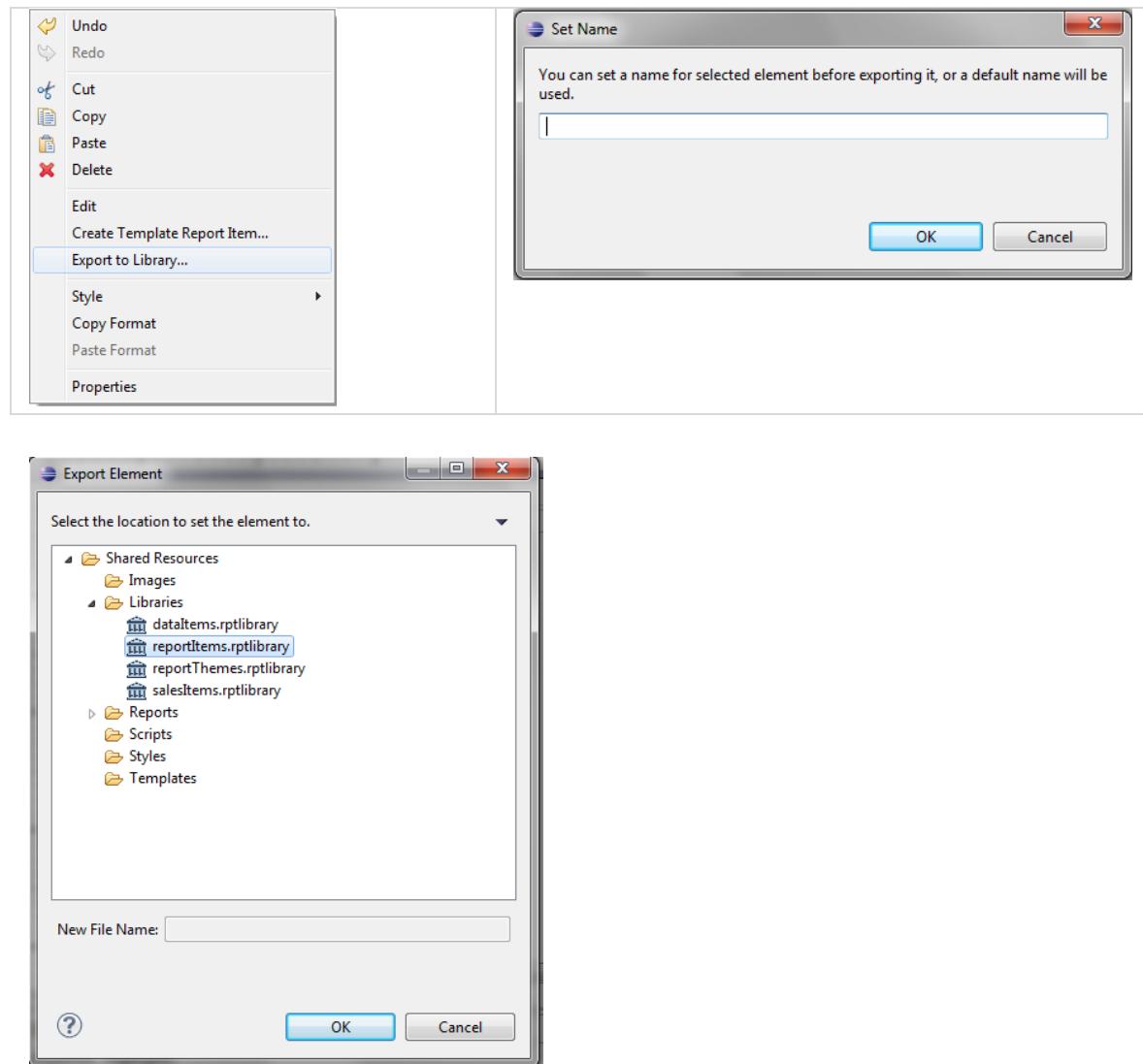
Choose what type of Element you want to insert. Click Ok.



The second way is to drag and drop from the Report Item Palette straight to the “Report Items” section.

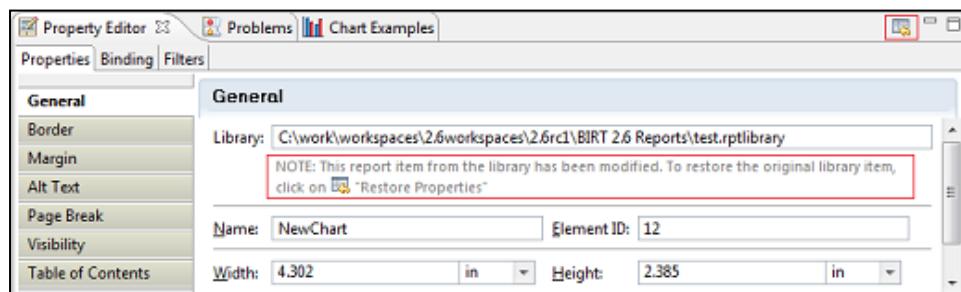


The third way is to right click on the Report Item you want to add to the library. Choose "Export to Library ...". Name the item. Choose which library to add it to.



5.1.1 Modifying an Item From a Library

Once you have added a Report Item from the Library to the Report Design, BIRT allows you to modify certain properties of the item. This doesn't affect the actual Item in the Library, it only is reflected in the Report Design it's used in. Click the little icon in the top right part of the Property Editor to restore the Item back to its original library item. In BIRT 2.6, if an Item has been changed, you will see the following message in the Property Editor.



5.2 Establishing your initial Common Report Items Library – What every library should start with

There are a few properties to be set and several report items that should be in your initial Report Items library. These include:

- Setting the Library to use the Styles Library
- Setting the default Theme
- Common Master Pages
- Common Report Header
- Common Report Footer

5.2.1 Setting the Library to use the Styles Library

Set the library to use the Styles library “reportThemes.rptlibrary”. This makes the library aware of the Styles we created already.

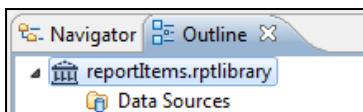
Right click on the “Libraries” section and choose “Use Library ...”. Choose the “reportThemes.rptlibrary”. You will see the “reportThemes” library has been added to the “Libraries” section. The lock on the icon indicates that the item came from a library.



5.2.2 Setting the Default Theme

Set the default Theme to the Main Theme containing all of your styles.

Click on the root “reportItems.rptlibrary” section



On the General Properties window, change the Theme to the “reportThemes.mainReportTheme”.

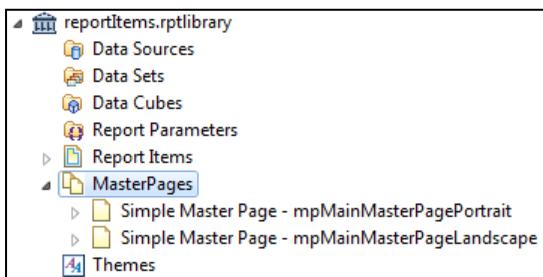
Themes: reportThemes.mainReportTheme

5.2.3 Common Master Pages

A common requirement is to create a common page layout, report header and report footer. We will first create a common Master Page, Report Header and Footer in the “reportItems” library. We’ll then use the report items in a new report.

Master Page

- Open the “reportItems.rptlibrary” by double-clicking and click the “Outline” tab.
- Expand the “MasterPages”.
- Rename the existing Master Page by right clicking and choosing “Rename”. Rename it to “mpMainMasterPagePortrait”.
- This Master Page will serve as our default Master Page.
- Right click on the MasterPages section and choose “Insert Item”.
- Rename the Master Page to “mpMainMasterPageLandscape”.
- Change the settings of the Master Page to landscape settings.



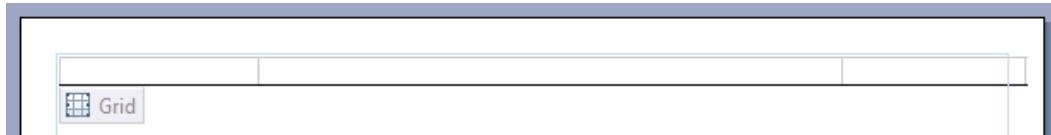
Depending on your requirements, you may have the need for several different kinds of “Master Pages”. This is a way to define the page size, page margins, borders and other settings for your reports. Some reports might be Portrait, others Landscape, and still others may require page size A4. Defining multiple Master Pages in the library will allow you to re-use them as you develop new reports.

5.2.4 Common Report Header

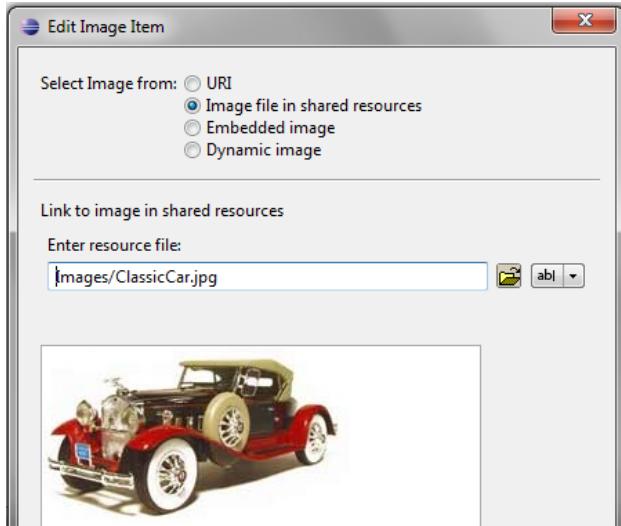
A common requirement is to create a Report header that is shown on each report. Use the Grid to create a holding place for other report items. Create the grid in the library so it can be re-used across Master Pages.

1. Insert a Grid Control (1 row, 3 cols) *into the Library*. Set the width of the 1st and 3rd columns to 20% and the middle column to 60%. Set the width of the whole grid to 100%. Put a border on the bottom line of the grid.

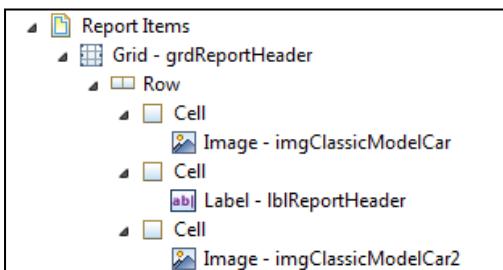


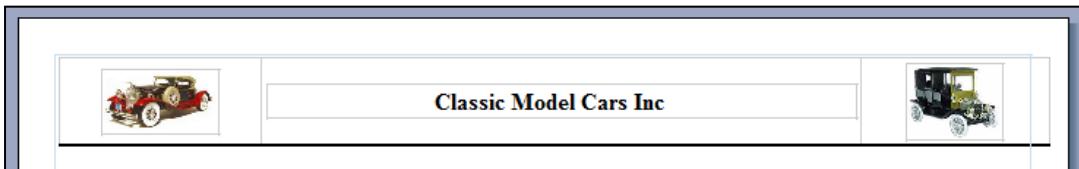


2. Insert an Image Control in the left column of the grid. Choose the Classic Car image. Set dimensions: width: 85px, height: 45px



3. Insert a Label into the middle column of the grid. Insert text "Classic Model Cars Inc".
o Set the style to be "lblReportHeader14Bold"
4. Insert the second image into the right column of the grid. Set dimensions: width: 70px, height: 55px
5. Rename the components to have description names.



***Tip**

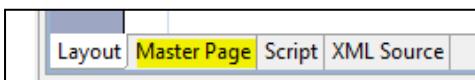
Add the Grid report item to the library first, instead of dragging and dropping them directly to the Report Header. If added to the Report Header first, they remain specific to the Report Header. This is fine for only one Master Page. However, if you have multiple Master Pages, it's better to add the item to the library first so they can be re-used.

* It is not necessary to name every report item in the report. However, it is beneficial to rename the re-usable report items so other developers know what they are.

* When using a border on a control, the default thickness is medium. Change to thin for a thin line. Sometimes, the BIRT Designer does not pick up the thin width setting. If you notice the line is still thicker than what you want, open the XML and change the thickness of the line to "1px". This means the line will be 1 pixel thin, which is the thinnest you can get.

Set the Report Header on the Master Page

Open the Master Page "mpMainMasterPagePortrait". Click on the "Master Page" tab.



The report header is defined by the blue dotted line on the Master Page. This area's dimension is a property on the master page.



Drag and drop the Report Header Grid from the library onto the area defined by the blue dotted line. This item is now on the Report Header. Rename the grid item. The grid item remains linked to the library.

The screenshot shows the BIRT Report Designer interface. At the top, there is a report preview window displaying a header section with two images of classic cars and a central label "Classic Model Cars Inc". Below this is a library tree on the left side of the screen. The tree structure is as follows:

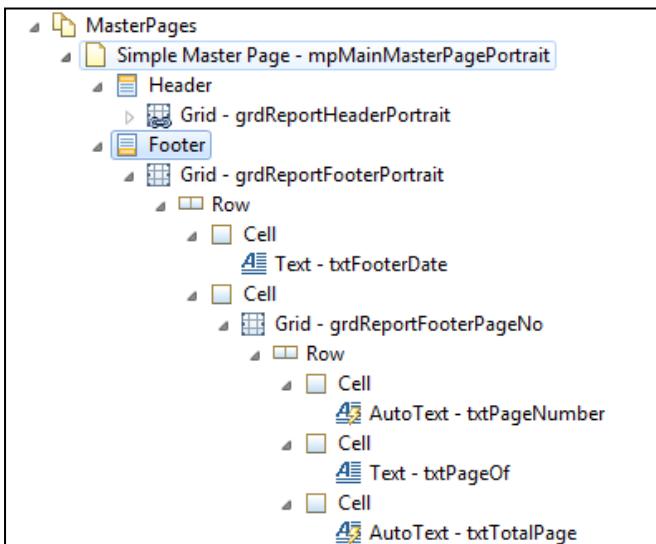
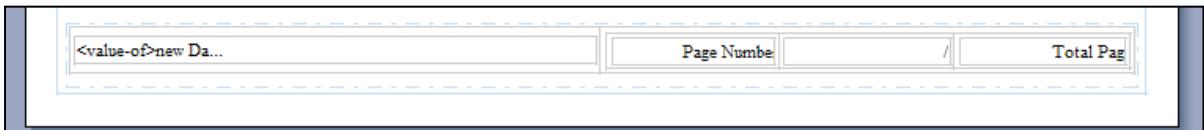
- Report Items
 - Grid - grdReportHeader
- MasterPages
 - Simple Master Page - mpMainMasterPagePortrait
 - Header
 - Grid - grdReportHeaderPortrait
 - Row
 - Cell
 - Image - imgClassicModelCar1
 - Cell
 - Label - lblReportHeader1
 - Cell
 - Image - imgClassicModelCar21
 - Footer
 - Simple Master Page - mpMainMasterPageLandscape

Do the same for the Landscape Master Page and other Master Pages you might have created.

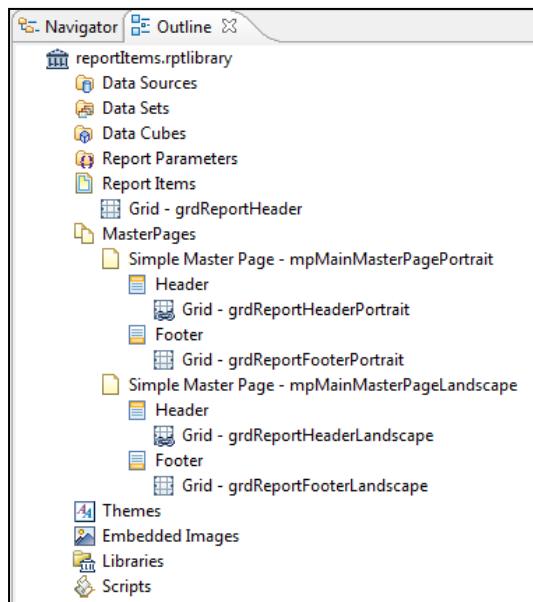
5.2.5 Common Report Footer

The BIRT Report Designer has a set of very convenient “AutoText” report items specifically for use in the Footer of the Master Page. Unfortunately, they cannot be used outside the Footer section or stored into a library by themselves. As a result, we cannot create a re-usable grid containing Report Footer AutoText items. However, what we can do, is create the footer grid directly onto the Master Page. This is the best we can do for creating a re-usable Report Footer.

Open each Master Page item in the library, switch to the Master Page tab, and drag and drop from the BIRT Palette, a grid, “Page n of m” and “Date” AutoText items onto the Footer section. Apply styles as necessary to the items.



Do the same for all Master Pages in your library. Use the “copy” command to copy the Footer grid to other Master Pages. Remember, this creates a copy that is not re-usable.



We will be adding more Report Items to our Library later, but for now, this gives a good base to use going forward.

***TIP**

Notice the value of the Date control that was dropped into the footer:

```
<VALUE-OF>new Date();</VALUE-OF>
```

This tag can be used to output the value of a Javascript expression. It will evaluate the expression at runtime.

In BIRT 2.6, there is a new tag called <VIEWTIME-VALUE-OF>. This tag will also output the value of a Javascript expression, except it will do it at view time instead of runtime.

5.3 Limitations of BIRT Libraries

There are a couple of limitations of the BIRT libraries that developers should be aware of.

5.3.1 Designer Quirks when using Libraries

If you open a library to make changes, and a report that is using that library is already open in the Designer, you may encounter some error messages. BIRT will first warn you that the library has changed and will ask you if you want to refresh the report. There is often an error when refreshing. Just close the report design and re-open and everything should be fine.

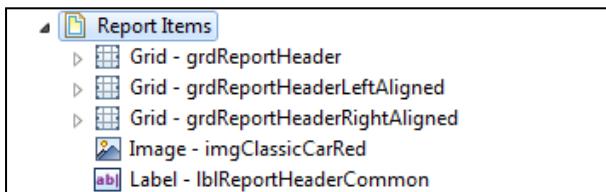
***TIP**

When working on a library, it's best to close all other report designs first.

5.3.2 Unable to re-use report items in the same library

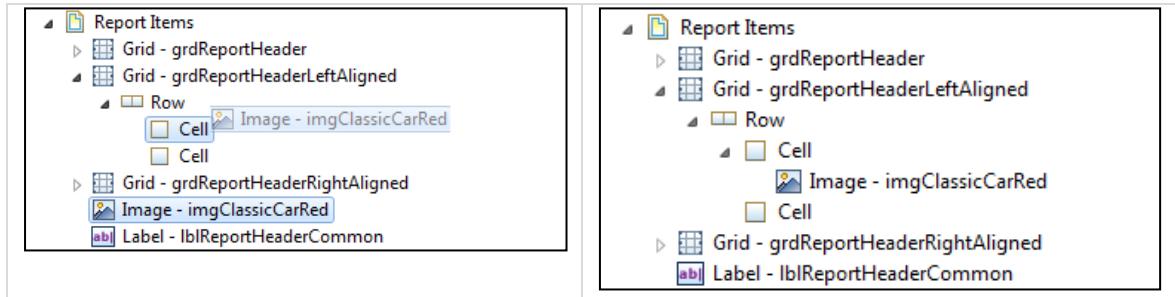
A developer can easily add report items to a library and re-use them in Report Designs, Report Templates and even other libraries. However, the items cannot be re-used in the same library. If you try to use a report item in the same library, it will just move the item to where you wanted to use it.

For example, say we want to create a common image and label item in the library and re-use it in two different grids, which were also located in the same library.



If I click on the grid "grdReportHeaderLeftAligned", the grid will appear on the main window. If I click on the image "imgClassicCarRed" to drag it into the grid, the tool will switch focus to the image instead.

In the Outline tab, if I drag the image to the grid, it will move the image to the grid. If I drag and drop the image to the other grid, it will move the image there.



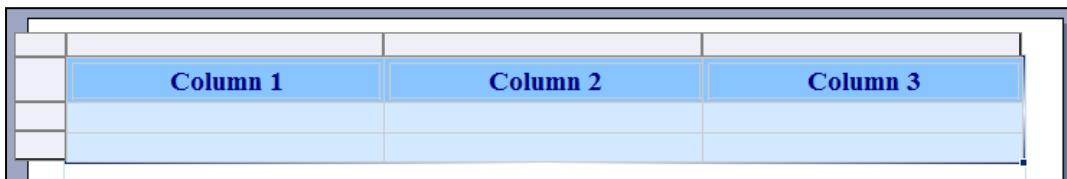
There is no way to create a reference from one report item to another report item in the same library. If you need to do this, you'll need to move one of the items to a different library. Is this really that big of a deal? I don't think so. It might be a little bit of a pain, but definitely able to work around. Duplicating a few of the same items a couple times here and there in the library is easier than having to put it in a different library just for the sake of re-use. Since the item is in a library, that in itself provides a single point of change for the future. If you find yourself creating and re-using several of the same items over and over again in the same library, then it would be worth putting those in a separate library.

5.3.3 Can inherit but not extend

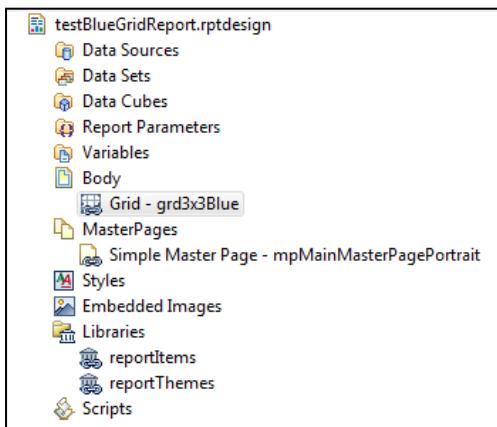
When using a report item from a library in a Report Design, Report Template or another library, BIRT enables you to override the property settings. However, you will not be able to change the structure of the report item.

For example, say you have a perfectly formatted 3x3 grid that you wanted to re-use on your reports. However, some reports might only need 2 columns or an additional label in one of the grid cells. This is considered changing the structure and the BIRT Designer will not let you do this.

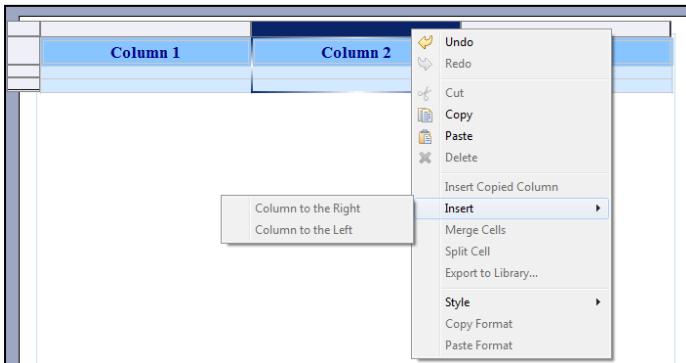
Formatting Grid item in the Library:



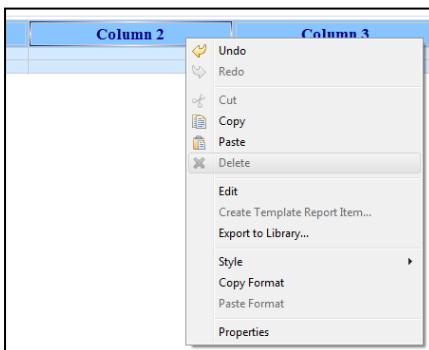
Using the grid in a report



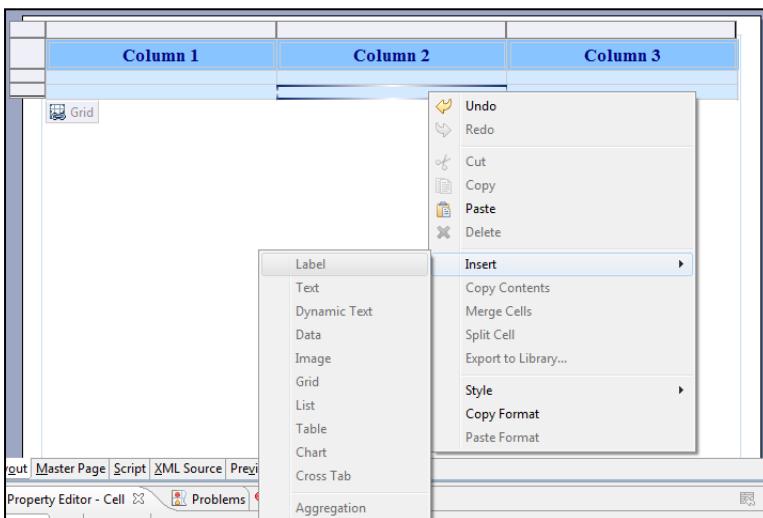
Won't be able to delete or insert a column or row.



Cannot delete an existing label

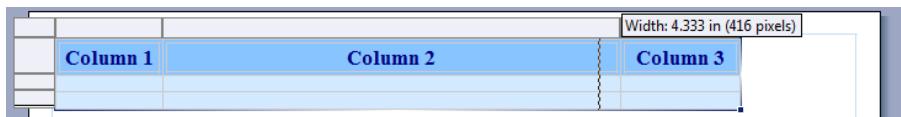


Cannot add a label to a grid cell



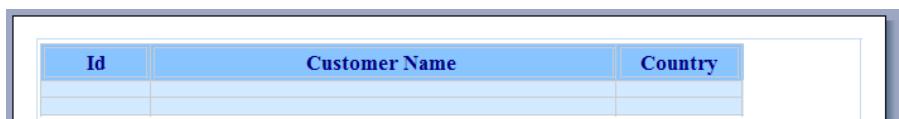
5.3.4 Can change the grid properties

By clicking on the grids, the properties are available to override. The size of the grid can change as well. This doesn't change the original grid in the library, but allows you to customize the item in the report design. Any properties customized in the report design will remain as is, even if those same properties change in the library.



5.3.5 Can change the label properties

Each individual label properties can be changed as well.



The limitation of not being able to change the structure or extend the report item is probably the most annoying limitation of the BIRT Libraries. This example uses a grid, but the same limitation applies to tables. Creating generic, re-usable tables (or grids) that can work with any Data Set, number of fields or rows are not possible with BIRT. If this is approach you want to take, the best you'll be able to do is create several variations in the libraries to be used in the reports. For example, create a 2x2, 2x3, 3x2, and 3x3 grid items for the different combinations you might need.

5.3.6 Can only inherit once

BIRT only supports one level of inheritance. Once an item has been created in a library, a new report item cannot be based off it. For example, if a label was created in a report library, I cannot create a second label that inherits from the first label in that same library.

5.4 Recommendations when using BIRT Libraries

Because of these limitations, Le BIRT Expert recommends creating common re-usable items when they are truly re-usable as a whole item. This most likely will apply to a specific purpose for an item that is needed on several reports. As you develop the reports for your project, and you find that you need to re-use an element that is on another report – promote that item to the library from the other report and re-use. If you try to create several very generic items for re-use in any situation, it will turn into more effort than it's worth.

As you are developing the reports, and find that certain report items are recurring in multiple reports, put those items in the library. No need to obsess over libraries and re-use, but continually be on the look-out for ways to re-factor your report designs and libraries. If a report item is specific to one report, leave it in the report. However, if it is used in multiple reports, put it in the library to be shared.

Chapter 6

Establishing a Secure and Consistent Way to Access Data

In this Chapter:

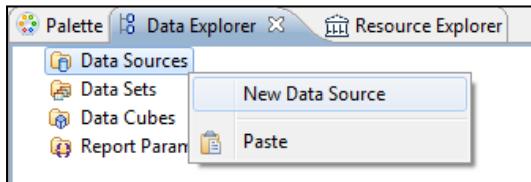
- Creating the Data Source
 - Flat File Data Source
 - Web Services Data Source
 - XML Data Source
 - JDBC Data Source
 - Using SQL
 - Dataset Best Practices
 - Tips and Tricks
 - Connection Pools
 - Scripted Data Source
 - Using POJO's, Spring, Hibernate and Other Java Frameworks with BIRT
 - Other Forms of Open Data Access (ODA)
 - Joint Data Sets
 - Other Types of Data Access using Actuate BIRT
 - Using Data Cubes
 - What is the Best Way?
 - Best Practice – Putting Data Sources and Data Sets into Libraries
 - Best Practice – Externalizing Connection Properties
-

This is one of the most important topics when discussing best practices for BIRT Report Designs. Nothing will have more affect on your reports than how you decide to access your data. Having a secure and consistent way to access your data will enable report designs to be created much quicker.

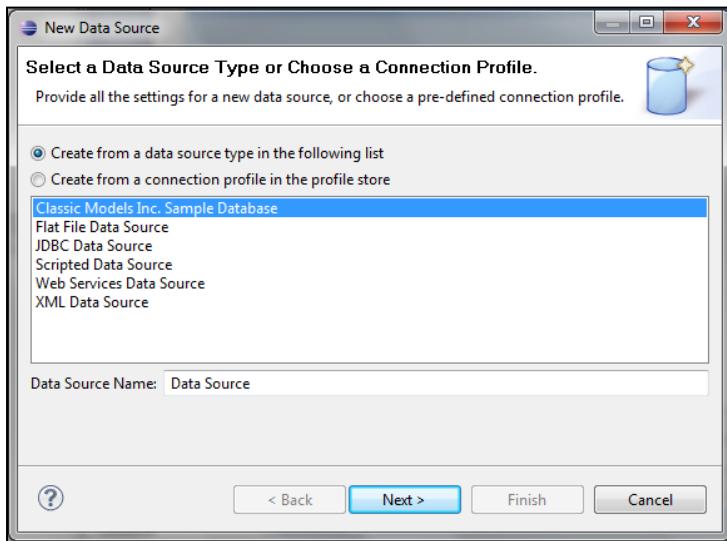
BIRT supports many different types of Data Sources. There are many different ways to retrieve data in a secure way. This chapter will discuss the different approaches and offer recommendations.

6.1 Create the Data Source

In the Data Explorer, right click on “Data Sources” and choose “New Data Source”.



The types of Data Sources available by default through BIRT are listed below. These data sources will be discussed below. Creating a connection profile in the profile store is covered as well.



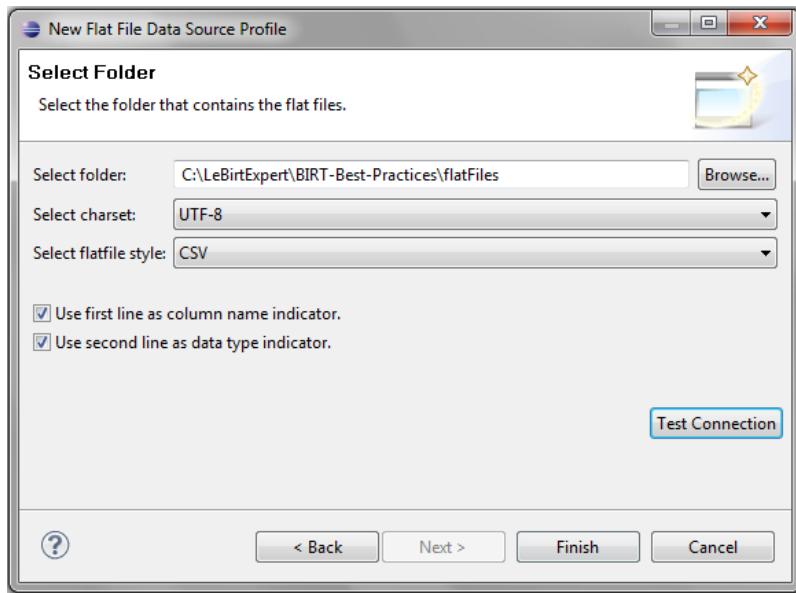
Classic Models Inc. Sample Database

This Data Source is used for examples only. It is a Java Apache Derby database and something that is provided with the Eclipse installation to allow developers to start playing around with the tool right away. See the Setup section (included with the code download) on how to create the MySQL version.

<http://www.eclipse.org/birt/phoenix/db/>

6.2 Flat File Data Source

The BIRT Designer offers a convenient way through the interface to select a flat file for a Data Source.



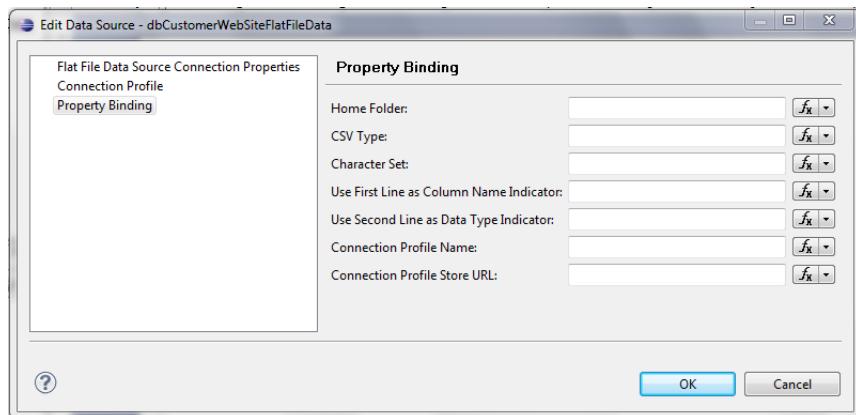
A wide variety of charsets are available. You'll mostly use UTF-8, but if you are reading files with international characters or text, you might need to use a different character set.

Flat file style lets you choose between comma (CSV), pipe (PSV), semicolon (SSV) and tab (TSV) styles.

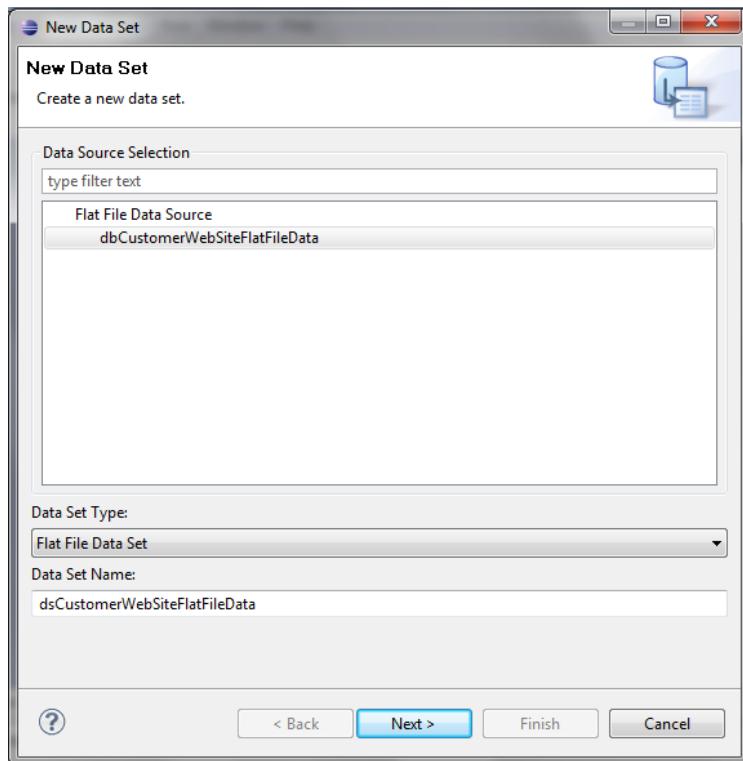
***Tip**

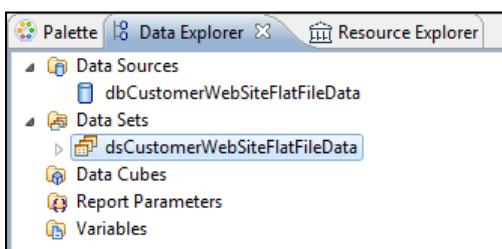
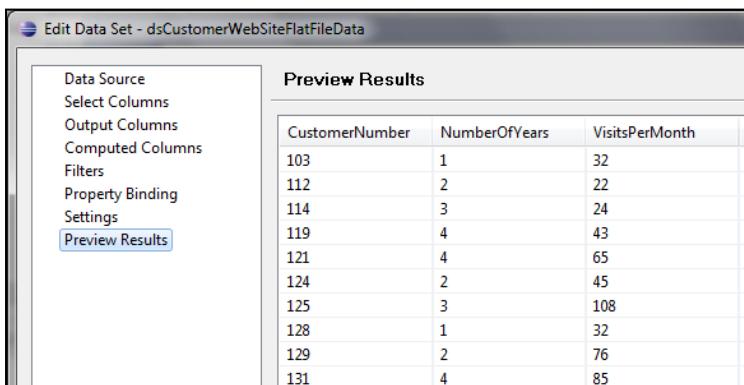
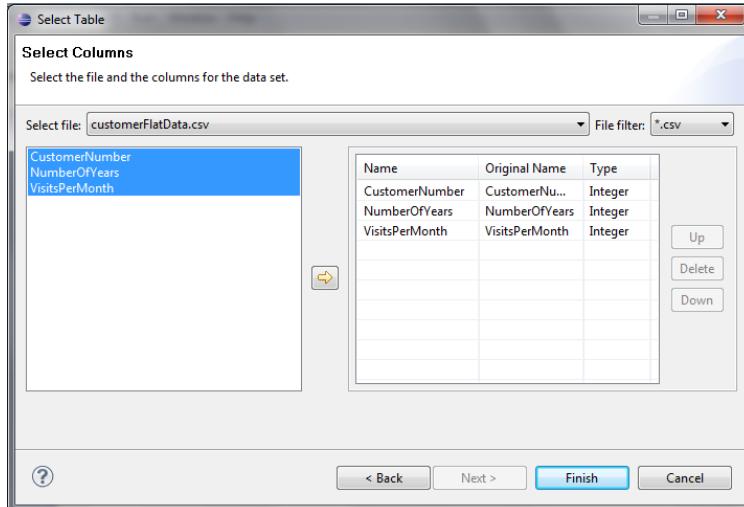
Create the Data Source only lets you choose the directory that the file is stored. The data set will let you choose the actual file. BIRT assumes that all of the flat files in the directory are of the same charset, style and format. If they are not, you will have to create a new Data Source for each different charset, style and format. It is best if you are either able to keep all of your flat files in the same format, or create different directories for different styles of flat files.

The Property Binding option allows the developer to set the connection properties at run time. If the properties need to change dynamically, either variables or an expression can be used. An example of this would be to set the folder path at runtime based on the user.



A new data set must be created to read the flat file.





Flat file access does have its place. Often times, applications have a whole process in place where data files are copied from the main frame and ftp'd to a certain directory for reporting. BIRT offers an easy and convenient way to read most flat file types. Anything more complex than what is offered, the developer would have to use a scripted data source to parse the file themselves. Or they could import the file into a database table.

6.3 Web Services Data Source

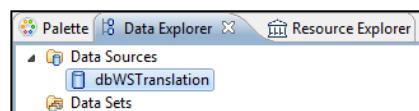
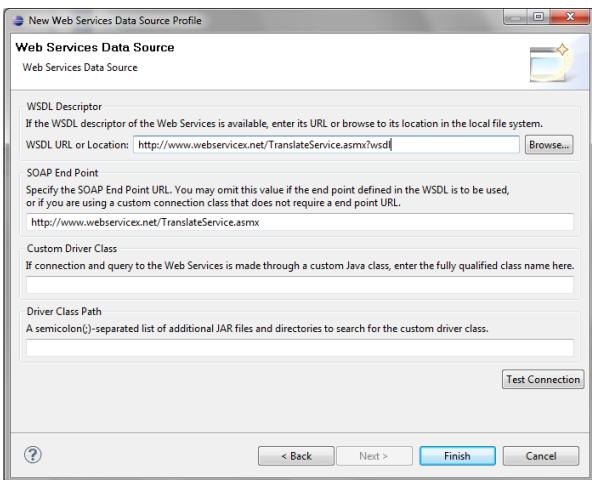
The BIRT Designer offers a way to connect to a SOAP Web Service using the Web Services Data Source. For a wizard based connection, it does offer several powerful ways to connect to a Web Service. It also provides ways to easily extend the functionality using a Custom Driver class. It is beyond the scope of this book to go into all of the implementation details of the Web Service ODA. However, using a combination of the Interface options and Custom Driver class, you should be able to connect to most SOAP web services.

To show a simple example, several publically available web services are available at <http://www.webservicex.net>. These are basic open web services that anyone can call to retrieve information.

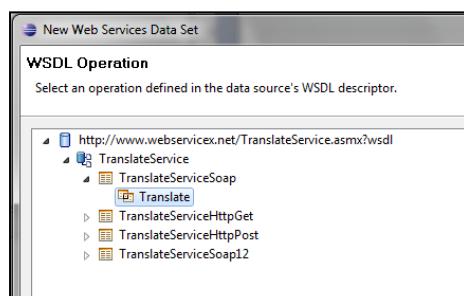
This particular web service will provide a translation from English to French.

<http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=47>

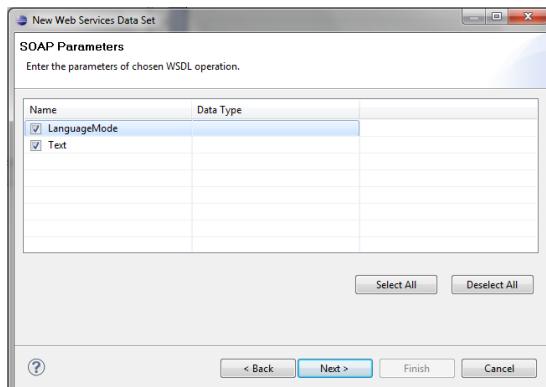
Create Data Source



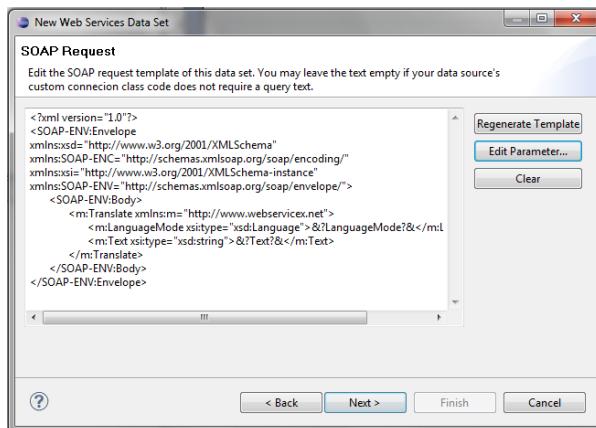
Create Data Set



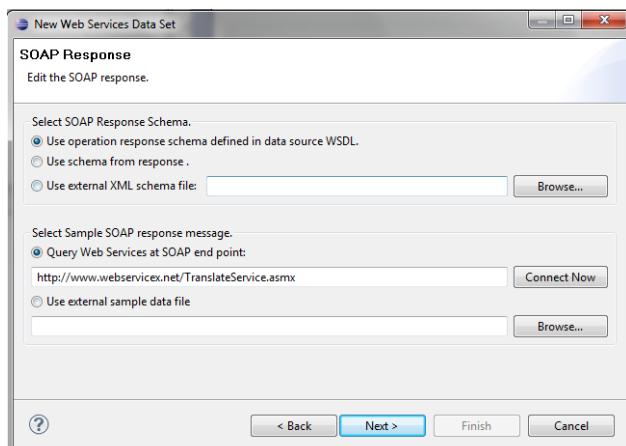
BIRT will automatically identify any parameters needed.



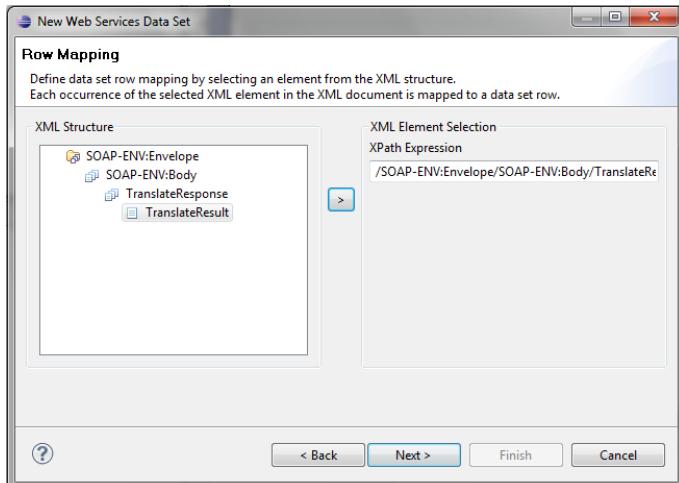
BIRT gives you the option of editing the SOAP message



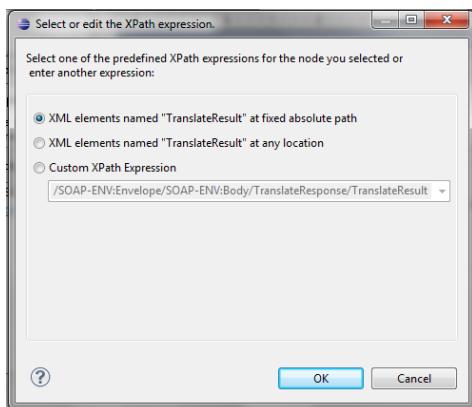
And modify the way the response is handled.



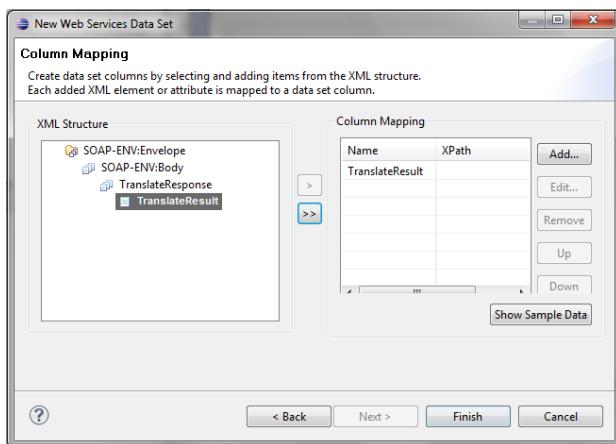
Map the response to the Data Set Row



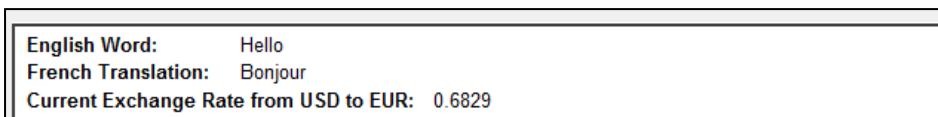
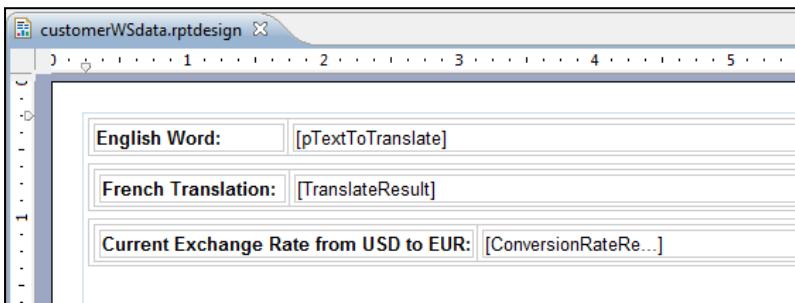
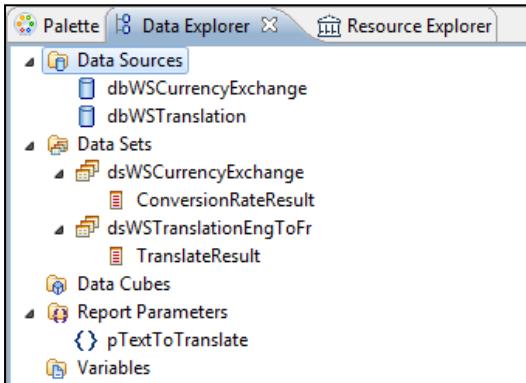
Edit the XPath expression



Map the Columns. In this example, there is only one column



Add another web service data source connection. This connection will connect to the Currency Exchange rate web service to calculate USD to EUR exchange rate.



6.3.1 Custom Driver Class

If you need to do more complex things when connecting to the Data Source, such as setting an application id or security token in the Header, the Custom Driver class can be used. The Custom Driver Class must implement some methods such as “connect()”, “queryText()” and “executeQuery()”. There is no Interface or Parent Class to extend. Just create a normal Java Class with the methods:

```
public static Object connect(java.util.Map connectionProperties,java.util.Map appContext) {  
  
    // code here  
}  
public Object executeQuery( String queryText, Map parameterValues, Map queryProperties ){  
// code here  
}  
public void close(){  
    // code here  
}
```

More information about creating a Custom Driver Class can be found at: <http://www.birt-exchange.org/devshare/designing-birt-reports/591-introduction-to-birt-web-services-data-source/#description>

More information about creating a Web Services Data Source can be found on the web:

Web Service Data Source Tutorial

<http://www.birt-exchange.org/devshare/designing-birt-reports/218-web-services-as-data-sources-birt-2-2-1-tutorial-series/#>

<http://www.birt-exchange.org/devshare/designing-birt-reports/175-birt-web-services-example/#description>

HTTPS Support

<http://www.birt-exchange.org/devshare/designing-birt-reports/893-birt-webservice-driver-with-https-support/#description>

Web Service ODA Information

<http://wiki.eclipse.org/BPS50>

<http://birtworld.blogspot.com/2007/10/birt-web-service-oda.html>

6.3.2 Recommended Approach

Most web applications that connect to web services already have an existing set of Java proxy classes that are built specifically for that purpose. If your application has this already, Le Birt Expert recommends that you use your existing proxy classes to access the web service. It is most likely that security and all other implementation details have already been figured out and tested. These classes can be called in BIRT by using a Scripted Data Source. This will save time and effort and provide consistency with the rest of your application.

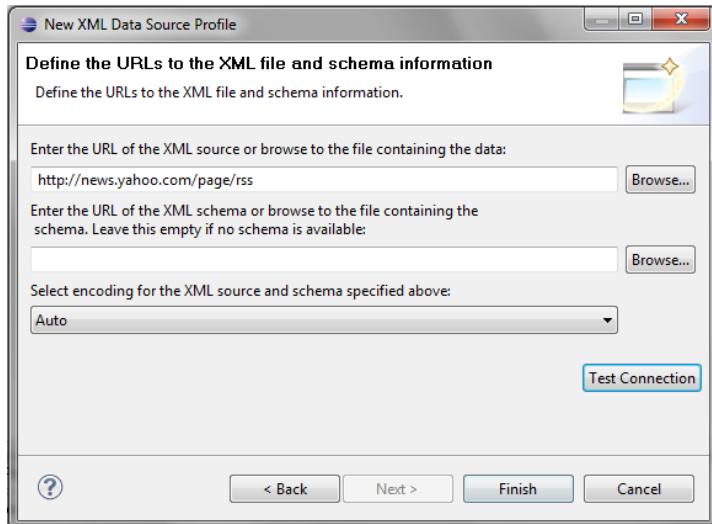
However, if you don't have re-usable Java proxy classes or if you just need to make a few quick calls to a relatively straight forward web service and do not have any unusual security tokens or other unique requirements, then the Web Service Data Source wizard is perfectly suited for that. The custom driver class will give you the ability to extend the functionality to make more advanced calls.

6.4 XML Data Source

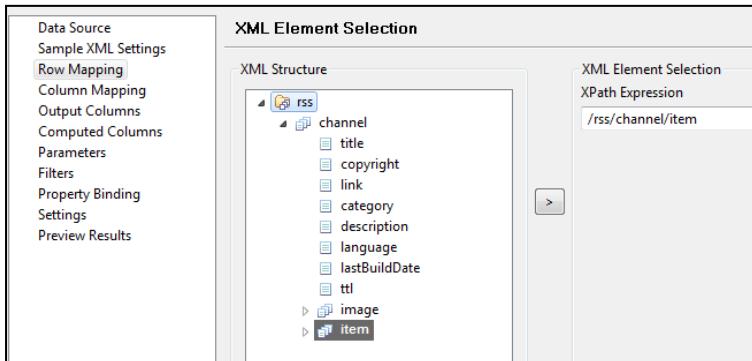
The BIRT Report Designer offers a Data Source to read and parse XML files. The wizard interface should be able to handle most types of XML files. The GUI offers the developer the ability to read from a local file or from a URL. The location of the XML file can also be dynamically changed at runtime.

6.4.1 Creating an XML Data Source

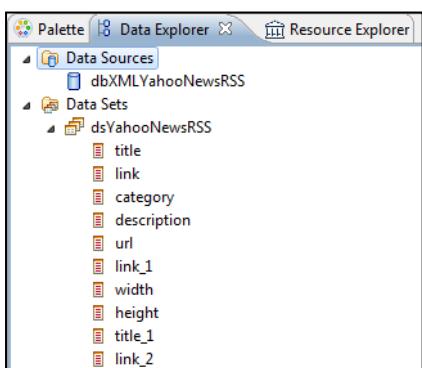
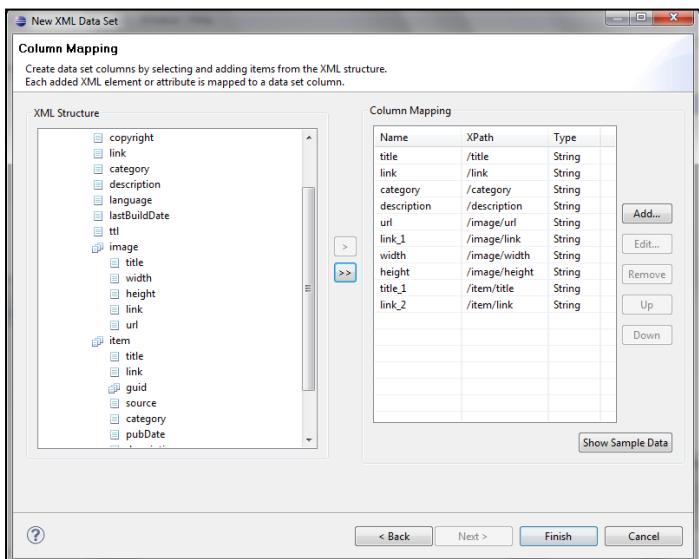
For this example, we'll use an RSS feed from Yahoo News



Map the Columns



Choose Column Mapping



category	title	description
[category]	[title]	row["description"]
Detail Row		
Group Footer Row (category)		
Footer Row		

customerXMLdata.rptdesign

Show Report Parameters Note: Current maximum number of data rows is limited to 500. [Click to change Preview Preferences](#)

category	title	description
business	Obama implores top bankers to increase lending (AP)	 AP - President Barack Obama implored top bankers Monday to help keep the fragile recovery from faltering by boosting lending to small businesses and getting behind an overhaul of financial regulation. "We rise and fall together," he declared.
entertainment	Mexico: Grammy winner sang at drug cartel's party (AP)	 AP - Latin Grammy winner Ramon Ayala was rounded up during a raid on a drug cartel's Christmas party in a wealthy gated community, a lavish night that showed the audacity of traffickers in gathering despite being hounded by police.
odd	Police: Mass. suspect mutilated fingers to hide ID (AP)	AP - Police say a Boston man wanted for drug trafficking tried to conceal his identity by cutting the tips of his fingers to hide the prints. State Police spokesman David

More information on creating reports with XML data source.

<http://www.birt-exchange.org/devshare/designing-birt-reports/563-birt-xml-data-source-tutorial/#description>

6.4.2 Recommendation

The BIRT Designer offers an easy way to parse standard XML files and RSS feeds. Be aware that parsing very large XML files can be memory intensive and is much slower than reading directly from a database. If your application already has existing code that processes large and complex XML files, you will want to consider using your existing classes in a BIRT Scripted Data Source. The XML and Web Service data sources may not perform as well as custom code.

If you need to connect to a URL using a special type of authentication to authorize you to read the XML, you are better off using custom Java classes in a Scripted Data Source. BIRT should be able to parse most XML streams, but if you find that it is too much trouble to use the GUI to do the parsing, the alternative is to use a Scripted Data Source with custom Java classes that parse the XML.

6.5 JDBC Data Source

The most common Data Source is JDBC access to a SQL database. Examples of creating reports using a SQL database are all over birt-exchange.org, so I will not provide a step-by-step example here. However, we will discuss different aspects of JDBC connection and best practices.

6.5.1 JDBC Drivers

Make sure that you have the appropriate JDBC drivers for your database. If you are using open-source BIRT, you will need to find the JDBC drivers for your database. Below is a table of popular databases and where to find the JDBC drivers.

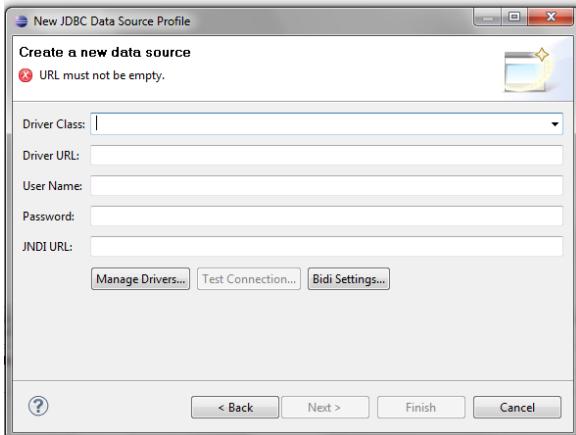
Database	Driver
MySQL	http://www.mysql.com/downloads/connector/j/
SQL Server	http://jtds.sourceforge.net/
DB2	http://www.ibm.com/developerworks/data/library/techarticle/dm-0512kokkat/
Oracle	http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html
Sybase	http://www.sybase.com/products/allproductsاز/softwaredeveloperkit/jconnect

A good reference for JDBC driver class names and url syntax can be found at:

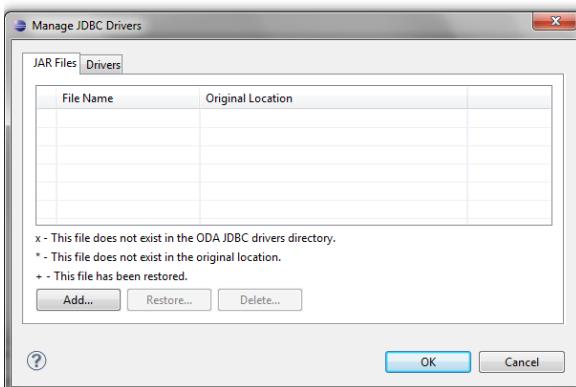
<http://www.devx.com/tips/Tip/28818>

The Actuate version of BIRT ships with several built-in JDBC drivers for the most popular RDMS databases. These drivers are from a company named Data Direct, and they are available purchase separately. (www.datadirect.com)

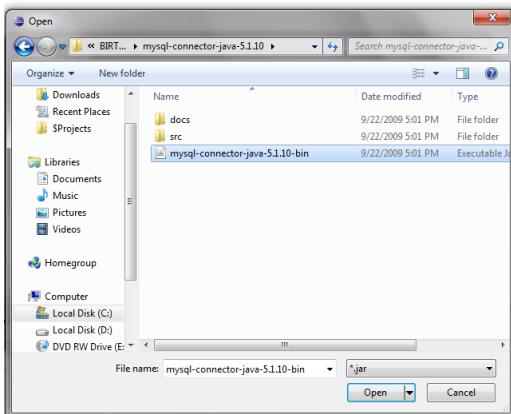
There are two ways to register a JDBC driver with BIRT. When creating a new JDBC Data Source, do the following:



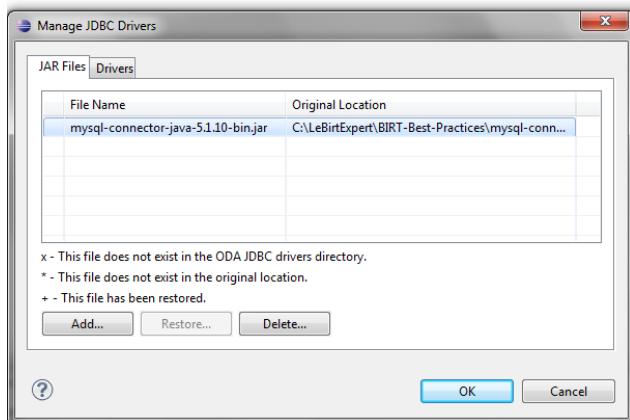
Click Manage Drivers ...



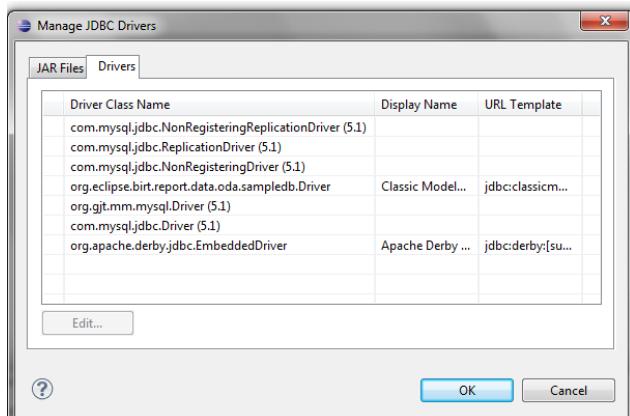
Click "Add..."



Choose the mysql-connector-java-5.1.10-bin.jar and click “Open”. The file should be where you unzipped the MySQL JDBC driver.



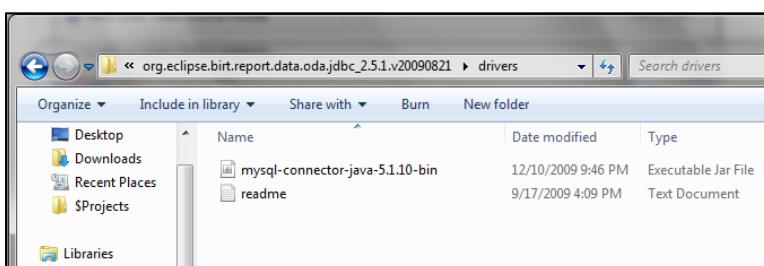
Click the “Drivers” tab to confirm that the driver was loaded.



Click “OK”.

By default, the BIRT Report Designer copied the jar file to the following directory:

C:\<eclipse-directory>\eclipse\plugins\org.eclipse.birt.report.data.oda.jdbc_2.5.1.v20090821\drivers



6.5.2 Alternative Method to register JDBC driver with BIRT Designer

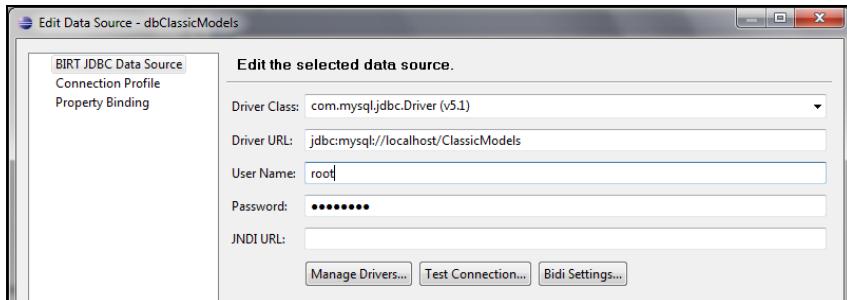
Copy the “mysql-connector-java-5.1.10-bin.jar” file to the directory:

C:\<eclipse-directory>\eclipse\plugins\org.eclipse.birt.report.data.oda.jdbc_2.5.1.v20090821\drivers
Restart Eclipse. Eclipse will automatically pick up the driver and register it.

***Tip**

When deploying BIRT reports into your application or onto the Actuate iServer, remember to also deploy the JDBC drivers! If you are using JDBC drivers with Actuate’s version of BIRT, then iServer already has the JDBC drivers deployed for you.

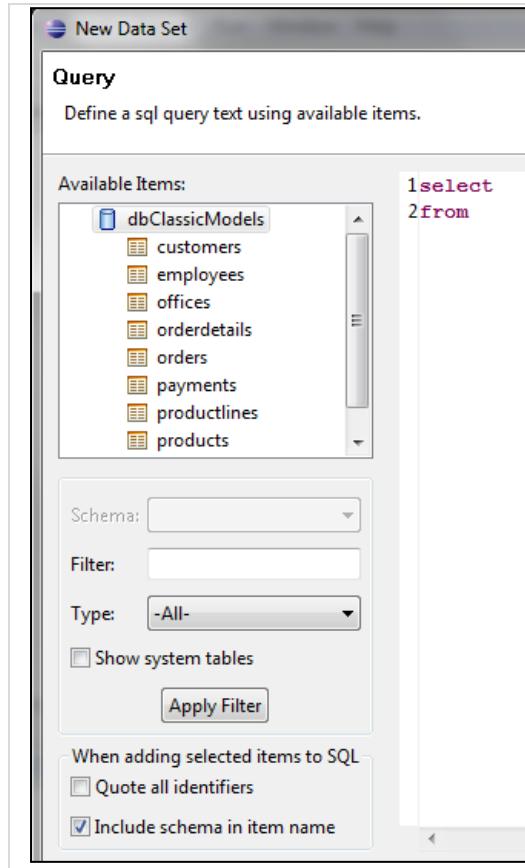
Refer to Appendix II for more information about deployment.



When creating a Data Source, be sure to have the correct database driver URL format. If you are not sure what it is, search the site where the driver was downloaded (or Google it). Use the “Test Connection” button to confirm that BIRT can log into the database using the credentials entered.

***Tip**

Just because the “Test Connection” came back successfully doesn’t mean everything is okay. The proof is when you create a Data Set and BIRT attempts to browse the database for available tables and stored procedures. This is a good test to confirm that the database user can see everything and the JDBC driver is able to pull back all the database information correctly. If you notice some irregularities, confirm the results with a different database application (like TOAD).

***Tip**

The BIRT Designer will allow the developer to browse the database schema when creating a new Data Source. For a large database schema, the designer might take a longer time to load the table information. Use the filter to find a specific table, view or stored procedure.

If you notice some tables, views or stored procedures/functions do not show up in the browser, check the definition in the database and confirm there is nothing unusual. Also double check the results with another database tool. Sometimes, custom data types have been known to cause problems. Certain JDBC drivers have been known to be a little sensitive when retrieving all database information.

6.5.3 SQL vs Stored Procedures

In the software application world, there has been an on-going religious debate about using inline SQL versus stored procedures. Since both methods are widely used, this guide will offer best practices for both.

***Tip**

Le BIRT Expert recommends using stored procedures for security, performance and re-usability. However, since many applications use inline SQL, SQL best practices will also be covered. Database specific Best Practices are outside the scope of this book, but general guidelines are offered.

6.5.4 Stored Procedures

This is Le BIRT Expert's recommended approach for accessing data that is secure, fast and reusable.

Pros:

- Secure
 - Stored procedures can greatly reduce the threat of the “SQL Injection” attack.
 - The DBA can set role-based security privileges on the stored procedure objects.
- Performance
 - Stored procedures are compiled and executed on the database server, offering improved performance over dynamically generated SQL.
- Development and Testing
 - Allows a DBA (or other developers) to work independently on creating and unit testing the stored procedures for the Reports. This will speed development and testing.
- Maintenance
 - Stored procedures encapsulate the logic of accessing and retrieving the data, while providing an interface for the report to access it.
 - Provides a single point of change for data access.
 - If a change needs to be made in the query, it can be done so in the stored procedure and not require any code changes or redeployment of the reports.

Cons:

- Expertise
 - Creating a library of stored procedures requires some expertise about the database and knowledge on how to write the procedures. For smaller companies or teams that do not have a dedicated DBA, this can create a bottleneck during development.
 - Dynamic and ad-hoc queries can be harder to write and maintain.
- Portability
 - Stored procedures are not portable across database systems. If you create a set of Oracle stored procedures, they cannot be moved to DB2. This is only a weakness if you plan to support multiple databases.

Database-specific best practices for writing stored procedures is outside the scope of this book. This section will offer how to connect using a stored procedure and Best Practices when calling the procedure from BIRT.

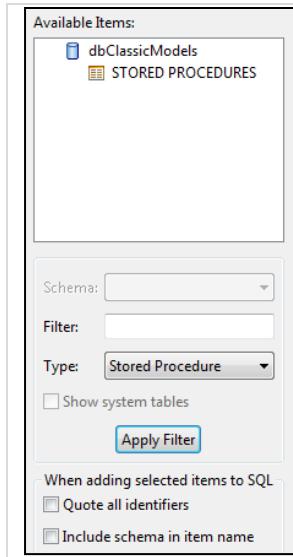
Create a new report. Create the data source. Create a new report parameter called “pCountryName”.

The screenshot shows the 'New Parameter' dialog box. The 'Name' field is set to 'pCountryName'. The 'Prompt text' field contains 'Enter Country Name'. The 'Data type' is 'String' and the 'Display type' is 'Text Box'. Under 'Display As', 'Format as' is 'Unformatted' with a preview of 'USA'. The 'List Limit' is set to 'values'. Under 'Validation', 'Is Required' is checked. The 'Default value' is 'USA'. The 'List of value' section has 'Static' selected. The 'Palette' window on the left shows 'Data Sources' containing 'dbClassicModels', 'Report Parameters' containing 'pCountryName', and 'Variables'.

Confirmed that the stored procedure “GetCustomerByCountry” has been created in MySQL. See the Setup section (including in the code download) to do this.

Create a Data Source using a Stored Procedure

The screenshot shows the 'Create a Data Source using a Stored Procedure' dialog box. The 'Data Set Type' is set to 'SQL Stored Procedure Query'. The 'Data Set Name' is 'dsCustomerByCountrySP'.



Try to find the stored procedure in the database schema browser. It cannot be found, even though we know it exists.

We know the stored procedure exists because we can call it from the MySQL command prompt. I'm not sure if this is a bug in the JDBC driver or the BIRT Designer. It may happen from time to time. Just be aware of it and know that there are always work-arounds.

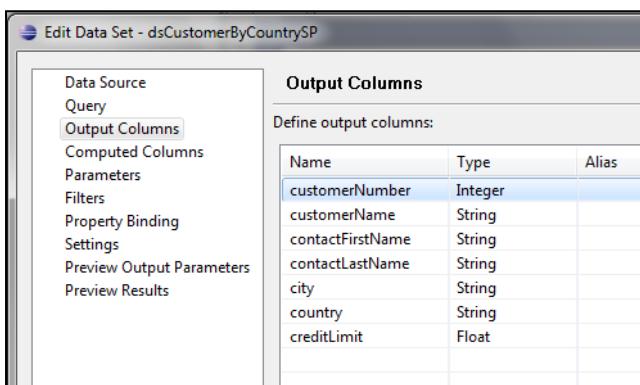
Since we can't find the procedure in the browser, type the CALL command directly into the Window. Click Finish.

CALL GetCustomerByCountry(?)

***Tip**

Always use "?" when passing a parameter. BIRT Designer will automatically recognize the "?" as being a parameter and it will automatically create a data set parameter for you.

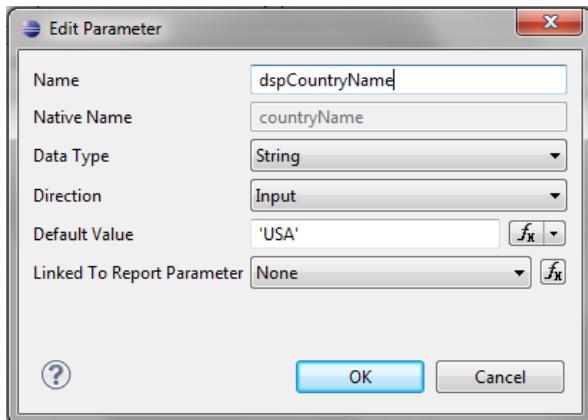
The BIRT Designer will automatically recognize the output columns. If there is an error in the store procedure, the Designer will let you know.



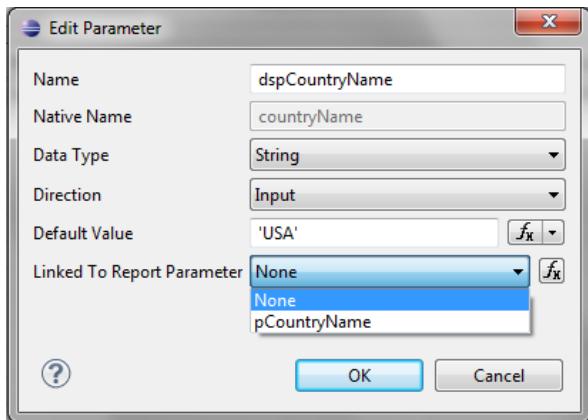
The parameters have been automatically created as well. Remember our naming conventions? Rename the “countryName” parameter to “dspCountryName”. This is so we know that this parameter is a data set parameter and not any other kind of report parameter or variable.

Name	Native Name	Data Type	Direction	Default Value	Linked To Report Parameter
1	countryName	String	Input		None

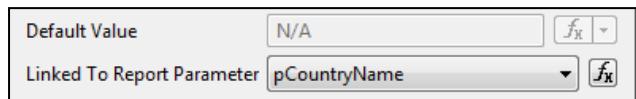
Note that even though this parameter is defined as a String, the default value still must have single quotes around it for it to be passed to the stored procedure correctly. This is not necessary if it's linked to a report parameter.



To link this parameter to a report parameter, first define the report parameter. The BIRT Designer will offer you a choice to choose between any of the Report Parameters available.



The BIRT Designer forces you to choose between providing a default value and linking to a report parameter.

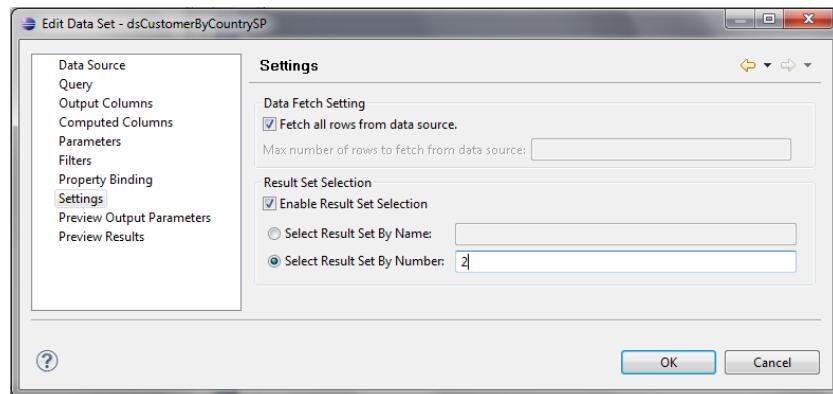


6.5.4.1 Stored Procedure Options

The BIRT Designer now offers several advanced options for Stored Procedures.

Settings

This is used when there are multiple result sets returned by the stored procedure. This option allows the developer to choose which result set to open. The result set can be chosen by Name or Number.



Preview Output Parameters

When the stored procedure has output parameters, this can be used to see the values.

Name	Native Name	Data Type	Direction	Default Value	Linked To Report Parameter
1 dspCountryName	countryName	String	Input	'USA'	None
2 dspTotal	total	Integer	Output		None

The screenshot shows the 'Preview Output Parameters' section of the BIRT Data Source configuration. On the left, a sidebar lists options: Data Source, Query, Output Columns, Computed Columns, Parameters, Filters, Property Binding, Settings, Preview Output Parameters (which is selected and highlighted in blue), and Preview Results. Below this is a preview table with one row for 'dspTotal' containing the value '36'. At the bottom, there is a toolbar with icons for 'dsCustomersCountByCountrySP' and '{ } dspTotal'.

6.5.5 Testing Stored Procedures

Often, it is useful to have your preferred database tool open to run stored procedures to compare them to the results you see in the report. It is sometimes easier to validate the data using your database tool than it is to view a 20 page report.

6.5.6 Using Stored Functions

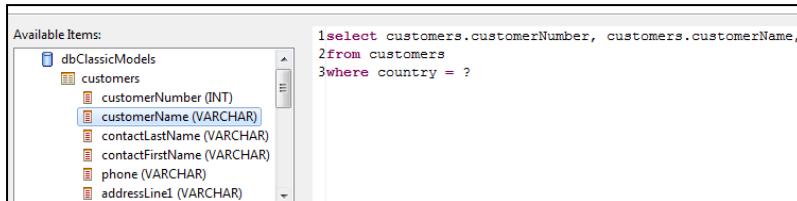
BIRT can also execute Stored Functions. Stored Functions are similar to a stored procedure, but they only return one value. They can be used for calculations, look-ups and other quick things. They can be called within Inline SQL (select function_name(?)) or executed directly using its own stored procedure Data Set.

6.5.7 Using SQL

Using SQL will be by far the most popular way of querying a database for most BIRT beginners. There are many examples on birt-exchange and elsewhere on how to write SQL statements to query data, so we will not be covering that. Database-specific best practices for writing good SQL statements is outside the scope of this book. There are a lot of good resources about SQL on the web. This section will offer Best Practices when querying the data source from BIRT using SQL.

6.5.8 Designing the Query

The BIRT Designer does not offer an advanced screen to design your query. The database schema browser and query window interface is a bit clunky. To add a table or field to the text window, make sure your cursor is where you want the item to go, and then double click on the table or column name to add it to your text. Then type “,” or “ ” or other text to add to your query.

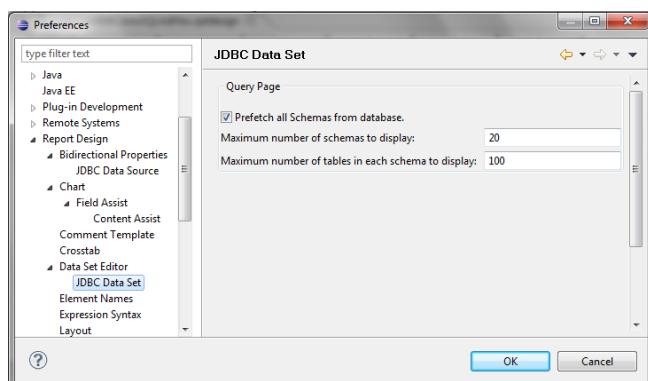


It is recommended that you use another database tool to create your queries. TOAD (freeware version available - <http://www.toadsoft.com/>) or other open source tools are available for most major databases. Once you have written and tested your queries, copy and paste them into BIRT. Replace hard coded parameter values with “?”.

*Tip

To assist in writing queries, a SQL formatter utility will come in handy. When copying and pasting SQL between BIRT and your database tool, you may find it helpful to reformat the code to make it easier to read. Many SQL formatters are available on the web.

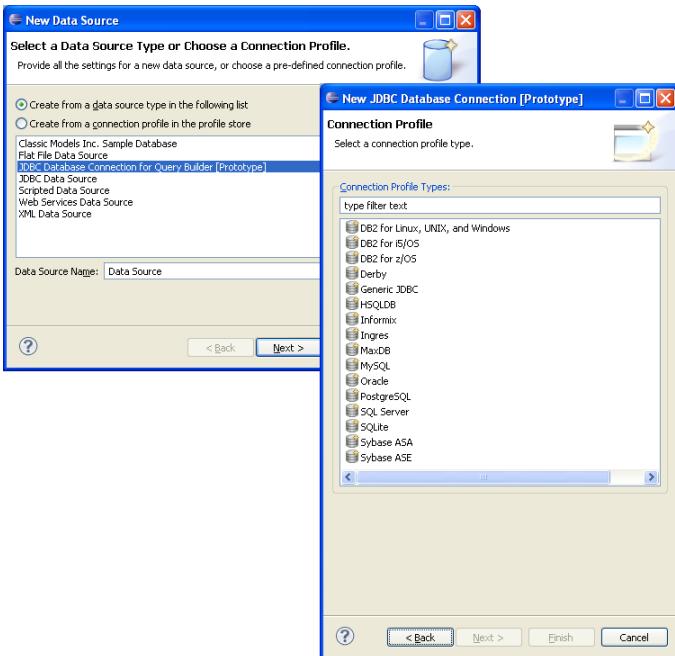
If you notice that some tables or schemas are not displaying in BIRT, double check the Data Set Editor properties by going to Window → Preferences → Data Set Editor → JDBC Data Set



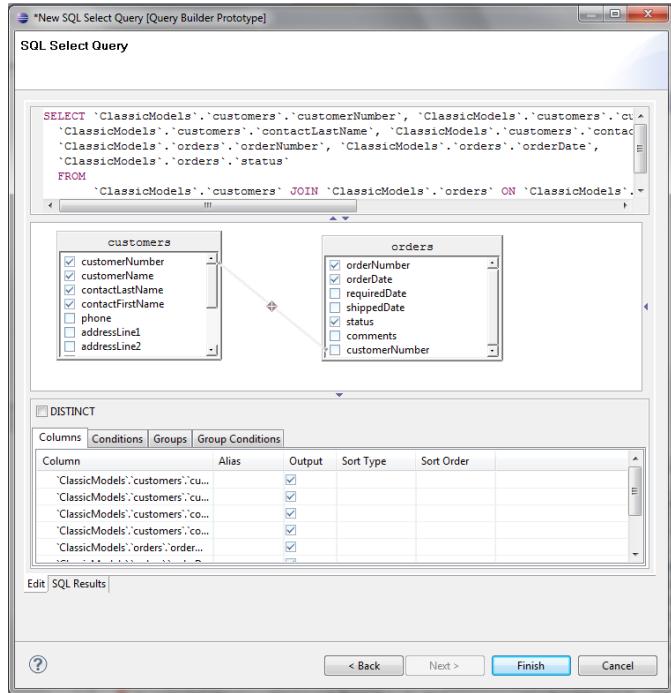
6.5.9 Using DTP to design your Query

A relatively new feature in Eclipse is called the “Data Tools Platform” (DTP). This is an eclipse plug-in that is designed to provide a common way for all Eclipse projects to access a database. It provides drivers and connection properties for most major databases. Once a connection Profile is created, it can be shared among Eclipse projects that support DTP. BIRT supports DTP and allows the developer to use the SQL Query Builder interface to design their queries.

In BIRT versions prior to 2.5.2, a connection profile had to be created to use the DTP. This can be created when creating a Data Source. Connection Profiles will be covered later in section 6.18.7. Version 2.5.2 and later support using the DTP SQL Query Builder directly through the Data Source interface.



For now, DTP offers a better interface than the native BIRT interface for creating SQL queries. It still feels a little clunky and doesn't offer many advanced features, but it does allow you to visualize your query and to copy and paste SQL into the window for more advanced SQL. As noted in the tool itself, the Query Builder is a Prototype, so it might be a little buggy.



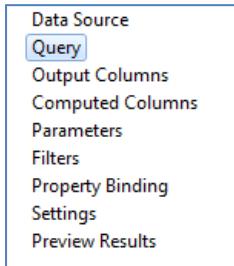
More information on the DTP SQL Query Builder can be found at:

<http://www.birt-exchange.org/devshare/designing-birt-reports/782-eclipsecon-2009-using-and-extending-eclipse-data-tools-dtp-/#description>

<http://birtworld.blogspot.com/2010/03/birt-252-enhancements.html>

6.6 Dataset Best Practices

Regardless of the Data Source or Data Set Query, BIRT offers several common properties for the Data Set. We'll discuss their use and pros and cons.



6.6.1 Computed Columns

The Computed Columns interface provides an easy way to perform calculations or evaluate expressions based on data from the Data Set. This can be very useful, particularly when using Flat File, Web Service or XML data sources. Often, the developer doesn't have much control over these data sources, and the Computed Columns feature is very useful in performing calculations and evaluating expressions.

When using SQL queries, there is debate about whether it is better to put these calculations and expressions into the SQL or have the reporting tool evaluate them. We recommend that you put the calculation or expression in the SQL statement itself. This way, the expression is run on the database side, which will improve performance of the query.

One exception to this is if your SQL statement contains many calculations and expressions. If so, your SQL statement can get very messy and hard to read and maintain. As a rule of thumb, mathematical calculations should be done on the database side, perhaps even using stored functions to perform the calculation. Simple evaluative statements can be done in the SQL as well. For lengthy case statements and other evaluative situations, it would be easier and cleaner to let BIRT do it. It can be done using Computed Columns, Script code (discussed later) or other methods in the Data Control (discussed later).

When using Stored Procedures, it is also recommended that the calculation or expression is put into the stored procedure to improve performance and keep the business logic in the stored procedure. However, many times, stored procedures are written to be somewhat generic, and the report developer will not have access or be able to modify the stored procedure. In other

cases, the report developer needs to do a certain calculation just in one particular report. In these cases, the computed columns can be used.

No matter what data source or data set you are using, having BIRT handle your Computed Columns can be very convenient. However, over a large dataset, this can affect performance. BIRT is doing the calculation in memory as it is reading all of the rows in the Data Set. It is best for the calculation or expression to be evaluated before the data arrives in BIRT.

6.6.2 Parameters

Data Set Parameters are essential to querying the database and filtering the data. BIRT should automatically recognize parameters for the query and list them in the screen below. If the query was edited from its original source, sometimes it does not pick up the change. Check the parameters screen to confirm that all defined parameters match up to the “?” in the Query. If not, BIRT will throw error messages when attempted to run the query.

Parameters					
	Name	Native Name	Data Type	Direction	Default Value
1	dspCountryName		String	Input	N/A
2	dspMinCreditLimit		String	Input	50000

New... Edit... Remove Up

***Tip**

There are some cases where the same parameter needs to be used twice in one query. In this case, the developer will need a “?” for each parameter, and BIRT will create a data set parameter for *each* “?”. Each data set parameter must have a default value or be linked to a report parameter. Even if two data set parameters represent the same piece of data, the “?” will still need to be specified for each parameter.

6.6.3 Linked Parameters Versus Default Value

Some developers might assume that it's okay to just link all of the data set parameters to report parameters. This is a security risk and dataset parameters should only be exposed through Report Parameters if you want the end user to see them.

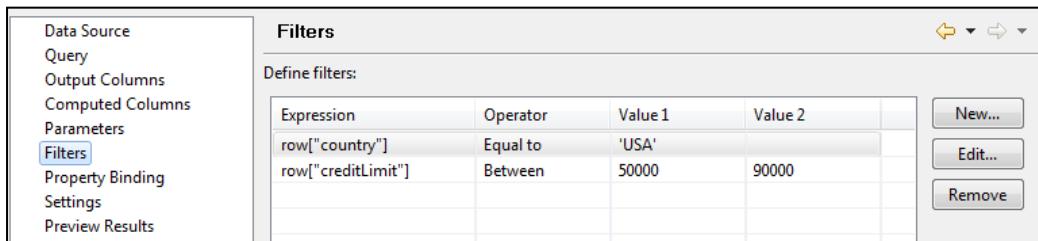
Only create report parameters that you want the user to see and adjust when running the report. Only link data set parameters to report parameters if they need to be. Otherwise provide Default values for them or set them programmatically.

***Tip**

There are times when you will need to set non-exposed data set parameter based on data from another query or other piece of information. This can be done programmatically using Scripting. An example of this would be detecting what user is running the report, and passing in the user name as a parameter directly into the dataset to retrieve data specifically for that user. If the user was exposed as a report parameter, then any user name could be passed in.

6.6.4 Filters

The BIRT Designer provides a couple ways to filter data in the Report Design. One way is to use the Filter option on the Data Set wizard. This interface provides a convenient way to filter data based on an expression.

***Tip**

Le BIRT Expert recommends that whenever possible, to filter the data on the database side. In using the data set filter, BIRT is pulling all of the data to the report, and then filtering the rows as it reads each one in memory. This is memory intensive and will cause performance problems in your report.

When using a SQL statement, use the “WHERE” clause to have the database handle the filter.

```
1select customers.customerNumber,  
2from customers  
3where country = ?  
4and customers.creditLimit > ?
```

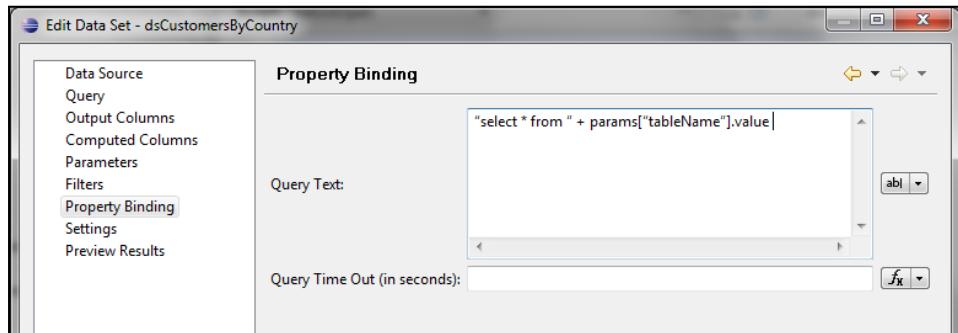
When using a Stored Procedure, have the stored procedure filter the data before returning it. It might be necessary to have additional parameters on the procedure to let the report pass in filter criteria.

When using a flat file, XML file or Web Service, the report developer most likely will not have control over where the data is coming from. If possible, split up the files according to data requirements. Or contact the author of the Web Service and request a filter parameter.

If the data is not able to be filtered by the time it reaches BIRT, using the data set filter is the next best option. The interface offers a convenient way to do this. Just be aware that this is memory intensive for large data sets and may cause performance problems.

6.6.5 Property Binding

A relatively new feature in BIRT is the Property Binding screen. Before, this was only available by scripting. It is a feature that lets a developer override the Query or timeout settings based on a variable or expression at runtime. This feature can be used to create dynamic queries on the fly.

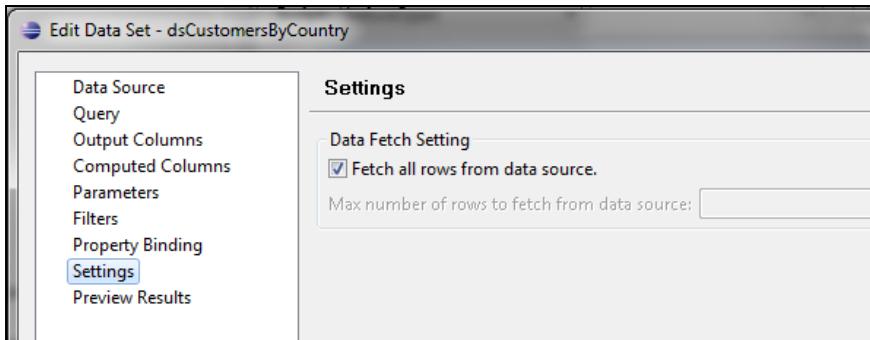


NOTE: The SQL shown in this example is not secure and just provided as an example on how to use the interface. There are other concerns when creating a dynamic SQL statement which are discussed later in this section.

See section 6.2, 6.7.3 and 6.8 for more information on property binding.

6.6.6 Settings

In most cases, the developer will want to fetch all records. However, in the case where they want to restrict the number of rows being returned, this setting can be used.



6.7 Tips and Tricks

This section offers other advice on data access in BIRT.

6.7.1 Don't Use Database Schema Name or Namespace

A mistake that has been made from time to time is for the developer to include the database schema name or namespace in the SQL code or call to the stored procedure. This sometimes happens if they use a database tool that generates SQL that includes the name. It may not be evident that there is a problem right away. However, by specifying a specific database name or namespace in the query, your report may be pulling data from a database different than what you intended.

The data source connection should establish which database you are connecting to, and your database security should determine which namespace to use. Your query should not determine this. An example of this scenario is of reports that have been moved to production, but are still pulling data from test or dev databases.

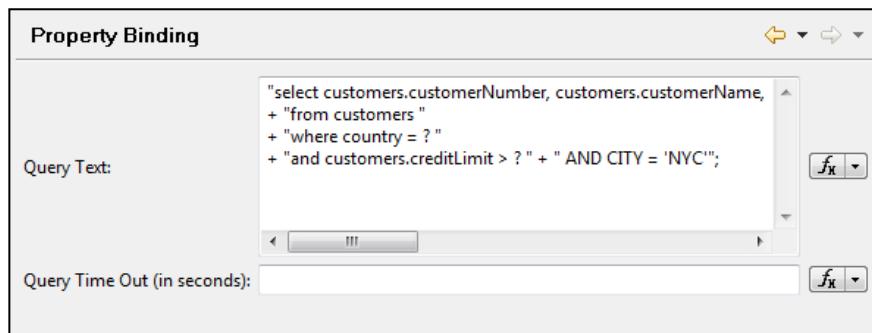
6.7.2 Using Views

Views are SQL queries that have been saved on the database. They offer better performance and a convenient way to re-use queries. It is beyond the scope of this book to talk about Views Best Practices, but Views can be leverage by BIRT in the same way as writing normal SQL queries.

6.7.3 Changing SQL at Runtime

A question that is often brought up is how to change the SQL of a data set at runtime. BIRT provides three ways on how to do this.

- 1) Use the Property Binding feature of the Data Set properties. This method offers a convenient way to dynamically update the SQL at runtime. However, there are a few drawbacks you need to know before using it.
 - a. When using the Property Binding interface, the developer cannot refer to the original Query. Using "this.queryText" will cause BIRT to throw errors. The developer must repeat the whole query. This can be a pain when your query is very large.



- b. The Property Binding interface also requires the developer to reformat the query to be used within Javascript context. This means no line breaks without ending quotations, using the "+" character to join strings together, and using other Javascript syntax. If the developer just copies and pastes their original query into this window and tries to add something at the end, you will get Javascript errors when running the report.
 - c. When Previewing the results of the query, BIRT does not use the Query Text Property Binding. This is a huge drawback, because the only way to test this is by running the actual report. You may notice that the preview results give one result, but running the report will give a different result.

- 2) Use Javascript to modify the Query Text at runtime. This is Le BIRT Expert's preferred method. Scripting will be covered later in the book, but the following is a quick example on how to do this. Scripting allows the developer to reference the original query and use other Javascript methods and keywords to construct the query of choice. The results of the Scripting are also reflected in the Preview Results option of the Data Set.

Original Query:

```
select customers.customerNumber, customers.customerName,
customers.contactLastName, customers.contactFirstName, customers.city,
customers.country, customers.salesRepEmployeeNumber,
customers.creditLimit

from customers

where country = ?

and customers.creditLimit > ?
```

Click on the Data Set, and then click the “Script” tag in the other window. You should see the name of the Data Set on the right corner of the window. This creates a script for that particular item. If not, you picked the wrong item. Choose the “beforeOpen” method.

Type in:

```
// Changes the Query Text at run time.

this.queryText = this.queryText + " AND city = 'NYC' ";
```

The screenshot shows the BIRT IDE interface. At the top, there is a toolbar with various icons. Below the toolbar, a script editor window is open. The 'Script:' dropdown menu is set to 'beforeOpen'. The code in the editor is:

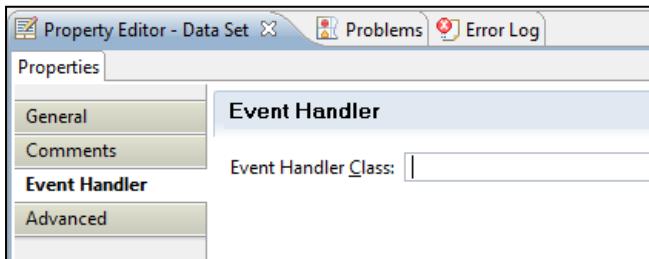
```
// Changes the Query Text at run time.
this.queryText = this.queryText + " AND city = 'NYC' ";
```

Below the script editor is a preview results table. The table has columns: customerNumber, customerName, contactLastName, contactFirstName, and city. The data in the table is:

customerNumber	customerName	contactLastName	contactFirstName	city
131	Land of Toys Inc.	Lee	Kwai	NYC
151	Muscle Machine Inc	Young	Jeff	NYC
181	Vitachrome Inc.	Frick	Michael	NYC
424	Classic Legends Inc.	Hernandez	Maria	NYC
456	Microscale Inc.	Choi	Yu	NYC

- 3) Using Java Event Handlers to modify the Query Text at runtime. This technique is a little heavy-handed since using Scripting should be plenty enough, however it is possible to change the query text this way.

- a. Click on the Data Set, and click on the “Event Handler” option in the Property Editor of the Data Set. This allows the developer to name the Java Event Handler class that will be called at runtime. Java Event Handlers will be covered later in this book.



6.7.4 Dynamic SQL Cautions

Using dynamic SQL can be very useful in many scenarios, however, it is more vulnerable to security holes. Dynamically combining existing SQL code with report parameters without any precautions is a recipe for disaster. Here are a few guidelines when constructing dynamic SQL.

- 1) Never allow full select, from or where clauses to be passed in as a report parameter and used directly in a SQL statement. Allowing a user to pass in a whole query, or even whole clauses of a query means opening up your database to anyone.
- 2) Never allow users to pass in column names or use column names directly when doing dynamic filtering or sorting. Even though it's easier, it's a security risk.
- 3) When using the value of a parameter in a SQL statement, consider validating the value to confirm it does not contain SQL Injection commands.
- 4) Be mindful of putting in the correct single quotes for strings, formatting dates properly, and other SQL syntax formatting needs when constructing a SQL statement dynamically. Otherwise, run time database errors will occur over incorrectly formatted SQL strings.

6.7.5 Using dynamic SQL to Secure Your Queries

A technique used often is to include the user id or other information in the SQL itself to limit the data to only what the user has access to see. BIRT can retrieve the user id (or other user information) from the HTTP Session, ApplicationContext, iServer variable, parameters, or other ways and include it into the SQL where clause. Look to take advantage of this when you need to restrict data based on users or roles. Inclusion in the where clause or the use of sub-queries can help solve this.

More information on doing this in an Actuate environment:

<http://www.birt-exchange.org/devshare/designing-birt-reports/122-filter-birt-jdbc-query/#description>

6.7.6 Dynamic Filter and Sorting Example

Add two parameters to the report called pCityName and pSortField. For the pSortField, add two static options for “CustomerName” and “CreditLimit”. Confirm they are not required parameters on the report.

Add the following script to the “beforeOpen” method of the Data Set. This code allows dynamic filtering and sorting to the SQL based on the report parameters.

```
// Changes the Query Text at run time.

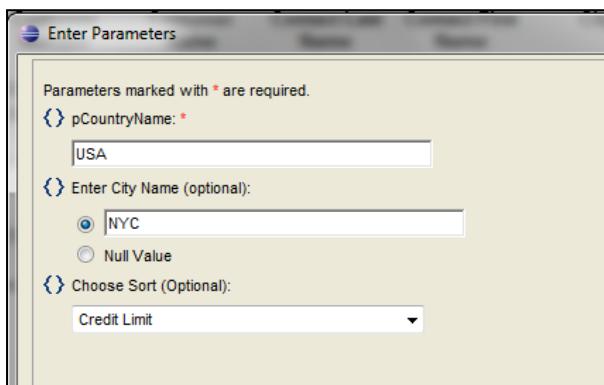
/******************
** if user chose city name, then filter on city
** - for extra security, do a check to confirm that
**   the city name passed in is a valid value (no sql injection)
*****************/
// Get City Name parameter
var paramCityName = reportContext.getParameterValue("pCityName");
// check if user entered a city
if( paramCityName != "" & paramCityName != "null" && paramCityName != null) {
    this.queryText = this.queryText + " AND city = '" + paramCityName + "'";
}

/******************
** if user chose sort field, then sort on field
*****************/
// Get City Name parameter
```

```

var paramSortField = reportContext.getParameterValue("pSortField");
// check if user chose a sort field
if( paramSortField == "CustomerName") {
    this.queryText = this.queryText + " ORDER BY CustomerName";
}
// notice the column name is used in the condition and not
// the SQL directly.
if( paramSortField == "CreditLimit") {
    this.queryText = this.queryText + " ORDER BY CreditLimit";
}

```



Customer Number	Customer Name	Contact Last Name	Contact First Name	City	Country	Sales Rep Employee Number	Credit Limit
456	Microscale Inc.	Choi	Yu	NYC	USA	1286	39800
424	Classic Legends Inc.	Hernandez	Maria	NYC	USA	1286	67500
181	Vitachrome Inc.	Frick	Michael	NYC	USA	1286	76400
131	Land of Toys Inc.	Lee	Kwai	NYC	USA	1323	114900
151	Muscle Machine Inc	Young	Jeff	NYC	USA	1286	138500

6.7.7 Passing in a delimited list of values for filtering, using IN or NOT IN

Using a list of values in a SQL statement is possible, but it has to be added to the SQL manually with a little scripting. The Designer Tool does not handle it automatically.

For example, if a delimited value was passed in as a report parameter (String) with the value as 'USA', 'France', 'Japan'; and the query was:

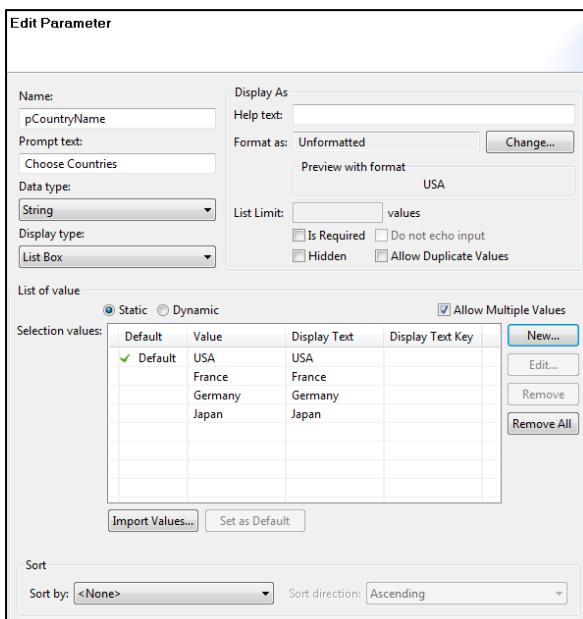
```
select *
from customers
where country IN (?)
```

BIRT would not handle it correctly. It will treat all of the options as one long string ““USA’, ‘France’, ‘Japan””.

To handle this correctly, use Scripting to dynamically add this to the SQL.

```
// Changes the Query Text at run time.
/*************************************
** if user chose country name, then filter on country
** - for extra security, do a check to confirm that
**   the city name passed in is a valid value (no sql injection)
*********************************/
// Get Country Name parameter
var paramCountryName = reportContext.getParameterValue("pCountryName");
// check if user entered a country
if( paramCityName != "" && paramCityName != "null" && paramCityName != null) {
    // normally wouldn't add complete WHERE clause - example only
    this.queryText = this.queryText + " where country IN (" + paramCountryName + ")";
}
```

If the report parameter is defined as a multi-select list, then BIRT will pass the selected values as a Javascript Array. The developer will need to use Javascript to iterate through the array to get the values, build the IN clause, and append it to the SQL.

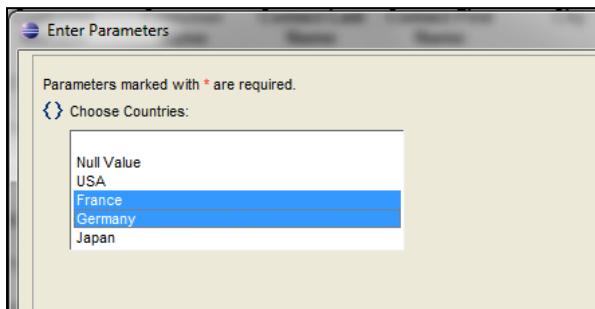


Script on the “beforeOpen” method of the Data Set.

```
// gets array of country values
var arrCountryList = reportContext.getParameterValue("pCountryName");
var strCountryIN = "";

// iterate through array, getting values, building IN clause
for(var i=0;i<arrCountryList.length;i++) {
    strCountryIN += " " + arrCountryList[i] + " ";
    if(i < (arrCountryList.length - 1) ) {
        strCountryIN += ",";
    }
}

// append IN clause to end of SQL - should validate value
this.queryText += " WHERE Country IN (" + strCountryIN + ")";
```



Customer Number	Customer Name	Contact Last Name	Contact First Name	City	Country	Sales Rep Employee Number	Credit Limit
103	Atelier graphique	Schmitt	Carine	Nantes	France	1370	21000
119	La Rochelle Gifts	Labrune	Janine	Nantes	France	1370	118200
128	Blauer See Auto, Co.	Keitel	Roland	Frankfurt	Germany	1504	59700
146	Savely & Henriot, Co.	Savely	Mary	Lyon	France	1337	123900
171	Daedalus Designs Imports	Rancé	Martine	Lille	France	1370	82900
172	La Corne D'abondance, Co.	Bertrand	Marie	Paris	France	1337	84300

As a warning, dynamically manipulating the SQL statement to add IN or NOT IN clauses opens the developer to coding mistakes, especially leaving open the possibility of SQL Injection attacks.

Whenever constructing SQL on the fly, be sure to validate the values coming in from the report parameters and take precautions on how the SQL is constructed.

For additional information on securing IN/NOT IN clauses, see this article:
<http://birtworld.blogspot.com/2009/03/birt-multi-select-statements.html>

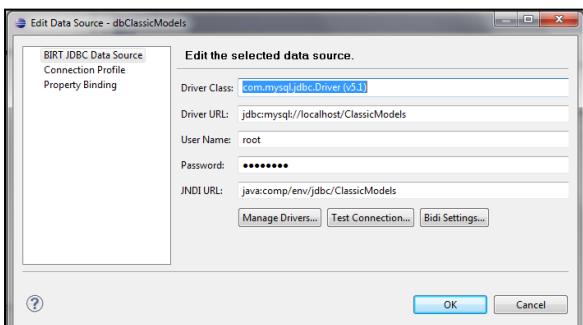
6.7.8 Ad-hoc queries

Any ad hoc query capabilities will need to be programmed manually into the report design. This is not hard to do with a little scripting, but it does require some code. (Refer to *Section 6.7.3 Changing SQL At Runtime* for examples on how to dynamically change the query at runtime.) There are no out-of-the-box ad hoc capabilities in open source. Actuate offers some commercial products that do ad hoc reporting. Visit this link for more information:
<http://www.birt-exchange.org/devshare/interactive-reporting/>

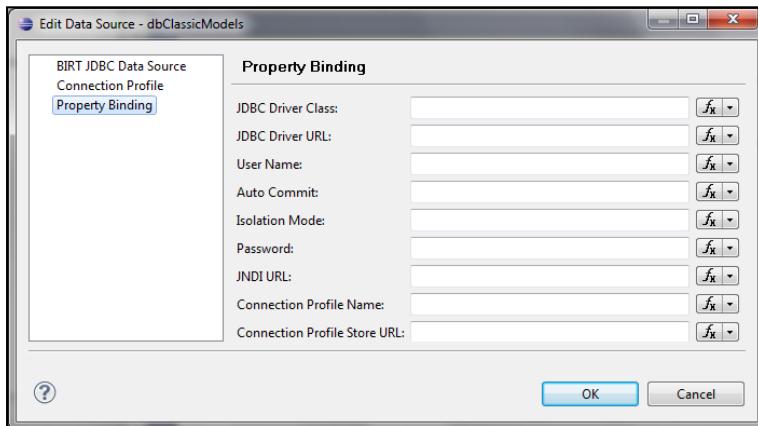
6.8 Connection Pools

If your web application uses database connection pooling, you can configure BIRT to use the connection pool instead of opening its own connection to the database. There are a couple ways to do this, as outlined below. Getting the connection pool to work with BIRT might be a little tricky and will take some experimentation to get working properly. Full implementation details are outside the scope of this book, but URL's to find more information are provided.

- 1) Use a JNDI Connection Pool. In the BIRT Designer, it provides a way to retrieve the connection from JNDI. In the Data Source properties → BIRT JDBC Data Source, the JNDI name can be set. BIRT will look for this at runtime. If it cannot find it, it will use the supplied connection details. This is most convenient way of using a connection pool. The other ways are manual coding efforts to retrieve the connection pool and feed it into the report.



These settings, including JNDI, can be set dynamically using Property Binding or Scripting ("beforeOpen" method).



How to setup a connection pool with Tomcat and use in a BIRT report, visit:

http://wiki.eclipse.org/index.php/BIRT/FAQ/Data_Access#Q:_How_can_I_pass_a_connection_object_to_BIRT.3F

- 2) Create an Extension point to use a Driver Bridge from the Data Tools Project. This involves creating a class that retrieves the Pooled Connection object from the Application Context and passes it to the BIRT Report to use. Exact implementation of this is outside the scope of this document, but more information can be found at: <http://birtworld.blogspot.com/2007/01/birt-connection-pooling-continued.html>
- 3) Passing the connection object to the report through the Application Context. This is a simplified version of the previous step involving the Driver Bridge. More information about this approach can be found at: <http://birtworld.blogspot.com/2008/11/birt-connection-pooling-continued-again.html>
- 4) Using a scripted datasource to retrieve the connection object and use it to retrieve the data. Implement a Java event handler (or could be done with scripting too) that will leverage your existing Java classes to access the connection pool and retrieve data for the report.

6.9 Scripted Data Source

Scripted data sources provide a completely flexible way to access your data however you want. If none of the previously covered methods of data access work for you, then creating a custom scripted data source is the option for you. Scripted data sources allow the developer to write JavaScript or Java code to connect to their data store in any way they wish. They are more cumbersome (and a little frustrating your first time!) to implement, but provide the best of both worlds – complete flexibility on how your data is accessed and the ability to still use all of the features of the BIRT Designer.

There are two ways to implement a Scripted Data source:

- (1) Using JavaScript to implement methods of the Scripted Data Source
- (2) Using Java Event Handlers to implement methods of the Scripted Data Source.

This section shows the developer how to implement a Scripted Data Source using these two methods. It is not meant to cover every possible scenario, but to show the developer what is possible and point them in the right direction. Best Practices for general scripting and Java event handlers are covered later in the book.

6.9.1 Using JavaScript

This section shows how to use JavaScript to create a new Scripted Data Source.

As a Best Practice, when creating a Scripted Data Source and Data Set, it's best to encapsulate as much code as possible in a Java Class that the Script can call. It is possible to do everything with Script, and it might be a little easier or more convenient. However, it is better to do all of the heavy lifting and exception handling in a separate Java class. This allows another developer to work on it if needed and it will be easier to develop, debug and unit test with pure Java. Once you have a class that retrieves the data you need, use JavaScript to call that object and populate the Data Set.

Scripting with JavaScript can appear to be easier initially, however, it can also be a pain to debug, especially when trying to call a whole series of Java Objects.

6.9.1.1 Use Java Objects

The heavy lifting of retrieving the data and parsing it has been put into the trusty hands of a capable Java Class. The following example shows how Java classes could be used to read and parse data. The class “ReadCSVForBIRTDataSet” will do the work, and will return an instance of a populated “CustomerBIRTDataSet” class. The “ReadCSVForBIRTDataSet” class will open a csv file, parse it, and return a populate instance of “ReadCSVForBIRTDataSet” for each row of data. Notice that the methods of “CustomerBIRTDataSet” will directly fit in with how the Scripted Data Source and Data Set will be calling it.

A separate Java Project can be created to hold these Java Objects.

```
package com.leBirtExpert.birtBestPractices.scriptedDS;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

/**
 * This class is used to read and parse a CSV
 * file into an ArrayList of CustomerBIRTDataSet
 * @author David Mehi (Le BIRT Expert)
 */
public class ReadCSVForBIRTDataSet {

    private FileReader fileReader;
    private BufferedReader bufferedReader;

    /**
     * Opens connection to CSV File
     * @param dbFile
     * @return result msg
     */
    public String openFileConnection(String dbFile) {

        String resultMsg = "";

        try {
            // file reader
            fileReader = new FileReader(dbFile);

            // buffered reader
            bufferedReader = new BufferedReader(fileReader);
        }
    }
}
```

```

        resultMsg = "SUCCESS";
    } catch (FileNotFoundException fnfe) {
        // log this exception
        resultMsg = "ERROR: " + fnfe.toString();
    }
    return resultMsg;
}

/**
 * Closes File connection
 */
public void closeFileConnection() {
    if(fileReader != null) {
        try {
            fileReader.close();
        } catch(IOException ioe) {
            // log exception
        }
        fileReader = null;
    }
    if(bufferedReader != null) {
        try {
            bufferedReader.close();
        } catch(IOException ioe) {
            // log exception
        }
        bufferedReader = null;
    }
}

/**
 * Reads and discards first two lines of file
 * @return
 */
public String skipFirstTwoLines() {

    String resultMsg = "";
    String dataLine = "";

    try {
        // Since we know the first two lines of the CSV file
        // contain field and type information, we can skip these
        dataLine = bufferedReader.readLine(); // skip line 1
        dataLine = bufferedReader.readLine(); // skip line 2

        resultMsg = "SUCCESS";
    } catch(IOException ioe) {
        // log message
    }
}

```

```
        resultMsg = "ERROR: " + ioe.toString();
    }
    return resultMsg;
}

/**
 * Reads the next line of the file, parses,
 * returns CustomerBIRTDataSet object
 * @return
 */
public CustomerBIRTDataSet getNextDataSetRow() {

    String dataLine;
    CustomerBIRTDataSet dataSetRow = null;

    try {
        dataLine = bufferedReader.readLine();
        if ((dataLine) != null){
            // even though the data is numbers, we'll split them into
            // a string array for now
            String columndataArray[] = dataLine.split(",");
            dataSetRow = new CustomerBIRTDataSet(columndataArray);
        }
    } catch(IOException ioe) {
        // log message
    }
    return dataSetRow;
}

}

package com.leBirtExpert.birtBestPractices.scriptedDS;

/**
 * Object representing the Customer BIRT Data Set
 * @author David Mehi (Le BIRT Expert)
 */
public class CustomerBIRTDataSet {

    private int customerNumber;
    private int numberofYears;
    private int visitsPerMonth;

    /**
     * Parses a string array of data, converts to int and

```

```

    * sets local variables
    * @param columndataArray
    */
public CustomerBIRTDataSet(String[] columndataArray) {

    this.setCustomerNumber(Integer.parseInt(columndataArray[0].trim()));
    this.setNumberOfYears(Integer.parseInt(columndataArray[1].trim()));
    this.setVisitsPerMonth(Integer.parseInt(columndataArray[2].trim()));

}

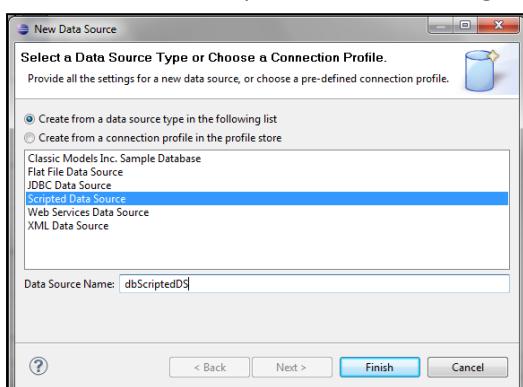
/**
 * @return the customerNumber
 */
public int getCustomerNumber() {
    return customerNumber;
}
/**
 * @param customerNumber the customerNumber to set
 */
public void setCustomerNumber(int customerNumber) {
    this.customerNumber = customerNumber;
}
// Rest of getters/setters here ...
}

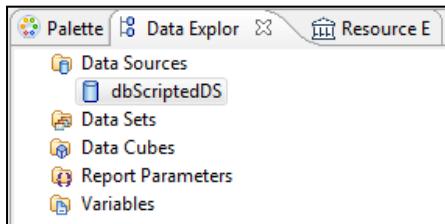
```

Imagine doing all that with Javascript! ☺

6.9.1.2 Create a New Scripted Data Source

To create a new Scripted Data Source, right click on Data Sources, and choose “New ...”





The Scripted Data Source allows the developer to control how the Data Source is opened and closed. The methods available to the developer are:

- *beforeOpen* – called just before BIRT opens the data source
- *open* – establishes a connection to the data source
- *afterOpen* – called just after BIRT opens the data source
- *beforeClose* – called just before BIRT closes the data source
- *close* – closes connection to the data source
- *afterClose* – called just after BIRT closes the data source

***Tip**

In a Scripted Data Source, putting the code that opens and closes the data source in the Data Source item itself is optional. If this Data Source will be used by more than one Data Set, it is recommended that the data source is opened and closed here. However, since the developer has complete control on how the data source is opened and closed, then they can choose to open and close the data source in the open/close methods of the Data Set instead.

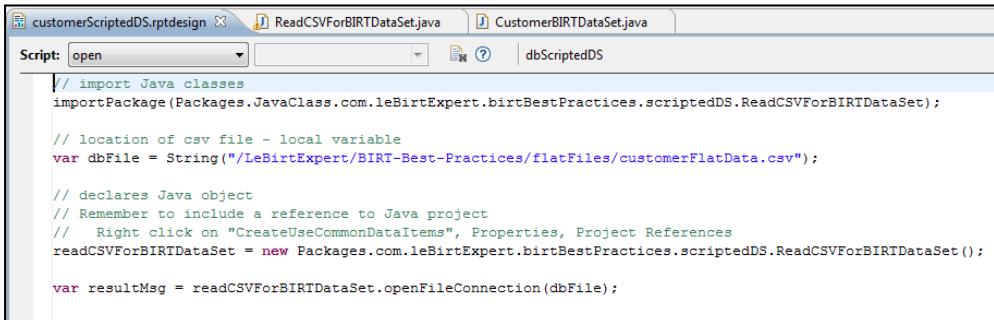
6.9.1.3 Scripted Data Source Example

“open” method of the Data Source is where the connection can be opened. Pass in the full path to the CSV file so the Java program can find it.

By using “var” to declare a Javascript variable, it keeps the scope local to the method. If you do not use “var”, then the variable is considered global and can be used anywhere in the Report.

The variable “readCSVForBIRTDataSet” is referenced later in the DataSet. The DataSource will open the connection with that variable. However, the DataSet will use it to extract the data.

To put the Java files created earlier in the classpath, right-click on the project, go to Properties, and Project References. Click the Java project that holds the Java Objects.



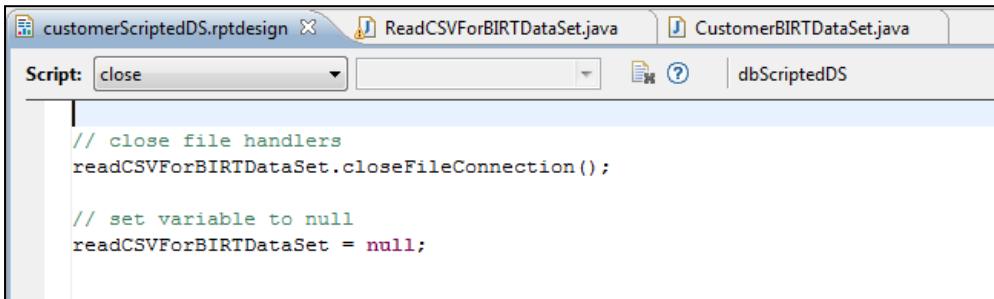
```
// import Java classes
importPackage(Packages.JavaClass.com.leBirtExpert.birtBestPractices.scriptedDS.ReadCSVForBIRTDataSet);

// location of csv file - local variable
var dbFile = String("/LeBirtExpert/BIRT-Best-Practices/flatFiles/customerFlatData.csv");

// declares Java object
// Remember to include a reference to Java project
// Right click on "CreateUseCommonDataItems", Properties, Project References
readCSVForBIRTDataSet = new Packages.com.leBirtExpert.birtBestPractices.scriptedDS.ReadCSVForBIRTDataSet();

var resultMsg = readCSVForBIRTDataSet.openFileConnection(dbFile);
```

“close” method of the Data Source is where the connection can be closed.



```
// close file handlers
readCSVForBIRTDataSet.closeFileConnection();

// set variable to null
readCSVForBIRTDataSet = null;
```

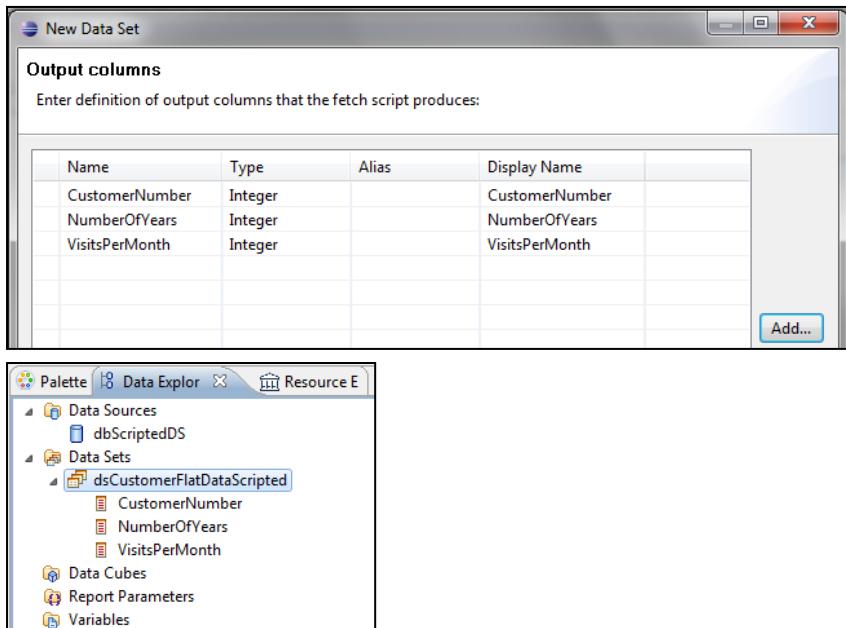
6.9.1.4 Create a Scripted Data Set

The Scripted Data Set allows the developer to control how the Data Set is opened, closed and read (fetched). The following methods are available to script in:

- *beforeOpen* – called just before BIRT opens the data set
- *open* – opens the data set
- *afterOpen* – called just after BIRT opens the data set
- *fetch* – creates, populates and returns an object of type DataRow
- *onFetch* – called just after BIRT fetches data row from the data set
- *describe* – describes the result set
- *beforeClose* – called just before BIRT closes the data set
- *close* – closes the data set
- *afterClose* – called just after BIRT closes the data set

To create a scripted data set, right-click on “Data Sets” and create a new one.

Manually add the columns (name and type) of the Result Set. BIRT won’t recognize them, you must add them yourself.



6.9.1.5 Scripted Data Set Example

The “open” method will take the Java object that was created in the Scripted Data Source and skip the first two lines of the file. It also declares a record counter variable.

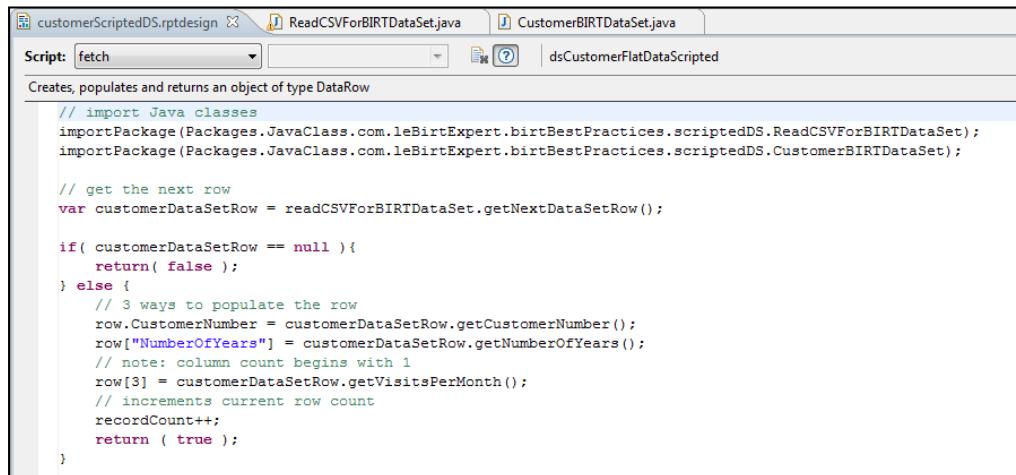
The screenshot shows the BIRT Designer script editor with the tab "customerScriptedDS.rptdesign" selected. The "Script" dropdown is set to "open". The code area contains the following Java code:

```
// import Java classes
importPackage(Packages.JavaClass.com.leBirtExpert.birtBestPractices.scriptedDS.ReadCSVForBIRTDataSet);

recordCount = 0;

// skip first two lines
readCSVForBIRTDataSet.skipFirstTwoLines();
```

“fetch” method will retrieve one data row at a time and populate the Data Set row. Notice the three different methods used to populate the BIRT Data Set row.



```
customerScriptedDS.rptdesign | ReadCSVForBIRTDataSet.java | CustomerBIRTDataSet.java
Script: fetch | dsCustomerFlatDataScripted
Creates, populates and returns an object of type DataRow


```
// import Java classes
importPackage(Packages.JavaClass.com.leBirtExpert.birtBestPractices.scriptedDS.ReadCSVForBIRTDataSet);
importPackage(Packages.JavaClass.com.leBirtExpert.birtBestPractices.scriptedDS.CustomerBIRTDataSet);

// get the next row
var customerDataSetRow = readCSVForBIRTDataSet.getNextDataSetRow();

if(customerDataSetRow == null){
 return(false);
} else {
 // 3 ways to populate the row
 row.CustomerNumber = customerDataSetRow.getCustomerNumber();
 row["NumberofYears"] = customerDataSetRow.getNumberofYears();
 // note: column count begins with 1
 row[3] = customerDataSetRow.getVisitsPerMonth();
 // increments current row count
 recordCount++;
 return (true);
}
```


```

Running the Report

CustomerNumber	NumberofYears	VisitsPerMonth
103	1	32
112	2	22
114	3	24
119	4	43
121	4	65
124	2	45
125	3	108
128	1	32
129	2	76
131	4	85
141	3	34
144	2	65
145	1	34
146	2	23
148	4	35
151	4	68
157	2	98
161	2	54
166	4	34

6.10 Using Java Event Handlers

Java Event handlers allow a developer to extend the use of BIRT by using pure Java. Writing code in pure Java allows better integration with your existing frameworks, allows you to leverage your development tools to debug and unit test, and provides strong type casting and other benefits that JavaScript cannot offer. However, it is more cumbersome to use Java Event Handlers, as it is more work to write the code and connect it to the reports.

We will implement the same example, except using Java Event Handlers.

6.10.1 Implementing the Scripted Data Set in Java

The implementation is a little different in Java. In this example, another Event Handler could have been created for the Data Source Event Handler, similar to how it's done with JavaScript. However, that would have required another class just to open the file, and place it in the reportContext as a global variable. Then the Data Set Event Handler would have had to retrieve it from the reportContext to use it.

Instead of creating a second class, all of it is implemented in the Data Set. This is only recommended if the Data Source is only going to be used once by one Data Set.

NOTE: This uses the same Data Set from the previous example. A Scripted Data Set requires you to create the Data Set by manually adding all required fields.

```
/**  
 * Customer CSV Data Set Event Handler  
 * @author Le BIRT Expert  
 */  
public class CustomerCSVDataSetEH extends ScriptedDataSetEH {  
  
    private Logger logger = Logger.getLogger("CustomerCSVDataSetEH");  
    private ReadCSVForBIRTDataset readCSVForBIRTDataset;  
    private int recordCount = 0;  
  
    /**  
     * Constructor  
     */  
    public CustomerCSVDataSetEH() {  
    }
```

```
/*
 * before Open
 * @param dataSet
 * @param reportContext
 */
public void open(IDataSetInstance dataSet) {

    try {
        // location of csv file - local variable
        String dbFile = "/LeBirtExpert/BIRT-Best-
Practices/flatFiles/customerFlatData.csv";

        // declares Java object
        // Remember to include a reference to Java project
        // Right click on "CreateUseCommonDataItems", Properties,
Project References
        readCSVForBIRTDataSet = new ReadCSVForBIRTDataSet();

        String resultMsg =
readCSVForBIRTDataSet.openConnection(dbFile);
        logger.log(Level.INFO, resultMsg);

        recordCount = 0;

        // skip first two lines
        readCSVForBIRTDataSet.skipFirstTwoLines();

    } catch(Exception ex) {
        logger.log(Level.FINE, ex.toString());
    }
}

/**
 * after close
 * @param reportContext
 */
public void close(IDataSetInstance dataSet) {

    try {
        // close file handlers
        readCSVForBIRTDataSet.closeFileConnection();

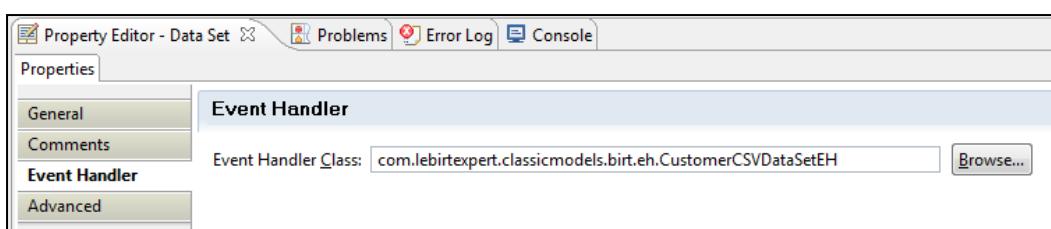
        // set variable to null
        readCSVForBIRTDataSet = null;

    } catch(Exception ex) {
        logger.log(Level.FINE, ex.toString());
    }
}
```

```
/**  
 * fetch  
 * @param dataSet  
 * @param row  
 */  
public boolean fetch(IDataSetInstance dataSet, IUpdatableDataSetRow row) {  
  
    try {  
        // get the next row  
        CustomerBIRTDataSet customerDataSetRow =  
readCSVForBIRTDataSet.getNextDataSetRow();  
  
        if( customerDataSetRow != null ) {  
  
            // 3 ways to populate the row  
            row.setColumnValue("CustomerNumber",  
customerDataSetRow.getCustomerNumber());  
            row.setColumnValue("NumberOfYears",  
customerDataSetRow.getNumberOfYears());  
            // note: column count begins with 1  
            row.setColumnValue(3,  
customerDataSetRow.getVisitsPerMonth());  
            // increments current row count  
            recordCount++;  
            return true;  
        }  
  
        catch(Exception ex) {  
            logger.log(Level.FINE, ex.toString());  
        }  
        return false;  
    }  
}
```

6.10.2 Referencing the Java Event Handler in the Report Design

The Java Event Handler should be declared in the Data Set of the Report Design.



6.10.3 Testing the Report

When the report is run, the data from the CSV should be displayed.

CustomerNumber	NumberOfYears	VisitsPerMonth
103	1	32
112	2	22
114	3	24
119	4	43
121	4	65
124	2	45
125	3	108
128	1	32
129	2	76
131	4	85
141	3	34
144	2	65
145	1	34
146	2	23
148	4	35
151	4	68
157	2	98
161	2	54
166	4	34

6.11 Using POJOs, Spring, Hibernate and other Java Frameworks with BIRT

Using a Javascript or Java Event Handler “Scripted Data Source” allows the developer to leverage existing Java Frameworks to access their data. Many teams already have a data access layer that they want BIRT to use, or they want to be able to retrieve the data on their own, and feed in a populated object into the Report for formatting. This is all possible using the Scripted Data Source and Data Set. This topic is out of scope for this book, but additional resources are listed below.

See section 6.9 for more on how to leverage existing POJO's by using a Scripted Data Source.

For more information on General Data Access:

http://wiki.eclipse.org/BIRT/FAQ/Data_Access

POJO's

http://wiki.eclipse.org/BIRT/FAQ/Data_Access#Q:_How_do_I_get_data_from_a_POJO_.28Plain_Old_Java_Object.29.3F

<http://www.vogella.de/articles/EclipseBIRT/article.html>

Hibernate

<http://yekmer.blogspot.com/2008/11/how-to-integrate-eclipse-birt-with.html>

http://dieroster.blogspot.com/2008/05/integrating-birt-with-hibernate-and_30.html

Spring

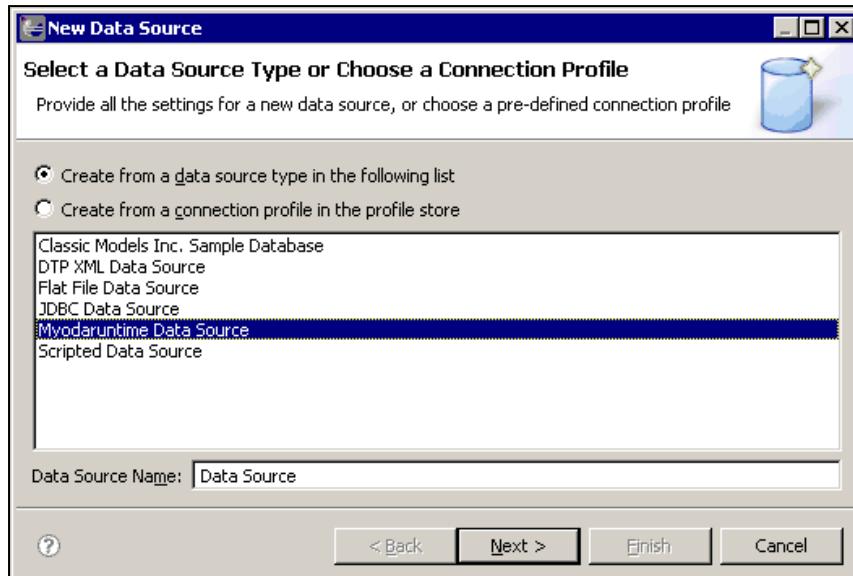
<http://www.birt-exchange.org/devshare/designing-birt-reports/1100-calling-spring-objects-from-the-expression-builder/#description>

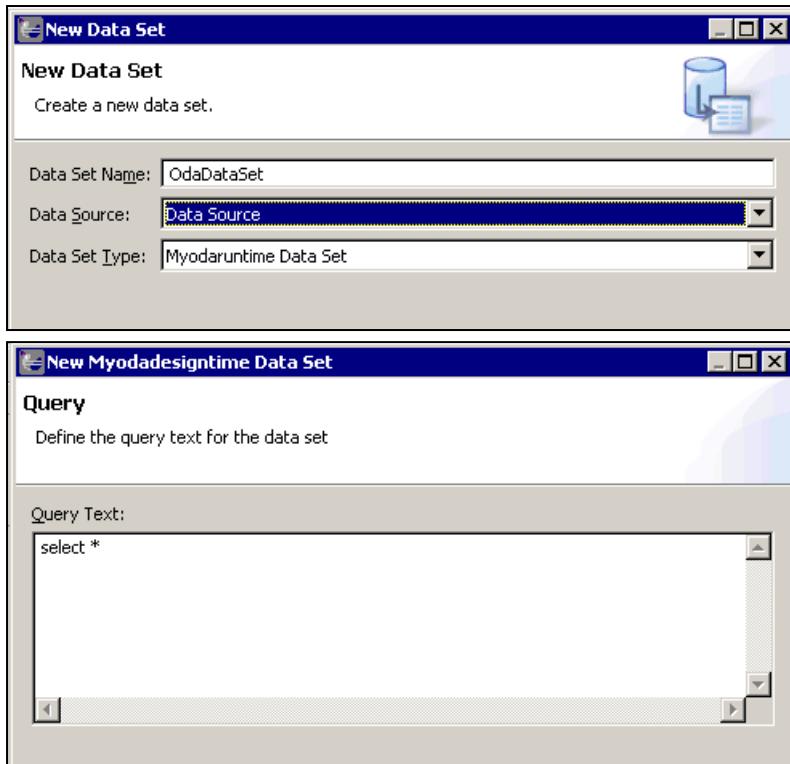
<http://birtworld.blogspot.com/2009/11/accessing-spring-beans-from-birt.html>

<http://birtworld.blogspot.com/2009/11/more-on-birt-and-spring.html>

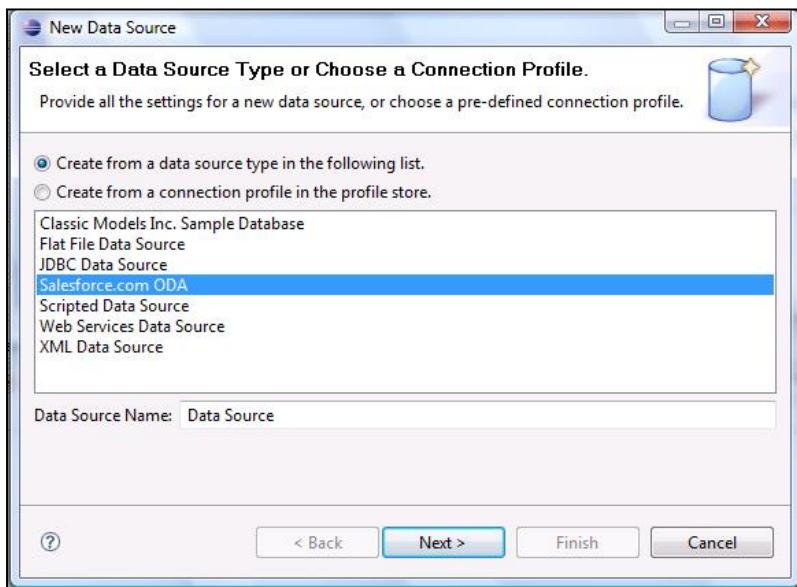
6.12 Other forms of Open Data Access – ODA

BIRT also offers the ability for a developer to create their own ODA (Open Data Access) Data Source plug-in. What this means is that a developer can create an Eclipse BIRT plugin that is specific to their custom Data Source. This allows the developer to add their own Data Source to the “Create Data Source” wizard. It also allows the developer to create their own custom Data Set and Data Set Properties Interface.





Another example is the Salesforce ODA Plugin specifically for accessing Salesforce data through BIRT



The ability to create ODA plug-ins are great for software companies who want to embed BIRT into their own software or if a company has a custom Data Source that they want to provide an easy-to-use interface for accessing the data for reporting.

It is outside the scope of this book to go into implementation details of creating ODA plug-ins, however the following resources will help guide you in the right direction.

An example on how to create a simple ODA Driver

<http://www.birt-exchange.org/devshare/interactive-reporting/129-using-oda-plug-in-wizard-to-create-a-simple-oda-driver/#description>

SalesForce BIRT ODA Driver

<http://www.birt-exchange.com/be/marketplace/app-showcase/?app=38>

An open-source company named OpenMRS who created a BIRT plugin-in for their data source

http://openmrs.org/wiki/BIRT_ODA_Plugin_Project

A good overview of connecting to any kinds of data in BIRT

http://www.birt-exchange.org/wiki/Connecting_to_Data_in_the_BIRT_Designer/

XML/A Driver for MS SQL Server Analysis, SAP, Essbase and more

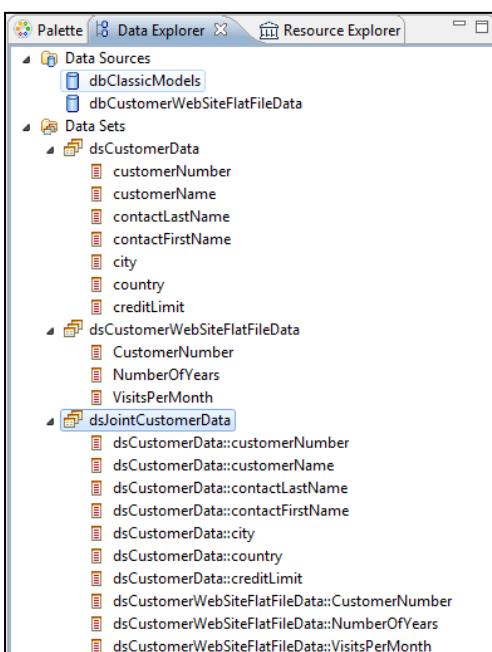
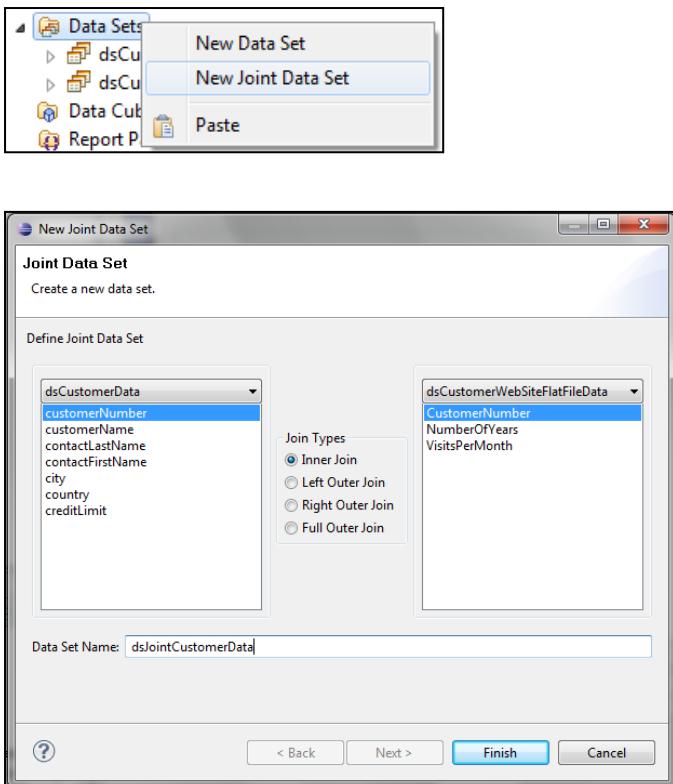
<http://www.birt-exchange.org/devshare/designing-birt-reports/1173-xml-for-analysis-data-connector/#description>

Using JSON in a Scripted Data Source

<http://www.birt-exchange.org/devshare/designing-birt-reports/1053-scripted-data-source-using-json/#description>

6.13 Joint Data Sets

Joint Data Sets are a feature of BIRT that allows developers to join two or more Data Sets together. This can be extremely useful in joining Data Sets together from completely different sources. An example of this would be to join the data from a flat file to data from a database.



Customer Number	Customer Name	Contact Last Name	Contact First Name	City	Country	Credit Limit	Number Of Years	Visits Per Month
103	Atelier graphique	Schmitt	Carine	Nantes	France	21000	1	32
112	Signal Gift Stores	King	Jean	Las Vegas	USA	71800	2	22
114	Australian Collectors, Co.	Ferguson	Peter	Melbourne	Australia	117300	3	24
119	La Rochelle Gifts	Labrune	Janine	Nantes	France	118200	4	43
121	Baane Mini Imports	Bergulsen	Jonas	Stavern	Norway	81700	4	65
124	Mini Gifts Distributors Ltd.	Nelson	Susan	San Rafael	USA	210500	2	45

***Tip**

It is never recommended to join two Data Sets from the same Database. If you are joining two Data Sets from the same database, create a joined SQL statement so the database does the join instead of BIRT. Only when the Data Sources are completely unrelated is it okay.

* One limitation of a Joint Data Set is that it only allows the Data Sets to be joined on one Data Field. If you want to join the Data Sets on more than one field, you will need to create a composite key field and join the Data Sets on that. Create a calculated column in each Data Set that creates the Composite Key. Then join the Data Sets together on the Composite Key field.

Joint Data Sets are very useful when joining data from completely different sources. Be aware though that BIRT is retrieving all of the data from both sources, and then joining them together in memory. For large data sets, this can result in slow performance.

If you want to join more than 2 data sets together, you will have to first create a Joint Data Set with two of the data sets. Then, join the Joint Data Set with another Data Set, producing a second Joint Data Set.

6.14 Other Data Access Using Actuate BIRT

The Actuate commercial version of BIRT offers a few more methods of accessing data. One is using a *rptdocument* as a Data Source for another report. When a BIRT report is run, the developer can save the output into a neutral format called “*rptdocument*”. From this format, BIRT can then convert it into HTML, Word, Excel, and others. BIRT can also use this type of output (*rptdocument*) as a Data Source for another report design. This can be useful for caching data from the database into an *rptdocument* and using it at a later time.

Another Data Source method is to use Actuate's Information Objects. Information Objects allows a developer to create a layer of MetaData about their data on the BIRT iServer. BIRT can use these Information Objects in a query to retrieve the report data.

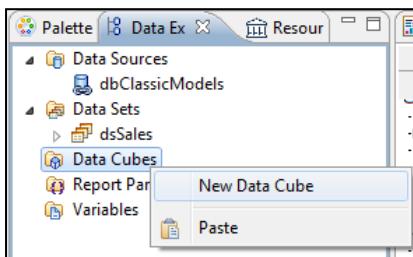
6.15 Using Data Cubes

Data Cubes are used as a Data Source for the Cross Tab Item. Data Cubes allow the developer to organize the data from a Data Set into a grouped format that fits into a Cross Tab structure. A Cross Tab Item cannot use a Data Set directly. A Data Cube must be created, using an existing Data Set, for the Cross Tab to reference.

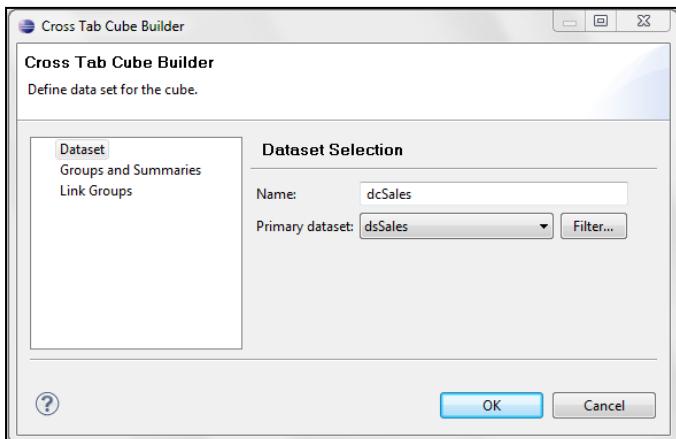
6.15.1 Creating a Data Cube

In the Designer, create a Data Set containing the data you want to use for the Cross Tab.

Right click on Data Cube and select New



Select the Data Set



In Groups and Summaries, choose how you want to group and sum your data.

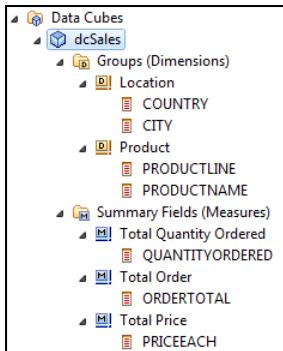
Create Groups using the “Add” button. Or, when dropping a field on the Groups icon, it will prompt you to create one automatically.

It is recommended to create Groups based on a business purpose. In this example, a group for “Location” and a group for “Product” were created. The related data fields were dragged and dropped under the appropriate group.

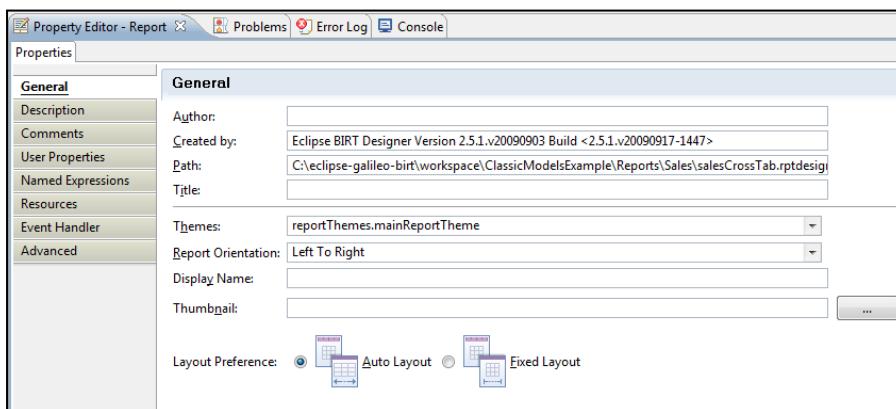
The screenshot shows the 'Groups and Summaries' interface. On the left, under 'Available Fields', there is a list of data fields from the 'dsSales (Primary)' table, including ORDERNUMBER, PRODUCTCODE, QUANTITYORDERED, PRICEEACH, ORDERLINENUMBER, ORDERTOTAL, orderDate, STATUS, CUSTOMERNUMBER, COUNTRY, STATE, CITY, CUSTOMERNUMBER_13, PRODUCTNAME, PRODUCTLINE, and PRODUCTCODE_16. Two arrows (right and left) are positioned between the two columns. On the right, under 'Groups and Summaries', there are two main categories: 'Groups (Dimensions)' and 'Summary Fields (Measures)'. Under 'Groups (Dimensions)', there are two groups: 'Location' (containing COUNTRY and CITY) and 'Product' (containing PRODUCTLINE and PRODUCTNAME). Under 'Summary Fields (Measures)', there is one summary field: 'Total Quantity Ordered' (containing QUANTITYORDERED(SUM)). On the far right, there are three buttons: 'Add', 'Edit', and 'Delete'.

If the Data Fields are related, then add them to groups based on business purpose. If not, create new groups for each one and label appropriately.

This screenshot shows the same 'Groups and Summaries' interface as the previous one, but with more summary fields added under 'Summary Fields (Measures)'. The 'dsSales (Primary)' table fields are listed on the left, and the 'Groups and Summaries' section on the right now includes 'Total Order' (containing ORDERTOTAL(SUM)) and 'Total Price' (containing PRICEEACH(SUM)). The 'Add', 'Edit', and 'Delete' buttons are also present on the right side.

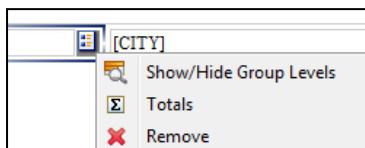


If you expect a large cross tab report, be sure to set your report to “Auto Layout”. Otherwise, the cross tab will be restricted to the width of the page as defined by your master page.



Drag and Drop the required fields onto the Report Design.

Use the following icon to select/unselect group levels.

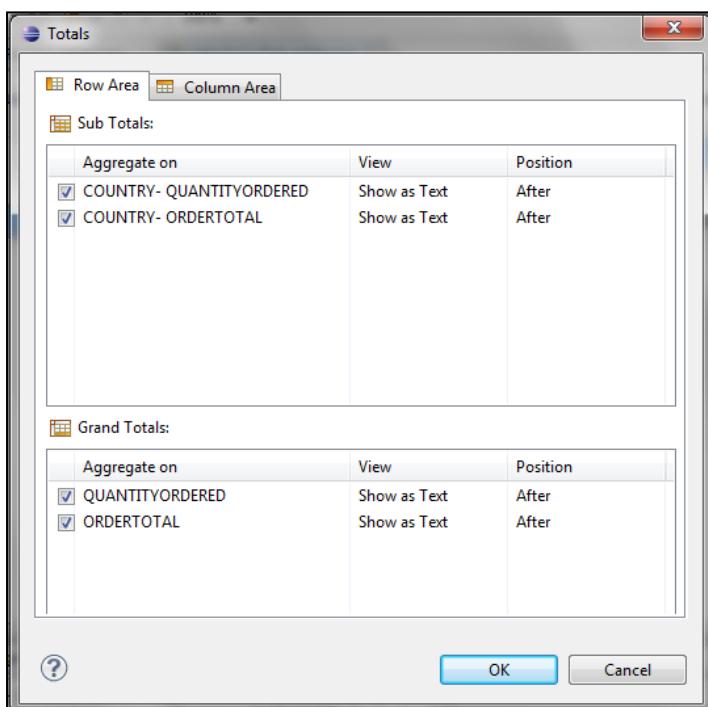
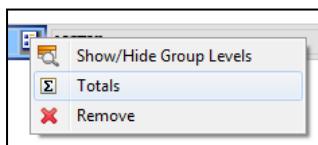


6.15.2 Formatting Cross Tab Items

Use Styles to format accordingly

Classic Model Cars Inc Detroit, MI																
	Classic Cars			Motorcycles		Planes		Ships		Trains		Trucks and Buses		Vintage Cars		
	QTY Ordered	Order Total	QTY Ordered	Order Total	QTY Ordered	Order Total	QTY Ordered	Order Total	QTY Ordered	Order Total	QTY Ordered	Order Total	QTY Ordered	Order Total	QTY Ordered	Order Total
Australia	Chatswood	387.00	34,840.67	46.00	4,825.28	243.00	18,877.85	24.00	2,063.76		162.00	16,035.78	739.00	57,263.78		
	Glen Waverly	119.00	11,072.13			63.00	3,094.77	32.00	2,346.24				491.00	39,352.88		
	Melbourne	451.00	48,009.87	490.00	50,014.36	419.00	35,204.86					166.00	16,326.52	400.00	31,029.46	
	North Sydney	744.00	79,040.51	219.00	16,401.80						286.00	27,396.76	220.00	14,195.15		
Austria	South Brisbane	117.00	15,002.29	121.00	13,341.75	88.00	8,090.56			33.00	1,886.61	91.00	10,085.92	95.00	6,783.03	
	Graz	202.00	21,533.09								203.00	19,562.53	127.00	9,964.37		
	Salzburg	735.00	79,993.29	197.00	18,039.49	200.00	16,063.72	113.00	7,712.17				197.00	15,671.40		
Belgium	Brussels	30.00	4,080.00					315.00	25,590.98	70.00	6,187.56			381.00	34,993.04	
	Charleroi	117.00	14,379.90			41.00	5,624.79	28.00	2,264.08	27.00	1,627.56			65.00	5,320.85	
Canada	Montréal	256.00	31,735.22					218.00	16,472.01			166.00	15,959.17	77.00	4,811.27	
	Tsawassen	25.00	3,264.00	41.00	3,726.90	317.00	23,540.47	268.00	20,133.15				222.00	16,147.48		
	Vancouver	175.00	24,660.46								351.00	30,406.54	177.00	15,055.19		

Click on the small icon and display/hide Totals



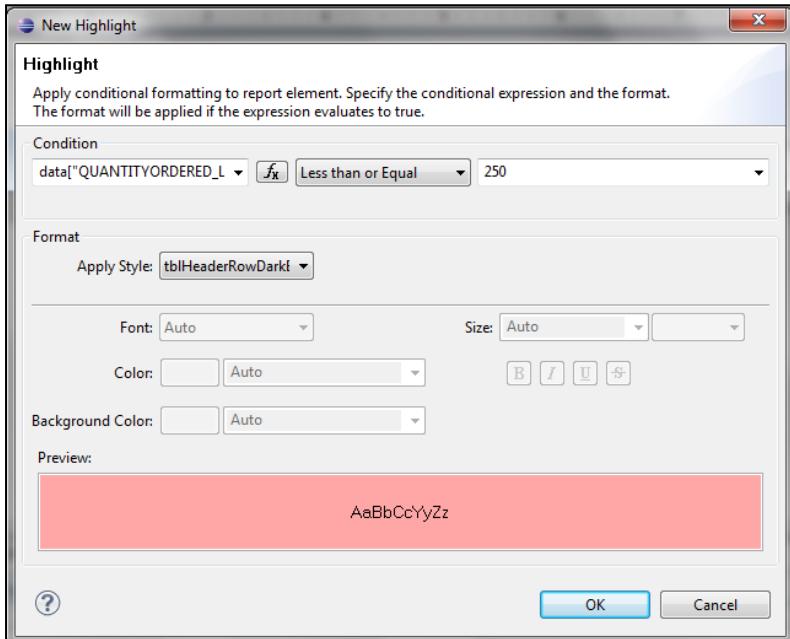
		Classic Model Cars Inc															
		Detroit, MI				Cars				Trucks and Buses				Vintage Cars		Grand Total	
		Classic Cars		Motorcycles		Planes		Ships		Trains		Trucks and Buses		Vintage Cars		Grand Total	
		Qty Ordered	Order Total	Qty Ordered	Order Total	Qty Ordered	Order Total	Qty Ordered	Order Total	Qty Ordered	Order Total	Qty Ordered	Order Total	Qty Ordered	Order Total		
Australia	Chatswood	387	34,840.67	46	4,825.28	243	18,877.85	24	2,063.76			162	16,035.78	739	57,263.78	1,601	133,907.12
	Glen Waverly	119	11,072.13			63	3,094.77	32	2,346.24					491	39,352.88	705	55,866.02
	Melbourne	451	48,009.87	490	50,014.36	419	35,204.86					166	16,326.52	400	31,029.46	1,926	180,585.07
	North Sydney	744	79,040.51	219	16,401.80							286	27,396.76	220	14,195.15	1,469	137,034.22
	South Brisbane	117	15,002.29	121	13,341.75	88	8,090.56			33	1,886.61	91	10,085.92	95	6,782.03	545	55,190.16
	Australia Total	1,818	187,965.47	876	84,583.19	813	65,268.04	56	4,410.00	33	1,886.61	705	69,844.98	1,945	148,624.30	6,246	562,582.59
Austria	Graz	202	21,533.09									203	19,562.53	127	9,964.37	532	51,059.99
	Salzburg	735	79,993.29	197	18,039.49	200	16,063.72	113	7,712.17					197	15,671.40	1,442	137,480.07
	Austria Total	937	101,526.38	197	18,039.49	200	16,063.72	113	7,712.17			203	19,562.53	324	25,635.77	1,974	188,340.06

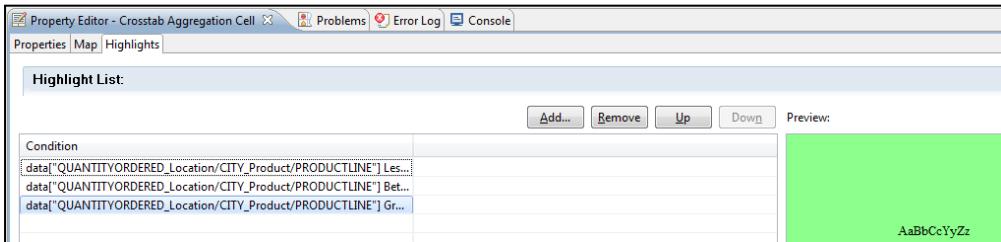
Use conditional formatting to highlight cells and font color.

Expression Builder

Type an expression in the Expression field. Browse the lists of available objects and double click on them to insert them into the expression.

1 data["QUANTITYORDERED_Location/CITY_Product/PRODUCTLINE"]





Classic Model Cars Inc
Detroit, MI



		Classic Cars		Motorcycles		Planes		Ships		Trains		Trucks and Buses		Vintage Cars	
		Qty Ordered	Order Total	Qty Ordered	Order Total	Qty Ordered	Order Total	Qty Ordered	Order Total	Qty Ordered	Order Total	Qty Ordered	Order Total	Qty Ordered	Order Total
Australia	Chatswood	387	34,840.67	46	4,825.28	243	18,877.85	24	2,063.76			162	16,035.78	739	57,263.78
	Glen Waverly	119	11,072.13			63	3,094.77	32	2,346.24					491	39,352.88
	Melbourne	451	48,009.87	490	50,014.36	419	35,204.86					166	16,326.52	400	31,029.46
	North Sydney	744	79,040.51	219	16,401.80							286	27,396.76	220	14,195.15
	South Brisbane	117	15,002.29	121	13,341.75	88	8,090.56			33	1,886.61	91	10,085.92	95	6,783.03

6.15.3 Using Charts in the Cross Tab

Cross Tabs support displaying charts in the report. Click on the small icon and choose “Display as Chart”.

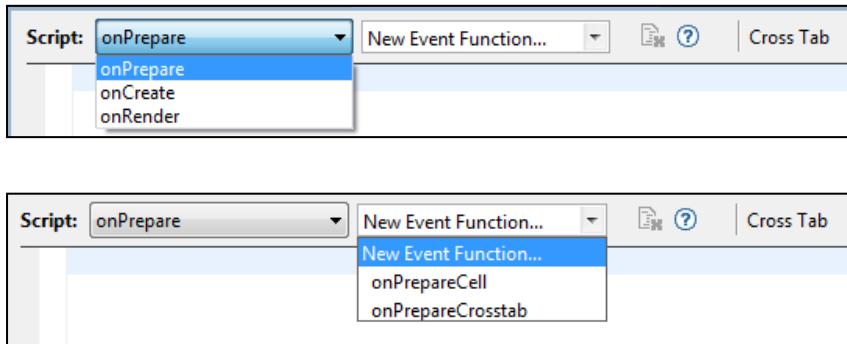
The screenshot shows a cross-tab report with columns for Country, City, Product Line, and various order metrics. A context menu is open over a chart area, with the '2 Show as Chart' option selected. Below the report, a bar chart is displayed showing quantity ordered by product line.

		[COUNTRY]		[CITY]		[PRODUCTLINE]		Grand Total	
		Qty Ordered	Order Total						
[COUNTRY]	[CITY]	[QUANTITYORDERED]	[ORDERTOTAL]	[QUANTITYORDERED]	[ORDERTOTAL]	[QUANTITYORDERED]	[ORDERTOTAL]	[QUANTITYORDERED]	[ORDERTOTAL]
Grand Total		[QUANTITYORDERED]	[ORDERTOTAL]	[QUANTITYORDERED]	[ORDERTOTAL]	[QUANTITYORDERED]	[ORDERTOTAL]	[QUANTITYORDERED]	[ORDERTOTAL]

Double-click on the chart to see additional chart options.

6.15.4 Cross Tab Scripting

The Cross Tab Item now supports manipulation via scripting. Click on the item and click on the Script tab.



6.15.5 Additional Cross Tab Resources

There are plenty of tutorials and additional information Cross Tabs available.

http://www.birt-exchange.org/documentation/BIRT_231/wwhelp/wwimpl/js/html/wwhelp.htm#href=crossstab.17.1.html

<http://www.birt-exchange.org/devshare/designing-birt-reports/202-birt-cross-tab-example-basic-/#description>

6.16 So what is the best way?

After describing all of the possible methods of retrieving data in BIRT, what is Le BIRT Expert's recommend way?? Well, it all depends on your Data Source!

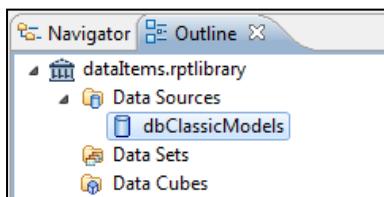
Each section talks about the best way to access your data. If your data is coming from a Web Service, then use the Web Service Data Source. If you have an existing Data Access framework that you want to re-use, use a Scripted Data Source. If you're using a typical Relational Database, use JDBC with stored procedures.

No matter what technique you use to access your data, the best practices associated with that method have been covered in this chapter. Using this information, you will have to determine what method of data access is best for your application and Data Source.

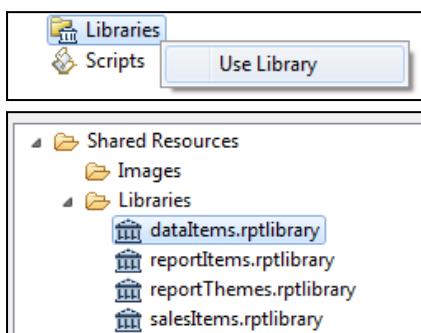
6.17 Best Practice - Putting data sources and data sets into libraries

No matter what technique you have chosen to access your data, it is absolutely necessary to componentize it into a re-usable library. Each Report Design should be using the same method of accessing your data. Putting your Data Source items into a library is essential in making sure every report is accessing the data in the same way. It is also a single point of change for when database properties change.

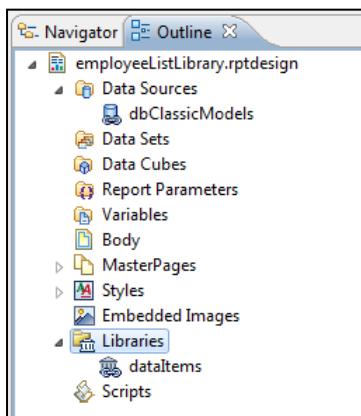
Open the “dataItems.rptlibrary”. Right-click on the Data Source to create a new Connection. Fill in the connection properties.



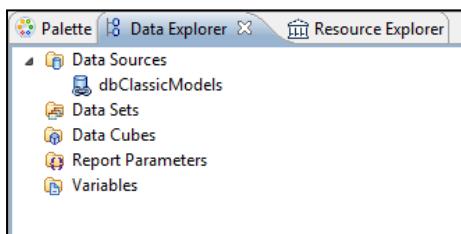
Create a new report. Use the Data Source from the library in the Report.



In the report, drag and drop the “dbClassicModels” item into the “Data Source” section of the Report Outline.

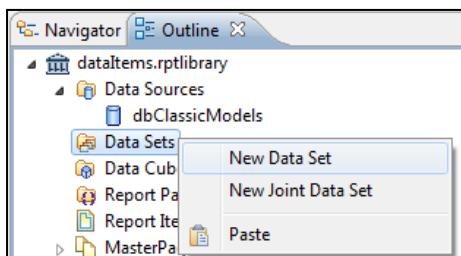


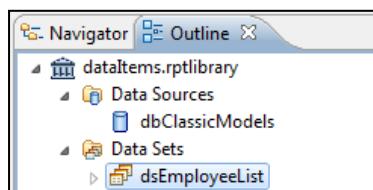
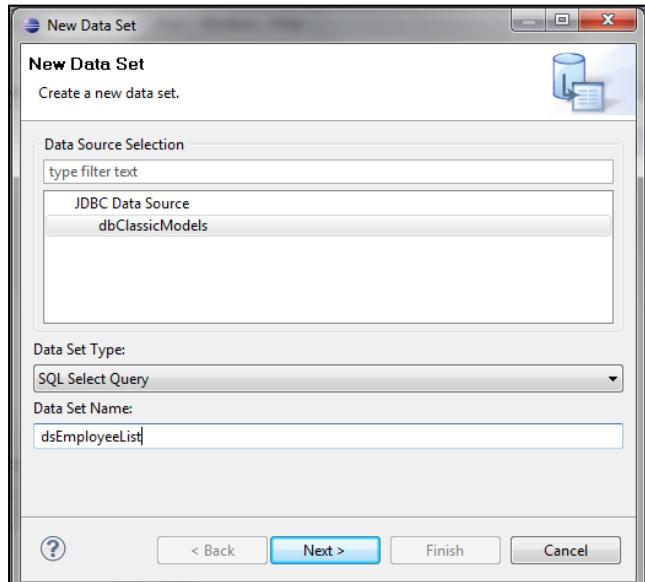
In the Data Explorer of the Report, you will see the Data Source from the Library.



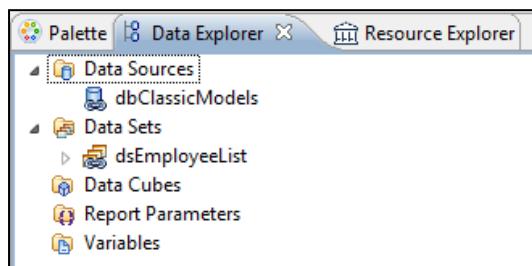
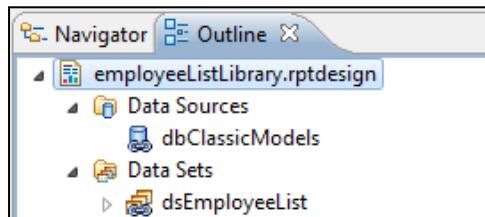
From this point on, the Report will use the Data Source that is defined in the library.

Create a Data Set in the Library.





Drag and drop this Data Set into the Data Sets section of the Report Outline

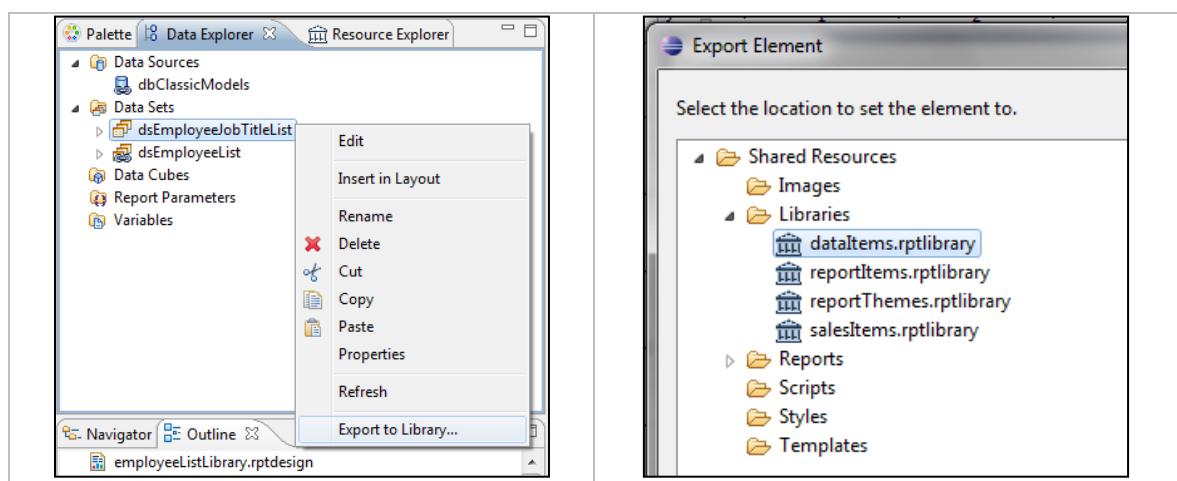


Employee Number	Last Name	First Name	Office Code	Reports To	Job Title
1002	Murphy	Diane	1		President
1056	Patterson	Mary	1	1002	VP Sales
1076	Firrelli	Jeff	1	1002	VP Marketing
1088	Patterson	William	6	1056	Sales Manager (APAC)
1102	Bondur	Gerard	4	1056	Sale Manager (EMEA)
1143	Bow	Anthony	1	1056	Sales Manager (NA)
1165	Jennings	Leslie	1	1143	Sales Rep

***Tip**

Should the developer add every Data Set to the library? Le BIRT Expert says no, it's not necessary. Only Data Sets that you will re-use in other reports should be put into the library. Sometimes, Data Sets are unique to an individual report. If this is the case, then no need to put it into a report. However, if you find that later down the road, you need to use the Data Set in another report, you can always export that Data Set to a library for re-use.

To export the Data Set to a Library ...



Employee Number	Last Name	First Name	Office Code	Reports To	Job Title
1002	Murphy	Diane	1		President
1056	Patterson	Mary	1	1002	VP Sales
1076	Firrelli	Jeff	1	1002	VP Marketing
1088	Patterson	William	6	1056	Sales Manager (APAC)
1102	Bondur	Gerard	4	1056	Sale Manager (EMEA)
1143	Bow	Anthony	1	1056	Sales Manager (NA)
1165	Jennings	Leslie	1	1143	Sales Rep

6.18 Best Practice - Externalizing connection properties

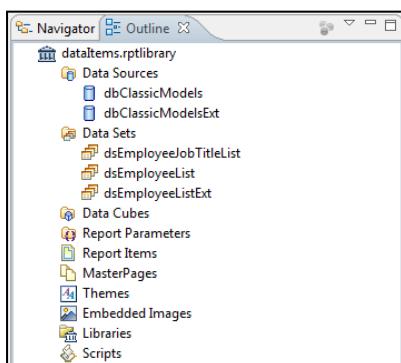
A common question among developers is how to externalize Data Source connection properties. When switching environments (developer's machines, development server, test servers, production), having a single point of change is essential in making this process go smoothly. Externalizing database properties should also provide a secure way to store the database password.

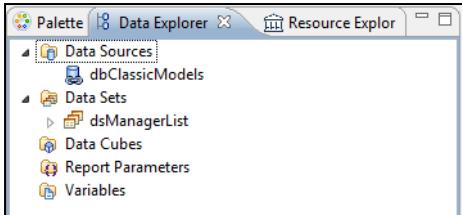
There are several ways to do this. This section will cover how to do it and the advantages/disadvantages of each approach.

- Externalize the Data Source in a Report Library
- Using BIRT's Default Option
- Using a JDBC Connection Pool
- Use JNDI
- Pass Properties Through the Report Context or Session
- Use a Scripted Data Source
- Use a Connection Profile
- Use a Custom Java Class Called by Scripting or Event Handler

6.18.1 Externalize the Data Source in the Report Library – Best Practice

No matter what technique you choose to externalize your Data Source properties, the Data Source Item itself should be in a Report Library for all reports to use. This technique provides a single point of change for any Data Source changes. When switching environments, the only file that will need to be updated is this Library file. The reports will automatically pick up the changes from the Library.





6.18.2 Using BIRT's Default Option

This is the easiest solution, provided out of the box. It works well enough for demo or test situations. The BIRT Designer will automatically encrypt the Data Source password using Base64 encryption. By externalizing the Data Source in the Report Library (as outlined in the previous section), it provides a single point of change for any database properties.

```
<encrypted-property name="odaPassword" encryptionID="base64">cGFzc3cwcmQ=</encrypted-property>
```

The Library file is just an XML file, so database properties can be set outside of the designer in a build script.

```
<data-sources>
  <oda-data-source extensionID="org.eclipse.birt.report.data.oda.jdbc" name="dbClassicModels" id="7">
    <list-property name="privateDriverProperties">
      <ex-property>
        <name>contentBidiFormatStr</name>
        <value>ILYNN</value>
      </ex-property>
      <ex-property>
        <name>metadataBidiFormatStr</name>
        <value>ILYNN</value>
      </ex-property>
    </list-property>
    <property name="odaDriverClass">com.mysql.jdbc.Driver</property>
    <property name="odaURL">jdbc:mysql://localhost/ClassicModels</property>
    <property name="odaUser">root</property>
    <encrypted-property name="odaPassword" encryptionID="base64">cGFzc3cwcmQ=</encrypted-property>
  </oda-data-source>
</data-sources>
```

The password can be reset using the report designer. Open the Library in the Report Designer, change the password, save. The encrypted value of the new password will be in the XML code.

The encrypted version of a new password can also be achieved using Java Code. This will be more practical for those that use a build/deploy script.

```
import org.eclipse.birt.report.model.metadata.SimpleEncryptionHelper;
import org.eclipse.birt.report.model.metadata.MetaDataDictionary;

SimpleEncryptionHelper eh = SimpleEncryptionHelper.getInstance( );
String encryptedPass = eh.encrypt("password");
```

When switching environments, the only file that will need to be updated is this Library file. If the database server, port, id or password is changed, only the Library file will need to be updated. The reports will automatically pick up the changes from the Library.

BIRT provides an extension point to override the default Encryption type.

The new encryption class must implement the *org.eclipse.birt.report.model.api.extension.IEncryptionHelper* Interface.

The new encryption class must be defined in plugin.xml:

```
<extension point="org.eclipse.birt.report.model.encryptionHelper">
    <encryptionHelper
        class="org.eclipse.test.encryptionHelper1"
        extensionName="org.eclipse.test.encryptionHelper1"/>
</extension>
```

This extension is not very well documented for Open Source BIRT, and only mentioned here because it is possible. It is not clear which plugin.xml file it needs to be added to. This approach is possible, but will need some additional research to get this to work.

http://www.birt-exchange.org/documentation/BIRT_220/EngineJavadoc/model/api/org/eclipse/birt/report/model/api/extension/IEncryptionHelper.html

http://dev.eclipse.org/viewcvs/index.cgi/source/org.eclipse.birt.doc.isv/ref/ext/org_eclipse_birt_report_model_encryptionHelper.html?root=BIRT_Project&view=co

*Note

The manual for Actuate Commercial BIRT provides a much better explanation about how the BIRT encryption works within Actuate BIRT. See *Chapter 14, Working with BIRT Encryption in Actuate Java Components*, in the *Using Actuate BIRT PDF* that comes with Actuate BIRT for more information on modifying the encryption in BIRT.

Disadvantages:

- 1) It's not completely externalized, as the connection properties are still located in a Report Library.
- 2) The default base64 encryption is not strong and can be easily decrypted.
- 3) Creating a BIRT extension point to provide a different type of encryption is not well documented for the Open Source version. More information specific to Actuate BIRT is available in the manual.

[6.18.3 Using a JDBC Connection Pool](#)

Using a Connection Pool in your BIRT report removes the need for BIRT to store the connection properties. The connection properties are stored in the JNDI Connection Pool properties. See section 6.8 for more information on connection pools.

Disadvantages:

- 1) The developer will still need to put the connection values in the Data Source properties when using the Designer.
- 2) This technique only works for JDBC Data Sources and those applications running in an Application Server.

[6.18.4 Use JNDI](#)

Even if you are not using a JDBC Connection Pool, you can still put your connection information in JNDI. Use a Java class to retrieve the information from JNDI and pass it into the Data Source properties.

Disadvantages:

- 1) This technique only works for those applications running in an Application Server.
- 2) You will still have to enter the connection details as normal in the Designer

6.18.5 Pass Data Source Connection Properties Through the Application Context

This technique requires code to set the values in the Application Context before the BIRT Runtime Engine is used to generate the Reports. This also works if passed through the HTTP Session. It will also require a little Scripting to pick up the values and place them in the Data Source. The password should be sent in encrypted, and BIRT should decrypt it using a Java Class it can call in the Script.

Disadvantages:

- 1) Only works when Report execution is handled by your custom BIRT Runtime Engine. This is not possible when running in the BIRT report designer or on the BIRT iServer.

6.18.6 Use a Scripted Data Source

A Scripted Data Source gives the developer complete control on how the Data Source connection is managed. Whether it is through opening the connection manually, using another data-access framework, or working with objects passed in through the Application Context – the developer can implement the best and most secure way to access the Data Source. The developer can decide how and where to store the password and other connection details. They can use existing encryption/decryption classes to secure the password and other details.

Disadvantages:

- 1) Creating a scripted data source is a little more cumbersome than the other techniques. It requires the data source to be manually handled, which may not be ideal for all data sources. It would be overkill for normal database reporting.

6.18.7 Using a Connection Profile

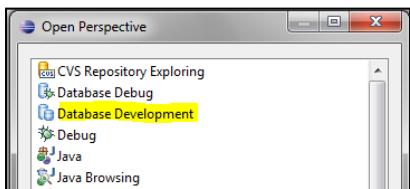
As mentioned earlier in this chapter, a developer can create a Connection Profile through Eclipse that can work across Eclipse Plugins, including BIRT. It provides a way to externalize connection properties in a binary file and offers a default encryption method.

There are two different types of connections to create in a connection Profile. The “Database Connection” profile is used only with the DTP SQL Select Query Prototype Builder. The “ODA” (Open Data Access) Data Sources is used with normal BIRT Data Sets (SQL and Stored Procedures).

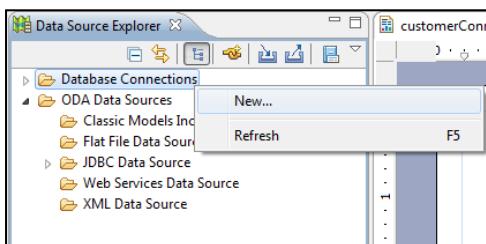
The Database Connection Profile has limited use, however, the ODA Data Source Connection Profile is a simple and very convenient way to externalize your connection properties.

6.18.7.1 Database Connection Profile

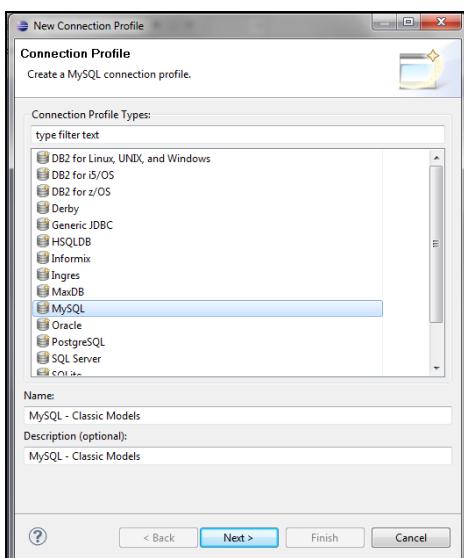
The Database Connection Profile can be created in the “Database Development” perspective of Eclipse. Switch to this Perspective.



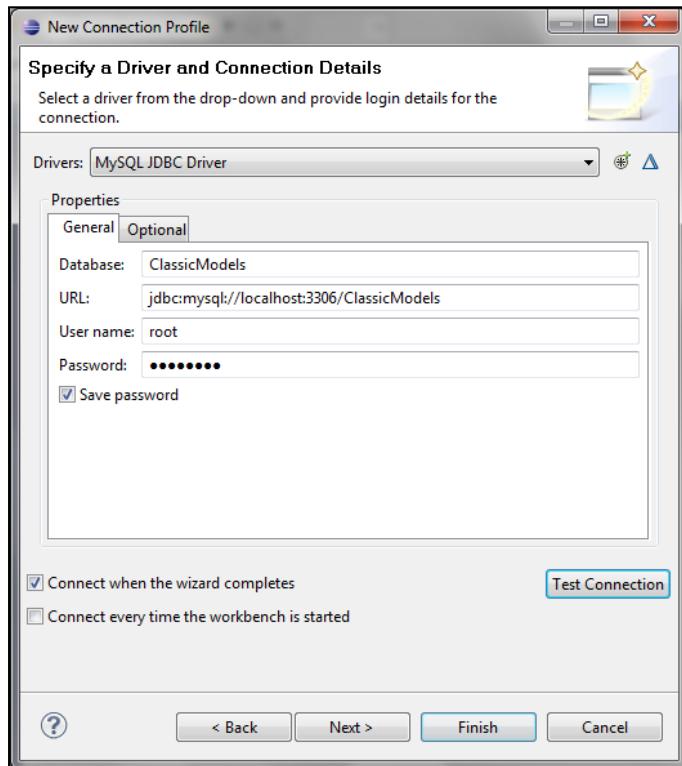
In the Data Source Explorer, right click on “Database Connections” and select “New..”.



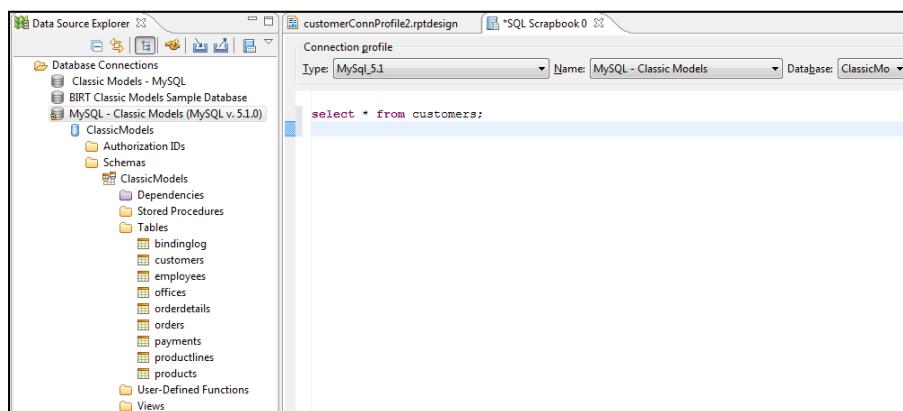
Choose MySQL and enter the name of your connection



Enter in the connection properties.



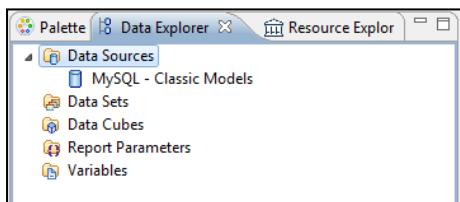
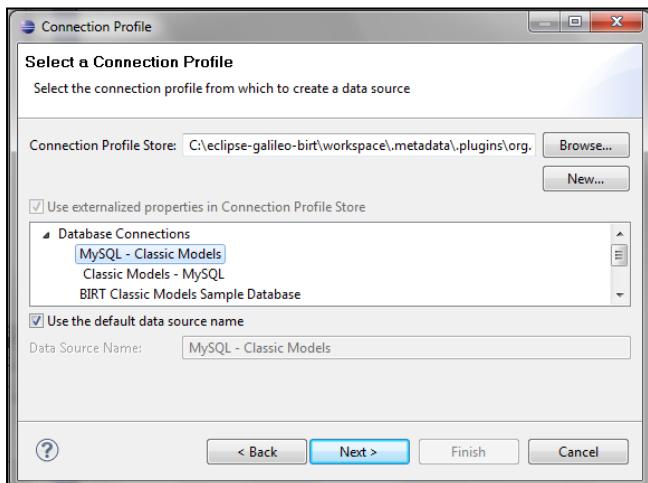
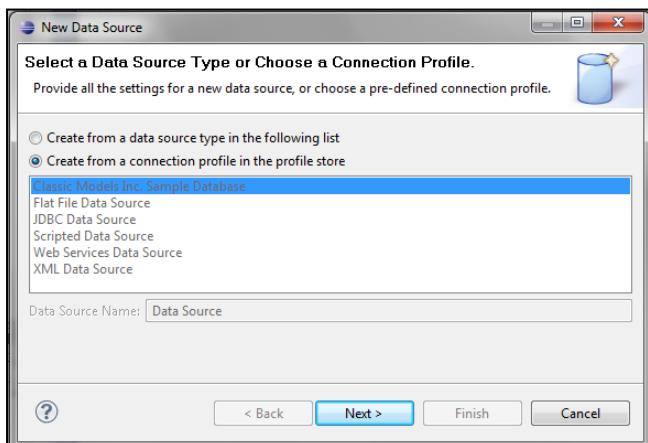
In the new profile, the developer can browse the database through the Data Source Explorer. SQL can also be executed using the SQL Scrapbook.



This profile has been saved, by default, in the following file:

C:\<birt directory>\workspace\.metadata\.plugins\org.eclipse.datatools.connectivity\Server Profiles.dat

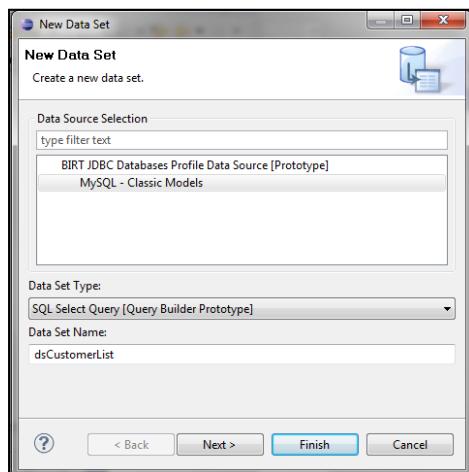
In the Report Library (or Design or Template), right click on Data Sources to create a new connection. Choose “Create from a connection profile in the profile store”. Under “Database Connections”, choose the Database Connection you created earlier. Click Next and then Finish.



The Base64 version of the password is still stored in the XML. This is just used by the Designer. When the report executes, it will retrieve the connection information from the connection profile.

```
<property name="odaDriverClass">com.mysql.jdbc.Driver</property>
<property name="odaURL">jdbc:mysql://localhost:3306/ClassicModels</property>
<property name="odaUser">root</property>
<encrypted-property name="odaPassword" encryptionID="base64">cGFzc3cwcmQ=</encrypted-
property>
<property name="OdaConnProfileName">MySQL JDBC - Classic Models</property>
<property name="OdaConnProfileStorePath">C:\eclipse-galileo-
birt\workspace\.metadata\.plugins\org.eclipse.datatools.connectivity\ServerProfiles.dat</
property>
```

When creating a new Data Set, the developer is limited to only using the DTP SQL Query Builder Prototype.

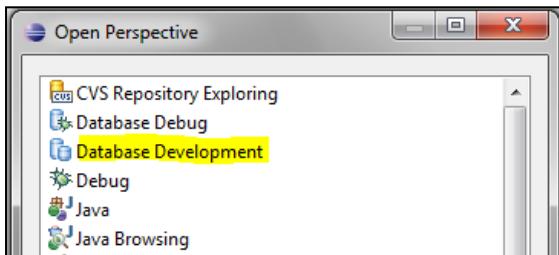


It is Le BIRT Expert's opinion that unless you are doing Eclipse Plug-In development and need to share connection properties, or plan to use the SQL Query Builder Prototype, using a Database Connection Profile is more trouble than it's worth for BIRT Reports. This feature might be improved on in future versions. (If you want to use the DTP SQL Query Builder, but do not want to use a Database Connection profile, then upgrade to BIRT 2.5.2 or higher) In the meantime, using an ODA Data Source (next section) should provide a much better solution.

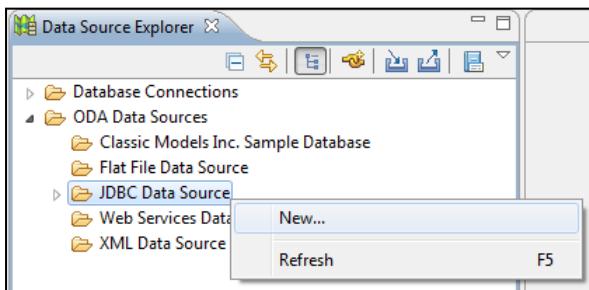
More information on Database Connection Profiles and the Query Builder a can be found at:
<http://www.birt-exchange.org/devshare/designing-birt-reports/782-eclipsecon-2009-using-and-extending-eclipse-data-tools-dtp/#description>

6.18.7.2 ODA Data Source Profile

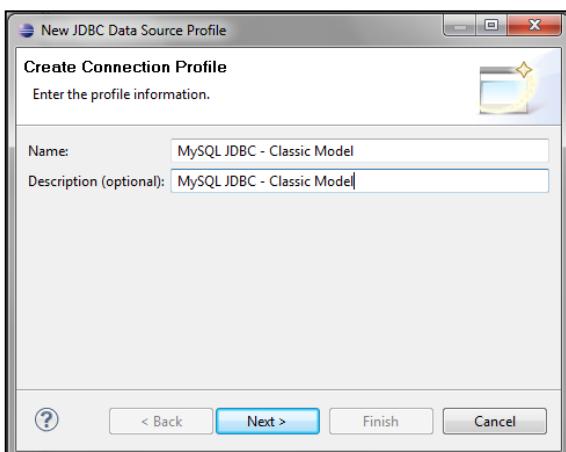
If you are using a Connection Profile, this is the recommended approach. The ODA Data Source Connection Profile can be created in the “Database Development” perspective of Eclipse. Switch to this Perspective.



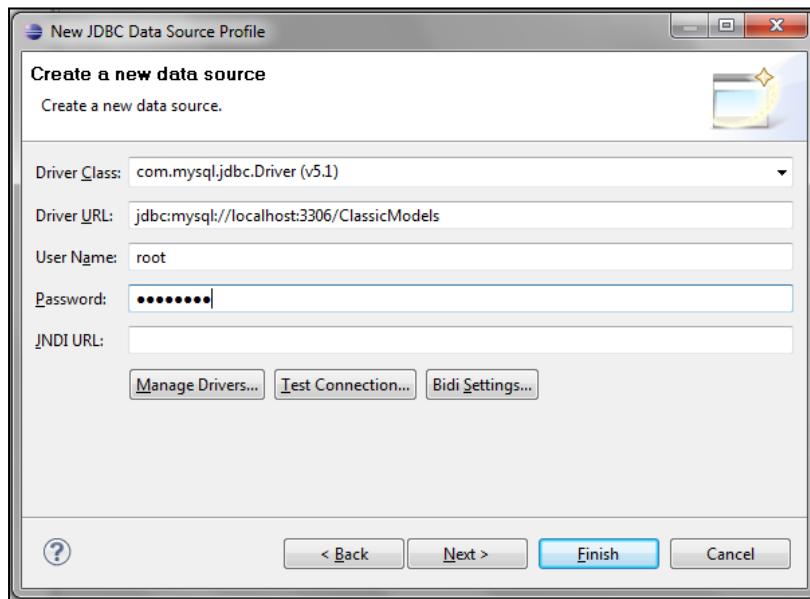
Under the “ODA Data Sources” section, right click on “JDBC Data Source” and click “New ...”. (Flat File, Web Services and XML Data Sources are supported too.).



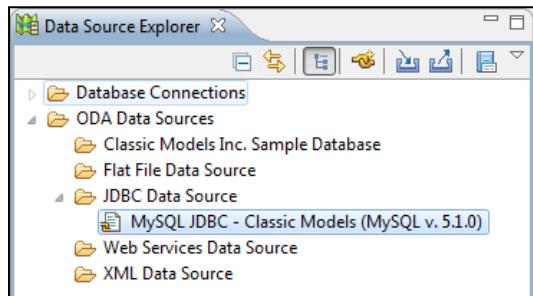
Enter a name for your profile



Enter in the connection information



Click Finish



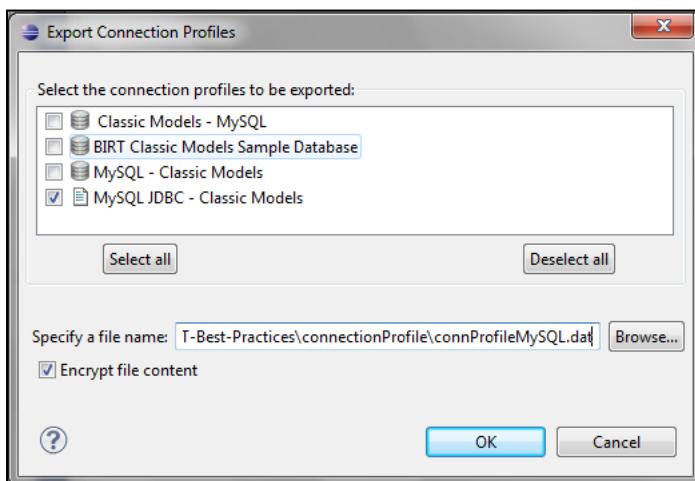
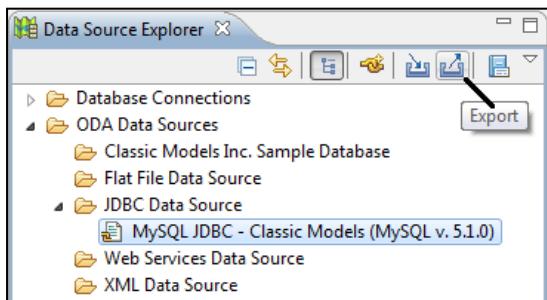
***Tip**

By default, Eclipse will save the connection properties in the connection profile file, located at:

C:\<birth
directory>\workspace\.metadata\.plugins\org.eclipse.datatools.connectivity\Server
Profiles.dat

Once you have created your Connection Profile file, it is recommended that you export the connection profile to another location.

In the Data Source Explorer, click the little export icon at the top right corner of the tab.



Check the box to encrypt the file content. It is not known what kind of encryption is used.

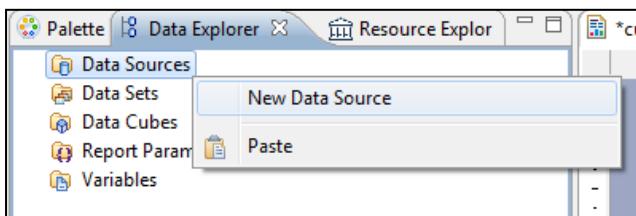
If the file is not encrypted, it will look like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<DataTools.ServerProfiles version="1.0">
<profile autoconnect="No" desc="" id="fb1d5e70-e9b2-11de-a62e-f4ac418e24df" name="dbJDBConnMySQL">
<baseproperties>
    <property name="org.eclipse.datatools.connectivity.db.connectionProperties" value="" />
    <property name="org.eclipse.datatools.connectivity.db.savePWD" value="true" />
    <property name="org.eclipse.datatools.connectivity.drivers_defnType" value="org.eclipse.d...
    <property name="jarList" value="C:\LeBirtExpert\BIRT-Best-Practices\mysql-connector-java-...
    <property name="org.eclipse.datatools.connectivity.db.username" value="root" />
    <property name="org.eclipse.datatools.connectivity.db.driverClass" value="com.mysql.jdbc...
    <property name="org.eclipse.datatools.connectivity.db.databaseName" value="ClassicModels"...
    <property name="org.eclipse.datatools.connectivity.driverDefinitionID" value="DriverDefin...
    <property name="org.eclipse.datatools.connectivity.db.password" value="passw0rd" />
    <property name="org.eclipse.datatools.connectivity.db.version" value="5.1" />
    <property name="org.eclipse.datatools.connectivity.db.URL" value="jdbc:mysql://localhost:...
    <property name="org.eclipse.datatools.connectivity.db.vendor" value=" MySql" />
</baseproperties>
<org.eclipse.datatools.connectivity.versionInfo>
    <property name="server.version" value="5.1.0" />
    <property name="technology.name.jdbc" value="JDBC" />
    <property name="server.name" value="MySQL" />
    <property name="technology.version.jdbc" value="3.0.0" />
</org.eclipse.datatools.connectivity.versionInfo>
<driverreference>
    <property name="driverName" value="MySQL JDBC Driver" />
    <property name="driverTypeID" value="org.eclipse.datatools.enablement.mysql.5_1.driverTem...
</driverreference>
</profile>
</DataTools.ServerProfiles>
```

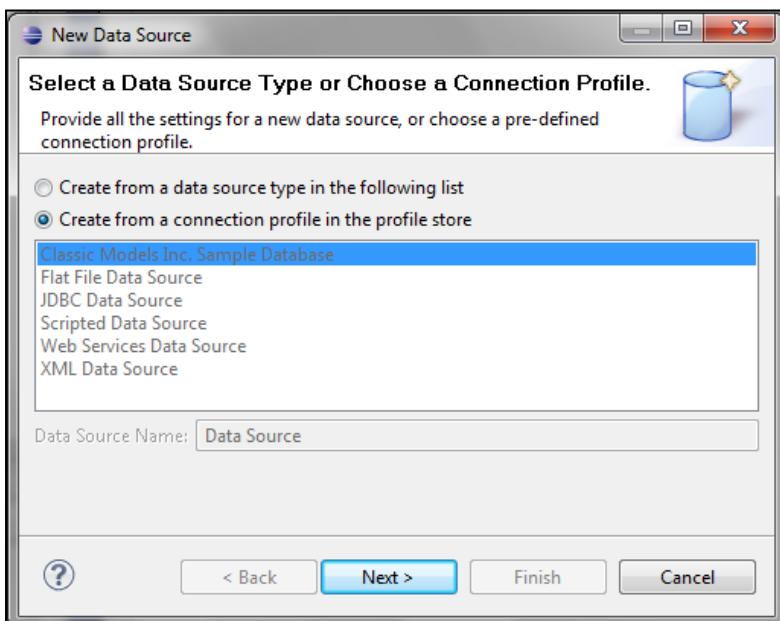
Establishing a Secure and Consistent Way to Access Data

If it is encrypted, it will look like this when opened in a text editor.

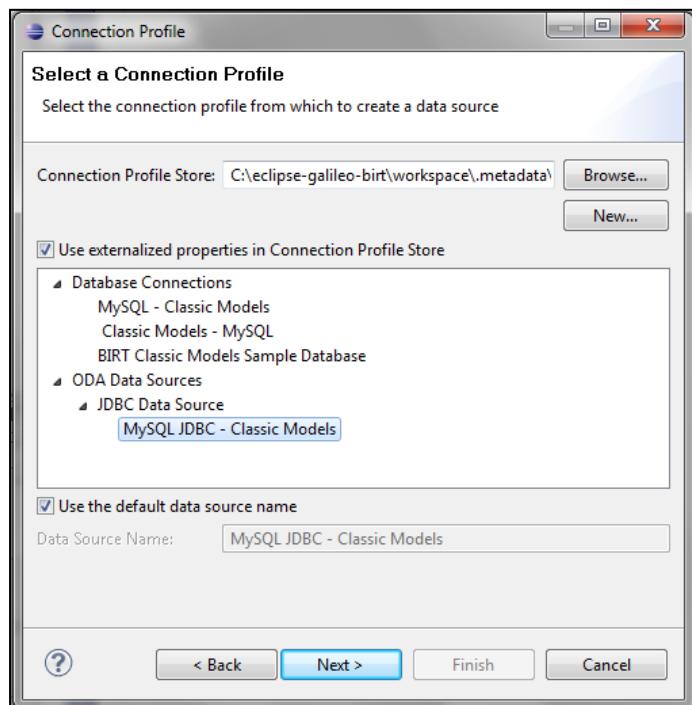
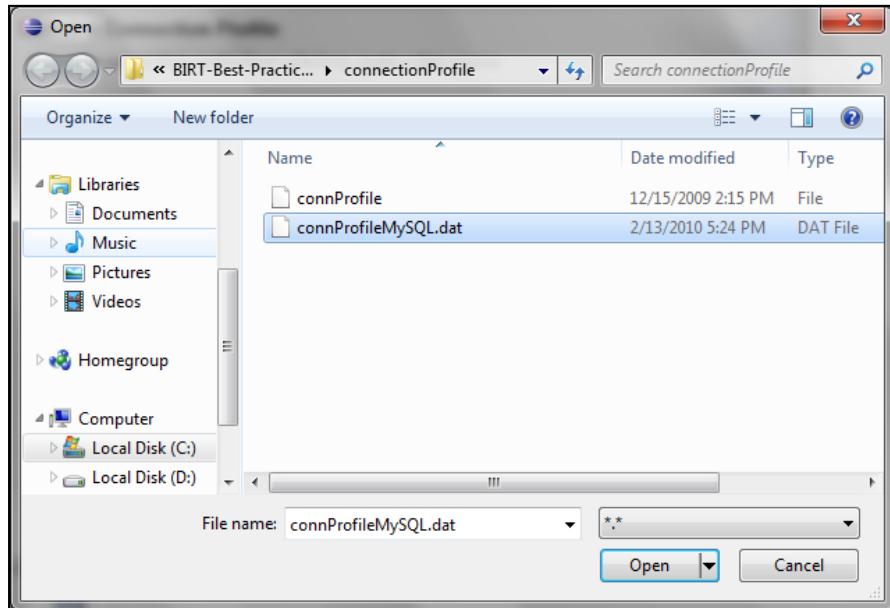
In the Report Library (or Template or Design), right click on Data Source and select “New Data Source”.



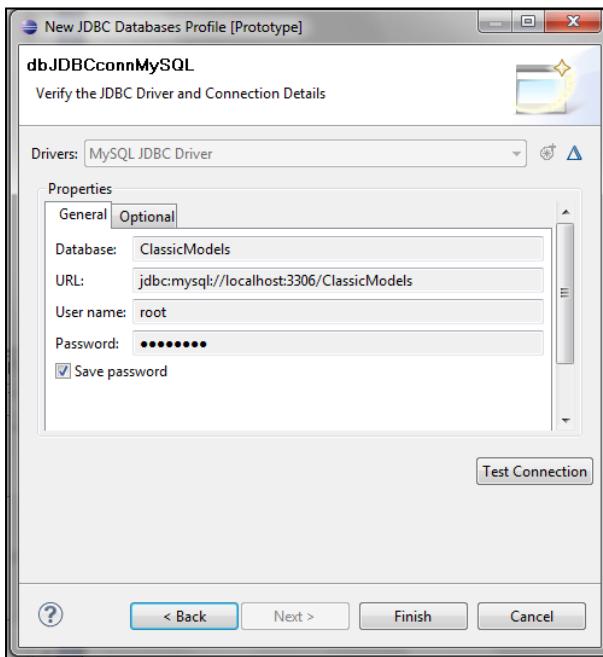
Select “Create from a connection profile in the profile store”



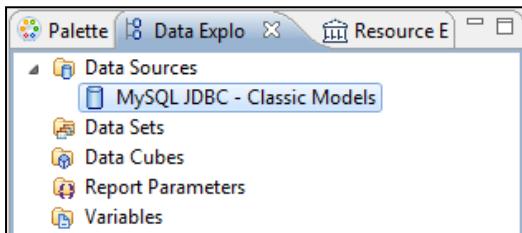
Select your Connection Profile dat file.



Click Next



Confirm details and Click Finish.



The Base64 version of the password is still stored in the XML.

```
<property name="odaDriverClass">com.mysql.jdbc.Driver</property>
<property name="odaURL">jdbc:mysql://localhost:3306/ClassicModels</property>
<property name="odaUser">root</property>
<encrypted-property name="odaPassword" encryptionID="base64">cGFzc3cwcmQ=</encrypted-
property>
<property name="OdaConnProfileName">MySQL JDBC - Classic Models</property>
<property name="OdaConnProfileStorePath">C:\eclipse-galileo-
birt\workspace\.metadata\.plugins\org.eclipse.datatools.connectivity\ServerProfiles.dat</
property>
```

It is not known why BIRT still puts the connection information in the XML, but it is just used by the Designer. When the report executes, it will retrieve the connection information from the connection profile.

6.18.8 Using a Custom Java Class Called by Scripting

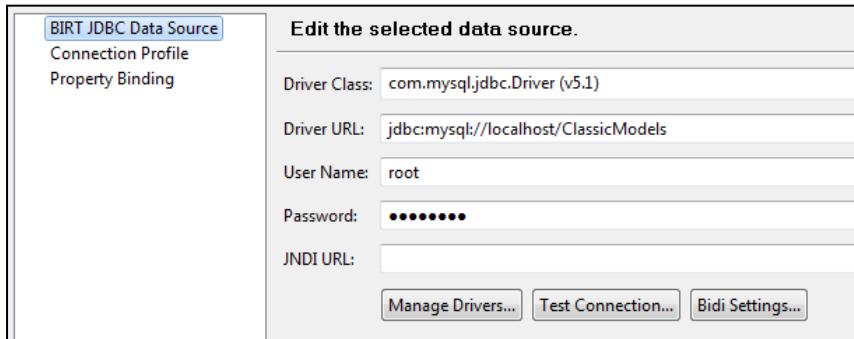
This technique is the best way if you need your Data Source connection properties to be extra secure. It allows the developer to provide a custom way of reading the connection properties, but still allowing them to use the Tool without interference. This involves using a custom Java Class called by Scripting (or a Java Event Handler). This technique will work for all Data Sources and inside or outside the Designer or an Application Server.

Store the Data Source properties in an external file:

- 1) Create a text or xml file holding the Data Source properties
- 2) Encrypt the password (and any other properties you wish – or even the whole file) using your favorite encryption technique.
- 3) Create a Java class that will read the file and retrieve the property values. Use existing classes if your application already does this.
- 4) Create “getter” methods that the BIRT report can call to retrieve the Data Source Property Values. The getter method for the password should decrypt it before sending the value to BIRT
- 5) The BIRT report should contain some Script (or use a Java Event handler) in the “beforeOpen” method of the Data Source to call the Java Class. The Java Class should accept the path of the property file, read the file, and provide the property values via getter methods. The BIRT scripting will call those getter methods to get the values and set the Data Source properties.

Disadvantage:

One disadvantage is that you will still need to put in your db connection properties directly into the Data Source property interface if you want to browse the Data Source schema. For some reason, BIRT uses those values to initially connect to the Data Source to allow the developer to view the tables (or other data) and construct the query. However, when the developer previews the data or runs the report, it will cause the script to run which will get the Data Source values from the external property file.



The following code is provided to give you an idea of how to use a custom Java class to read and parse the external property file and return the values to the Report Design:

```
package com.leBirtExpert.birtBestPractices.dataSourceProps;

/**
 * This class is a stub class. It provides a placeholder for
 * implementation. The BIRT Report will call this class
 * via JavaScript to retrieve Data Source properties from
 * an external property file
 *
 * @author David Mehí (Le BIRT Expert)
 */
public class DataSourceProperties {

    private String dbURL = "";
    private String dbUserId = "";
    private String dbPassword = "";

    /**
     * Open, read, parse property file
     * @param filePath
     * @return result (success/fail)
     */
    public String readPropertyValues(String filePath) {

        // TODO: OPEN PROPERTY FILE
        // TODO: PARSE VALUES
        // TODO: STORE IN LOCAL VARIABLES

        // hardcoded values for example
        this.dbURL = "jdbc:mysql://localhost:3306/ClassicModels";
        this.dbUserId = "root";
        this.dbPassword = "passw0rd"; // SHOULD BE ENCRYPTED VALUE
    }
}
```

```

        return "SUCCESS";
    }

    /**
     * @return the dbPassword
     */
    public String getDbPassword() {

        // DECRYPT PASSWORD BEFORE RETURNING
        return dbPassword;
    }

    /**
     * @return the dbDriverURL
     */
    public String getDbURL() {
        return dbURL;
    }

    /**
     * @return the dbUserId
     */
    public String getDbUserId() {
        return dbUserId;
    }

}

```

The “beforeOpen” method of the Data Source will look something like this:

```

// import Java classes
importPackage(Packages.JavaClass.com.leBirtExpert.birtBestPractices.dataSourceProps.DataSourceProperties);

// Instantiate Properties Class
var dataSourceProps = new Packages.com.leBirtExpert.birtBestPractices.dataSourceProps.DataSourceProperties();

// Read Files - pass in complete path to file (could be read from an env variable)
dataSourceProps.readPropertyValues("path-to-file.xml");

// Sets values on the Data Source
this.setExtensionProperty("odaURL", dataSourceProps.getDbURL());
this.setExtensionProperty("odaUser", dataSourceProps.getDbUserId());
this.setExtensionProperty("odaPassword", dataSourceProps.getDbPassword());

```

6.18.9 Summary

Whichever way your team decides to externalize the Data Source properties, it is important to put the Data Source Item in the library for all reports to use.

BIRT provides a default Base64 encryption of the password in the XML file. However, except for quick demos or test reports, this encryption will not be strong enough.

An ODA Connection Profile provides a very convenient, simple and a more secure way (better than base64) to externalize the connection properties. However, the connection profile still can be read in by eclipse (without any additional security), and it's not known what type of encryption is used. But for projects that don't have a tough security need, this technique should be okay.

Using an external Java class allows the developer to provide a custom encryption technique (or use existing classes) to secure the data source properties, without interfering on how the Designer is used. If you have strong encryption needs or want to leverage existing code, this is the best option.

6.19 Externalizing Connection Properties with Commercial BIRT and iServer

The commercial version of BIRT Designer offers a stronger method of encryption in the XML for the password. In the XML, the encryption method is listed as “jce” (instead of “base64”). It is not clear what the encryption key is, but the iServer knows how to decrypt it properly.

Through the BIRT iServer Enterprise Edition, there is another way to externalize connection properties for all types of reports. In the Management console, there is a way to specify a global connection profile that will override any settings in the Reports. Refer to the iServer documentation for more information.

Chapter 7

Report Parameters

In this chapter:

- Linking a Report Parameter to a Data Set
 - Parameter Grouping
 - Cascading Parameters
 - Hidden Parameters
 - Parameter Validation
 - Use the Default BIRT Parameter Page or Create Your Own?
 - Parameter Validation Considerations
 - Parameter Bad Practices
-

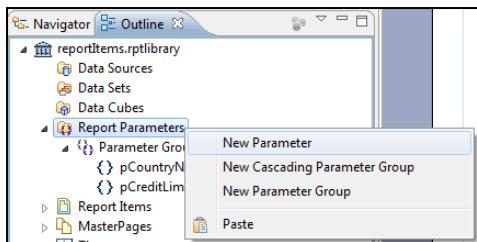
This chapter covers best practices as it relates to using report parameters. BIRT offers a way to store parameter items in a report library for future use. There are also other features in BIRT that support linking parameter values to data sets and cascading parameters. This chapter also discusses creating custom parameter screens and security concerns with parameters.

7.1 Re-Usable Report Parameters

Often times, the same set of parameters are used in a set of Reports. BIRT allows the developer to place parameters in a library and re-use them in reports. A report parameter has several properties, and the ability to re-use these parameter objects can be very valuable for a large set of reports.

7.1.1 Creating a Re-Usable Report Parameter

To create a report parameter in a library, open the “reportItems.rptlibrary” and switch to the Outline view. Right click on the “Report Parameters” section and choose “New Parameter”.



Enter the name and properties

New Parameter

New Parameter

Name: pCity
Prompt text: Enter a City Name
Data type: String
Display type: Text Box

Display As
Help text: Enter the city
Format as: Unformatted
Preview with format: NYC
List Limit: values
Is Required
Hidden
Do not echo input
Allow Duplicate Values

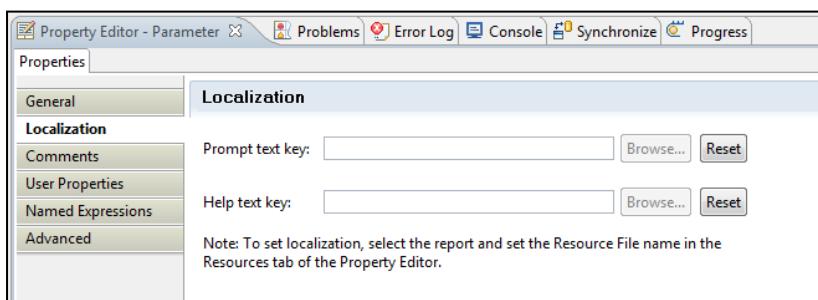
List of value
Default value: NYC

OK Cancel

reportItems.rptlibrary

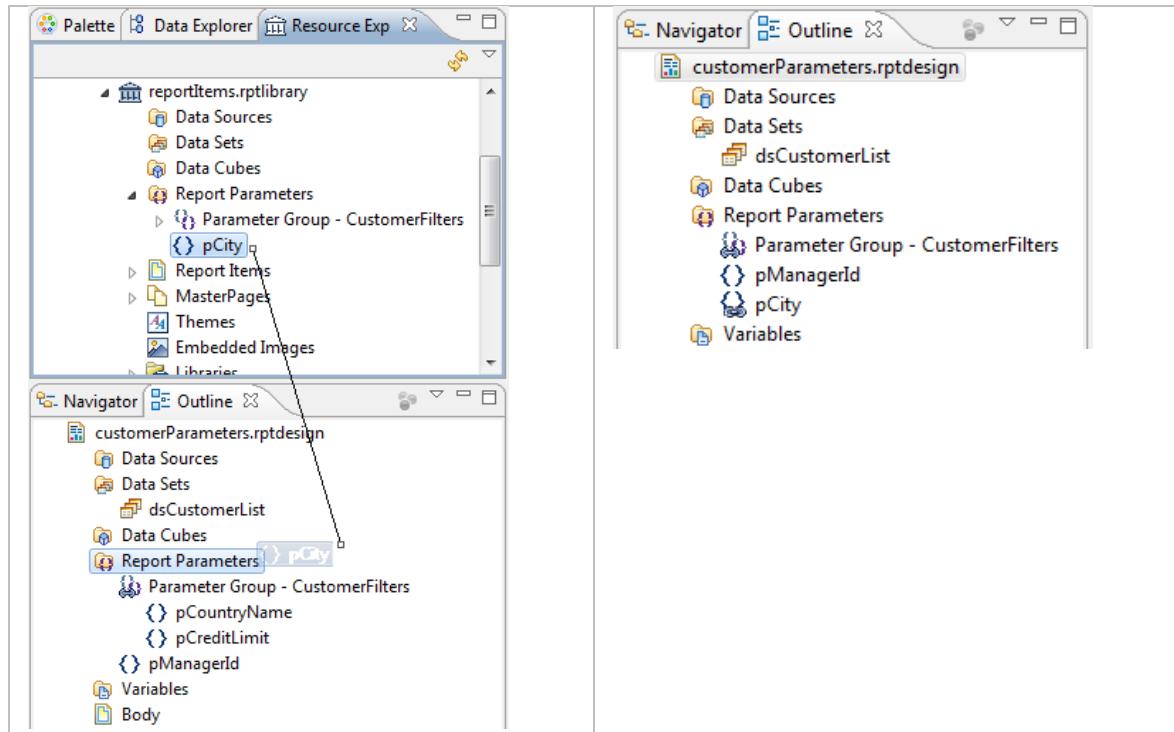
- Data Sources
- Data Sets
- Data Cubes
- Report Parameters
 - Parameter Group - CustomerFilters
 - { } pCountryName
 - { } pCreditLimit
 - { } pCity
- Report Items

Once the parameter has been created, select the report item in the Data Explorer (or Outline) to see more parameter properties. Click the “Localization” tab to set the localization key values for the Prompt and Help text.



7.1.2 Using a Re-Usable Report Parameter

The parameter in the library can be used like any other report item from a library. Drag and drop the parameter item from the library to the report design.

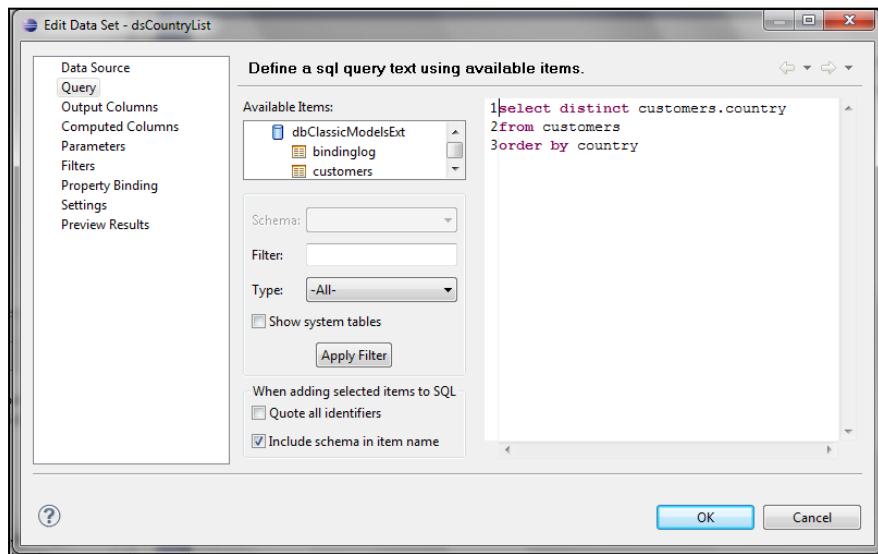


Once the parameter is a part of the report design, it can be used like a normal parameter. The nice thing is that it's already configured. Keeping common parameter items in a library is a great way to ensure consistency among your reports and shorten development time.

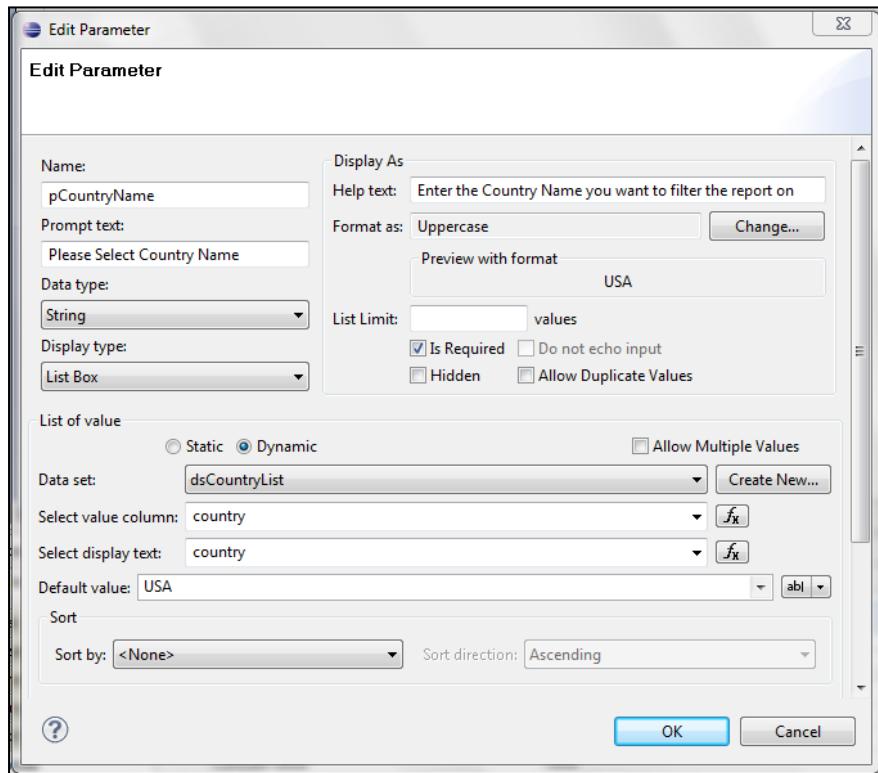
7.2 Linking a Report Parameter to a Data Set

Often, it is necessary to link a Parameter to a Data Set. For example, giving the user a drop-down list of Car Models (come from the database) to choose from. BIRT provides several out-of-the-box properties for Parameters that will be useful.

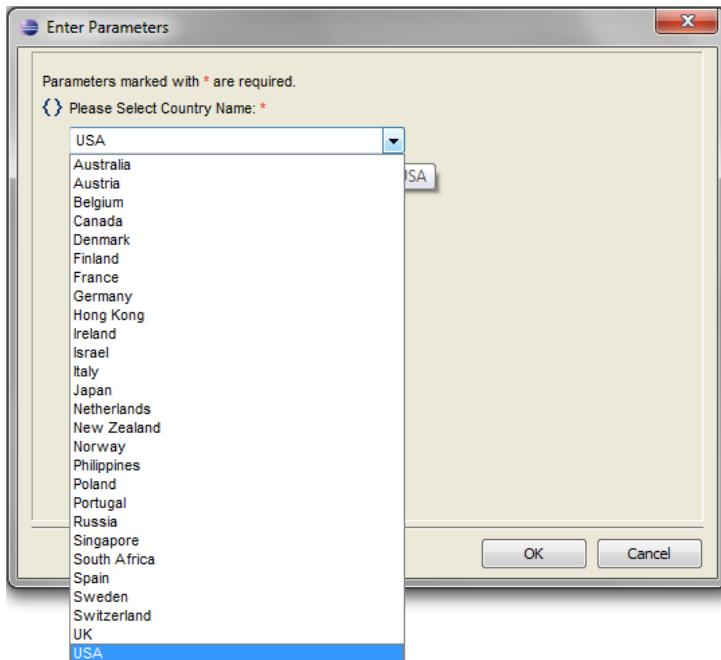
The Data Set “dsCountryList” contains a list of all possible Country names a user can choose.



BIRT allows the developer to link a Parameter List Box to this Data Set



The user will be able to choose from the List.

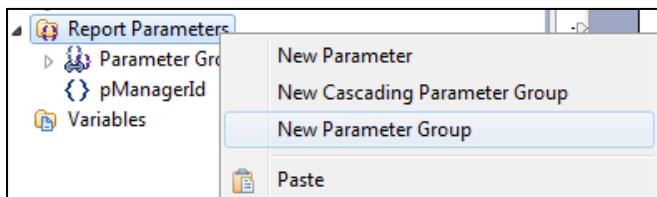


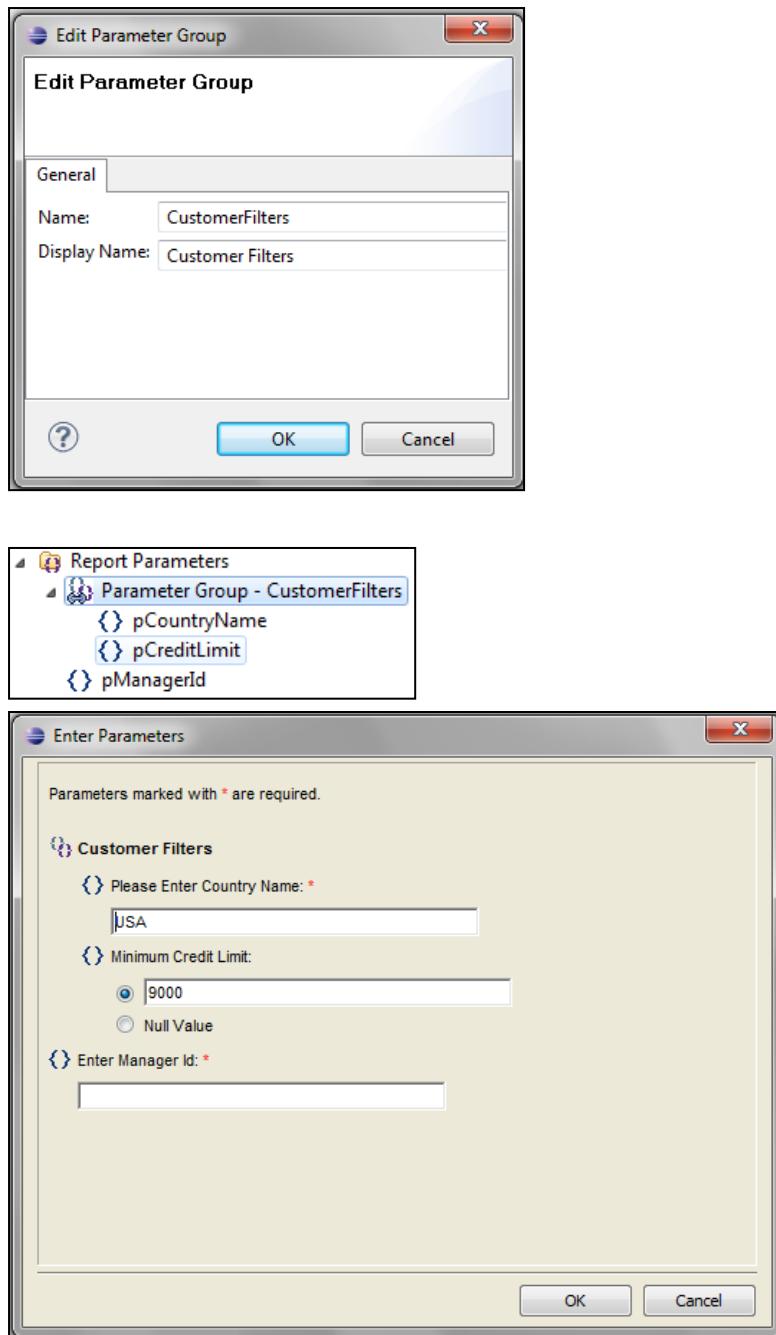
7.3 Parameter Grouping

BIRT allows the developer to group like parameters together. This is useful when you want to display parameters for a certain business function together.

***Tip**

This grouping is for display and organizational purposes only and doesn't provide any programming or logic benefits.





7.4 Cascading Parameters

BIRT provides cascading parameters functionality out of the box. This is a great way to allow your users to narrow down the choices from a large data set. For example, a user chooses from a list of countries. Based on what country they chose, a second drop-down box is populated

with a list of Customers from that Country. In BIRT, this involves linking the second parameter (and data set) to the first parameter (and data set).

This topic is covered well on other web sites. Check out these examples on Cascading Parameters.

<http://www.eclipse.org/birt/phoenix/examples/reports/birt2.1/cascade/index.php>

<http://www.birt-exchange.org/devshare/designing-birt-reports/969-birt-cascading-parameter-example/#description>

<http://birtworld.blogspot.com/2009/04/birt-cascaded-parameters.html>

7.5 Hidden Parameters

Hidden Parameters can be useful when certain values need to be passed in the report, however, they don't need to be displayed to the user. For example, elsewhere in your application, the user might have made certain decisions on what account they were working with. Since the user already chose to work with this account number, they shouldn't have to type it in the parameter box again. However, the report still needs to know what account it should retrieve the data for. This account number can be passed into the report through a hidden parameter without the user's knowledge.

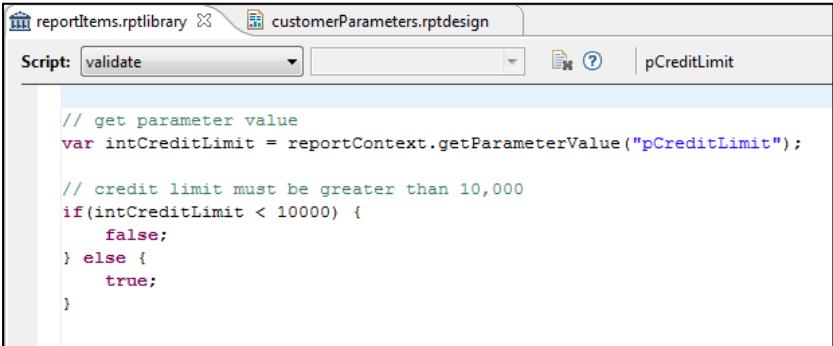
Hidden Parameters are useful when you need to pass in relevant data that you don't want to bother the user with. However, don't assume that they are secure just because they are hidden.

7.6 Parameter Validation

A feature that BIRT offers is to do validation on the incoming Parameter. This validation is done with JavaScript or with Java Event handlers. The validation is done server side, after the user has already entered the values and clicked "Ok". Currently, BIRT does not support client side validation out of the box. The developer would manually have to code this.

For parameter validation, select the parameter item and select the "Script" tab. Choose the "validate" method. In this example, we want to make sure the value is over 10,000.

Even though we are writing code for this particular parameter, we will still need to access the parameter value from the “reportContext”. The “validate” method must return either a true or false, depending on the outcome of the validation. JavaScript syntax is important here. If there is an error, the BIRT viewer will display a general Exception message instead of explaining exactly what is wrong. In this method, the final value must be “true” or “false”. If the developer types a “return true” or “return false”, they will get a JavaScript exception.

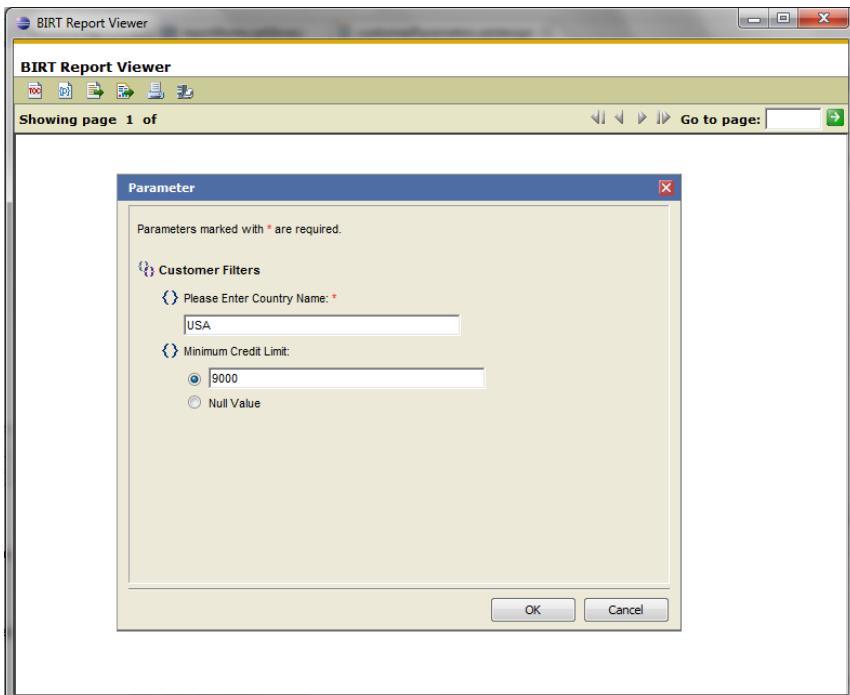


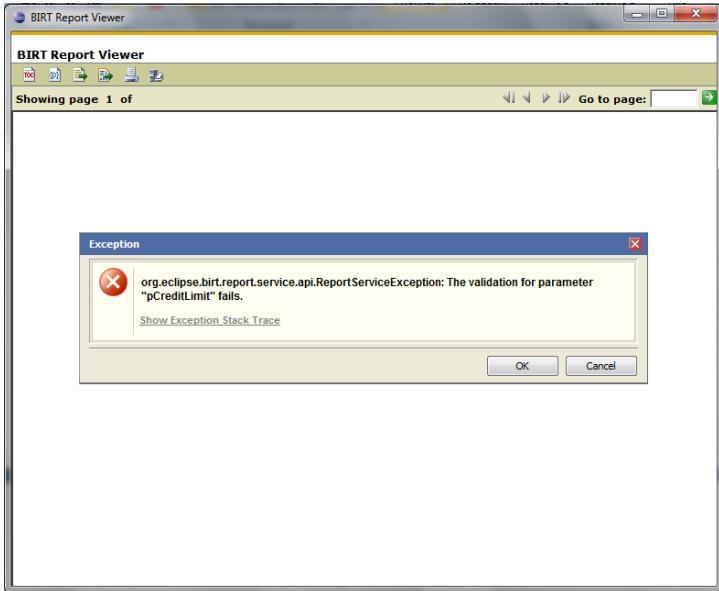
The screenshot shows the BIRT Report Designer interface. The title bar says "reportItems.rptlibrary" and "customerParameters.rptdesign". A dropdown menu labeled "Script" is set to "validate". Below it is a code editor window containing the following JavaScript:

```
// get parameter value
var intCreditLimit = reportContext.getParameterValue("pCreditLimit");

// credit limit must be greater than 10,000
if(intCreditLimit < 10000) {
    false;
} else {
    true;
}
```

If true, the report continues to run. If false, the BIRT Viewer will display an exception message.





BIRT offers a very basic way of doing parameter validation, and no out-of-the-box way to display a better message to the end user. Most end users do not want to see an Exception Stack Trace message. To display any other type of message, or to even return the user to the parameter screen will involve some manual coding in the BIRT Viewer web app.

See *Chapter 16, Exception and Error Handling*, for more information.

7.7 Should You Use the Default BIRT Parameter Page or Create Your Own?

Using the BIRT Web Viewer (or Actuate's iPortal or JSAPI) offers a convenient way to display the parameter choices to the end user. It allows the developer to control the parameter choices and parameter lookup values from within the report. It offers the ability to do cascading parameters, hidden parameters and simple validation. For basic parameter pages, the default way of handling parameters should be fine.

Custom parameter pages offer complete control over how you display report parameter choices to your end user. Using the BIRT Runtime Engine (or BIRT iServer's IDAPI), report parameter details can be extracted from the report and used to display the appropriate parameter controls. This even includes the values of a drop-down list specified by the parameter in the BIRT Design.

Consider creating customer parameter pages in the following scenarios:

- If the parameter choices have a lot of business logic tied to them, it would be better to keep that business logic in your application and outside of the reports. For example, specific users should only see certain values based on certain criteria.
- The logic to retrieve and display parameter data is used elsewhere in your application. This code can be re-used to save effort.
- If your parameters have special validation needs or want to offer client side validation.
- If you have a very tightly integrated solution, and want to keep a consistent look and feel throughout the application. This is possible by modifying the BIRT viewer code, however it might be better for maintenance just to keep the code in your application.

Drawbacks of a custom parameter screen:

- More coding effort. BIRT provides parameter pages out of the box with no extra effort
- The developer must come up with a way to tie the parameter controls to the report parameters. Some do this with an XML file or use a database table. For example, if a report has a “country” parameter, the application should know to display a list of countries. (To a certain extent, this can be done by referencing the Report Design)

For basic parameter page needs, the default parameter page in the BIRT Viewer should be fine. However, for more complex validation and display needs, it is recommended to create custom parameter screens.

A hybrid approach would involve defining the parameters (and list of values) in BIRT, using the BIRT Runtime Engine (or iServer’s IDAPI) to extract the parameter information, and displaying your custom parameter control according to the type of parameter the report is using.

***NOTE**

If you are using Actuate’s iPortal or JSAPI, the version must match the version of the iServer to maintain support. If you make a lot of customizations to the iPortal, you will need to re-apply these customizations when upgrading. If you decide to customize the iPortal (instead of creating your own parameter page), be sure to document every change made. You will need to re-apply those changes when going to a new version. Since it’s a web application, there is no way around this. If your parameter pages require customization, it is recommended to use the JSAPI or IDAPI to build your own parameter pages.

7.8 Parameter validation considerations

Whether you are using the default parameter page in the BIRT Viewer or developing your own custom parameter pages, parameter validation is very important to report security. Anytime a parameter is declared in a report, it is like opening a tiny window to the world. Even with report parameter pages, a smart end user or hacker can manipulate the URL to send in their own parameter choices to see data that they are not allowed to see.

The only way to ensure this does not happen is to do another round of validation in the report. The developer will need to determine if this level of security is necessary for their reports. Some environments and applications may already be securely tied down enough or the report data is not sensitive enough to need this extra level of security. However, any report that restricts data to only what a user has access to, this extra level might be necessary.

There are two ways to do this extra validation. One is to write Javascript in the report that checks the parameter value to confirm that the values are correct. Typically, this can be done in the “initialize” script method of the report design or even in the parameter “validate” method. This can involve using JavaScript or invoking a Java class to do the validation. If the values are not correct, either correct them (reformat) or don’t run the report and display an error message to the user instead.

The other way is to put extra validation measures in the data source query. For example, in SQL (or stored procedure), put an additional sub-query to limit values to only what a specific user should see. If an end user happened to sneak additional parameters values via URL manipulation, the query will only return values that they have access to. It is better to display a report with no data than it is to display one with data they don’t have access to.

7.9 Parameter Bad Practice

Think of a report parameter as a gateway to your database. Anything that is passed in can be passed down to your database without proper precaution. Le BIRT Expert does *not* recommend declaring the user id or any data source connection properties as a report parameter. This can be easily hacked, causing one user to pass themselves off as another user. If you need to use the user id in the report query, have the report pick it up through the HTTP Session or another mechanism. For data source connection properties, look at section 6.18 “Externalizing Connection Properties”.

Chapter 8

Scripting using JavaScript

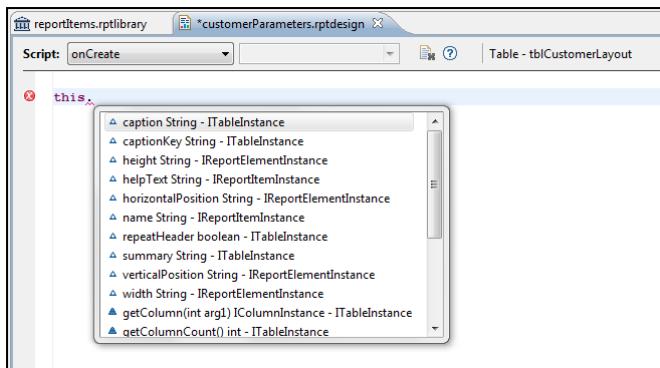
In this chapter:

- Using the Script Palette
 - Advantages and Disadvantages of JavaScript
 - Scripting Guidelines
 - Calling External Java Objects
 - Creating and Leveraging Re-Usable Scripting Libraries
 - Runtime Manipulation of the Report Design
 - Don't want to use JavaScript? A Simple Alternative
-

The use of JavaScript in the Report Design is a powerful way to extend the use of BIRT. Things that cannot be done through the Designer interface can usually be done with Scripting. Each Report Item supports the use of Script to modify its properties or functionality in some way. However, there are some drawbacks to using Scripting in the report. There are some general guidelines to follow if you do decide to use Script.

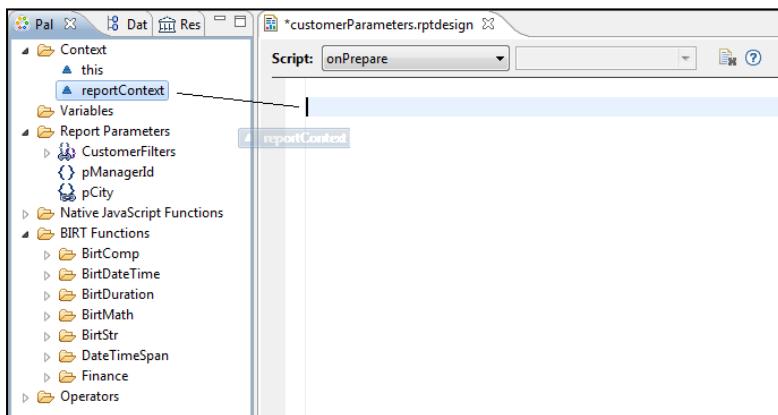
To add a Script to any Report Item, choose the item you want to add the Script for, and click the “Script” tab. This window will give you the area to enter your Script. Choose which method you want to type the script in for. The name of the Report Item that the Script will be applied to is listed in the upper right corner. Certain JavaScript key words will be highlighted. To take advantage of intelli-code, type in a key word, followed by “.” to see the options available.

For example, type “reportContext.”. If the list of methods do not appear, type “control-space” to see. Use the word “this” to refer to the current control. Typing “this.” will display methods available.



8.1 Using the Script Palette

When you are in the Script area, a list of commonly used scripting functions are shown in the Palette window. The developer can drag and drop these function names into the Script area.



8.2 Advantages and Disadvantages of JavaScript

Advantages of using JavaScript:

- Easier to code
- No need to deploy since its part of the Report Design
- Convenient
- Can control most basic functions in the report
- Limited intelli-code help
- Can call normal Java objects

Disadvantages of Using JavaScript:

- No type-casting
- Limited syntax checking
- Error messages do not give enough information to fix the problem
- More room for error. Little typos can lead to frustration
- Limited intelli-code help

8.3 Scripting Guidelines

Covering the detailed implementation of Scripting is out of scope for this book, but Le BIRT Expert offers a few guidelines to consider when writing your own Script:

- Always leave comments in your code. Comments can be left by using the syntax “//” or “/* ... */”.
- Declaring your variables with “var” will keep the variables local to that method or function. Not using “var” will make the variables global to the whole report.
 - reportContext.setGlobalVariable() is used to pass variables into the reportContext, which can be picked up by the Report Engine or Java Event Handler.,
 - reportContext.setPersistentGlobalVariable() makes the variable persistent to the rptdocument (can be used in rendering tasks)
- Add “;” at the end of each line command.
- Put common re-usable functions into an external “js” file. (See next section)
- Leave the heavy lifting to external Java classes. This allows more sophisticated exception handling, logging and other features available to Java. For example, if you have to read and parse a file, it can be done with JavaScript. However, it is recommended that you read and parse the file with a Java class, and call it from Script.
- When making heavy use of the BIRT Runtime or Design Engine, it is better to use a Java Event Handler instead. Doing simple things with the Runtime or Design Engine is fine (such as adding a grid or table column, etc), but when doing more complex things (such as creating a whole new grid or table and adding it to the report design), it is recommended to use a Java Event Handler

Javascript can be very useful for certain manipulation in the Report Design. However, because there is no type-casting, limited syntax help and non-descriptive error messages, putting a lot of logic in JavaScript can be frustrating to debug and get working properly. It can definitely be done, however, using a Java Event Handler would speed the development effort.

8.4 Calling external Java Objects

As long as the Java class is in the class path, JavaScript can call a Java Class. JavaScript must import the Java package, and then reference the object as it needs.

```
importPackage(Packages.java.util);
var objCustomPOJO = new CustomPOJO();
objArrList = new ArrayList();
objArrList.add(objCustomPOJO);
```

8.4.1 Using the Java String Object

Sometimes it is better to do String manipulation with a Java class than with JavaScript. There have been cases where the Rhino Javascript Engine (the Javascript engine that BIRT uses – open sourced by Mozilla) does not handle Strings as the developer might expect. Some developers have reported issues when using the “replaceAll()” function and when converting to dates. If you are experiencing issues with String or Data manipulation in Javascript, use the direct Java equivalent classes instead.

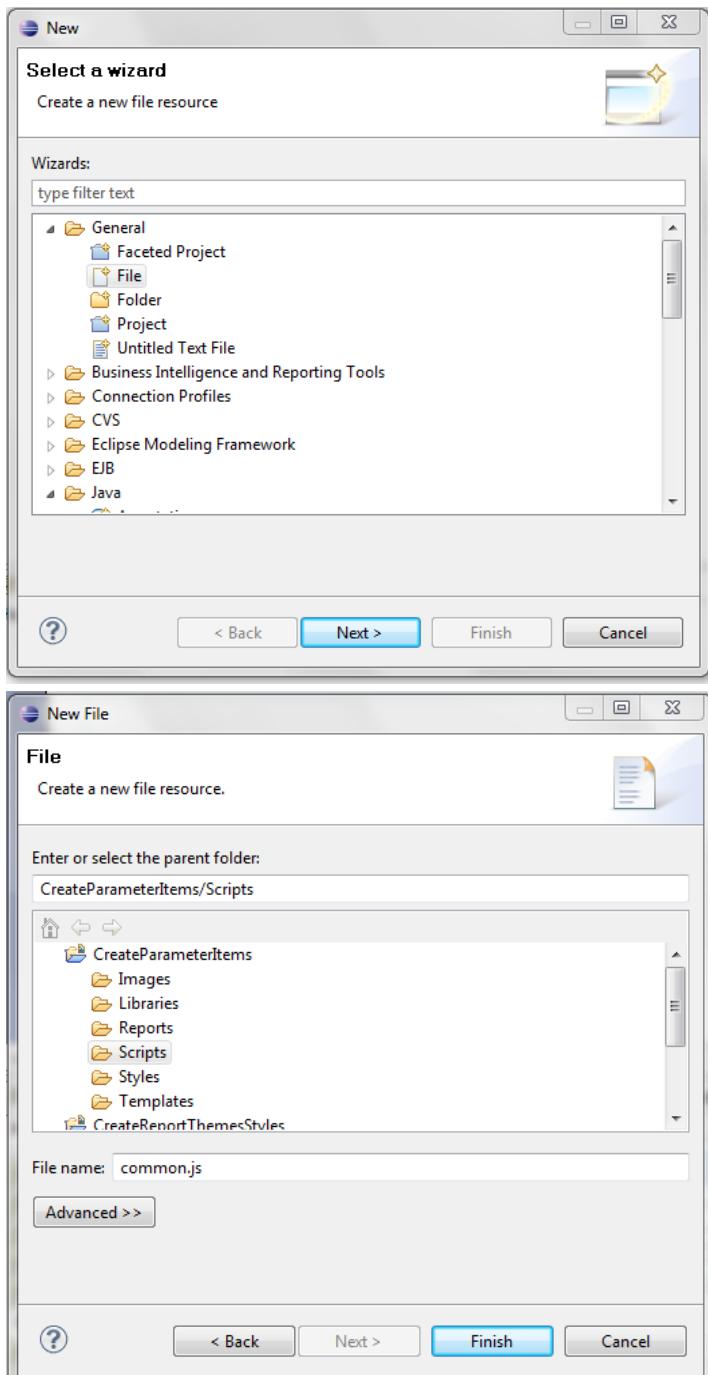
```
var myStringObj = new Packages.java.lang.String();
myStringObj = "Hello World";
myStringObj = myStringObj.replaceAll("o", "0");
```

8.5 Creating and leveraging re-usable scripting libraries

BIRT allows a developer to create JavaScript functions, store them into an external “js” file, and then include them in other report designs. This can be extremely useful when wanted to re-use functions that have been written in JavaScript.

8.5.1 Creating a Scripting Library

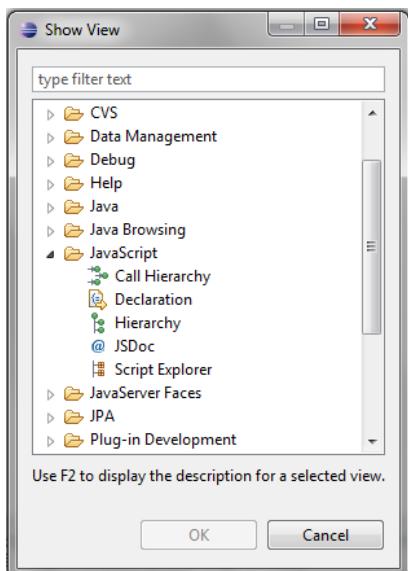
Create a new file in Eclipse, and name it using a “js” extension. Create your desired functions in the file and save. Javascript functions are declared with the word “function”. The “Javascript” Eclipse Perspective is available for additional Javascript syntax assistance. This Perspective is only available when coding external “js” files.



JavaScript Eclipse Perspective



In the JavaScript Perspective, additional “Views” can be added for more insight into the JavaScript file. Go to Window → Show View → Other.



```


// Checks to see if the string passed in is either
// null or empty
*/
function isEmptyString(strValue) {
    if(strValue == null || strValue == "" || strValue.length == 0)
        return true;
    else
        return false;
}

/*
 * check date JavaScript function (US format - mm/dd/yyyy)
 * if date is valid then function returns true, otherwise returns false
*/
function isDate(strDate){
    var objDate; // date object initialized from the txtDate string
    var mSeconds; // milliseconds from txtDate

    // date length should be 10 characters - no more, no less
    if (strDate.length != 10) return false;

    // extract day, month and year from the strDate string
    // expected format is mm/dd/yyyy
    // subtraction will cast variables to integer implicitly
    var day   = strDate.substring(3,5) - 0;
    var month = strDate.substring(0,2) - 1; // because months in JS start with 0
    var year  = strDate.substring(6,10) - 0;
}

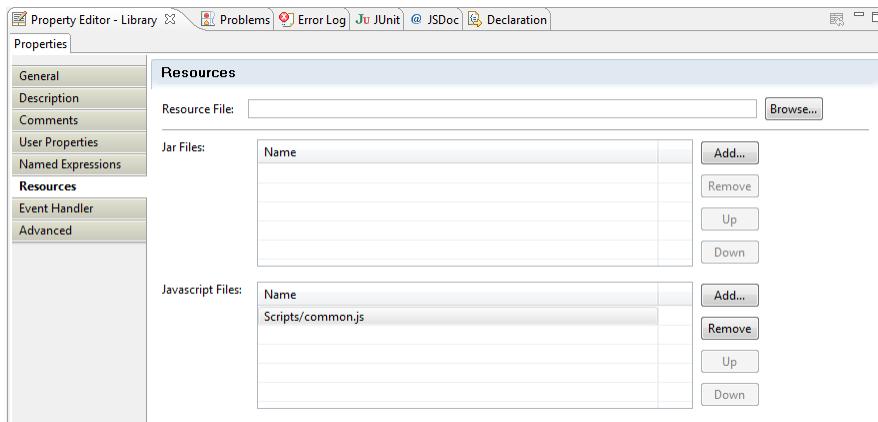

```

8.5.2 Using a Scripting library

If there are Javascript “js” files that you want to re-use, BIRT allows the developer to include them as a “resource”. It is similar to how an HTML page would include a JavaScript file. The “js” files can be included in a Report Library, Report Template or Report Design.

If your JavaScript file contains commonly used functions, include it in the “reportItems.rptlibrary”. If the functions may be used in specific scenarios, include them in the Report Design as needed.

To include a JavaScript file, select the Report Library, Report Template or Report Design that you want to add the file to. In the Property Editor window, select “Resources”. In the JavaScript section, click the “Add...” button. Select the JavaScript file to include.

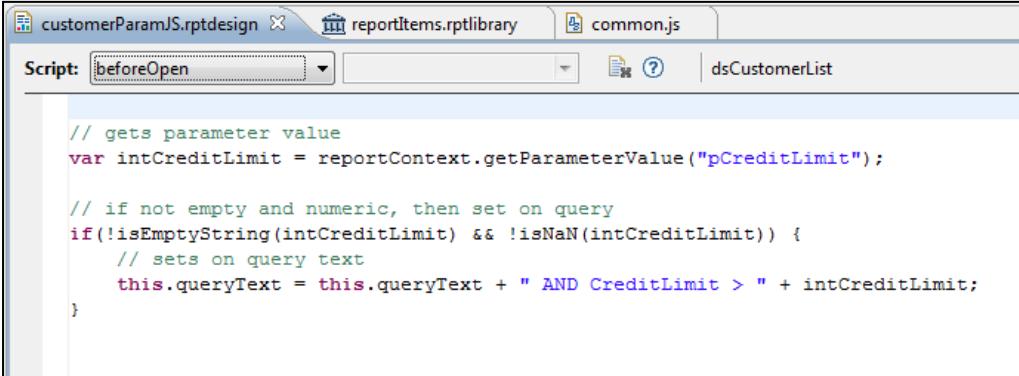


In the XML for the Library, Template or Design – notice the line where the file is included.

```
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns="http://www.eclipse.org/birt/2005/design" version="3.2.20" id="1">
  <property name="createdBy">Eclipse BIRT Designer Version 2.5.1.v20090903 Build
  <property name="units">in</property>
  <property name="theme">reportThemes.mainReportTheme</property>
  <list-property name="libraries">
    <structure>
      <property name="fileName">Libraries/reportThemes.rptlibrary</property>
      <property name="namespace">reportThemes</property>
    </structure>
  </list-property>
  <list-property name="includeScripts">
    <property>Scripts/common.js</property>
  </list-property>
```

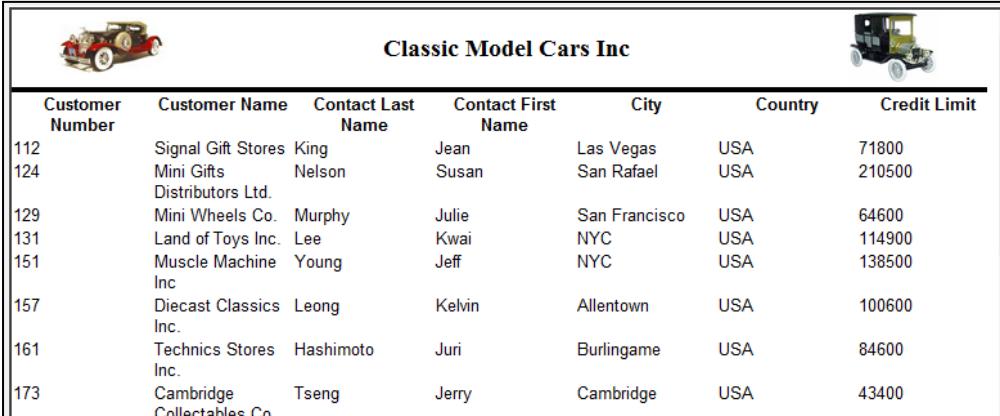
The screenshot shows the Eclipse BIRT Designer's XML editor with the file "reportItems.rptlibrary" open. The XML code is displayed in a syntax-highlighted editor. A yellow box highlights the line containing the script inclusion: "<property name='includeScripts'>Scripts/common.js</property>". The rest of the XML code includes properties for creation details, units, theme, and library structures.

In the Report Design, use the JavaScript function as expected.



```
// gets parameter value
var intCreditLimit = reportContext.getParameterValue("pCreditLimit");

// if not empty and numeric, then set on query
if(!isEmptyString(intCreditLimit) && !isNaN(intCreditLimit)) {
    // sets on query text
    this.queryText = this.queryText + " AND CreditLimit > " + intCreditLimit;
}
```

Classic Model Cars Inc

Customer Number	Customer Name	Contact Last Name	Contact First Name	City	Country	Credit Limit
112	Signal Gift Stores	King	Jean	Las Vegas	USA	71800
124	Mini Gifts Distributors Ltd.	Nelson	Susan	San Rafael	USA	210500
129	Mini Wheels Co.	Murphy	Julie	San Francisco	USA	64600
131	Land of Toys Inc.	Lee	Kwai	NYC	USA	114900
151	Muscle Machine Inc	Young	Jeff	NYC	USA	138500
157	Diecast Classics Inc.	Leong	Kelvin	Allentown	USA	100600
161	Technics Stores Inc.	Hashimoto	Juri	Burlingame	USA	84600
173	Cambridge Collectables Co.	Tseng	Jerry	Cambridge	USA	43400

For more additional information on using JavaScript in BIRT, view this URL:
<http://www.eclipse.org/birt/phoenix/deploy/reportScripting.php>

8.6 Runtime Manipulation of the Report Design

BIRT allows runtime manipulation of the report design using Javascript. A little Javascript can go a long way in dynamically updating the dataset SQL, adding a table column or adding/removing report items to a report. This topic is well documented elsewhere on the web. Several good links are listed here:

Adding Columns

<http://www.birt-exchange.org/devshare/designing-birt-reports/1099-add-column-to-table-dynamically/#description>

Dropping a Table

<http://www.birt-exchange.org/devshare/designing-birt-reports/1027-drop-a-table-based-on-a-report-parameter/#description>

Dropping a Table Group

<http://www.birt-exchange.org/devshare/designing-birt-reports/1098-drop-group-with-script/#description>

Displaying Controls in a Table

<http://www.birt-exchange.org/devshare/designing-birt-reports/1097-row-per-cell-example/#description>

Using the Design API in the Report Design

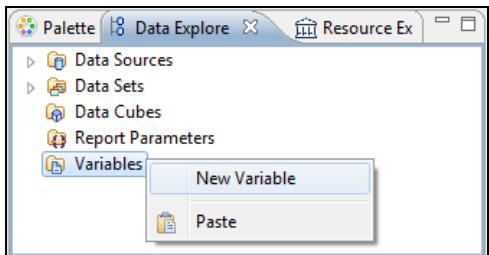
<http://www.birt-exchange.org/devshare/designing-birt-reports/930-birt-design-api-and-rptlibrary/#description>

8.7 Don't want to use JavaScript? A Simple Alternative

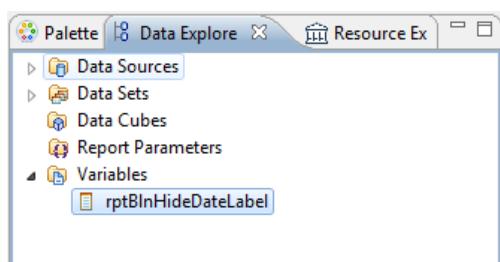
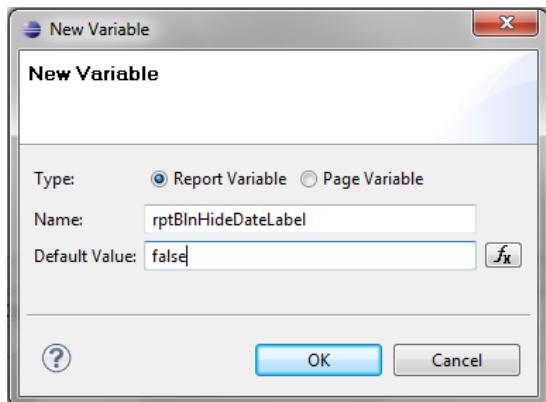
Don't want to mess with Scripting, but still have a need to use basic variables and expressions? BIRT now has a feature to create a "Variable" in the report design. These variables can be used for expressions throughout the report design. They are restricted to individual report designs and cannot be put into a library for re-use, however, they can serve report specific needs. Variables can be used in Scripting, but they can also provide a simple alternative to coding in JavaScript.

8.7.1 Creating a Report Variable

To create a Variable, navigate to the Data Explorer and right-click on “Variables.”



Enter the type, name and value of the Variable

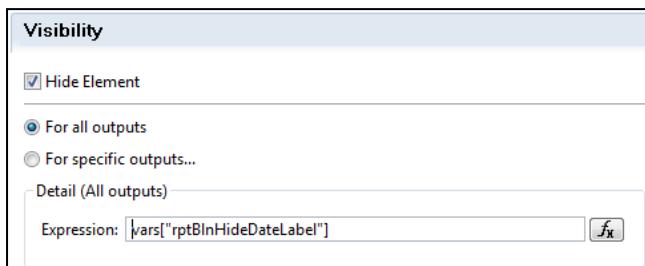
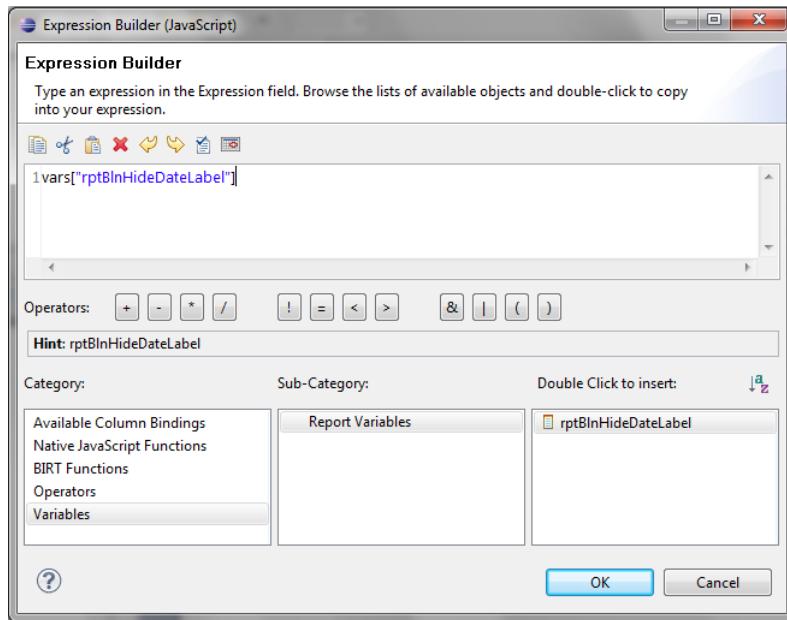


8.7.2 Using a Report Variable - Example

In this example, we have a text item that we want to hide based on the value of the Variable. For the text item's "visibility" property, refer to the report variable in the expression.



Choose the report parameter in the expression



Chapter 9

Create and Leverage Re-usable Java Components

In this chapter:

- Using Custom Java Classes
 - Java Event Handlers
 - BIRT Events
 - Creating a BIRT Event Handler Java Project
 - Using Java Event Handlers – Best Practices
 - Java Event Handler Example
 - Additional Resources
 - Deploying External Java Classes
-

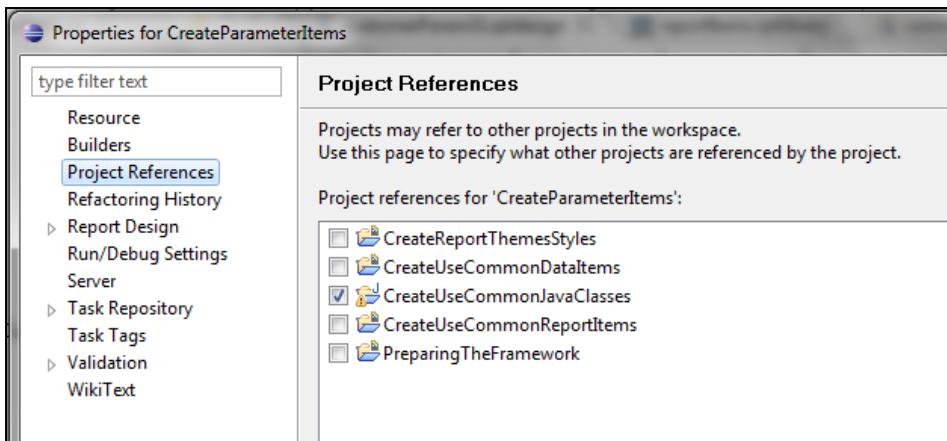
BIRT allows the developer to access normal Java Objects through Scripting or an Event Handler. This allows the developer to leverage existing business logic, Java frameworks, and custom Java classes.

9.1 Using Custom Java Classes

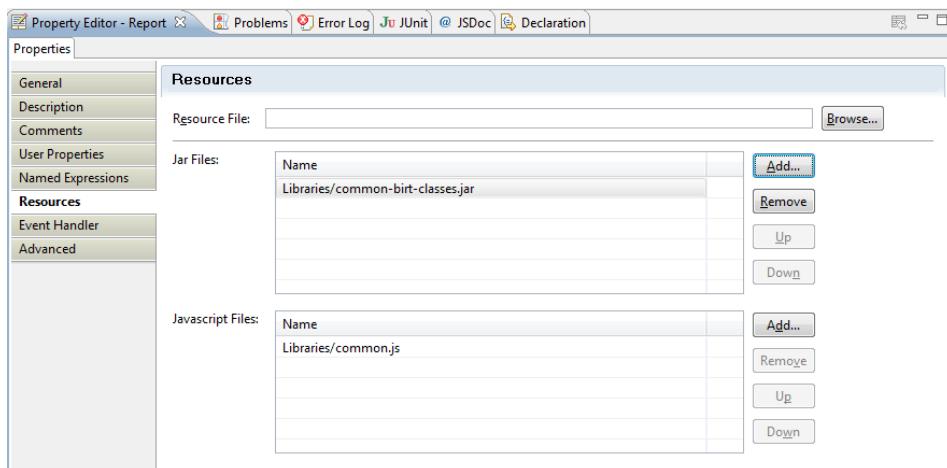
To use external Java classes, they need to be added to the class path for BIRT to find and reference them.

To add jar or class files to the class path, there are three ways.

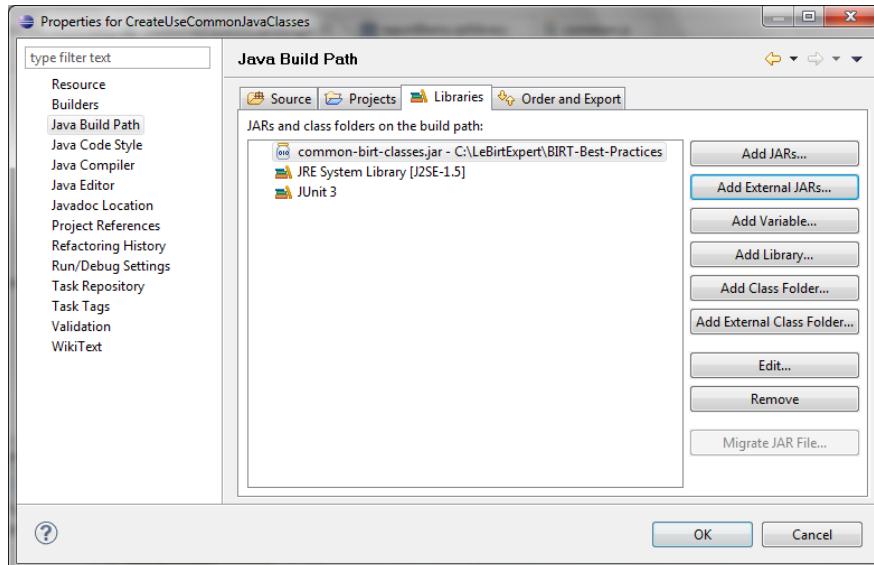
- 1) Put the Java files in a separate “Java” project in the same Eclipse instance. Right click on the BIRT Project and select Properties. Select Project References, and choose the Java Project that you want to reference. This automatically puts the class files from that project into your BIRT project class path.



- 2) The JAR file will need to be added to the Project Resources folder of the BIRT project. The reference can be added to a Report Library, Template or Design by going to the Property Editor and selecting “Resources”. Add the reference to the JAR file in this screen.



- 3) If the Eclipse Project is a Java project and not a BIRT project, there are more options to include additional JAR files in the class path. Just because a project is not a “BIRT” project doesn’t mean that a developer can’t create BIRT designs in the project.



Developers must use JavaScript to call custom Java classes from BIRT. See section 8.3, *Calling external Java Objects*, about how to do this.

9.2 Java Event Handlers

Java Event Handlers are a powerful way to extend the functionality of BIRT. Java Event Handlers are more tedious to implement than Scripting. With Scripting, the developer can type directly into the report design, and the report will execute it automatically at runtime. With Java Event Handlers, the developer must create the appropriate Java class, override the desired event methods, declare the name of the class in the report design, and deploy the jar file separately from the reports.

Advantages of using Java Event Handlers:

- Type-casting
- Syntax checking – compiler will catch coding errors
- Intelli-code assistance (through a Development UI like Eclipse)
- Better descriptive exception messages
- Easier to use existing Java classes
- Easier to debug than Javascript

Disadvantages of using Java Event Handlers:

- More cumbersome to code and deploy
- Download BIRT Runtime Engine classes
- Create individual classes for each item
- Deploy separately from Report Designs

***Tip**

It is recommended that for simple and quick manipulation, JavaScript be used in the Report Design. For more complex manipulation of the report design, Java Event Handlers are recommended.

9.3 BIRT Events

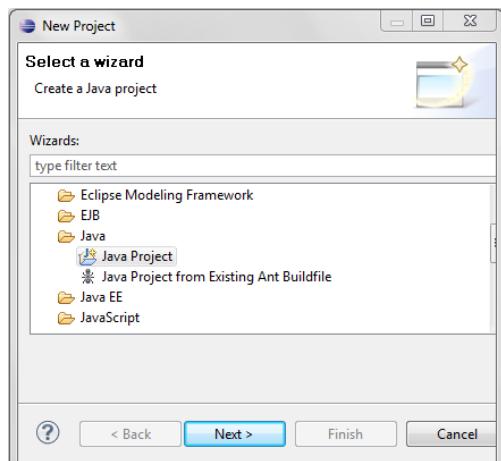
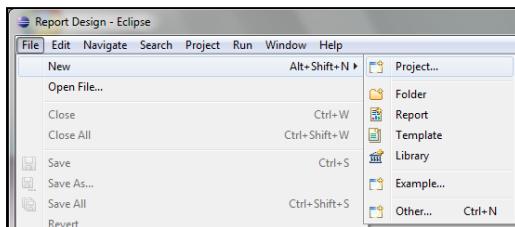
The Events that Java Event Handlers can override are the same events as Scripting.

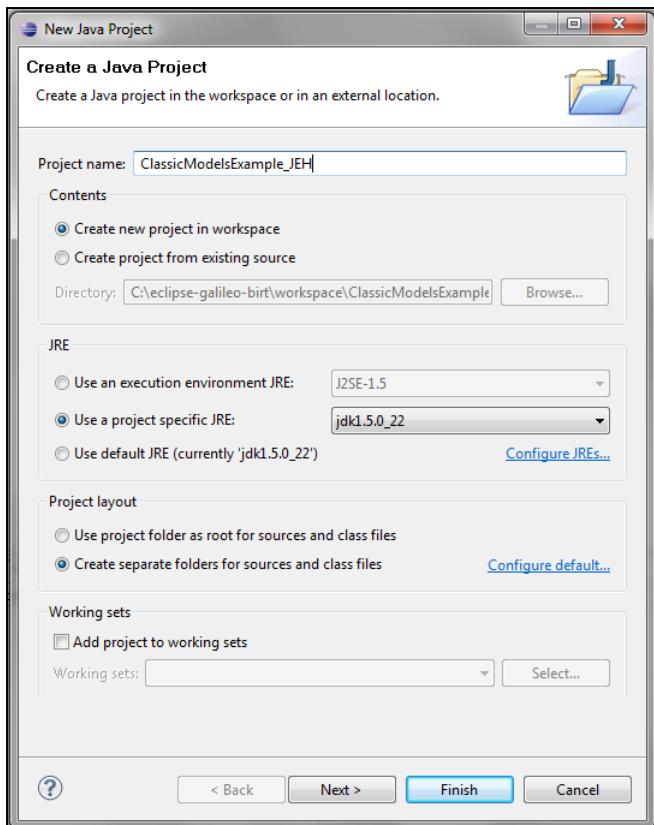
9.4 Creating a BIRT Event Handler Java Project

If you are planning to create new external Java objects, including Java Event Handlers, for your BIRT Project, it is best to create a new Java project within Eclipse. This way, you can keep the files separate, which will make it easier for development and deployment. Creating your project as a “Java” project will give you features of the Java perspective that is not available with a normal “BIRT Report Design” project in Eclipse.

Note: creating a Java Project is also covered in Chapter 3, “Preparing the Framework.”

A part of the Reporting Library is having an additional Java project dedicated to external Java classes, including Java Event Handlers.





9.4.1 JAR Files Needed

To create Java Event Handlers, you need to download the BIRT Runtime Engine. These JAR files will need to be added to the class path of the project when developing.

To download, go to the link: <http://download.eclipse.org/birt/downloads/>

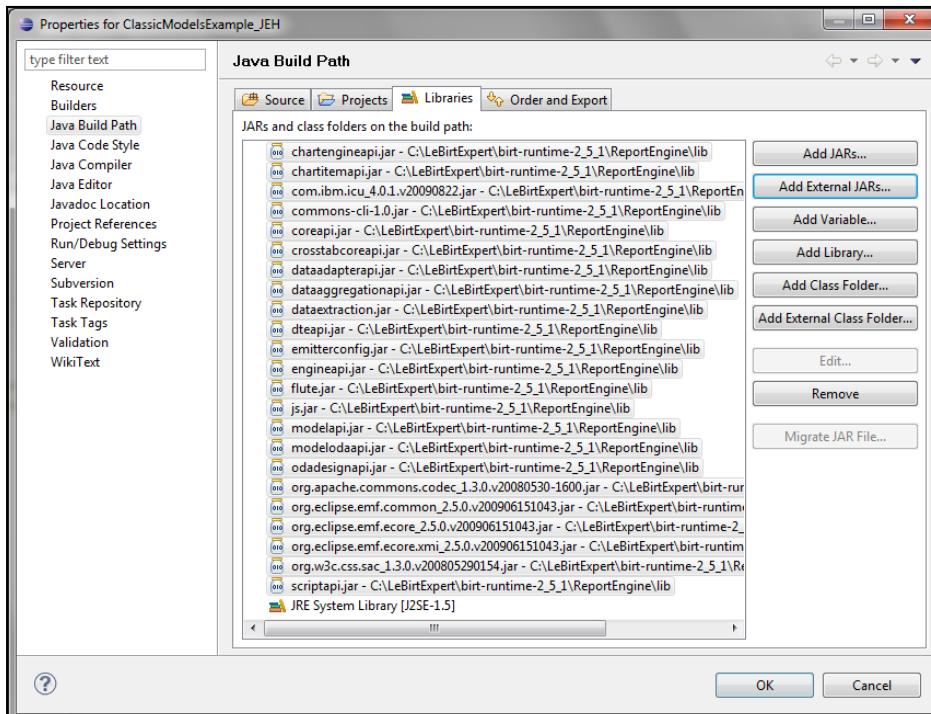
Download the Runtime Release Build.



Unzip the file into a directory

In Eclipse, right click on the project name → select properties → Java Build Path

In the “Libraries” tab, click the “Add External JARS ...” button. Select all of the JAR files in the “/ReportEngine/lib” folder of the BIRT Runtime Engine zip file that was just downloaded. Click OK.



9.4.2 Creating Base Classes

The Scripting API of the Runtime Engine offers us a set of Event Handler Interfaces or Event Adapter Classes to use.

Event Handler Interfaces (scriptapi.jar)	Event Adapter Classes (scriptapi.jar)
<ul style="list-style-type: none"> org.eclipse.birt.report.engine.api.script.eventhandler <ul style="list-style-type: none"> IAutoTextEventHandler.class ICellEventHandler.class IDataItemEventHandler.class IDataSetEventHandler.class <ul style="list-style-type: none"> IDataSetEventHandler <ul style="list-style-type: none"> afterClose(IReportContext) : void afterOpen(IDataSetInstance, IReportContext) : void beforeClose(IDataSetInstance, IReportContext) : void beforeOpen(IDataSetInstance, IReportContext) : void onFetch(IDataSetInstance, IDataSetRow, IReportContext) : void IDataSourceEventHandler.class IDynamicTextEventHandler.class IGridEventHandler.class <ul style="list-style-type: none"> IGridEventHandler <ul style="list-style-type: none"> onCreate(IGridInstance, IReportContext) : void onPageBreak(IGridInstance, IReportContext) : void onPrepare(IGrid, IReportContext) : void onRender(IGridInstance, IReportContext) : void ImageEventHandler.class LabelEventHandler.class ListEventHandler.class ListGroupEventHandler.class IReportEventHandler.class IRowEventHandler.class IScriptedDataSetEventHandler.class IScriptedDataSourceEventHandler.class ITableEventHandler.class ITableGroupEventHandler.class ITextItemEventHandler.class 	<ul style="list-style-type: none"> org.eclipse.birt.report.engine.api.script.eventadapter <ul style="list-style-type: none"> AutoTextEventAdapter.class CellEventAdapter.class DataItemEventAdapter.class DataSetEventAdapter.class DataSourceEventAdapter.class <ul style="list-style-type: none"> DataSourceEventAdapter <ul style="list-style-type: none"> DataSourceEventAdapter() <ul style="list-style-type: none"> afterClose(IReportContext) : void afterOpen(IDataSourceInstance, IReportContext) : void beforeClose(IDataSourceInstance, IReportContext) : void beforeOpen(IDataSourceInstance, IReportContext) : void DynamicTextEventAdapter.class GridEventAdapter.class ImageEventAdapter.class LabelEventAdapter.class <ul style="list-style-type: none"> LabelEventAdapter <ul style="list-style-type: none"> LabelEventAdapter() <ul style="list-style-type: none"> onCreate(ILabelInstance, IReportContext) : void onPageBreak(ILabelInstance, IReportContext) : void onPrepare(ILabel, IReportContext) : void onRender(ILabelInstance, IReportContext) : void ListEventAdapter.class ListGroupEventAdapter.class ReportEventAdapter.class RowEventAdapter.class ScriptedDataSetEventAdapter.class ScriptedDataSourceEventAdapter.class TableEventAdapter.class TableGroupEventAdapter.class TextItemEventAdapter.class

Event Adapter Class – Charts (chartengineapi.jar)

```

org.eclipse.birt.chart.script
  ↗ AbstractScriptContext.class
  ↗ AbstractScriptHandler.class
  ↗ ChartEventHandlerAdapter.class
    ↗ ChartEventHandlerAdapter
      ↗ ChartEventHandlerAdapter()
      ↗ afterComputations(Chart, PlotComputation) : void
      ↗ afterDataSetFilled(Series, DataSet, IChartScriptContext) : void
      ↗ afterDrawAxisLabel(Axis, Label, IChartScriptContext) : void
      ↗ afterDrawAxisTitle(Axis, Label, IChartScriptContext) : void
      ↗ afterDrawBlock(Block, IChartScriptContext) : void
      ↗ afterDrawDataPoint(DataPointHints, Fill, IChartScriptContext) : void
      ↗ afterDrawDataPointLabel(DataPointHints, Label, IChartScriptContext) : void
      ↗ afterDrawFittingCurve(CurveFitting, IChartScriptContext) : void
      ↗ afterDrawLegendEntry(Label, IChartScriptContext) : void
      ↗ afterDrawLegendItem(LegendEntryRenderingHints, Bounds, IChartScriptContext) : void
      ↗ afterDrawMarker(Marker, DataPointHints, IChartScriptContext) : void
      ↗ afterDrawMarkerLine(Axis, MarkerLine, IChartScriptContext) : void
      ↗ afterDrawMarkerRange(Axis, MarkerRange, IChartScriptContext) : void
      ↗ afterDrawSeries(Series, ISeriesRenderer, IChartScriptContext) : void
      ↗ afterDrawSeriesTitle(Series, Label, IChartScriptContext) : void
      ↗ afterGeneration(GeneratedChartState, IChartScriptContext) : void
      ↗ afterRendering(GeneratedChartState, IChartScriptContext) : void
      ↗ beforeComputations(Chart, PlotComputation) : void
      ↗ beforeDataSetFilled(Series, IDataSetProcessor, IChartScriptContext) : void
      ↗ beforeDrawAxisLabel(Axis, Label, IChartScriptContext) : void
      ↗ beforeDrawAxisTitle(Axis, Label, IChartScriptContext) : void
      ↗ beforeDrawBlock(Block, IChartScriptContext) : void
      ↗ beforeDrawDataPoint(DataPointHints, Fill, IChartScriptContext) : void
      ↗ beforeDrawDataPointLabel(DataPointHints, Label, IChartScriptContext) : void
      ↗ beforeDrawFittingCurve(CurveFitting, IChartScriptContext) : void
      ↗ beforeDrawLegendEntry(Label, IChartScriptContext) : void
      ↗ beforeDrawLegendItem(LegendEntryRenderingHints, Bounds, IChartScriptContext) : void
      ↗ beforeDrawMarker(Marker, DataPointHints, IChartScriptContext) : void
      ↗ beforeDrawMarkerLine(Axis, MarkerLine, IChartScriptContext) : void
      ↗ beforeDrawMarkerRange(Axis, MarkerRange, IChartScriptContext) : void
      ↗ beforeDrawSeries(Series, ISeriesRenderer, IChartScriptContext) : void
      ↗ beforeDrawSeriesTitle(Series, Label, IChartScriptContext) : void
      ↗ beforeGeneration(Chart, IChartScriptContext) : void
      ↗ beforeRendering(GeneratedChartState, IChartScriptContext) : void

```

Event Adapter Class – CrossTab (crosstabcoreapi.jar)

```

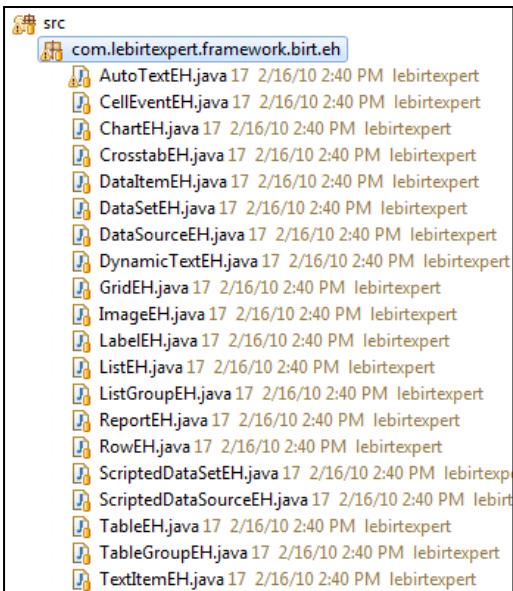
crosstabcoreapi.jar - C:\LeBirtExpert\birt-runtime-2_5_1\ReportEngine\lib
  ↗ org.eclipse.birt.report.item.crosstab.core
  ↗ org.eclipse.birt.report.item.crosstab.core.de
  ↗ org.eclipse.birt.report.item.crosstab.core.de.internal
  ↗ org.eclipse.birt.report.item.crosstab.core.i18n
  ↗ org.eclipse.birt.report.item.crosstab.core.script
    ↗ CrosstabEventHandlerAdapter.class
      ↗ CrosstabEventHandlerAdapter
        ↗ CrosstabEventHandlerAdapter()
        ↗ onCreateCell(ICrosstabCellInstance, IReportContext) : void
        ↗ onCreateCrosstab(ICrosstabInstance, IReportContext) : void
        ↗ onPrepareCell(ICrosstabCell, IReportContext) : void
        ↗ onPrepareCrosstab(ICrosstab, IReportContext) : void
        ↗ onRenderCell(ICrosstabCellInstance, IReportContext) : void
        ↗ onRenderCrosstab(ICrosstabInstance, IReportContext) : void

```

BIRT gives us the flexibility to work with either one. However, it is recommended to extend an Event Adapter. The developer only has to override the event method they need to use. And if additional events are added in the future, the class will still compile (versus implementing an Interface).

In addition, it is recommended to create a set of base classes for the Event Adapters. These classes should contain base functionality and be used when you are extending an Event Adapter for your project. Base classes also offer a buffer of protection from future API changes.

A set of base classes are included in the Reporting Framework.



```
/*
 * ReportEH.java
 */
package com.lebirtexpert.framework.birt.eh;

import org.eclipse.birt.report.engine.api.script.eventadapter.ReportEventAdapter;

/**
 * Report Event Handler Adapter Base Class
 *
 * @author Le BIRT Expert
 */
public class ReportEH extends ReportEventAdapter {

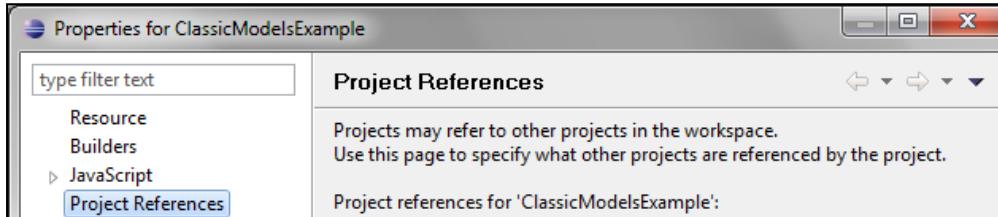
    /**
     * Constructor
     */
    public ReportEH() {
    }

}
```

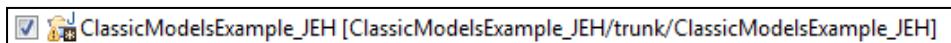
9.4.3 Referencing Java Project in BIRT Project

The great part of having the Java class in a separate project is that they can be worked on independently of the BIRT Report Designs. The BIRT Project just needs to reference the Java project in order to find the Java classes at runtime.

Right click on the BIRT Project → Properties → Project References



Check the Java Project containing your Java Classes



Click "OK"

9.5 Using Java Event Handlers – Best Practices

A couple tips when implementing Java Event Handlers.

9.5.1 Logging

Using logging will greatly help the debugging process. Using a simple logging mechanism or an existing logging method is fine.

```
Logger logger = Logger.getLogger("class-name");
```

9.5.2 Class and Event Implementation

Due to the nature of BIRT, each Report Item can only have one Java Event Handler class assigned to it. So if you want to override more than one event of an Item, you will need to put all of the event handlers in the same class. For example, if you wanted to override the “initialize()” and “afterFactory()” events of the Report Item, your Report Event Handler class should contain the two methods.

Name the class accordingly. If the Event Handler is only going to be used for the main report item of “Report1”, then name it something like “Report1ReportEH”. If the Event Handler has a generic use, such as formatting, name it something like “LabelNumberFormatEH”.

Add the Event Handlers in its own package. Be sure to extend from the Reporting Framework Base class

```
package com.lebirtexpert.classicmodels.birt.eh;

import java.util.logging.Logger;
import org.eclipse.birt.report.engine.api.script.IReportContext;
import com.lebirtexpert.framework.birt.eh.ReportEH;

/**
 * CustomerList Report Event Handler
 * @author Le BIRT Expert
 */
public class CustomerListReportEH extends ReportEH {
```

```
Logger logger = Logger.getLogger("CustomerListReportEH");

/**
 * Constructor
 */
public CustomerListReportEH() {
}

/**
 * Called when report is initialized
 * @param reportContext
 */
public void initialize(IReportContext reportContext) {

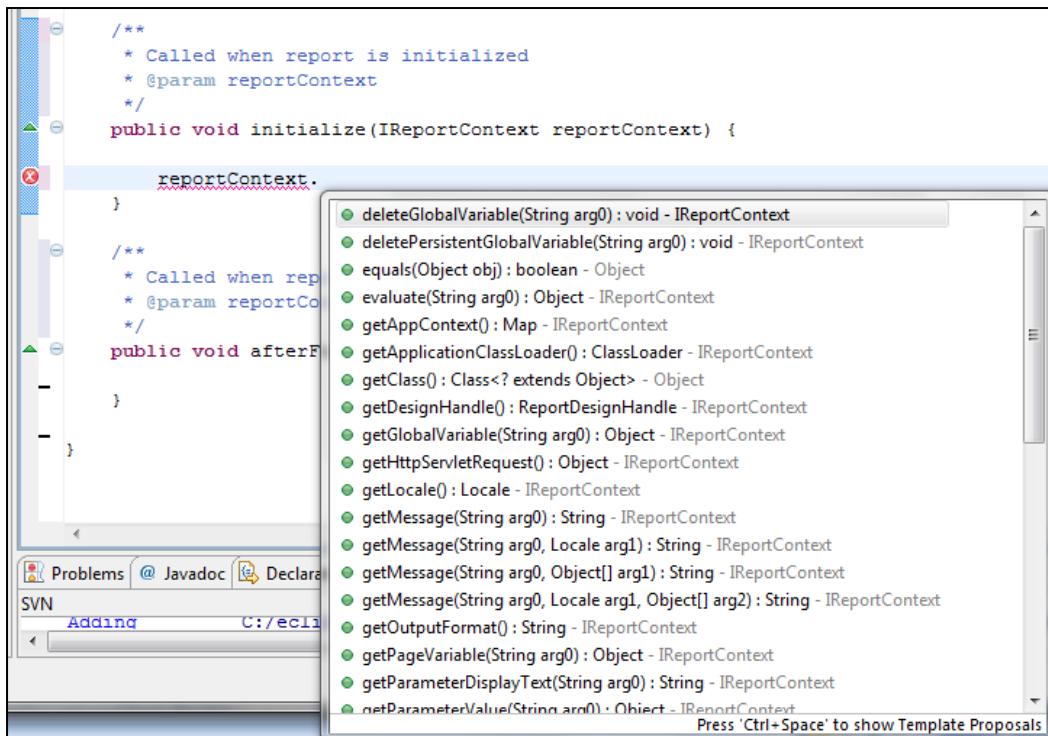
}

/**
 * Called when report is initialized
 * @param reportContext
 */
public void afterFactory(IReportContext reportContext) {

}
}
```

9.5.3 Referencing BIRT Related Objects

There are several BIRT related Objects that are available for access in the Event Handlers. One of the most commonly accessed Object is the “ReportContext.” The Report Context is passed into the Event Handler as a parameter in the method. Pretty much anything in the Report Design can be found in the Report Context.



To access the parameters of the report, use the following code:

```
String paramCountry = (String) reportContext.getParameterValue("param-name");
```

The Design Engine allows a developer to create new Report Items programmatically and add them to the Report Design. To access the Design Engine, use the following code:

```
ReportDesignHandle designHandle = reportContext.getDesignHandle();
```

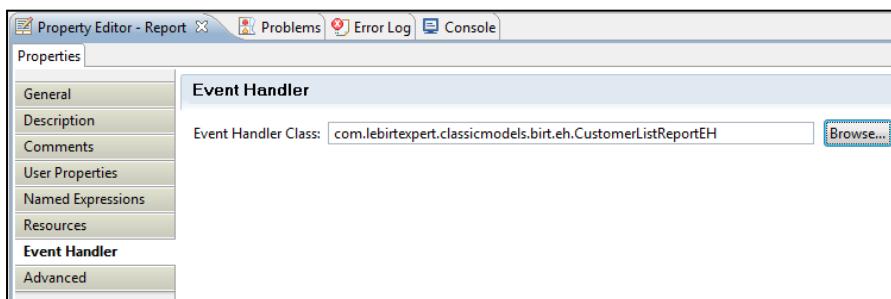
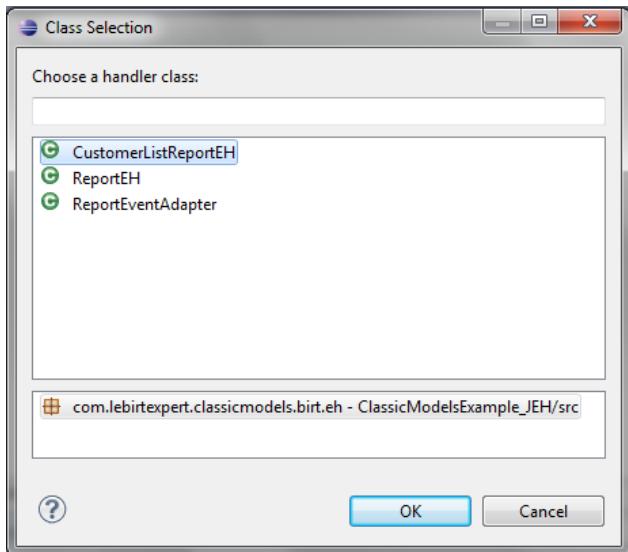
In a web environment, the HTTP Servlet Request is available to access in an Event Handler. To access the HTTP Servlet Request, use the following code:

```
HttpServletRequest httpRequest =
(HttpServletRequest) reportContext.getHttpServletRequest();
```

9.5.4 Setting in Report Design

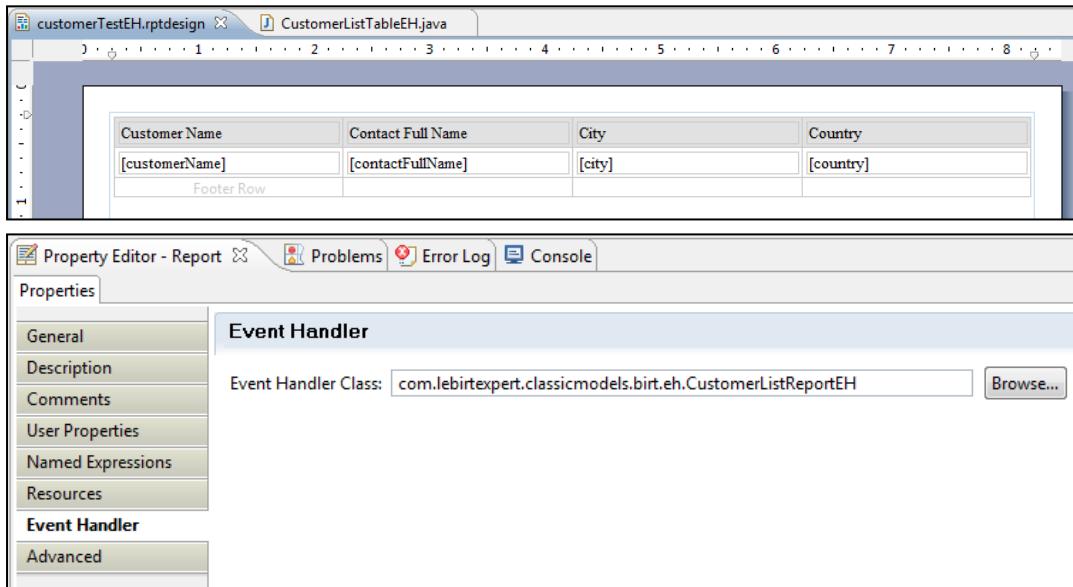
The Report Item in the Report Design must be aware of the Event Handler that was written for it. In the Designer, click on the Report Item, and navigate to the Event Handler property. Put the full package and class name in the text box.

By clicking Browse, Eclipse will give you a list of options that have that Event Adapter as a base class.



9.6 Java Event Handler Example

This example adds an additional column to the table at runtime.



Event Handler Code

```

package com.lebirtexpert.classicmodels.birt.eh;

import java.util.logging.Level;
import java.util.logging.Logger;

import org.eclipse.birt.report.engine.api.script.IReportContext;
import org.eclipse.birt.report.engine.api.script.instance.ITableInstance;
import org.eclipse.birt.report.model.api.CellHandle;
import org.eclipse.birt.report.model.api.DataItemHandle;
import org.eclipse.birt.report.model.api.LabelHandle;
import org.eclipse.birt.report.model.api.ReportDesignHandle;
import org.eclipse.birt.report.model.api.RowHandle;
import org.eclipse.birt.report.model.api.TableHandle;

import com.lebirtexpert.framework.birt.eh.TableEH;

/**
 * @author Le BIRT Expert
 */
public class CustomerListTableEH extends TableEH {

    Logger logger = Logger.getLogger("CustomerListTableEH");
}

```

```
/**  
 * Constructor  
 */  
public CustomerListTableEH() {  
}  
  
/**  
 * onCreate event handler  
 * @param table Table Instance  
 * @param reportContext Report Context  
 */  
public void onCreate(ITableInstance table, IReportContext reportContext) {  
  
    try {  
        ReportDesignHandle designHandle = reportContext.getDesignHandle();  
        TableHandle tableHandle = (TableHandle)  
designHandle.findElement("CustomerListTable"); // name defined in report design  
        tableHandle.insertColumn(5, 1);  
  
        // create new label  
        LabelHandle labelHandle =  
designHandle.getElementFactory().newLabel("lblCreditLimit");  
        labelHandle.setText("Credit Limit");  
        labelHandle.setStyleName("lblTableHeader10Bold");  
  
        // create new data item  
        DataItemHandle dataItemHandle =  
designHandle.getElementFactory().newDataItem("datCreditLimit");  
        dataItemHandle.setResultSetColumn("creditLimit");  
        dataItemHandle.setStyleName("dataItem10");  
  
        // get Header Table Row  
        RowHandle headerRowHandle =  
(RowHandle)tableHandle.getHeader().get(0);  
        CellHandle headerCellHandle =  
(CellHandle)headerRowHandle.getCells().get(4);  
        // add label to table header row  
        headerCellHandle.getContent().add(headerRowHandle);  
  
        // get Detail Table Row  
        RowHandle detailRowHandle =  
(RowHandle)tableHandle.getDetail().get(0);  
        CellHandle detailCellHandle =  
(CellHandle)detailRowHandle.getCells().get(4);  
        // add data item to detail cell  
        detailCellHandle.getContent().add(dataItemHandle);  
  
    } catch (Exception ex) {
```

```
        logger.log(Level.FINE, ex.toString());
    }
}
```

Final Result

Classic Model Cars Inc				
Detroit, MI				
Customer Name	Contact Full Name	City	Country	Credit Limit
Atelier graphique	Schmitt, Carine	Nantes	France	21000
Signal Gift Stores	King, Jean	Las Vegas	USA	71800
Australian Collectors, Co.	Ferguson, Peter	Melbourne	Australia	117300
La Rochelle Gifts	Labrune, Janine	Nantes	France	118200
Baane Mini Imports	Bergulfsen, Jonas	Stavem	Norway	81700
Mini Gifts Distributors Ltd.	Nelson, Susan	San Rafael	USA	210500
Havel & Zbyszek Co	Piestrzewicz, Zbyszek	Warszawa	Poland	0
Blauer See Auto, Co.	Keitel, Roland	Frankfurt	Germany	59700
Mini Wheels Co.	Murphy, Julie	San Francisco	USA	64600
Land of Toys Inc.	Lee, Kwai	NYC	USA	114900
Euro+ Shopping Channel	Freyre, Diego	Madrid	Spain	227600
Volvo Model Replicas, Co	Berglund, Christina	Luleå	Sweden	53100
Danish Wholesales Imports	Petersen, Jytte	Kobenhavn	Denmark	83400
Savalev & Henriet, Co.	Savalev, Mary	Lyon	France	123900

9.7 Additional Resources

Additional implementation details of Java Event Handlers are outside the scope of this book, however, there are plenty of resources available on the web for implementing Java Event Handlers.

BIRT Java Event Handlers

<http://www.eclipse.org/birt/phoenix/deploy/reportScripting.php>

<http://www.birt-exchange.org/devshare/designing-birt-reports/782-eclipsecon-2009-using-and-extending-eclipse-data-tools-dtp-/#description>

<http://digiassn.blogspot.com/2008/01/birt-creating-event-handlers-in-java.html>

BIRT Crosstab Scripting

<http://birtworld.blogspot.com/2010/02/birt-crosstab-scripting.html>

BIRT Chart Scripting

<http://birtworld.blogspot.com/2009/08/using-script-to-modify-birt-chart.html>

9.8 Deploying external Java classes

The book doesn't cover deployment of Reports, however, for the Java Event Handler to work in the BIRT Viewer, the class files must be deployed to the Web Viewer "SCRIPTLIB" directory. By default, the "SCRIPTLIB" directory is "BIRT_HOME/scriptlib." The path can be changed in the web.xml file.

In the Designer, deploy Java Event Handler JAR files to:

C:\<birt-directory>\eclipse\plugins\org.eclipse.birt.report.viewer_2.5.1.v20090821\birt\scriptlib

Other JAR files can be deployed to:

C:\<birt-directory>\eclipse\plugins\org.eclipse.birt.report.viewer_2.5.1.v20090821\birt\WEB-INF\lib

Chapter 10

Logging

In this chapter:

- JavaScript
 - Using Custom Java Classes to Log Messages
 - Java Event Handlers
 - Recommendation
 - Logging Example
 - Another Logging Example
 - Eclipse Log File
-

Logging usually gets skipped over when talking about report designs, however, it can be a very useful feature. Logging can help measure the performance of a report or help in the fixing of bugs.

10.1 Java Script

To log with JavaScript, invoke the Java Logger API. By creating an instance of the Logger object and creating a FileHandler object for the log file, messages can be sent to the log file easily.

```
importPackage(Packages.java.util.logging);

var fileHandler = new FileHandler("c:\\birt.log", true);
var baseLogger = Logger.getLogger("");
baseLogger.addHandler(fileHandler);

baseLogger.info("log-message");
baseLogger.severe("log-message");
```

Full set of Logging API calls can be found here:

<http://java.sun.com/j2se/1.4.2/docs/api/java/util/logging/Logger.html>

***Tip**

When using the Java Logging API, a new log file is created each time the report is run in the Designer. Be aware of this because it might be necessary to clean the log the directory every so often. If Eclipse gives a log file locking error, close Eclipse, delete the log directory and restart Eclipse.

10.2 Use custom Java classes to log messages

If you have existing Java classes that handle logging, you can use them in BIRT. Confirm that the JAR file is in the class path and use JavaScript to call the custom logging classes.

10.3 Java Event handlers

To leverage the existing logger in BIRT in the Java Event Handlers, use the Logger object.

```
import java.util.logging.Level;
import java.util.logging.Logger;

ILogger logger = Logger.getLogger(<Your class name>.class.getName());
```

10.4 Recommendation

Le BIRT Expert recommends creating a separate JavaScript file called “logging.js”. This file should contain functions that log messages to the log file.

10.5 Logging Example

In “logging.js”, there is a function named “initializeLogger()” that will initialize the Logger. This function is called automatically when the report runs, so it does not need to be called directly. However, any changes to the Logger can be made here.

```
importPackage(Packages.java.util.logging);

// Call function to initialize logging
initializeLogger();

/*************************************
 * Name:           initializeLogger
 * Description:   Initializes logger variables
 *
 *                 NOTE: This method can be customized
 *                 to use a Logger passed in
 *                 through the reportContext or
 *                 another type of custom
 *                 Java Class
 *
 * Parameters: string value
 * Returns:      true/false
************************************/

function initializeLogger() {
    // Log File Location
    filePath = "C:\\eclipse-galileo-birt\\log\\birt-reports.log";
    // Logger Name
    loggerName = "ClassicModelsExample";
    // Enable/Disable Logging
    blnLogging = true;
    // log file size limit (5mb)
    fileSizeLimit = 5000000;
    // number of log files
    numLogFiles = 1;

    // Enable/Disable Debug Window
    blnDisplayDebugWindow = true;
    //

    if(blnLogging) {
        // Create a file handler that uses 5 logfiles,
        // FileHandler fh = new FileHandler(pattern, limit, numLogFiles);

        //fileHandler = new FileHandler(filePath, true);
        fileHandler = new FileHandler(filePath, fileSizeLimit,
numLogFiles, true);

        // NOTE: This variable could use a Logger object
        //           passed in through the Report Context.
        //           Or, it could refer to another custom
        //           Java Object
        reportLogger = Logger.getLogger(loggerName);
        reportLogger.addHandler(fileHandler);
    }
}
```

The following function calls will send messages to the log file.

```
logMessageInfo("** logMessageInfo **");
logMessageWarning("** logMessageWarning **");
logMessageSevere("** logMessageSevere **");
```

10.6 Another Logging Example

A proven and highly recommended logging component, created by Innovent Solutions (<http://www.innoventsolutions.com>), is available at <http://code.google.com/p/birt-functions-lib/wiki/BirtLogger>

This uses a Java class to do the Logging. The Java class is then called by JavaScript.

10.7 Le BIRT Expert Debug Popup Window

Another quick way to logging is to use the Le BIRT Expert's Debug Popup window. This is a very convenient way for the developer to print debug messages to a popup window directly in Eclipse. Refer to Chapter 17 for more information.

10.8 Eclipse Log File

Eclipse uses a general log file to log any Eclipse related messages. The log file can be found in your Eclipse Workspace folder:

...\\workspace\\.metadata\\.log

More information on Logging:

- <http://wiki.eclipse.org/BIRT/FAQ/Scripting#Logging>
- http://wiki.eclipse.org/Logging_The_Events_-_Show_the_Typical_Log_Stack_%28BIRT%29

Chapter 11

Creating and Using Templates – A Starting Point for Future Reports

In this chapter:

- Creating a Template
 - Adding Report Items to a Template
 - Registering a Template
 - “New Report Wizard” Template Changes
 - Using a Template
 - Template Drawbacks
-

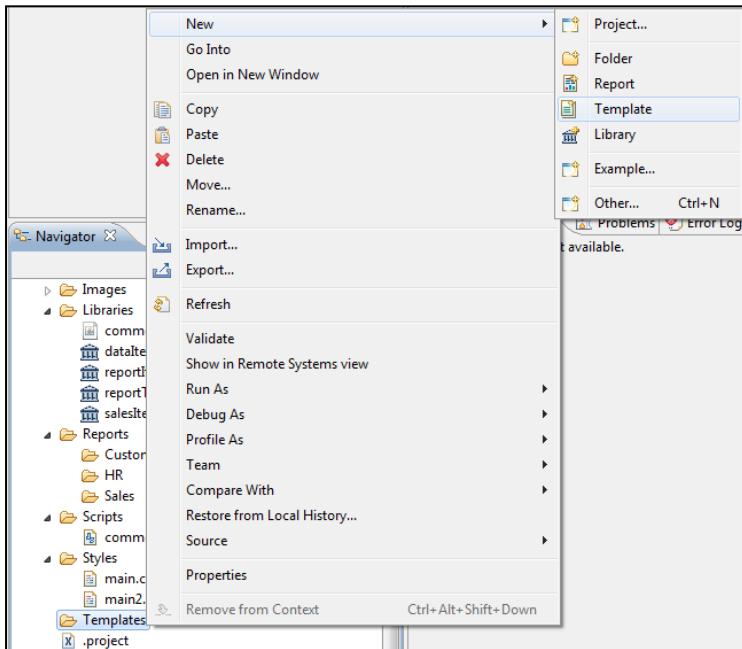
BIRT templates allow the developer to combine all of the components of their Reporting Framework into a single starting point. A Template is like a partially finished Report. The template will have the libraries, styles, Data Source and layout already in place, however, with minimal or no design. A developer will use the template as a starting point when creating a new report.

It is often necessary to create several different types of templates for different types of reports. For example, if you have a report that has a Landscape layout, there should be a “Landscape” template to choose. Same with a specific Theme, Data Source or other Layout (Portrait, Web, etc).

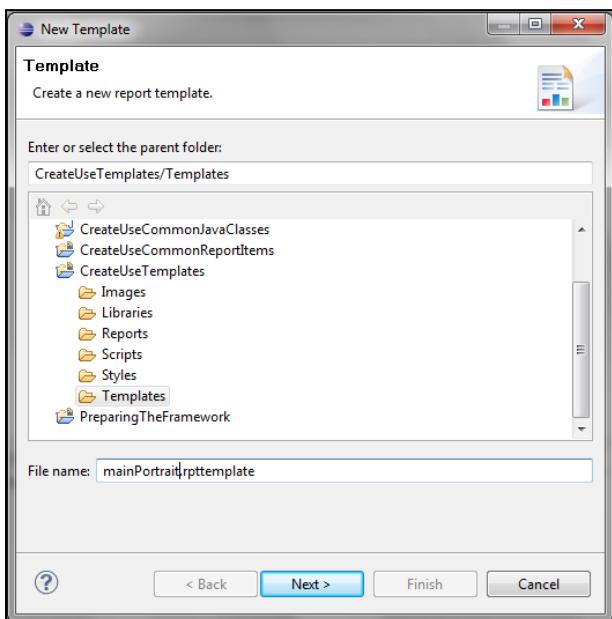
Registering the Template in the BIRT Designer allows the developer to easily choose the Template when creating a new Report.

11.1 Creating a template

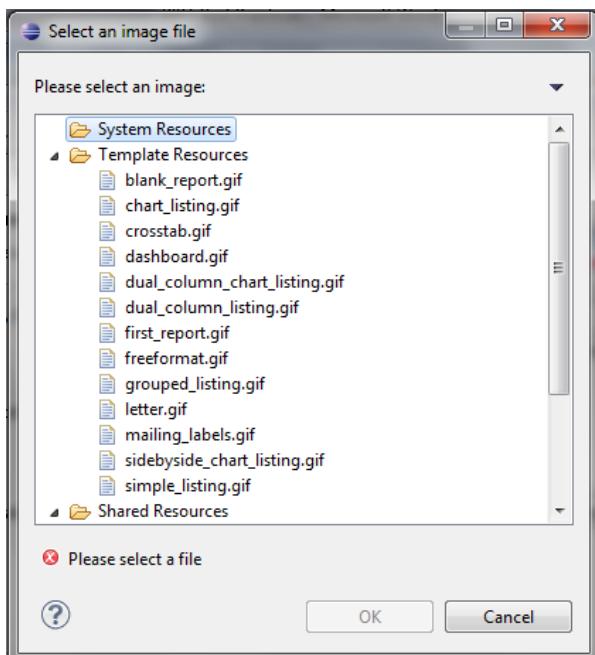
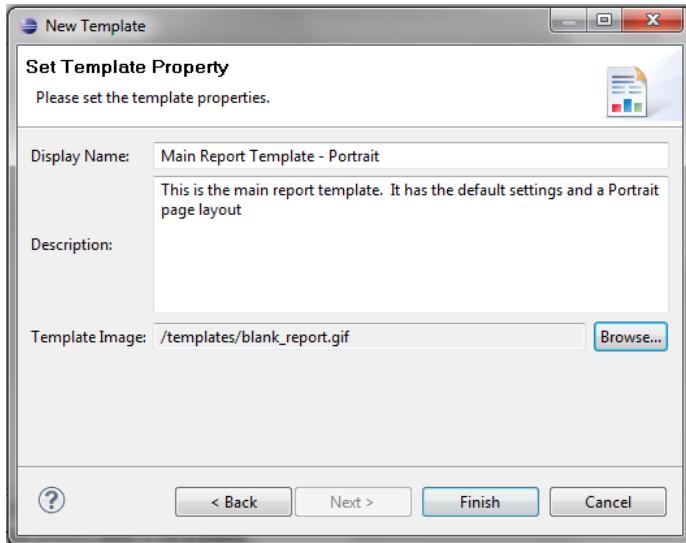
To create a template, right click on the “Templates” folder, select New → Template



Name the Template



Enter the Report Template Name and Description



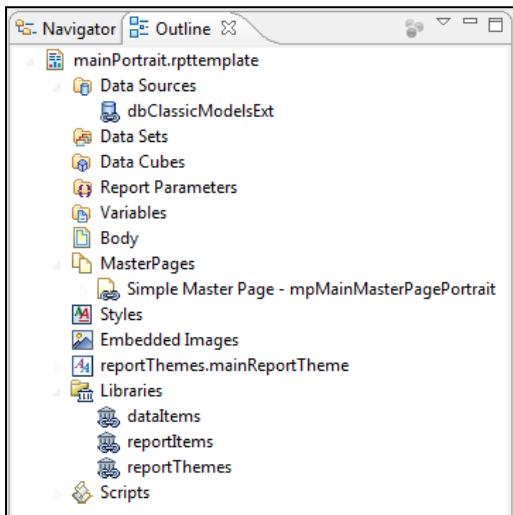
Click “Browse ...” to choose an Image File. If you don’t have a custom image to represent the template, you can choose “blank_report.gif” for now. Click “Finish”.

11.2 Adding Report Items to a Template

A Template is just like a normal BIRT Report in many ways, except that it cannot be executed directly. Adding Report Items to a Template is done in the same way as adding to a BIRT report. The goal is to create a starting point for a future report. All of the libraries, Themes, external scripts, Master Page Layouts should be set on the Template. When a developer creates a new report based on the chosen Template, everything should already be in place.

Add the following Report Items to the Template:

- All Libraries
- Set Theme
 - Delete Default Report Styles (*These should already be in your Theme*)
- Master Page (in this example, it will be Portrait)
- External Resources (such as a Resource File, Javascript or JAR files)
 - It is recommended to add these to the library instead of the Template or Report Design. However, if there are Resource files specific to the Template or Report Design, then they can be added here.
- Default Data Source

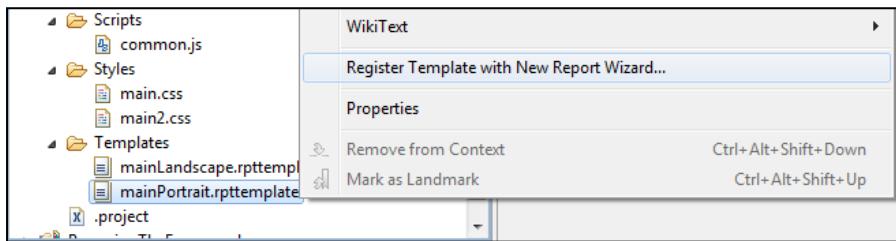


This example is a blank Report Design with everything in place. You can create more fancy and developed Templates with Data Sets, Tables and other logic already built in.

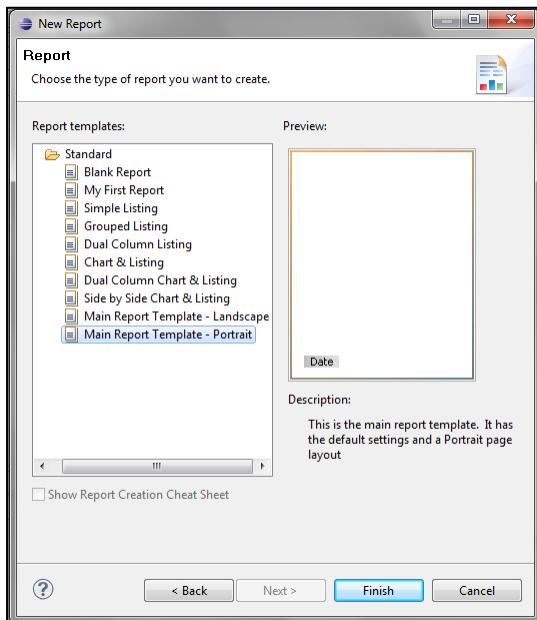
11.3 Registering a template

Having the template registered in the “New Report” wizard makes it very easy for a developer to use it as a starting point when creating a new Report. To register a Template with the “New Report” wizard, to the following:

Right click on the Report Template and choose “Register Template with New Report Wizard ...”

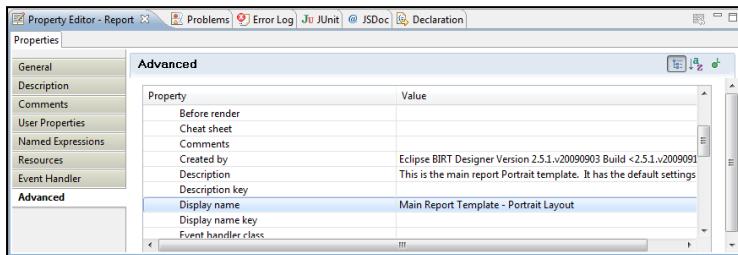


When creating a New Report using the “New Report” wizard, you will see your templates in the list, along with the image you chose for the Template.

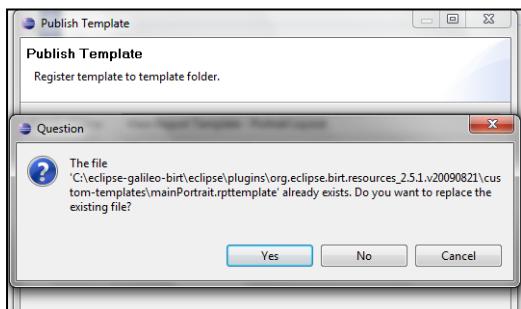


11.4 “New Report Wizard” Template Changes

To update the template in the “New Report” wizard, open the Template and make the appropriate changes. Save the Template Design.



Re-publish the Template. Right click on the Template file, select “Register with New Report Wizard ...”. Click Yes to replace the previous Template.



***Tip**

The Template must be re-published for the BIRT Designer to know about the changes.

Delete the Template from the Navigator by right clicking on the Template File and choosing “Delete”.

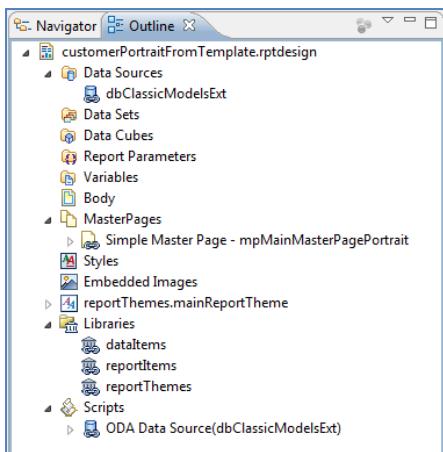
***Tip**

For the Template to be removed from the “New Report” wizard, navigate to the folder “..\\eclipse\\plugins\\org.eclipse.birt.resources_2.5.1.v20090821\\custom-templates”. (*Choose the folder that matches your Eclipse BIRT installation.*) Delete the Template from this directory. The Template will no longer be displayed in the “New Report” wizard.

11.5 Using a template

To use the Template, create a new Report using the “New Report” wizard. From the wizard, choose the template you want to start with. When the Report Design opens in BIRT, you will see all of the Report Items from the Template in place.

Notice BIRT still puts in two custom Report Design Styles for “crosstab” and “crosstab-cell”. These should already be defined in your Theme. These two styles can be removed.



11.6 Template drawbacks

Templates are an essential part of a reporting framework for establishing a starting point for all reports. It is much better for a developer to start with a template than to create a new report and manually add all of the pieces together each time.

One of the drawbacks of using a Template is that once the Report is created, any link to the Template is no longer maintained. If the Template changes (a new header is added for example), the Report Design is not updated. The Report Design carries forth as is. In BIRT, there is currently no way to create a “base” report.

To avoid this, it is best to rely on common Libraries, and just use Templates as a starting point for all Reports. Using Templates will save time and ensure that all developers are starting off at the same point.

Part II

Continuing Best Practices in Your Report Design

In this section:

- *Chapter 12 – BIRT Tables (Binding, Filtering, Sorting)*
 - *Chapter 13 – Using Charts*
 - *Chapter 14 – Planning Ahead for Internationalization (i18n)*
 - *Chapter 15 – Separating Business Logic from Display Logic*
 - *Chapter 16 – Exception and Error Handling*
 - *Chapter 17 – Debugging Reports*
 - *Chapter 18 – Improving Performance*
 - *Chapter 19 – Preparing for Different Output Types*
-

Creating and using a reporting framework is very much a part of using best practices in the report design! This section offers recommendations in other areas of your design.

Chapter 12

BIRT Tables

In this chapter:

- Understanding Column Bindings
 - Data Set Bindings
 - BIRT Table Bad Practices
-

BIRT Tables are an essential part of almost any Report Design.

Using Tables in Libraries have limited use since the Tables are not that flexible. As mentioned in section 5.3, only Table properties (not the structure) can be changed in a Table brought in from a library. If the same table (same data and layout structure) is re-used on multiple reports, it should definitely be put into a library.

12.1 Understanding Column Bindings

Column bindings are an important concept to understand when working with BIRT Reports. Column Bindings act as a layer between the Data Set and the Report Items, such as a Table or Data Control. Column Bindings define where the data is coming from and how to reference it in the Report Item.

By selecting the Table, and clicking the “Bindings” tab, the developer can view all of the Table Bindings.

The screenshot shows a BIRT Report Designer interface with a table component. The table has four columns: Customer Name, City, Country, and Credit Limit. Each column contains a data binding expression: [CustName], [city], [country], and [creditLimit]. Below the table, there is a row labeled "Footer Row" and a button labeled "Table".

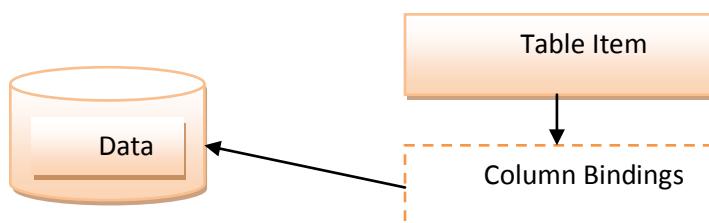
Customer Name	City	Country	Credit Limit
[CustName]	[city]	[country]	[creditLimit]
Footer Row			
Table			

The screenshot shows the 'Property Editor - Table' dialog box. At the top, there are tabs for 'Properties', 'Binding', 'Groups', 'Map', 'Highlights', 'Sorting', and 'Filters'. The 'Binding' tab is selected. Under 'Data Set:', 'dsCustomerList' is chosen. The main area is titled 'Data Column Binding:' and contains a table with the following data:

Name	Display Name ID	Display Name	Data Type	Expression	Function	Filter	Aggregate On
city			String	dataSetRow["city"]			N/A
contactFirstName			String	dataSetRow["contactFirstName"]			N/A
contactLastName			String	dataSetRow["contactLastName"]			N/A
country			String	dataSetRow["country"]			N/A
creditLimit			Float	dataSetRow["creditLimit"]			N/A
customerName	CustName		String	dataSetRow["customerName"]			N/A
customerNumber			Integer	dataSetRow["customerNumber"]			N/A
salesRepEmployeeNumber			Integer	dataSetRow["salesRepEmployeeNu..."]			N/A
state			String	dataSetRow["state"]			N/A

On the right side of the dialog box, there are buttons for 'Add...', 'Add Aggregation...', 'Edit...', 'Remove', and 'Refresh'.

Any Data Control in the Table will refer to the Binding name, and not the DataSet Name.



Important Column Binding Properties:

- *Name* – name of Column Binding
- *Display Name* – This is the name that is displayed in the Report Design. Useful if the original column binding name is very long.
- *Data Type* – data type of the column binding
- *Expression* – specifies where the data is derived. This can be from a Data Set Row, Javascript function/variable or another type of expression.

***Note**

When referring to a piece of data in the Data Set, use the expression '`dataSetRow["name-of-ds-column"]`'. This is used when pulling a value directly from the Data Set.

When referring to a piece of data in the column binding, use the expression '`row["name-of-cb-column"]`'. This is used when referring to another column in the same row that already has an aggregation or expression assigned to it. (This is the default way in the designer)

When using nested tables, use the expression '`row._outer["name-of-outer-cb-column"]`' to refer to nested values.

When working with Tables, you may notice errors related to Data Binding every once in a while (Data Binding missing, etc). Be sure to examine your Table Column Binding to see if there is anything unusual.

Because of the nature of Table Bindings, it is not recommended to dynamically switch Data Sets of a Table at Runtime. Unless you can guarantee the Data Sets have the exact same column names, or you're willing to re-create the Data Bindings for the Table, you will run into potential Column Binding errors.

More information on Column Bindings in BIRT is available on the Web here:

<http://www.birt-exchange.org/devshare/designing-birt-reports/22-data-bindings-in-birt-reports/>

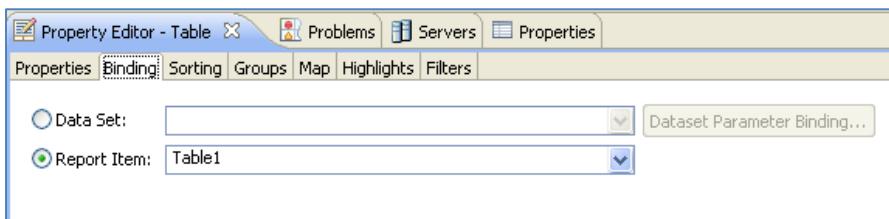
12.2 Dataset Bindings

In older versions of BIRT, every time a report item referred to the data set for the data binding, the query would re-execute. This caused performance problems when multiple tables used the same large data set. Starting in BIRT 2.3, there is a data set caching feature that is automatically used when two or more items use the same data set.

***Tip**

Dataset (Column) bindings are accessible by all report items within the Table.

In BIRT, there is a feature to use the same column bindings as another report item. If the developer has set up custom column bindings, they can be re-used by another Report Item in the same report.



12.3 BIRT Table Bad Practices

There are a few BIRT Table settings to avoid.

12.3.1 Filtering

Developers should do their best to avoid filtering in the Report Design. All filtering should be done on the data source side (in the database, web service, etc). If the Data Source does not support filtering, it is best to assign the filter to the Data Set itself.

An exception to this would be if, for some reason, two tables are using the same Data Set, and require different filters. A filter on each Table would be appropriate. It would be appropriate to add filtering to the group level if it cannot be added anywhere else.

The BIRT Designer makes it very easy to add Filters to Tables. This can be useful at times and an easy way to do things, however, keep in mind that BIRT is doing the filtering in memory, so large data sets may experience slow performance.

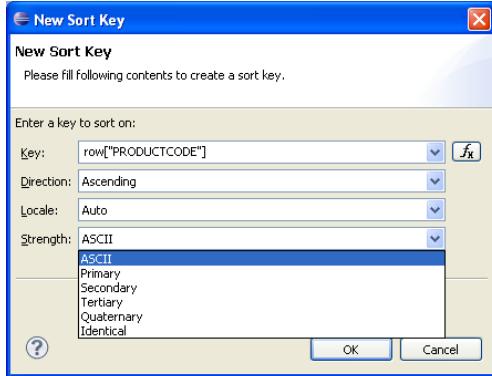
12.3.2 Sorting

As a best practice, it is best to have the Data Source do the sorting (in the database, web service, etc).

An exception to this is if the Data Source cannot sort its own data (flat file, etc). Using the table's sort feature will be handy for this scenario. Be aware that when adding a group to the table, BIRT will automatically sort the data based on the group key. If you are using grouping,

it's not necessary to add a sort key if the field is a group field. The BIRT Designer makes it very easy to add Sorting to Tables. This can be useful at times and an easy way to do things, but beware of the consequences. Keep in mind that BIRT is doing the Sorting in memory, so large data sets may experience slow performance.

In BIRT 2.5.2 and later, there is additional options for adding a sort key. These options include sorting on the locale specific version of the value and the strength of the sorting algorithm.



Chapter 13

Using Charts

In this chapter:

- Chart Output Formats
 - Which Format is Recommended?
 - Using Styles in Charts
 - Using i18n Messages in Charts
 - Chart Scripting
 - Chart Interactivity
 - Using Images in Charts
-

Charts are a powerful feature of BIRT that can help visualize the data. Charts are created through the Chart Builder interface, which provides many different options for formatting and displaying the Chart. Interactivity features allow the user to interact with the report in new ways.

Over 13 different Chart types are available to choose from. There are over 30 scripting methods available to enhance the Chart using Scripting. The developer can add markers to the Chart through the Chart Builder and dynamically using script.

In BIRT 2.6, there are many improvements to the Charting capabilities. These include:

- Radar/Polar Chart style
- Palette hashing (Using patterns for the Series Palette)
- Pattern image editor (Used for creating your own pattern for the Series Palette)
- Ability to choose the direction the slices appear in a Pie Chart
- Ability to sort on locale and sort strength
- Outside axis range indicator
- Better cube filter support
- Native PDF support for SVG charts

For a complete list of the BIRT 2.6 improvements, visit
<http://www.eclipse.org/birt/phoenix/project/notable2.6.php>.

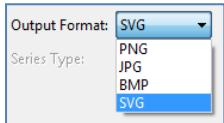
The Actuate commercial version goes one step further by providing a complete set of Flash-enhanced Charts, Maps and Gadgets that work right inside the Report Designer.

The Charting capability of BIRT is well documented elsewhere on the web (<http://wiki.eclipse.org/BIRT/FAQ/Charts2.2>). However, there are a few points in this chapter that will be covered about Charts in BIRT.

13.1 Chart Output Formats

BIRT Charts support the following output formats:

- PNG
- JPG
- BMP
- SVG



13.2 Which Format is Recommended?

Each format has its advantages, but Le BIRT Expert recommends either SVG or PNG.

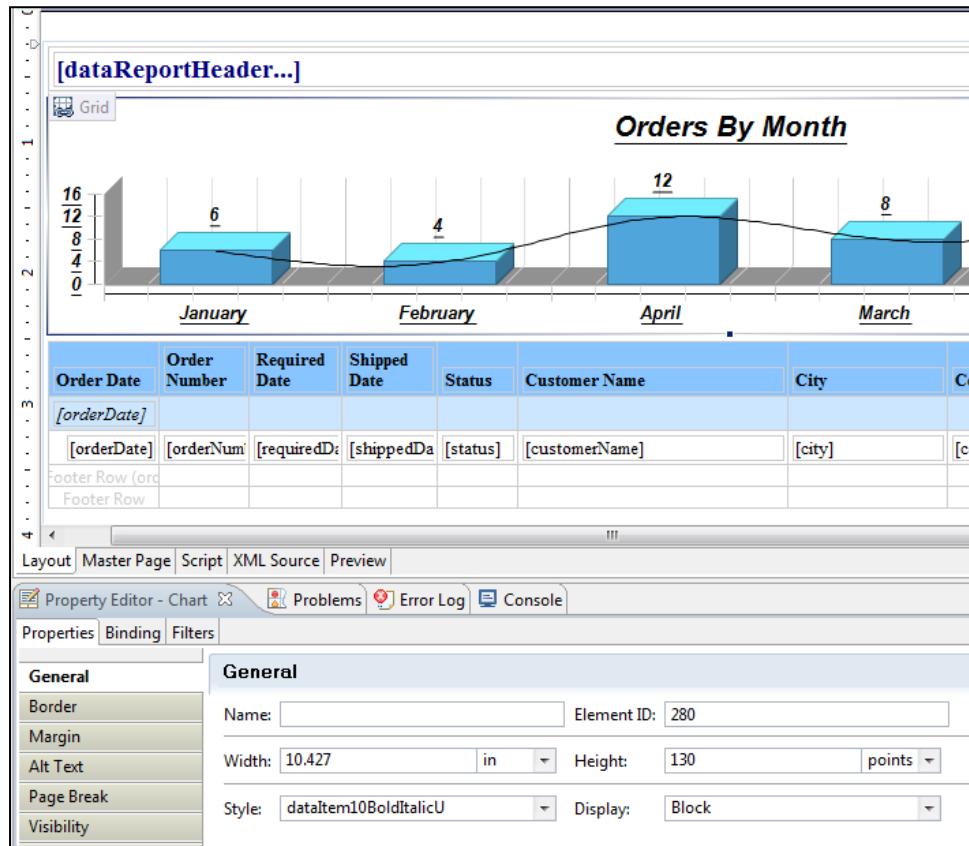
SVG – has the best interactivity support for BIRT Interactivity features. The only downside is that the user will need to install a SVG plugin. If the BIRT Viewer (either the Web Application or the Viewer in the Designer) detects that the SVG plugin is not installed, it will request the chart in PNG format. There is actually some Javascript code that runs to determine if the SVG plugin is installed. If it is, it will request the SVG version of the report. If not, then the PNG version. SVG output is only supported in HTML output (assuming the browser has SVG support). BIRT will use PNG for PDF and other report output formats. Starting in BIRT 2.6.2, BIRT provides native PDF support for SVG charts.

The SVG plugin is available at: <http://www.adobe.com/svg/viewer/install/>

PNG – has more limited interactivity support, but does support hyperlinks and tooltips (using an HTML Map). It is recommended over JPG and BMP because of its support of transparency, compression without loss and 32 bit color.

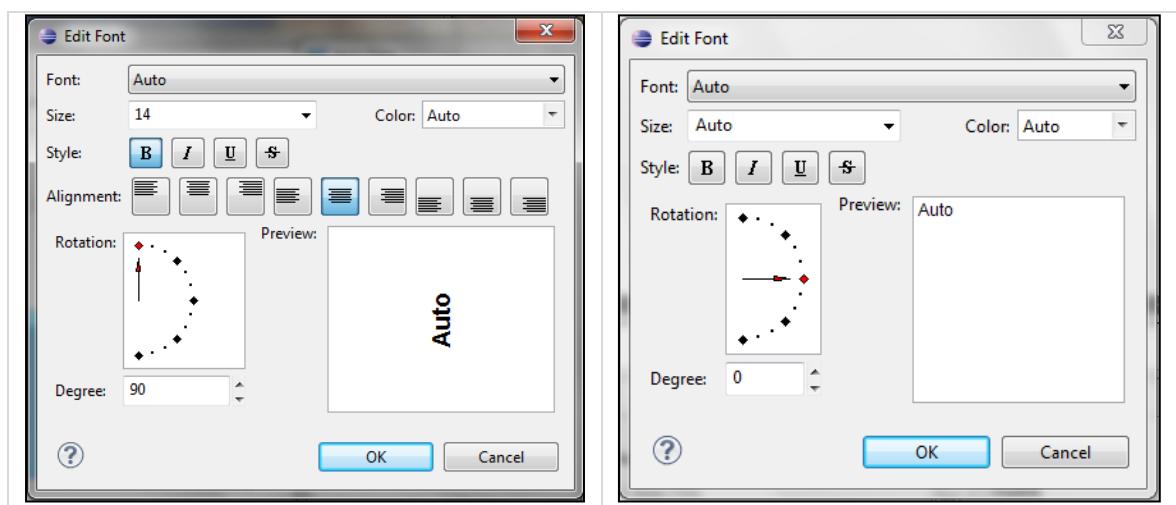
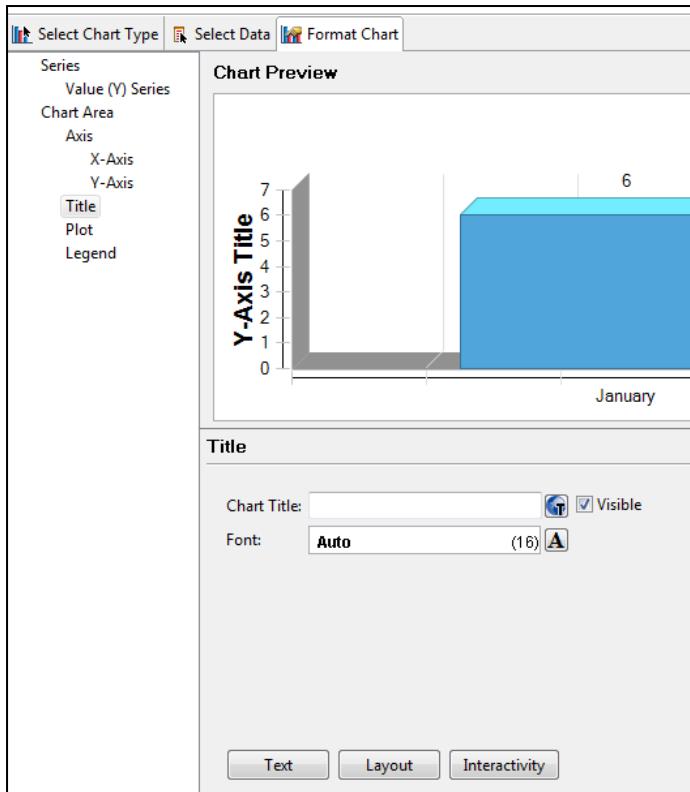
13.3 Using Styles in Charts

There is limited capability for using Styles on a Chart. The best the developer can do is apply one style to all of the text in the Chart. In the figure below, by clicking on the Chart, the Chart Properties will appear. The style option will allow the developer to choose which style to apply. In this example, “dataItem10BoldItalicU” is chosen, and all of the text in the Chart is size 10, Bold, Italic and with Underline.



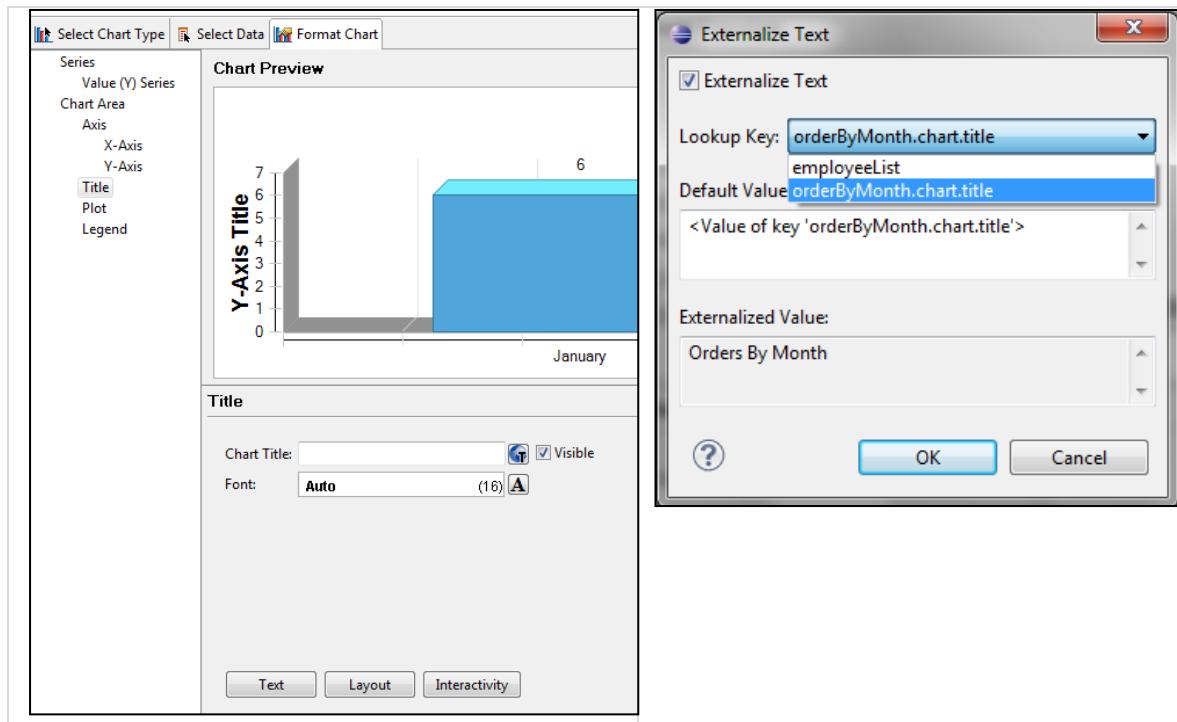
Unfortunately, the BIRT Chart Builder does not provide an interface to use existing styles for the individual Labels or other text in the Chart. The best the developer can do is use the Font properties window to adjust the size, font style, color, alignment, rotation and other properties of the text to make it fit in with the rest of the report.

By clicking the little “A” to the right of the Font text box, the following Font Properties’ window will pop up.

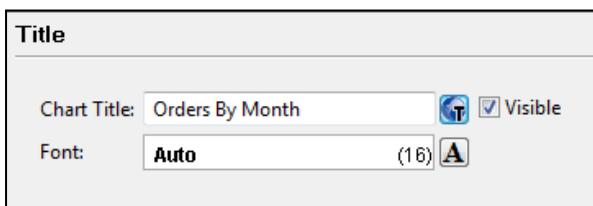


13.4 Using i18n Messages in Charts

The Chart builder allows the developer to use external messages in the Chart Labels. In the screenshot below, the Chart Title has a little “T” icon to the right of the text box. When clicked, the following screen will popup.

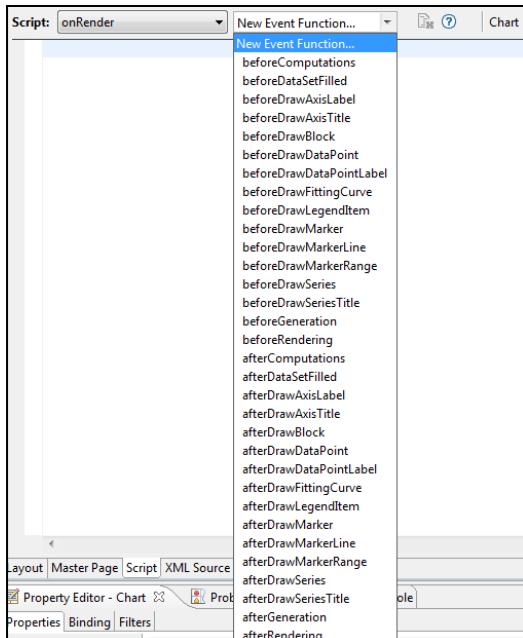


This screen allows the developer to choose text from an external resource. *If the report does not have a Resource file specified, then the option to Externalize Text will be grayed out.*



13.5 Chart Scripting

As with all other Report Items, Charts support event handler scripting. This is available for JavaScript and Java Event handlers. BIRT supports a large number of event functions related to Charts.



For a working example, see: BIRT Chart Scripting

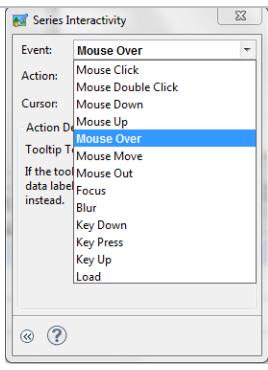
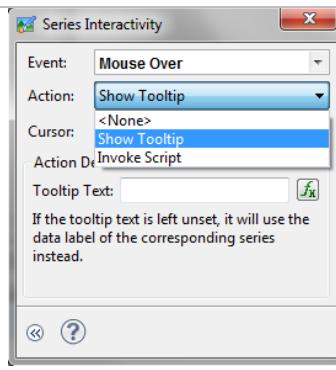
<http://birtworld.blogspot.com/2009/08/using-script-to-modify-birt-chart.html>

13.6 Chart Interactivity

BIRT provides a way to add interactivity to Charts. Events such as mouse-overs and mouse-clicks can be used to trigger actions such as tooltips or opening up new links.

In the Format Chart tab of the Chart Wizard, click the “Interactivity” button to configure the interactivity options for that particular chart item.

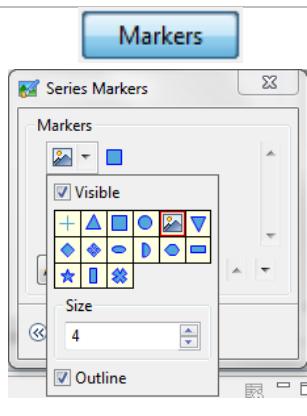
Interactivity

Events supported	Mouse-over actions supported
 <p>The screenshot shows the 'Series Interactivity' dialog with the following settings:</p> <ul style="list-style-type: none"> Event: Mouse Over Action: Mouse Click, Mouse Double Click, Mouse Down, Mouse Up Cursor: Mouse Over Action Description: If the tool tip text is left unset, it will use the data label of the corresponding series instead. 	 <p>The screenshot shows the 'Series Interactivity' dialog with the following settings:</p> <ul style="list-style-type: none"> Event: Mouse Over Action: Show Tooltip (selected) Cursor: Show Tooltip Action Description: If the tooltip text is left unset, it will use the data label of the corresponding series instead.

The PNG format only supports the mouse-over and mouse-click events. The other events are implemented in SVG, so SVG format must be enabled for the other event actions to work.

13.7 Using Images in a Chart

BIRT allows the developer to use images to improve the look of charts. An image can be used as a background or as a chart marker.

<p>In the Plot section of the Format Chart tab of the Chart Wizard, an image can be chosen for the background.</p>  <p>The screenshot shows the 'Plot' section of the 'Format Chart' dialog with the following settings:</p> <ul style="list-style-type: none"> Area Including Axes: Background: Transparent, Outline: Visible Area Within Axes: Background: Image (highlighted), Outline: None General: Name: [empty] 	<p>An image can be chosen for a chart marker</p>  <p>The screenshot shows the 'Markers' section of the 'Series Markers' dialog with the following settings:</p> <ul style="list-style-type: none"> Visible: Checked Marker icons: Various shapes like square, circle, triangle, diamond, etc., are displayed. Size: 4 Outline: Checked
---	---

For more information about how to use images in a chart, view Le BIRT Expert's article named "Spice Up Your Charts With Images and Additional Series" available at:

<http://www.lebirtexpert.com/wordpress/2010/03/18/internet-gambling-part1/>

Chapter 14

Planning Ahead for Internationalization (i18n) and Localization (l10n)

In this chapter:

- Using External Resource Files
 - Use Scripting to Display Formatted Values
 - Use Database Values for Localized Values
 - Use Existing Java Framework for Handing Localization
 - Recommendations
 - Using External Messages in Reports
-

There are a couple different ways to handle Internationalization and Localization in BIRT. The important thing is to plan ahead if you are going to need to localize your reports. Decide what method you are going to use and confirm that all translated values are available. Build the chosen method into your report framework so developers code it into the Report Designs right from the start.

14.1 Using external resource files

This is the default method in BIRT to handle Internationalization. This method involves creating the appropriate Java resource files for locale.

- <filename>_<ISO 639 language code>_<ISO 3166 region code>.properties
- reportLabels_en_US.properties – US English
- reportLabels_en_GB.properties – Great Britain English
- reportLabels_fr.properties – French

Each file should have a key/value representing a key name and the translated value

- thankyou=Thank you
- thankyou=Cheers
- thankyou=Merci

Be aware that all files must be ASCII encoded. Unicode formatting is not supported.

In the Report Design, do the following:

- Specify the resource file to use in the Report Design.
 - In the General Properties of the Report Design (or Template), scroll to the “Locale” tab. Specify the name of the file “reportLabels”. BIRT will pick up the appropriate resource file based on the locale.
 - BIRT only allows one resource file to be used. You will have to put all of key/value pairs into the same file in order to be used in a report. (In BIRT 2.6 and higher, the Designer allows multiple resource files to be used.)
 - Set the resource properties file in the “reportItems.rptlibrary”. This way, it is available to all reports.
- In the Report Labels (and other Report Items), refer to the “key”.
 - Select the label. Scroll to the “Localization” tab in the properties window. Choose the key you want to use in the Report.
- For Scripting, the localized values can be retrieved using the following syntax:
`reportContext.getMessage("thankyou");`
- The localized values are displayed at “presentation” time. Depending on the Locale, BIRT will pull the appropriate resource file and display the results.

More Information:

- <http://www.birt-exchange.org/wiki/BIRT:Internationalization/>
- <http://www.birt-exchange.org/devshare/designing-birt-reports/351-localization-of-birt-reports/>
- <http://www.birt-exchange.org/blog/2008-04-04/using-properties-files-to-localize-birt-reports/>
- <http://www.birt-exchange.org/devshare/designing-birt-reports/1119-localizing-birt-reports-with-report-parameters/>

14.2 Use Scripting to display formatted values

Using a set of hardcoded conditions, the developer can control what to display to the user based on locale. JavaScript can leverage Java Locale objects to display the correct formatting for Dates, Time and Number Formats.

BIRT offers limited formatting for different locales out of the box. Select a data item and go to “Formatting” options. You will see a “Locale” option box with “Auto” as the default. BIRT will automatically adjust the formatting based on the locale. However, using Scripting, the developer can have complete control on how values are formatted for different locales. (Just be aware that you’ll have to code all locale conditions for the formatting. Note: This can be done in a common script and re-used in other reports.)

Locale specific messages can be retrieved using JavaScript by accessing the “reportContext” variable. Using the “getMessage()” function to retrieve the locale message.

```
importPackage(Packages.java.util.Locale);  
  
var currLocale = new Locale("fr_CA"); // french canadian  
this.text = reportContext.getMessage("message-key-here", currLocale);
```

If you don’t know the locale, or want to dynamically retrieve the message according to the user’s locale, then use the “reportContext.getLocale()” method.

```
this.text = reportContext.getMessage("message-key-here",  
reportContext.getLocale());
```

More Information:

- <http://www.birt-exchange.org/devshare/designing-birt-reports/828-advanced-localization-technique/>
- <http://www.birt-exchange.org/devshare/designing-birt-reports/1119-localizing-birt-reports-with-report-parameters/>

14.3 Use Database Values for Localized Values

Another way to display localized values in the report is to retrieve it from the database. Depending on how your data is structured, this might require more hits on the database. This scenario is best for text data that has locale specific translation already stored in the database.

14.4 Use existing Java framework for handling Localization

If you need localization in your report, it is likely that you also need it in other parts of your application. If you already have an existing framework for handling localized messages, you can use JavaScript to call those classes, retrieve the messages and set them on the control.

14.5 Recommendations

- For Report Label Items, it is recommended to use the external resource files. Build the external resources (all locales) as you go.
- For local-specific formatting of data values such as Dates, Times and Numbers – it is recommended to use the out-of-the-box locale formatting options in BIRT. If these formatting options do not provide enough flexibility for your needs, then Scripting is the best alternative.
- For localized large text data, it is best to pull it from the database.

*Tip

When using external resource files and BIRT's out of the box method of locale formatting, this translation is done at "presentation" time. This means that the report has already run and has been generated. Using this technique, users from different locales can view an existing rptdocument and still see their respective locale values. If you are planning to store rptdocument files, enabling the report to do "presentation" translation of values is the recommended way.

* When using Javascript, Java or Database values to determine the locale-specific messages of the Report, this translation is done at "generation" time. If your report is on-demand only, then this method is fine, since the locale messages will be generated at run-time and saved into the rptdocument. If you need to archive the rptdocument files, you will need to run the report for each locale and save it appropriately. One way to avoid this is to use JavaScript in the "onRender" method instead of "onCreate". This ensures the value is set at presentation time instead of generation time.

14.6 Using External Messages in Reports

Using this approach allows the Report Developer to externalize the messages in the Report Design. This can be useful if you think the Report Headers, labels or other messages in the Report Design will change often, and you want to change the text without modifying the Report Design. The technique mentioned in this section will allow external messages. Using a resource property file to store the messages is not restricted to i18n use only.

Chapter 15

Separating Business Logic from Display Logic

A common mistake for Report Developers is to put too much business logic in the Report Design.

Business logic can be code that expresses business policy, how business objects interact with each other and any type of other business process flow. Business logic should remain in a central location and not be duplicated in different parts of the application.

As a best practice, avoid coding business logic into the report design itself. If you need to include business logic in the report, do it in either of the following ways:

- 1) Use existing Java classes that already have the business logic in them.
- 2) Put the logic in a stored procedure or call a Web Service that can perform the business function.
- 3) Put any business logic into a separate library, JavaScript file or Java class. It is best to put all business logic together in one place so you have a single point of change.

*Tip

An example of using business logic in a report design would be for the report to perform a special financial calculation based on the latest currency exchange rate. It is better to keep this special calculation out of the Report Design. If that is not possible, then keeping it in a common library, JavaScript file or Java Class with other business logic routines is the best way to go.

Chapter 16

Exception and Error Handling

In this chapter:

- No Data Messages
 - Catching an Exception in JavaScript
 - Handling Exceptions in Java Events Handlers
 - Forcing Exceptions to Fail a Report
 - Display Error Messages with Scripted Data Sets
 - Error and Exception Messages in the BIRT Viewer
 - Error and Exception Messages Using a Custom Viewer
-

BIRT 2.5.1 has dramatically improved over previous versions in giving the developer more descriptive error messages to try to solve the problem. However, to the average user, the exception messages look unfriendly. There are limited out-of-the-box ways to change how BIRT handles errors and exceptions. This section will go through some of the options of changing how BIRT handles and displays errors and exceptions.

***Tip**

Unfortunately, BIRT does not have a global mechanism to control Errors and Exceptions that might occur in Reports.

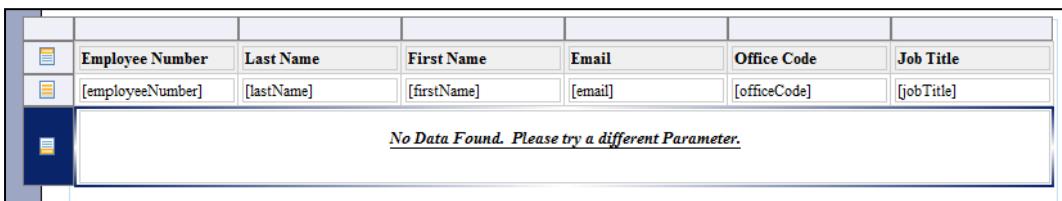
16.1 No Data Messages

If a report does not retrieve any data according to the parameters chosen, by default, BIRT will display an empty report. Some users would prefer a message saying something like “No data found”.

In the Table Footer, merge all columns together (if your report is using the Table Footer, add an additional footer row below). Add a label to that row, and enter the text “No Data Found. Please try a different Parameter”.

Employee Number	Last Name	First Name	Email	Office Code	Job Title
[employeeNumber]	[lastName]	[firstName]	[email]	[officeCode]	[jobTitle]
<u>No Data Found. Please try a different Parameter.</u>					
Table					

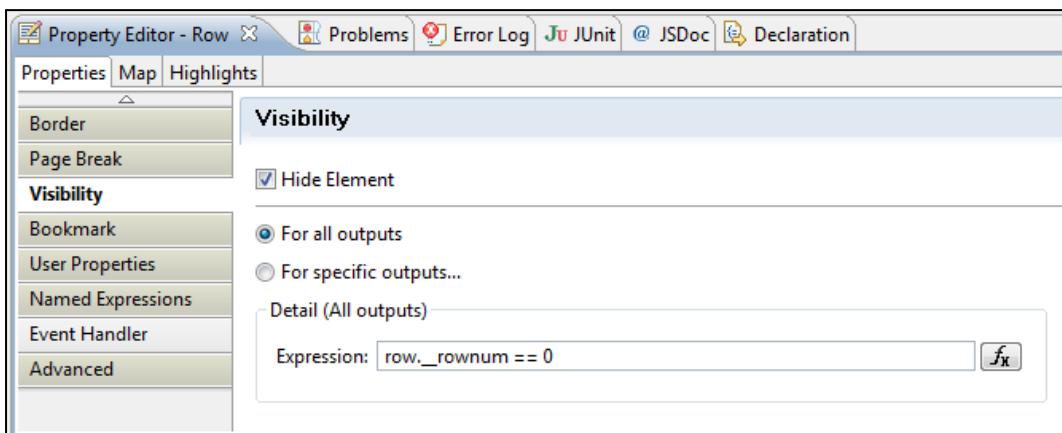
Select the Table Footer row that was just added. View the “Visibility” properties of the row.



The screenshot shows the Table Footer row selected in the report designer. The row contains the message "No Data Found. Please try a different Parameter." The "Visibility" properties are being viewed for this row.

Employee Number	Last Name	First Name	Email	Office Code	Job Title
[employeeNumber]	[lastName]	[firstName]	[email]	[officeCode]	[jobTitle]
<u>No Data Found. Please try a different Parameter.</u>					

Check “Hide Element” and “For all outputs”. Enter the expression “row.__rownum == 0”. This means to hide this element, unless there are no rows in the table. If there are no rows, then there is no data. If there is no data, then display this message.



Run the report with data

Classic Model Cars Inc					
Employee Number	Last Name	First Name	Email	Office Code	Job Title
1002	Murphy	Diane	dmurphy@classic1		President

Run the report without data

Classic Model Cars Inc					
Employee Number	Last Name	First Name	Email	Office Code	Job Title
<i>No Data Found. Please try a different Parameter.</i>					

***Tip**

The label that displays “No Data Found” message can be kept in the Library for re-use on other reports.

16.2 Catching an Exception in JavaScript

When using a lot of JavaScript code, especially when calling external Java Objects, it is a good idea to use the “try, catch and finally” statements. These statements are available in JavaScript and supported by BIRT.

```
try {  
    // JavaScript code  
} catch(err) {  
    // use "err.message" to get message  
} finally {  
    // do something  
}
```

16.3 Handling Exceptions in Java Event Handlers

Handling exceptions in Java Event Handlers should be done like any other piece of Java code. If the developer desires to raise the Exception up to the Report Level, they can use the “org.eclipse.birt.core.exception.BirtException” class. This allows them to set the message and severity level and add the Exception to the Report Context. The BIRT Viewer will automatically pull the exceptions from the Report Context and display them by default in the Browser.

Covering this subject in more detail is out of scope for this book, but more information can be found at: <http://birtworld.blogspot.com/2009/03/raising-errors-in-event-handlers.html>

16.4 Forcing Exceptions to Fail a Report

There are some situations where the developer may want to fail the report on purpose. This may be a result of bad or incomplete parameter data, incorrect data from the Data Source, or a number of other reasons. Raising the BirtException, as mentioned in the previous section, will allow the Developer to fail the report.

Using the “BirtException” class directly in the Script method of a report will cause the Report to stop running.

```
importPackage(Packages.JavaClass.org.eclipse.birt.core.exception.BirtException);
throw new BirtException("Something Happened in the Report Design");
```

The report will stop, however, it will not display an exception message. To get BIRT to display the Exception message, it must be added to the Exception list of the ReportContext. To do this, see the article listed at <http://birtworld.blogspot.com/2009/03/raising-errors-in-event-handlers.html>.

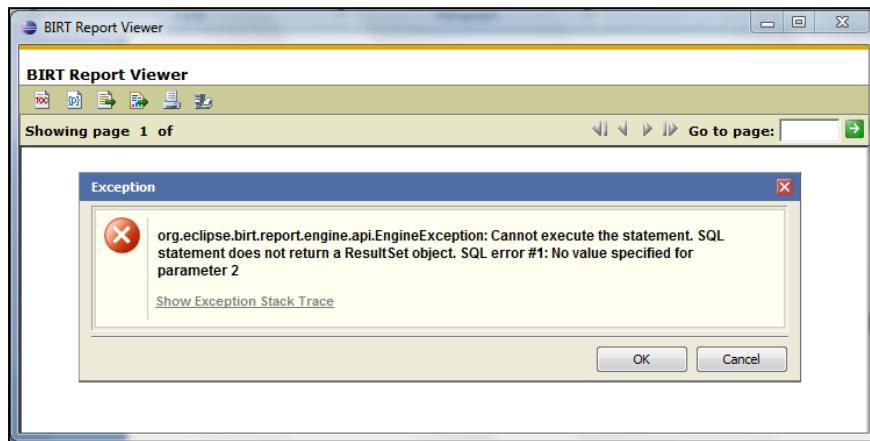
16.5 Display Error Messages with Scripted Data Sets

Reports with a typical Data Source and Data Set will by default display the appropriate exception message in the BIRT Viewer. Using a Scripted Data Source/Set is currently the only way to completely control the Exception and Error messages, especially when related to Data Access. If you are using a Scripted Data Source, you have complete control on how the Data Source and Data Set is managed. Any Exception or Error that occurs can be captured using your own code and handled appropriately.

If an Error occurs, use code to display a custom message in the Report. This means the Report will still run, but a message will be displayed in the Report saying the Report failed. This can be done with custom code. It may be necessary to drop most Report Items from the Report, and then add a Label with the error message.

16.6 Error and Exception Messages in the BIRT Viewer

The default BIRT exception message in the BIRT Viewer is fine for development. However, many typical business users may not like the default exception message that is shown by the BIRT Viewer. There are no out-of-the-box options in BIRT to control this exception message. Reports with a typical Data Source and Data Set will by default display the appropriate exception message in the BIRT Viewer.

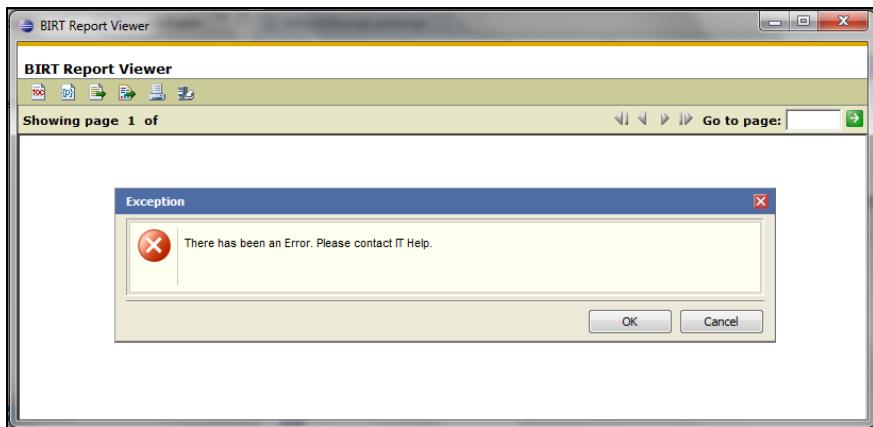


The best option is to make modifications in the BIRT Report Viewer. The JSP's in the BIRT Viewer responsible for displaying the Exception message are:

... \plugins\org.eclipse.birt.report.viewer_2.5.1.v20090821\birt\webcontent\birt\pages\birt\dialog\ExceptionDialogFragment.jsp

... \eclipse\plugins\org.eclipse.birt.report.viewer_2.5.1.v20090821\birt\webcontent\birt\pages\common\Error.jsp

In these JSP's, you will see the code for accessing the Exception messages. Custom Error messages can be created based on the type of Exception messages that are found. The above message has been changed to the following.

***Tip**

Specific Exception Messages give clues to hackers on vulnerability of your System. Le BIRT Expert recommends hiding Report Exceptions as much as possible to make your Reports friendlier to end users and to hide vulnerabilities from hackers. However, exception Messages should still be logged so developers can understand what exception occurred.

16.7 Error and Exception Message using a Custom Viewer

Creating a Custom Viewer allows the developer to completely control the Error messages your users see. Using the BIRT Report Engine, Exception messages can be extracted from the Report Context and the developer can choose to display a message of their choice. Creating a Custom Viewer will require a lot of extra coding, but will give the developer complete control over how the Report is displayed. This topic is out of scope for this book, but more information about building a custom BIRT Report Viewer using the BIRT Engine is available on the web.

More information on deploying with the BIRT engine can be found at
<http://www.eclipse.org/birt/phoenix/deploy/>.

Chapter 17

Debugging Reports

In this chapter:

- Using Logging
 - Using the Le BIRT Expert Debug Popup Window
 - Using the JavaScript Debugger
 - Using the Java Debugger for External Java Classes
 - Using the Process of Elimination
-

There are several ways to debug a report in the BIRT Designer. BIRT is a very flexible tool and allows complex logic, unusual Data Sources and calling external Java classes. The ability to successfully debug a report is essential to getting it to work correctly. This chapter discusses the options available to developers for debugging reports in the BIRT Designer.

17.1 Using Logging

Having a logging framework in place is a great way to help debug your reports. Printing out variable values and other messages can greatly help a developer find the root of the problem. Refer to Chapter 10 for more information on how to do logging in BIRT.

17.2 Using the Le BIRT Expert Debug Popup Window

A useful way to display quick messages inside the Designer is to use the following small JavaScript function. This function will open a “JFrame” inside of Eclipse and print a message to it. Using a Logging framework is preferred, but as an alternative, this method is a quick and dirty way of displaying messages during runtime.

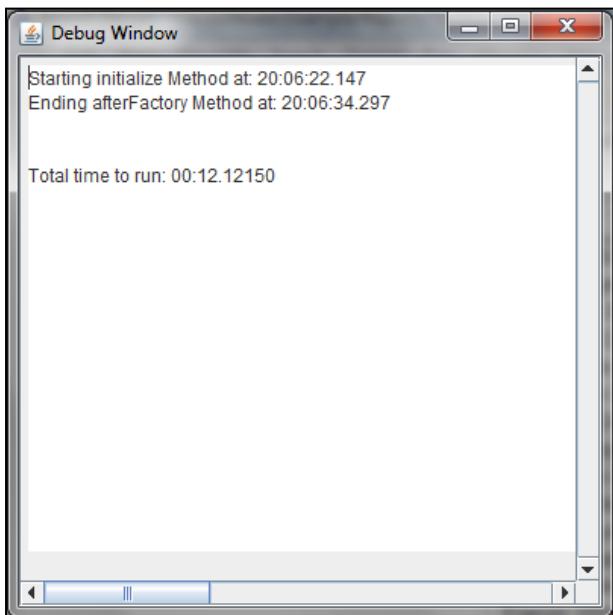
In the “logger.js” file, the “initialize()” function will determine if the debug window should be enabled or not. Set to false to disable the debug window.

```
// Enable/Disable Debug Window  
blnDisplayDebugWindow = true;
```

Use the “logToDebugWindow(msg)” function to log a message to the popup window.

```
logToDebugWindow("Starting initialize Method at: " +  
getTimeFromDate(getStartTime()));
```

The Window will popup and display the messages.



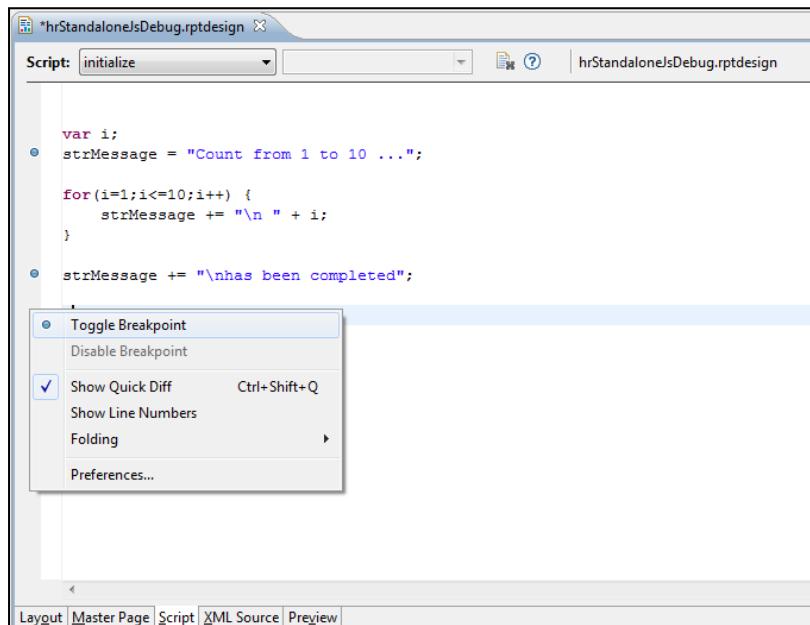
***Tip**

The Debug Popup Window will only appear when the report is running inside the Designer. It is recommended that the debug window messages be removed or the debug feature be disabled (using the blnDisplayDebugWindow variable) before the report is deployed.

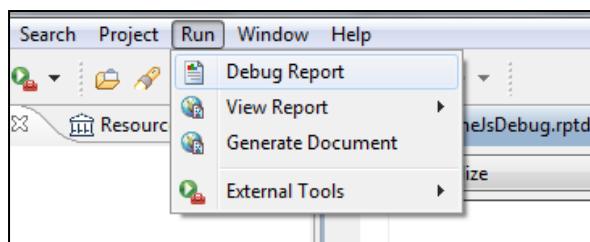
17.3 Using the JavaScript Debugger

As of version 2.3, BIRT ships with an integrated JavaScript Debugger. This can prove invaluable when debugging JavaScript in BIRT. To use the debugger, to the following:

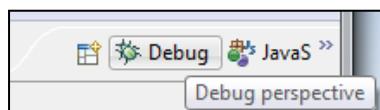
Set breakpoints in the Script. Navigate to the Script you want to debug. Double-click in the gray area on the left side of the Script window. Alternatively, right-click and choose “Toggle Breakpoint”.



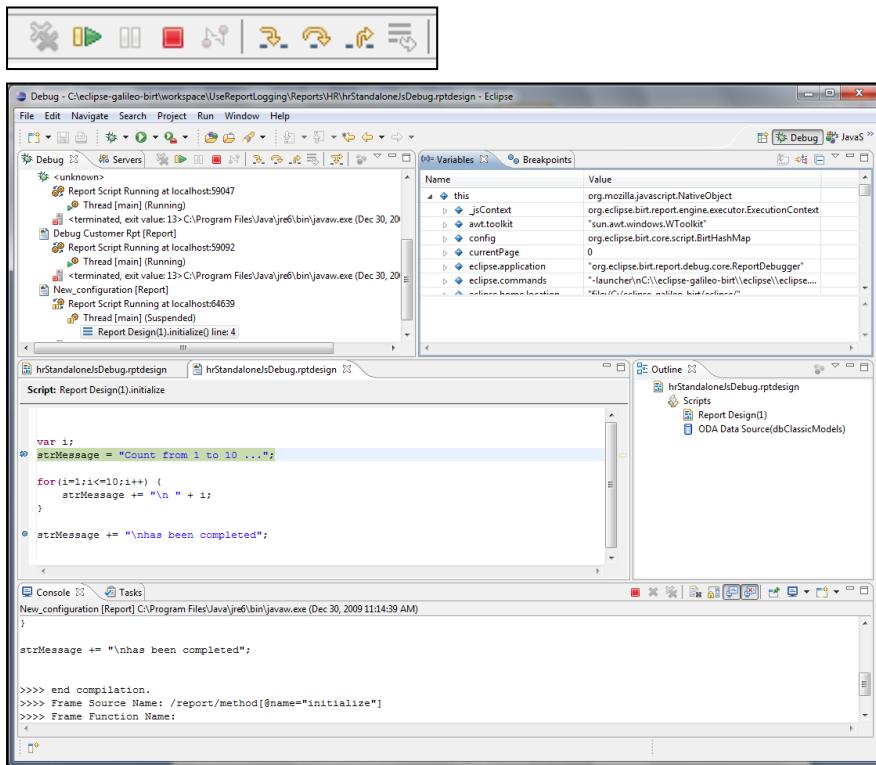
To run the report, go to the “Run” menu option of Eclipse and choose “Debug Report”.



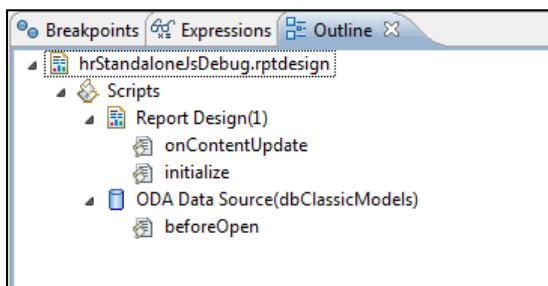
Click the “Debug” perspective to view all of the Debug Options



Step through the Script as needed. Notice the Variables window will display the value of the JavaScript variables.



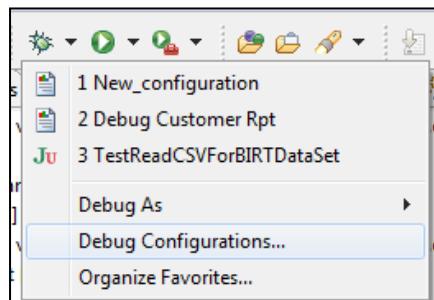
The outline tab will show where in the Report the Script is running



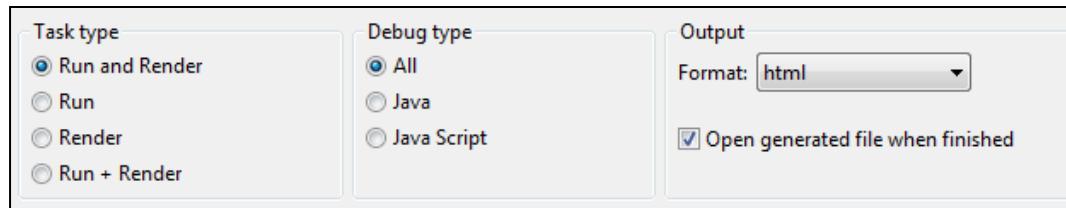
There are a few quirks with the JavaScript Debugger that the developer should be aware of.

- The JavaScript Debugger is relatively new, and thus a little buggy. Expect small bugs to occur, from the Debugger not launching to the output not being generated.
- If you are using external Java Classes, confirm that these classes have been added to the class path in the Debug Configuration screen.

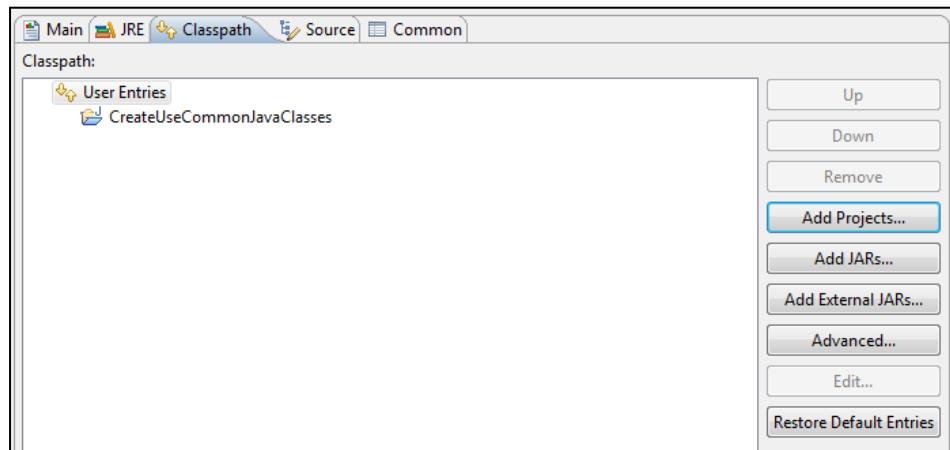
In the Debug Perspective, open the Debug Configuration screen



Confirm “JavaScript” or “All” has been chosen for “Debug Type”. If you wish to view the final Report output, check the box to “Open generated file when finished”.



Click on the class path tab and add the appropriate Java Projects or JAR files.



- The JavaScript debugger “Variable” window will only pick up the values of JavaScript variables that are declared locally using the “var” keyword. To see the value of a globally defined variable (a variable used without the “var” declaration), highlight the variable, right-click and choose “add to watch”. This will add the variable to the “Expressions” window and the value can be tracked from there.

The variables “this” and “i” are picked up, but not “strMessage”.

```

var i;
strMessage = "Count from 1 to 10 ...";

for(i=1;i<=10;i++) {
    strMessage += "\n " + i;
}

strMessage += "\nhas been completed";

```

Name	Value
this	org.mozilla.javascript.NativeObject
i	1.0

To view “strMessage”:

```

strMessage = "Count from 1 to 10 ..."
          1
          2
          3
          4
          5
          6
          7
          8
          9
          10"

```

- There seems to be a bug in the JavaScript Debugger that causes the Debugger not to pick up some external resources, including external JavaScript files. The Debugger also seems to not pick up all breakpoints, especially if those breakpoints are on Report Items in a Library. Le BIRT Expert has experienced mixed results when using the JS Debugger. Hopefully this will be fixed soon.
- The JavaScript Debugger works best with no external JavaScript “js” files and all code in the Report Design itself. If you are experiencing a very hard to solve bug and the JavaScript debugger is not working for your report, then copy the JavaScript code into a new standalone report (and remove the “js” file reference and other libraries) to use the JavaScript debugger. Once you find the problem, revert back to the original report and fix the code.

***Tip**

If you are having problems with the Debugger, check the main log file for Eclipse. It is located in your Eclipse Workspace folder: “...\\workspace\\.metadata\\.log”

17.4 Using the Java Debugger for External Java classes

The BIRT Designer allows the developer to take advantage of the Eclipse Java debugger for debugging external Java classes and Java Event Handlers. This can prove very useful when trying to solve problems with Java Event Handlers.

*Tip

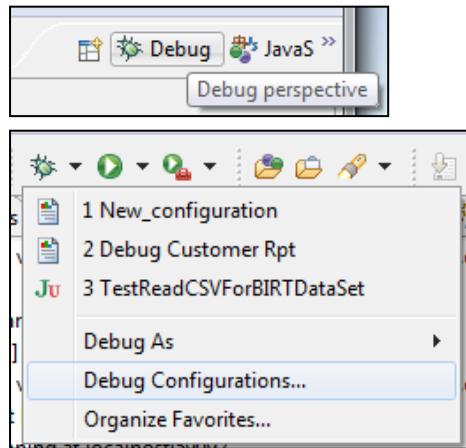
The Java Debugger in Eclipse is very stable and Le BIRT Expert hasn't really experienced many problems with it. The main thing is to confirm that the Java files are added to the class path of the Debug Configuration.

* Be aware that launching the debugger for Java in Eclipse can be memory intensive.
In the Java Source code, set the appropriate Break Points.

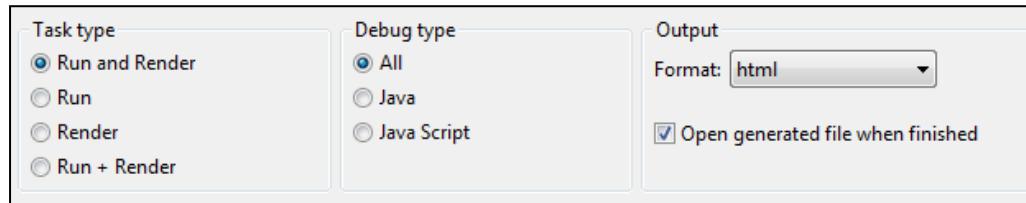
The screenshot shows the Eclipse IDE interface with the Java code for `DataSourceProperties.java`. The code defines a class with private fields `dbURL`, `dbUserId`, and `dbPassword`. It includes a comment block for a method that reads property values from a file, and a method implementation that sets hardcoded values for `dbURL` and `dbUserId`. A context menu is open over the code, with the 'Toggle Breakpoint' option highlighted. Other options in the menu include 'Disable Breakpoint', 'Go to Annotation' (with a keyboard shortcut of `Ctrl+1`), 'Add Bookmark...', 'Add Task...', 'Show Quick Diff' (with a keyboard shortcut of `Ctrl+Shift+Q`), 'Show Line Numbers', 'Folding', 'Preferences...', and 'Breakpoint Properties...'. The status bar at the bottom shows tabs for JUnit, Javadoc, Declaration, and Console.

```
public class DataSourceProperties {  
  
    private String dbURL = "";  
    private String dbUserId = "";  
    private String dbPassword = "";  
  
    /**  
     * Open, read, parse property file  
     * @param filePath  
     * @return result (success/fail)  
     */  
    public String readPropertyValues(String filePath) {  
  
        // TODO: OPEN PROPERTY FILE  
        // TODO: PARSE VALUES  
        // TODO: STORE IN LOCAL VARIABLES  
  
        // hardcoded values for example  
        this.dbURL = "jdbc:mysql://localhost:3306/ClassicModels";  
        this.dbUserId = "root";  
    }  
}
```

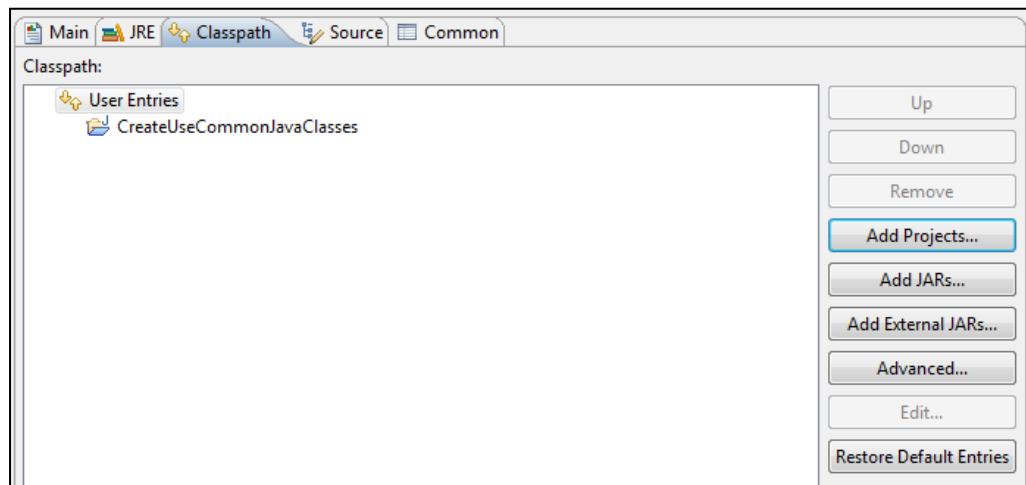
Open the Debug Perspective and open the Debug Configuration Screen.



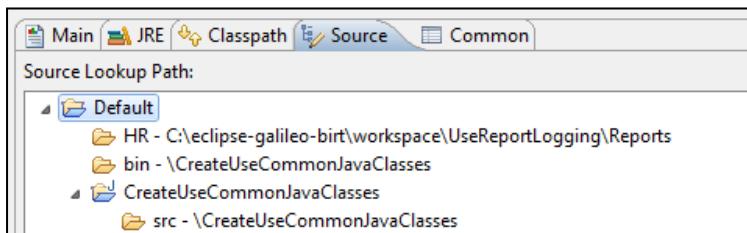
Confirm that “All” or “Java” is selected under “Debug Type”. Check the box if you want Eclipse to open the generated Report when finished.



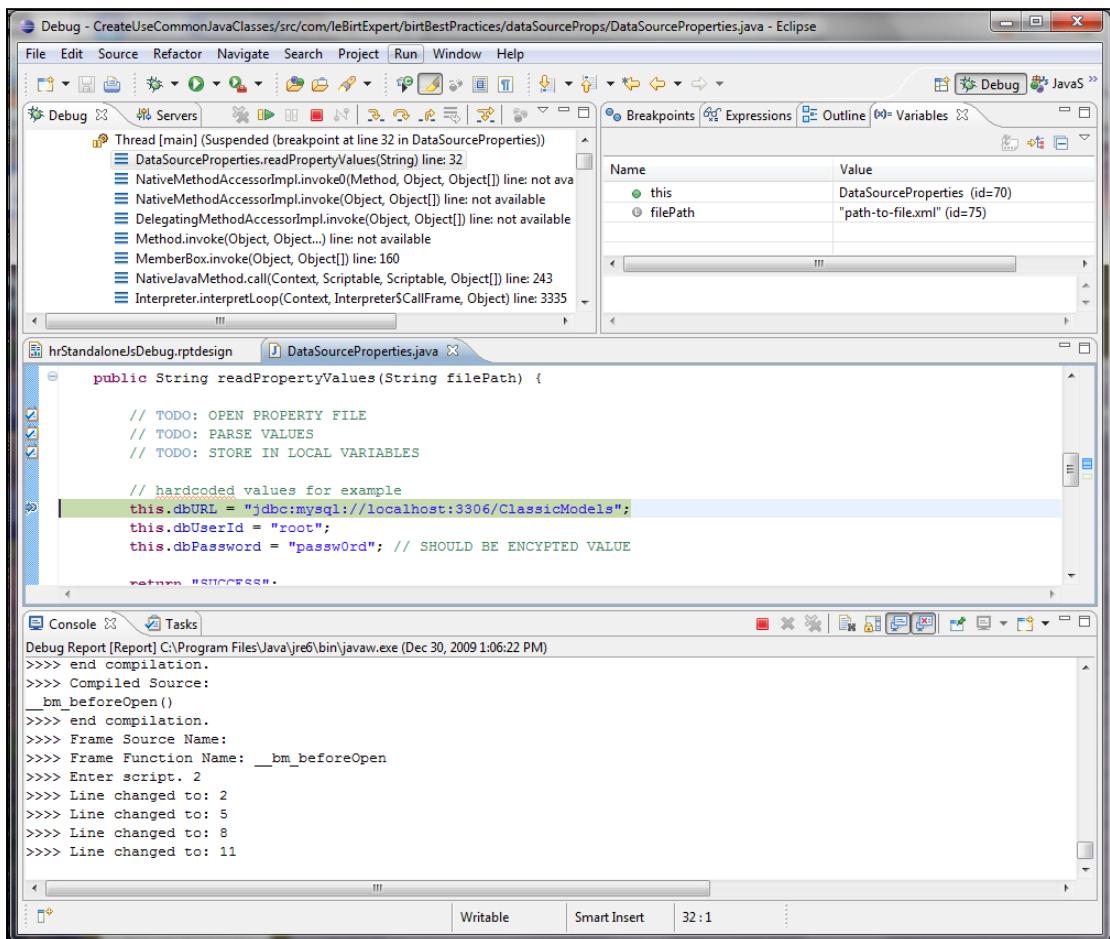
Click on the class path tab and add the appropriate Java Projects or JAR files. Confirm that your Java Event Handlers or other Custom Java files are in the Class Path.



Open the “Source” tab. Confirm that the Java Source code is listed.



Run the Report in Debug mode. The Debugger will stop at the breakpoint specified in the code.



17.5 Using the Process of Elimination

Never underestimate the process of elimination. If you are experiencing problems in your report, remove a section of the report or code piece by piece and re-test to try to find what is causing it. If necessary, go back to where it was last working, and slowly add the items or code again and re-test to try to find the problem.

Chapter 18

Improving the performance of BIRT Reports

In this chapter:

- Using Filters and Sorting in the Design
 - Simplify the Report Design
 - Measuring Performance
-

The performance of a BIRT report remains largely dependent on the Data Source Query. However, there are a few things in the report design that affects the performance of a report.

18.1 Using Filters and Sorting in the Design

It is recommended to let the data source do any filtering or sorting of the data. Adding a filter or sorting in the Report Design will slow down the processing of the BIRT report. The filtering or sorting is done in memory and will take longer to process than if it was done in the data source.

In BIRT 2.5.2 and later, there is additional options for adding a sort key. These options include sorting on the locale specific version of the value and the strength of the sorting algorithm. Try adjusting the strength of the algorithm to improve sorting results.



18.2 Simplify the Report Design

Excessive controls in the Report design cause the report generation to slow. The more items in the design, the more processing and memory it requires to execute. Eliminate unnecessary grids, tables and other report items in the design. Charts also take additional time to generate. The number of controls per page is directly related to the processing time. A report with 50 controls per page should take twice as long to process as one with 25 controls per page.

18.3 Measuring Performance

Use the timer functions in “common.js” to record the start and stop times of the report. Compare these times with the amount of time the database takes to retrieve the data. A large difference means that there is something in the Report Design that is causing additional processing time.

```
setStartTime(); // used to start the timer, typically put in initialize()  
setEndTime(); // used to stop the timer, typically in afterFactory()  
getTimeFromDate(dateObject); // gets time as string for output  
getTimerResults(); // gets time diff between start and end time for output  
  
Usage:  
setStartTime(); // used to start the timer  
logToDebugWindow("Starting initialize Method at: " + getTimeFromDate(getStartTime()));  
  
setEndTime(); // used to stop the timer  
logToDebugWindow("Ending afterFactory Method at: " + getTimeFromDate(getEndTime()));  
  
logToDebugWindow("Total time to run: " + getTimerResults());
```

Use operating system tools to measure CPU and Memory usage to determine the amount of resources being used.

- Windows Task Manager
- Windows Performance Monitor counters
- UNIX/Linuz: top, free -m, vmstat, iostat, sar, etc...

Adjust the Java Heap size accordingly to gain additional resources for the JVM. Tune your application server, BIRT iServer or other environment accordingly to gain the best performance.

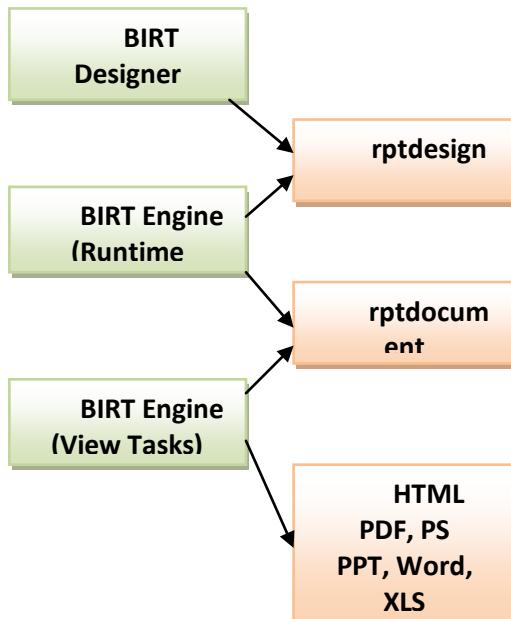
Chapter 19

Preparing for different Report Output Types

In this chapter:

- PDF
 - CSV
 - Excel
 - XML
 - Other Formats
-

An important feature of BIRT is to be able to view the report output in many different formats. The BIRT Engine will read the “rptdesign” file, execute the report, and create a “rptdocument”. The “rptdocument” can then be “rendered” into several different formats.

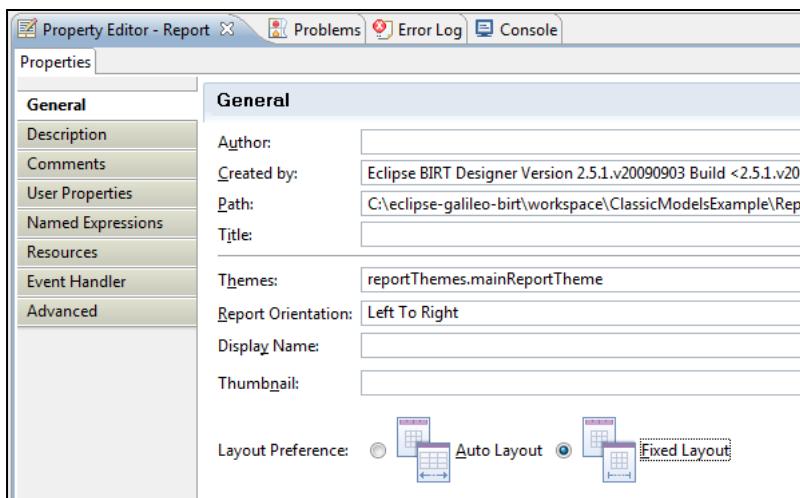


Getting the Report output to look consistent across all formats can be a little tricky. What looks good in HTML may not always fit properly in PDF or Word. There are a couple techniques that can be used to help this process.

19.1 PDF

Earlier versions of BIRT were notorious for not outputting correct PDF. However, BIRT has steadily improved and is much better at producing correct PDF output. The BIRT team is dedicated to improving PDF output in future releases. Here are a couple tips that will improve your PDF output.

- 1) The “Fixed Layout” option in the Report is best used for PDF output. Since PDF sets exact limits on the page size, using a Fixed Layout works best. This setting can be found in the Report properties, General tab.



- 2) For pixel-perfect type reports, use exact width measurements on the table and column widths. Don't use percentages. Use a ruler to measure the report on paper, and set the table and column widths to be exactly how you want them on paper. Set the table width, and then make sure all the column widths are set individually, but still add up to the total. If necessary, set the table height as well.

The screenshot shows the SAP BusinessObjects Designer interface with two open Property Editors.

Property Editor - Table:

- General Tab:** Shows settings like Name (Manager), Width (8 in), Height (8 in), Vertical alignment (Auto), Style (None), Font (Serif), Size (Medium), and Color (Black).
- Layout Tab:** Displays a hierarchical tree of table rows:
 - Manager (Detail Row): Contains fields [fullName_rt] and [jobTitle_16].
 - Detail Row: Contains field [fullName].
 - Group Footer Row (lastName_16): Contains field [extensi...].
 - Footer Row: Contains field [Table].
- Bottom Tabs:** Layout, Master Page, Script, XML Source, Preview.

Property Editor - Column:

- General Tab:** Shows settings like Width (1.5 in) and Vertical alignment (Auto).

- 3) Page breaks can be a little tricky when converting to PDF. Use the “Page Break Interval” setting to specify how many details rows should be printed per page. This setting is found in the Page Break tab of a table. This setting can be very helpful if you have just a normal table listing of data. However, if your table has many levels of grouping, images, charts, long descriptive fields or dynamic items that take up unexpected space, this setting will be of minimal help.

The screenshot shows the SAP BusinessObjects Designer interface with the Property Editor for a Table, specifically focusing on the Page Break tab.

Page Break Tab:

- Before: Auto
- Inside: Auto
- After: Auto
- Page Break Interval: 40
- Master Page: None
- Repeat Header

- 4) For very tricky page breaks and groupings, code may have to be written to force page breaks. Playing around with the page break settings of the table might help. Sometimes, it is a matter of trial and error to get everything lined up perfectly.

19.2 CSV

CSV is popular output that many projects require. There is no direct native way to “render” a report to CSV using an out-of-the-box emitter. Here are five different ways to generate CSV output from your Report Design.

- 1) Use the BIRT Engine’s DataExtractionTask
- 2) Use the experimental CSV Emitter
- 3) Use the BIRT Viewer,
- 4) Use the BIRT Server’s IDAPI
- 5) Use custom code

19.2.1 BIRT Engine

The BIRT Engine can be used to extract the data from a rptdocument file. Using the DataExtractionTask of the BIRT Engine, the developer can get the contents of the raw data, and format the CSV as they wish. This Data Extraction Task will access the data directly from the Data Set in the report. It will bypass any formatting in the report, and extract the raw data. Even if the data is not being used in the report, if it is coming back in the Data Set, it can still be extracted.

```
//Open previously created report document
IReportDocument iReportDocument = engine.openReportDocument("c:/CSVTest.rptdocument");

//Create Data Extraction Task
IDataExtractionTask iDataExtract = engine.createDataExtractionTask(iReportDocument);

//Get list of result sets
ArrayList resultSetList = (ArrayList)iDataExtract.getResultSetList();

//Choose first result set
IResultsetItem resultItem = (IResultsetItem)resultSetList.get( 0 );
String dispName = resultItem.getResultSetName();
iDataExtract.selectResultSet( dispName );

IExtractionResults iExtractResults = iDataExtract.extract();
```

More information on this DataExtractionTask can be found at:

<http://www.eclipse.org/birt/phoenix/deploy/reportEngineAPI.php#idataextractiontask>

19.2.2 CSV Emitter

There is an experimental CSV emitter that is available for use. This acts more like a traditional emitter. The emitter will take a Table Item, and extract the data directly from the table, in CSV format. It will keep the order of the data in the Table, but does not keep the formatting. The emitter will only take what data is displayed in the Tables.

More information about this Emitter can be found at:

<http://www.actuate.com/products/resources/?articleid=11719>

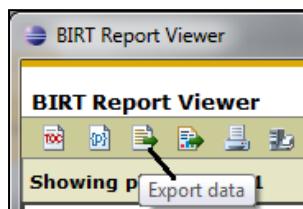
19.2.3 BIRT Viewer

A user viewing a report through the BIRT Viewer can download a CSV version of that report.

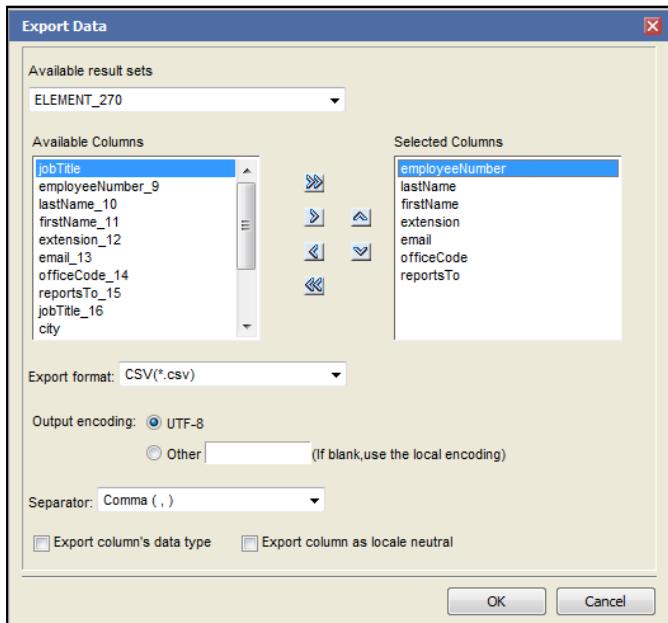
The screenshot shows the BIRT Report Viewer interface. At the top, there's a toolbar with icons for file operations and navigation. Below it, a header bar displays "Showing page 1 of 1". The main content area features a title "Classic Model Cars Inc" with a subtitle "Detroit, MI" and two decorative images of vintage cars. Below the title is a section titled "Managers and Direct Reports". A table lists employees under their respective managers. The table has columns for Manager, Full Name, Phone Ext, Email, and Title. The data is organized into sections for Bondur, Gerard (EMEA) and Bow, Anthony (NA).

Manager	Full Name	Phone Ext	Email	Title
Bondur, Gerard Sale Manager (EMEA)	Hernandez, Gerard	x2028	ghernande@classicmodelcars.com	Sales Rep
	Jones, Barry	x102	bjones@classicmodelcars.com	Sales Rep
	Gerard, Martin	x2312	mgerard@classicmodelcars.com	Sales Rep
	Bondur, Loui	x6493	lbondur@classicmodelcars.com	Sales Rep
	Bott, Larry	x2311	lbott@classicmodelcars.com	Sales Rep
	Castillo, Pamela	x2759	pcastillo@classicmodelcars.com	Sales Rep
Bow, Anthony Sales Manager (NA)	Thompson, Leslie	x4065	lthompson@classicmodelcars.com	Sales Rep
	Tseng, Foon Yue	x2248	ftseng@classicmodelcars.com	Sales Rep
	Jennings, Leslie	x3291	ljennings@classicmodelcars.com	Sales Rep
	Patterson, Steve	x4334	spatterson@classicmodelcars.com	Sales Rep
	Firelli, Julie	x2173	jfirelli@classicmodelcars.com	Sales Rep
	Vanauf, George	x4102	gvanauf@classicmodelcars.com	Sales Rep

Click on the “Export Data” icon.



Let the user choose which fields they want and which format.



	A	B	C	D	E	F	G	H
1	employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	
2	1370 Hernandez	Gerard	x2028		ghernande@classicmodelcars.com	4	1102	
3	1504 Jones	Barry	x102		bjones@classicmodelcars.com	7	1102	
4	1702 Gerard	Martin	x2312		mgerard@classicmodelcars.com	4	1102	
5	1337 Bondur	Loui	x6493		lbondur@classicmodelcars.com	4	1102	
6	1501 Bott	Larry	x2311		lbott@classicmodelcars.com	7	1102	
7	1401 Castillo	Pamela	x2759		pcastillo@classicmodelcars.com	4	1102	
8	1166 Thompson	Leslie	x4065		lthompson@classicmodelcars.com	1	1143	
9	1286 Tseng	Foon Yue	x2248		ftseng@classicmodelcars.com	3	1143	

Letting the user extract the CSV themselves is a great way to save some extra coding and allow the user more control on how they download the data.

If you do decide to let the user extract the CSV themselves, be sure to create user-friendly names for your Data Set column names.

19.2.4 BIRT iServer IDAPI

If you are using Actuate's BIRT iServer, there are a few IDAPI calls that can be used to extract the raw data from the rptdocument, similar to how the Data Extraction Task works. Use the "GetMetaData" and "DataExtraction" web service calls to achieve this. Refer to the iServer and IDAPI documentation for more information.

19.2.5 Custom Code

The above techniques work great if you have an existing Report that you want to extract CSV from. But who says that you *need* to create a BIRT report *just* to extract CSV? If your project requires CSV output as the only output, there is no need to create a BIRT report just to do that. Or, if the CSV output needs to be in a very special format that doesn't exactly fit the way the original BIRT report was designed, then use custom code to generate the CSV the way you want it to be. You can use your existing web framework to extract the data from the data source, format it, and return it to the user.

19.3 Excel Output

Excel output is another format that is often required. The default Excel output of BIRT is an XML document that is compatible with Excel 2003 and above.

For a more robust Excel output that uses Excel 2007 binary format, use the Tribix Excel Emitter.
<http://sourceforge.net/projects/tribix/>

Excel output limitations:

- Charts are converted into Excel as images, *not* live charts linked to data
- Formulas are not converted or kept intact in the spreadsheet
- Unable to create different “sheets” in Excel
- Unable to take advantage of any Excel specific features

For a more complete Excel output, look at Actuate's eSpreadsheet tool. This tool allows a developer to create data linked spreadsheets with pretty much all of Excel's features. Look for Actuate's commercial version of BIRT to offer enhanced Excel output in future versions.

<http://www.actuate.com/products/spreadsheets/>

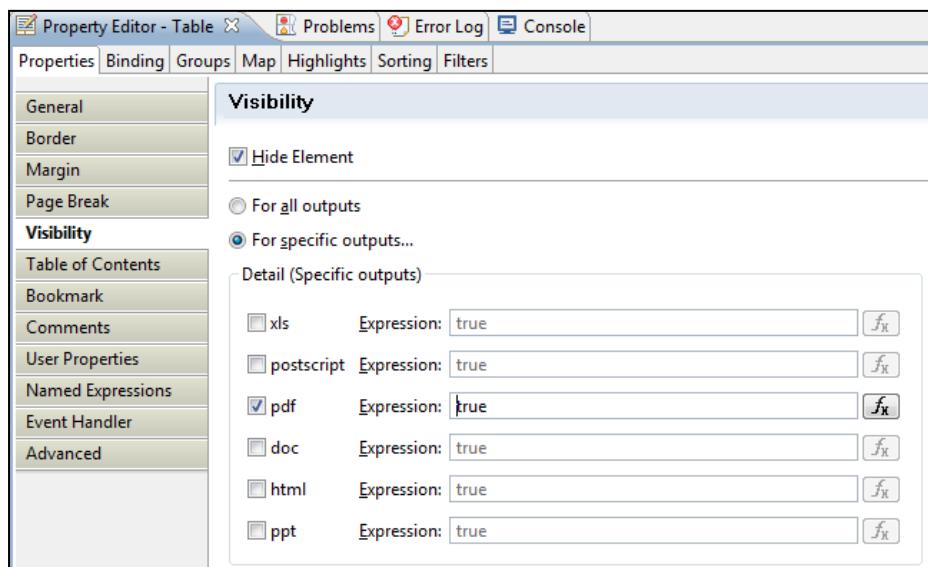
19.4 XML Output

XML output is not supported by BIRT “out-of-the-box”. However, there is an experimental XML emitter available for use.

<http://www.actuate.com/products/resources/?articleid=11719>

19.5 Other Formats

Depending on the complexity of your report, it can be tricky to make sure all formats look good. There might be parts of the report that only get shown in HTML and not PDF. Sometimes, different report item properties should be set depending on what format the report will be viewed in. Use the “Visibility” property and appropriate expression to control which items are visible in which formats.



*Tip

Sometimes, it might make sense to create different versions of the report design, each aimed at a particular format. If your design is very complex, or there are special items that may not convert to the appropriate format correctly, then this may be your best option.

Part III

Following Conventional Software Engineering Methods for your Reporting Project

In this section:

- *Chapter 20 – Using Source Control*
 - *Chapter 21 – Using BIRT in a Team Environment*
 - *Chapter 22 – Comments*
 - *Chapter 23 – Documentation*
 - *Chapter 24 – Solid Testing*
 - *Chapter 25 – Bug tracking*
-

Just because you are using a Reporting Tool doesn't mean you should forget about conventional Software Engineering methods. There are several ways in BIRT and the Eclipse environment that accommodate these methods.

Chapter 20

Using Source Control

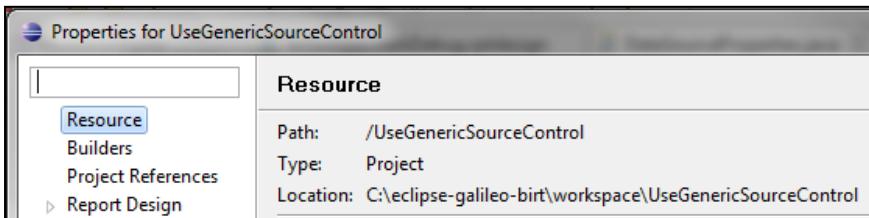
In this chapter:

- Using a Source Control Tool
 - Using CVS
 - Using Subversion
-

Report Designs often get overlooked when it comes to Source Control. However, Le BIRT Expert feels that it is very important to keep Report Designs and any other related resource files in Source Control. Source Control allows a team of developers to keep different versions of their design files, maintain backups and provide a secure location for their files. This chapter lists some of the options available to BIRT Developers.

20.1 Using a Source Control Tool

Because of Eclipse's open file and directory structure, virtually any Source Control product is compatible with Eclipse. To find the folder location of your project files, right-click on the Project Name and choose properties. The location of the folder will be displayed.



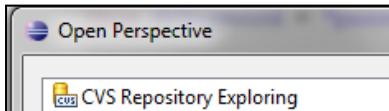
Files can be brought in and out of source control into this directory. By refreshing the Project in Eclipse, all files will be updated. To refresh, right-click on the Project Name and choose "refresh".



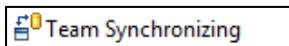
20.2 Using CVS

Eclipse comes with built in support for the source control tool CVS. In fact, Eclipse and BIRT source code are hosted on a publically accessible version of CVS. Two Eclipse Perspectives are available to interact with a CVS server.

The CVS Repository Explorer allows a developer to connect to and explore a CVS repository.



The Team Synchronization Perspective allows the developer to synchronize their project with CVS and to check-in and check-out files.



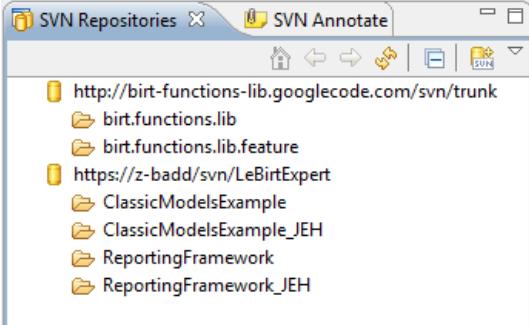
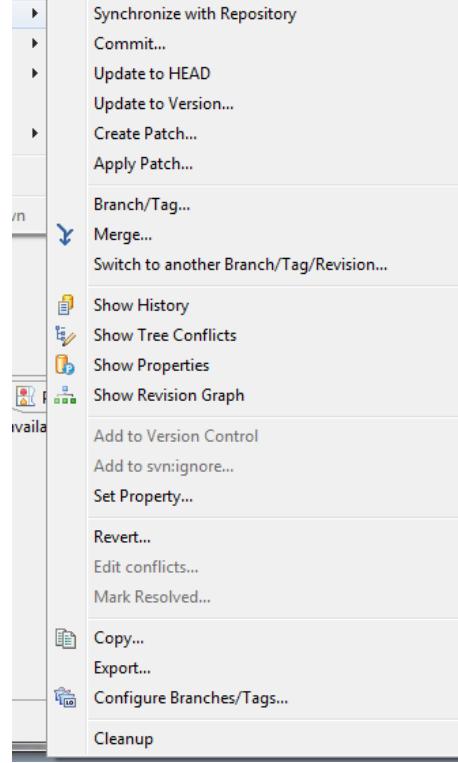
For more information on CVS, visit:

http://en.wikipedia.org/wiki/Concurrent_Versions_System

<http://www.nongnu.org/cvs/>

20.3 Using Subversion

Another popular open source tool that performs source control is “Subversion”. There is an Eclipse plug-in called “Subclipse” that allows Eclipse to connect to Subversion. Subversion can be setup as a server in your local environment. *The setup process is very easy and worth the time to ensure the code is in a central and safe location.* Popular code repository sites, such as Google Code, support subversion.

Switch to the SVN perspective to see details about repository connections.	Right click on the Project, Folder or file to manage with Source Control. Choose the Team item.
	

More information can be found at:

<http://subversion.tigris.org/>

<http://subclipse.tigris.org/>

***Tip**

It is recommended that developers use a Source Control that can plug into Eclipse directly, such as CVS or Subversion. Other methods of source control will work, but having the ability to check-in and check-out code directly from Eclipse is invaluable, especially in a team environment.

Chapter 21

Using BIRT in a Team Environment

In this chapter:

- Using Source Control to Collaborate
 - Using a Network Drive
 - Using the Resource and Template Folders
 - Configuring Eclipse in a Team Environment
 - Using iServer for Source Control
-

An important capability of Eclipse and BIRT is the ability to use the tool in a Team environment. This often involves several different developers working on Report Designs and other related components at the same time. The team should make it a priority to keep all developers on the same page and always working with the latest version of the code.

21.1 Using Source Control to collaborate

This is Le BIRT Expert's recommended way to work together as a Team. Eclipse allows a team of developers to work together by leveraging a Source Control System. By checking-in and checking-out files, the Team can keep in-sync using the latest libraries, templates and Report Designs. If you have a particular Source Control system already in place, Eclipse can take advantage of it. See the section about Source Control and BIRT for more information.

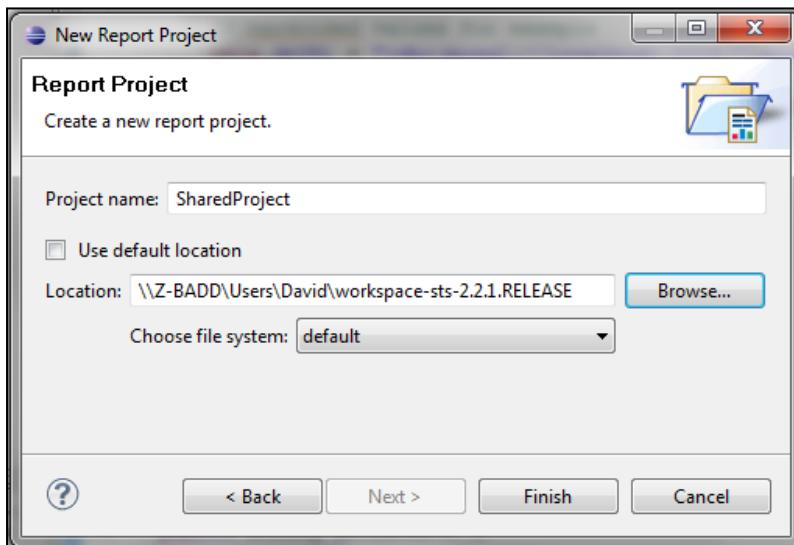
***Tip**

Setting up a CVS or Subversion server and instructing your team on how to connect using Eclipse will only take an hour or two and it will pay huge dividends going forward in your project. Spend the extra few hours to set the environment up in the beginning, it's well worth it!

21.2 Using a Network Drive

If a Source Control Tool is not available, an alternative is to use a Network or Shared Drive. Create a common folder on a networked drive that everyone in the team can access. If a networked drive is not available, use a shared folder on a team member's computer.

When creating a New Project, choose the Network Folder or Shared Folder location for the files. This way, all the team members are using the same folder for their files.



Any shared files, such as libraries, templates and other common files should be marked as "Read-Only" when they are not being used. In Eclipse, right-click on the file and select Properties. Check or Uncheck the "Read Only" box to mark or unmark the file as read-only. Having the file as "Read-Only" prevents accidental changes.

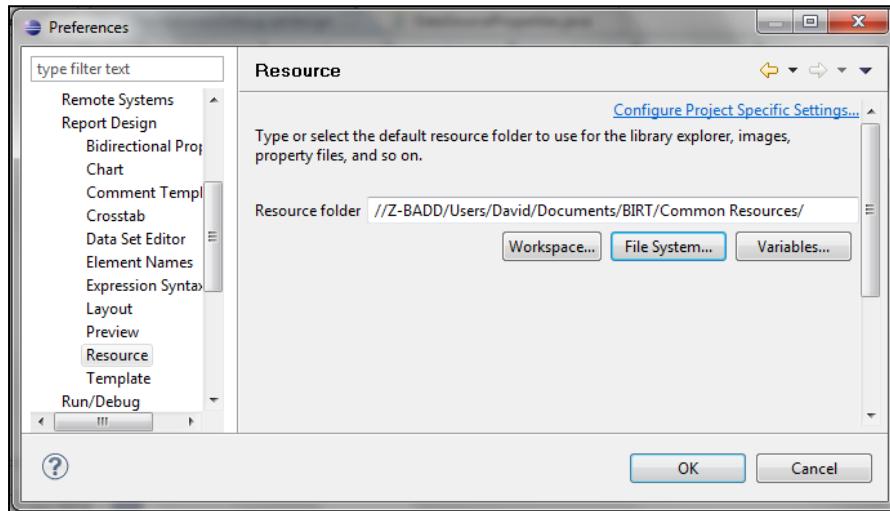
***Tip**

This technique must be used with care, as it is easy for other Team members to overwrite each other. Be sure to keep regular backups and versions. This will have to be tracked manually.

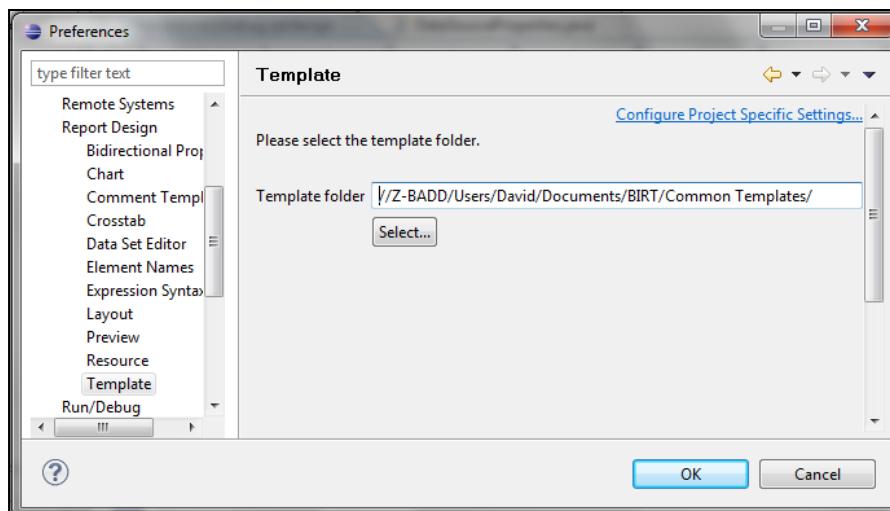
21.3 Using the Resource and Template Folders

It is possible to keep the developer files local to their computer, while sharing the common libraries, templates and other resources among the team. In Eclipse, there is a setting to specify the directory that contains the common resource folders and templates.

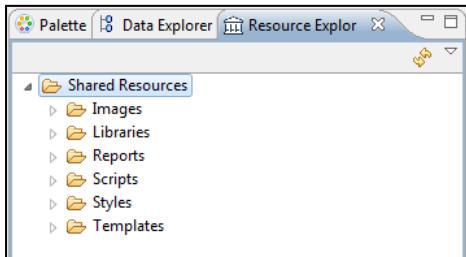
In the Designer, click the Window menu option. Go to Window → Preferences → Report Design → Resource. Check the folder to point to a common Network or Shared Folder.



Do the same for the Template Folder

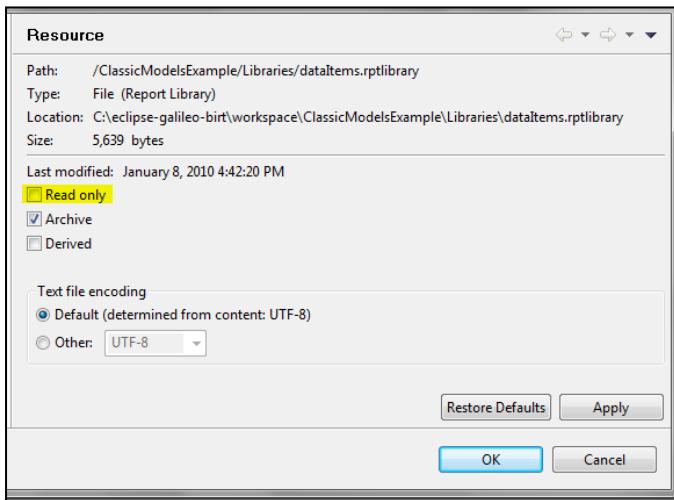


In the Shared Resources tab, the libraries available should appear.



In the New Report Wizard, the Report Templates should be available

Any shared files, such as libraries, templates and other common files should be marked as “Read-Only” when they are not being used. In Eclipse, right-click on the file and select Properties. Check or Uncheck the “Read Only” box to mark or unmark the file as read-only. Having the file as “Read-Only” prevents accidental changes.



*Tip

This technique must be used with care, as it is easy for other Team members to overwrite each other. Since the actual reports are in the developer environment, they are at less risk. However, be sure to keep regular backups and versions. This will have to be tracked manually.

21.4 Configuring Eclipse in a Team Environment

Even if you are sharing project files in a Team Environment, each developer's environment must be set up appropriately. Confirm that all JDBC drivers are in place, external JAR files and other Java Classes are in the class path and confirm all Report Templates are registered in Eclipse. If you attempt to create a Report based on a Template, and BIRT does not list the Template in the New Report Wizard, be sure to register the Template with the BIRT Designer.

21.5 Using iServer for Source Control

Those teams that are using Actuate BIRT Designer and have the BIRT iServer installed, can use the iServer as a make-shift source control and versioning system. The Actuate Professional version of the BIRT Designer allows the developer to connect to the iServer and view the folder structure and files on the iServer. The developer can drag and drop report designs, libraries, all other resource files between the iServer window and the Eclipse Navigator Window. By using a shared Resource and Template Folder and exchanging files through the iServer, a team could share files and keep in sync.

***Tip**

This technique is a little tricky and requires all team members to be familiar on how it should work. It's not a recommended approach but it is an option if you are using Actuate BIRT and the BIRT iServer.

Chapter 22

Comments

In this chapter:

- Comments in JavaScript and Java
 - Comments in the Report Design
 - Comments Best Practices
-

Leaving comments in the code are important for future maintenance of the report designs and components. Sometimes, after a long period of time, a developer can forget how their own code even works. Leaving comments that explain how the code works and what the reasoning behind it was can go a long way towards helping maintain the code in the future.

22.1 Comments in JavaScript and Java

To leave comments in JavaScript and Java, use the syntax “`/* ... */`” and “`//`”. The text will turn green if it is commented out. It is recommended to put at least one line of comment per line of code. Very obvious lines of code can be skipped. All Classes, Methods, JS Functions and variables should be commented so a future developer can understand what is the code is doing.

An example of a JavaScript function commented is:

```
*****
 * Name: isEmptyString
 * Description: Checks to see if the string passed in is either
 *               null or empty
 * Parameters: stringValue - String - original value of String to be evaluated
 * Return:      Boolean - true/false
*****
function isEmptyString(strValue) {
    // checks all possibilities of a variable being empty
    if(strValue == null || strValue == "" || strValue == "null" ||
       strValue.length == 0)
    {
        // returns true
        return true;
    }
}
```

```

    } else {
        // returns false
        return false;
    }
}

```

An example of a Java Class commented is:

```

/*
 * ReadCSVForBIRTDataSet.java
 */
package com.leBirtExpert.birtBestPractices.scriptedDS;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

/*************************************
 * This class is used to read and parse a CSV
 * file into an ArrayList of CustomerBIRTDataSet
 * @author Le BIRT Expert
 ************************************/

public class ReadCSVForBIRTDataSet {

    // Java File Reader used to open CSV File
    private FileReader fileReader;
    // Used to read input of CSV File
    private BufferedReader bufferedReader;

    /**
     * Opens connection to CSV File
     * @param dbFile full path to location of file
     * @return result message returned to BORT
     */
    public String openFileConnection(String dbFile) {

        // Result Message that is returned
        String resultMsg = "";

        try {
            // file reader handle
            fileReader = new FileReader(dbFile);

            // buffered reader
            bufferedReader = new BufferedReader(fileReader);

            resultMsg = "SUCCESS";
        }
    }
}

```

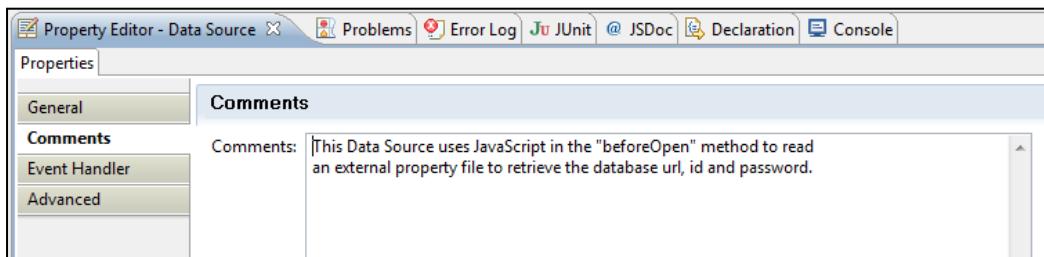
```
    } catch (FileNotFoundException fnfe) {
        // log this exception
        resultMsg = "ERROR: " + fnfe.toString();
    }
    return resultMsg;
}
// Continued
}
```

***Tip**

Create a set of coding standards to be used by your Team that enforce good commenting.

22.2 Comments in the Report Design

In the BIRT Designer, there is a way to leave comments for any kind of Report Item. Open a Report Design, Report Template or Report Library. In the Property Editor, you will see a tab for "Comments". This feature is available on all Report Items in BIRT.



22.3 Comments Best Practices

- Any JavaScript and Java code should have comments explaining what the code does.
- Any out-of-the-ordinary logic or design used to solve a problem should be commented.
- If a Report contains usual coding or special techniques to solve a problem, indicate that in the Comments section of the Report Design, Report Template or Report Library.
- Every Report Item does not need a comment. Only if the Item does something custom should it need comments to explain what it is doing.
- Each team will need to decide the best Comment syntax that works for them. The important thing is that the Comments *are* left behind for future developers who will maintain the system.

Chapter 23

Documentation

In this chapter:

- Report Documentation Contents
 - Using the Description field in the BIRT Designer
-

Documentation is important to help other developers maintain the Report Designs and components. 80% of the average software application lifetime goes to maintenance. Hardly any software is maintained for its whole life by the original author.

23.1 Report Documentation Contents

It is considered Best Practice to leave behind a document explaining how your software project works, and report designs are no exception. Surprisingly, many teams do not document their report designs, increasing the time it takes for a new developer to get up to speed on the project.

The documentation left behind should contain at least the following information about the report designs:

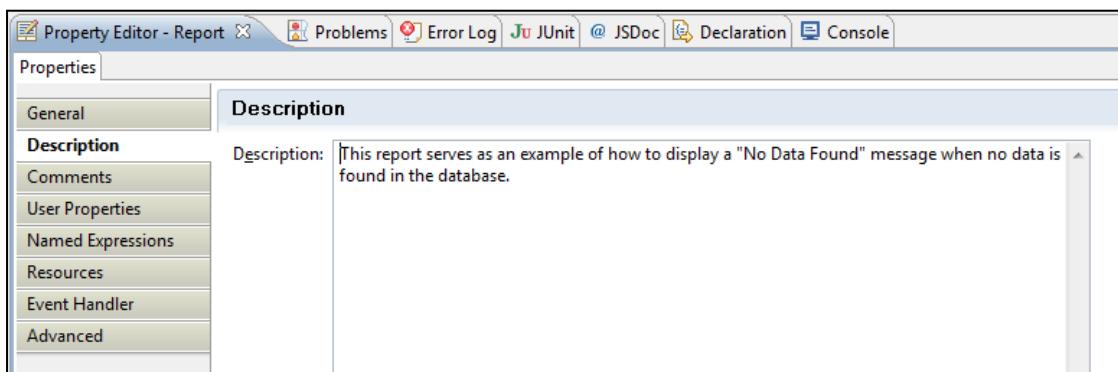
- Explanation of the reporting framework
 - Folder structure
 - Themes, Styles and Libraries
 - JavaScript and Java libraries
 - Templates
 - Any other items used in the framework
- Method of data access
- Any crucial design decisions made and why
- Business Purpose of the reports

- How the report designs work
 - Any special function, calculation or routine?
 - Event handlers
- How external components work
- How BIRT interacts with external components
- Remaining issues
- Any other useful information

For an example of how to identify Report Requirements and document them, refer to the soon-to-be-released book *Le BIRT Expert's Guide to Writing Bullet Proof Report Requirements*. Check the website for more information.

23.2 Using the Description Field in BIRT Designer

In the BIRT Designer, there is room for a developer to leave a Description of the Report Items. This is available on Report Designs, Report Templates and Report Libraries. It is recommended that the developer leaves a short description on what the business function the Report performs and a little about how the Report is used. Technical details can be put in the "Comments" section.



Chapter 24

Solid Testing

In this chapter:

- Developing a Test Plan
 - Testing the Data
 - Testing the Report Parameter Screens
 - Testing the Report Output
 - Testing Report Security
 - Testing Performance
 - User Testing
 - Improving Performance
 - Automated Testing
 - Testing Worksheet
-

Testing is an essential piece of software development and something that should not be overlooked when working with Reports. There are many aspects of a Report that need to be tested. Developing a solid Testing Plan is critical to ensuring your reports are ready for the real world.

24.1 Developing a Test Plan

No software application is complete without proper testing. Developing a solid Test Plan will allow your team to test all areas of the Report before deploying them to production.

The following Test Plan is suggested:

- Testing the Data
- Testing the Report Parameter Screens
- Testing the Report Output
- Testing Report Security
- Testing Performance
- User Testing

24.2 Testing the Data

The most important parts of the Report is the actual data that is retrieved and displayed. Some tips for testing the Data:

- Confirm that the query you are working with is retrieving the correct data. It may be necessary to run the query outside of BIRT to confirm this.
- During development, be sure to unit test the query.
- Ensure that you test the query with different Report Parameters.
- This is where stored procedures come in very handy. If the developer can create stored procedures and unit test them, this isolates the query from the Report Design. Any change to the query can be done in the stored procedure without change to the Report Design.
- Work with the DBA and Business Analyst to confirm that the correct data is being retrieved and displayed.
- Confirm that all calculations, totals and other expressions are being performed accurately. Manually calculate values to double check results.
- Confirm all Charts and Graphs accurately depict what the data shows.
- Check with your DBA to get a good list of Test Data
 - Having a correct set of Parameter values and Test Data is critical to accurately testing your reports.

24.3 Testing the Report Parameter Screens

Having correct Parameter Screens is important to how the end user will interact with the Reports. The Parameter Screens should be friendly, accurate, and allow the user to quickly enter the values to view the Report. Some suggestions for testing the Report Parameter Screens:

- Confirm that each Report displays the proper Parameters. Confirm “hidden” Parameters are indeed hidden.
- Confirm that all drop-down lists display accurate values.
- The Parameter Screen should perform some kind of validation on the values entered.
 - Confirm that the value entered is a valid:
 - Date
 - Number

- Phone Number
 - Account Number
 - Etc
- As mentioned in Chapter 7 (Report Parameters), BIRT has limited Parameter validation built in.
 - Don't rely completely on client side validation. Server side or in-report validation is very important as well.
 - If the parameters are invalid, consider how the application should handle the different situations. Should the report fail automatically? Run but show no data? Run but show an error message?

24.4 Testing the Report Output

Next to data accuracy, this is what the end user cares about the most! How the Report looks. Here are some suggestions for testing the Report Output:

- Confirm the Report output meets the user's expectations and matches the original Report requirements.
 - Header and Footers are correct.
 - Font type, color and size are correct
 - All Reports have a common look and feel
 - Colors, shading, images and Charts look correct
 - Labels, dynamic text or other data fields are not cut-off (shows the complete value)
- Don't rely only on the "Preview" feature of the BIRT Designer.
 - View the Report in the "Web Viewer" for a more accurate idea of what the report will look like in the BIRT Viewer.
 - Any exceptions that occur in the report will be shown in the Web Viewer. Not all exceptions are shown in the "Preview" window.
- Confirm the printed report meets the user's expectations and matches the original Report requirements.
 - Print the report through the browser and through your application. Confirm the printed report meets expectations
- Confirm the Report output in Different Formats meet user's expectations and matches the original Report requirements.

- View the Report in PDF and other required formats to confirm report layout accuracy, correct page dimensions, and correct color schemes.

24.5 Testing Report Security

Security should be a very important part of your application, and Reports are no exception. There are several places in a Report where security could be compromised. It is important to thoroughly test your Reports for security weaknesses. Here are some suggestions for testing Report Security.

- Report Security starts with ensuring that a user only has access to the Reports they are authorized to Run/View. Confirm that all users can only see Reports they have access to.
 - Use a role-based security system to limit Report access to a set of Roles.
 - If a user is not allowed to run/view a report, don't even give them the option of doing so!
- Confirm that parameter values displayed to the user are limited to what they are authorized to see.
 - For example, if a user is presented with a list of Account Numbers to choose from, confirm that the user has access to all of those Account Numbers.
 - Don't give the user the option of choosing a parameter value that they don't have access to.
- Confirm that all Report Parameters are valid
 - Confirm any parameter values typed in by the user are accurate
 - Confirm SQL Injection attacks do not work in your Report
 - Try to manipulate the URL of your Report to pass in unauthorized parameter values. Confirm that your Report does not display unauthorized data or results.
- Confirm that the Report displays only authorized data for the user
 - Confirm the user is able to see *only* what they are authorized to see.

24.6 Testing Performance

Performance is also an important factor for end users. Most users want reports quickly and don't like waiting for more than a few seconds. Here are some suggestions for Testing Report Performance.

- Confirm that each Report runs and displays in a time acceptable to the users
- Test each report with a combination of parameter values.
 - Some parameter values may return more results than others.
 - If you are not sure what parameter values to test with, check with your DBA to get a list of good test values.
- Test Reports with a **full database**
 - Some teams only test with a small set of Test Data. It is only after the Reports have been in production for a few weeks or months that they notice performance problems.
 - Testing with data as closest to production quality/quantity as you can get can give you an accurate picture on how the Reports will perform in Production.
- Load Testing
 - It is important to load test your application and Reports to get an idea of how much your system can handle.
 - Confirm that your Application Server, iServer installation or custom application can handle the anticipated request load.
 - Use a professional or open source load testing tool to do this.
 - Load the database with LOTS of data. Confirm that the queries and Reports perform well using a large data set.

24.7 User Testing

It's important to let the users take a look at the reports as soon as they are somewhat presentable. Many times, the users change their mind about what they want as soon as they see it on the screen. They also think of things that they might not have thought of before, or notice small details you might have missed, and can suggest likely usage scenarios for future testing. It's best not to wait to the end of your development process to show the user. *Involve them in the testing process early so there won't be any surprises later for either side.*

24.8 Improving Performance

If your Reports are experiencing Performance problems, there are a few areas to look at to improve performance. The key is to eliminate different components of the report until you find the bottleneck. Here are some suggestions to improve performance of your reports

- Data Source
 - Most performance problems occur in the Data Source.

- Execute the query or call the Web Service (or whatever your Data Source is) outside of BIRT.
 - The key is to isolate BIRT and the Data Source.
 - If you are experiencing similar performance problems in the Data Source, then tune the Data Source or Query to respond quicker.
 - If the database is taking 2 minutes to return the results of a query, BIRT won't be able to perform any faster!
 - Use professional or open source Database Query analyzing tools to find query bottle necks.
 - If the Data Source is responding quickly, but the Report is still taking a long time, then something is happening in the application or BIRT.
- Application Server or iServer
 - If all Reports or the application in general is running slow, your Application Server or iServer might be the culprit. Confirm the optimal memory, JVM and other settings are in place. Additional hardware or servers may be needed.
 - If you are using your own custom code to integrate BIRT into your application, confirm you are calling the BIRT Engine correctly.
 - If the report is running quickly in the Designer (using the same Data Source), but slowly on the Server, then the Server could be causing the problems.
- Network problems
 - Sometimes, the configuration of your Network may be the cause of Performance problems. Perform network tests from the actual Application Server or iServer to confirm there are no problems.
 - If you suspect network problems, try running the reports locally on that server. If they perform very quickly, but slowly on other computers in the network, then there may be some issues with your network.
- BIRT Design
 - For the most part, a typical BIRT Report design should not cause performance problems. Typically, a BIRT Report should only take *slightly* longer to run than the Data Source does to retrieve the data.
 - Some exceptions to this are:
 - If the BIRT Design is filtering or sorting a large Data Set in the Report Design, this could cause performance problems. As mentioned in Chapter 12, avoid filtering or sorting in the Report Design itself.
 - If the BIRT Design is making calls to external Java classes or other services, check these external components for delays.
 - If the BIRT Design uses a large number of Charts (a chart per data row) or large images, this may cause performance problems.

24.9 Unit Testing

Unit testing is incredibly important for each developer to do. The developer must test all different types of scenarios and parameters to ensure that the report is working the way the requirements define. Because of the visual nature of reporting and the use of the BIRT Designer, there is a certain element of manual work that is involved. The developer must run each report and visually inspect the report output to confirm it is correct.

With BIRT (and reporting in general), there is a limited way to create “Test Cases” to program against. All external Java classes can and should be unit tested using JUnit or other appropriate Java testing tools and frameworks. Stored procedures can and should be unit tested by the developer or DBA.

That is why this book has stressed the importance of isolating certain components outside of the report design (stored procedures, Java objects, business logic, etc). The core logic of the report should be in isolated components that can be unit tested, where the display logic is part of the report design that will be visually tested.

24.10 Automated Testing

Having automated tests is a popular way to do testing in software applications. Automated tests can be written to launch all reports, feed in parameter values and fetch results. These automated tests can be used to test performance, test security (to a certain point) and test data (to a certain point).

Having automated tests is a great initial start to Report Testing. However, due to the nature of Reports, using testing personnel for manual inspection is unavoidable to complete thorough testing of the reports.

Chapter 25

Bug Tracking

In this chapter:

- Bug Tracking
 - Tools to Track Defects
-

If you follow a solid Testing Plan to test your reports, it is more than likely that at least *some* defects will be found. It is important to track these defects so they can be fixed.

25.1 Bug Tacking

Each defect should have, at the very least, the following information:

- Good description of the defect.
 - Exact steps taken to duplicate the issue.
 - Screenshot of the error
- User who discovered the error and who can verify it has been fixed.
- Priority of defect
- Developer responsible for fixing it
- Status of Defect (assigned, needs testing, fixed, etc)

25.2 Tools to Track Defects

There are many tools available to track defects. Often, the same bug tracking tool used for your application can be used for your reports as well.

- MS Excel
 - Believe it or not, many software projects still use MS Excel to track defects.
 - Excel is very easy to use

- Its drawbacks are that it requires a lot of effort by the project manager to constantly email the file around and manually update it. Team members cannot search the file easily and there is not a good way to share the file among the team.
- Bugzilla
 - A very popular open source bug tracking tool
 - This Tool is used by the Eclipse and BIRT teams to track bugs
 - More information available at <http://www.bugzilla.org>
- Other Tools available:
 - http://en.wikipedia.org/wiki/Comparison_of_issue_tracking_systems

***Tip**

There are many tools available to track defects in your reports. Invest some time to set up the appropriate tool to make tracking bugs easier for your team.

Appendix I

Check List

Are you ready to start your BIRT project?! Before you write that first report, use this checklist to confirm that you have everything for your project in place.

<input checked="" type="checkbox"/>	Report Requirements established and signed off by Client/Users/Business
	Common naming conventions established
	Setup environment for Team Development
	Source Control setup and integrated with Eclipse
	Create folder structure
	Create Common Report Theme and Styles
	Common libraries (Theme, Data Items, Report Items) created
	Create a set of Master Pages
	A secure and consistent way to access data established
	Common parameters and parameter screen approach identified
	Common scripting “js” files created and added to Library
	Base Java Classes for Java Event Handlers
	Logging strategy in place
	Report Templates created for developers to use as a starting point
	Confirm an approach for i18n and l10n or external messages (if needed)
	Decide on an approach to separate business logic from display logic
	Confirm an approach for special exception and error handling
	Prepare Documentation Templates
	Do a proof-of-concept to confirm unique uses of BIRT will work as expected
	Keep a Testing Plan in Mind
	Bug / Defect Tracking Mechanism

Appendix II

Deployment

Are you ready to deploy and test your reports outside of the BIRT Designer environment?! This section was meant to help you do a quick deployment to the BIRT Runtime Engine and the BIRT Web Viewer for testing. Providing detailed deployment instruction is outside the scope of this book, but this section should help you get started in the right direction.

This section will refer to the “ClassicModelsExample” project that is available as part of the code download for this book. Visit the following pages for the code download:

<http://www.lebirtexpert.com/code/bestPractices/>

<http://code.google.com/p/lebirtexpert/>

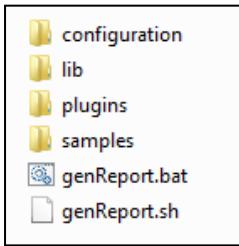
Deploying to the BIRT Report Runtime Engine

The BIRT Runtime Engine will allow you to run the report designs outside of the BIRT Designer environment. The BIRT Runtime Engine is available as a separate download through the Eclipse BIRT web site. Visit

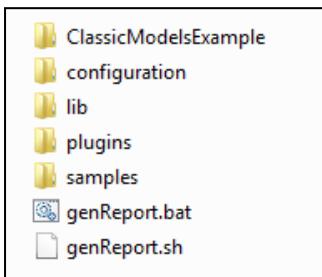
http://www.eclipse.org/downloads/download.php?file=/birt/downloads/drops/R-R1-2_5_2-201002221500/birt-runtime-2_5_2.zip to download the BIRT 2.5.2 Runtime environment. Once downloaded, unzip to a directory of your choice. Follow these instructions to deploy the reports to the Runtime Engine.

Set the “BIRT_HOME” system variable in your environment to point to the directory that the file was unzipped to.

Open the “Runtime” folder. You will see the following folder structure:



Copy and paste the “ClassicModelsExample” folder structure to the “Runtime” folder.



Copy and paste the MySQL driver named “mysql-connector-java-5.1.10-bin.jar” to the following directory in the Runtime Engine.

\birt-runtime-
2_5_1\ReportEngine\plugins\org.eclipse.birt.report.data.oda.jdbc_2.5.1.v20090821\drivers

In Eclipse, build the JAR file from the “ClassicModelsExample_Java” project. Copy the JAR file into the “lib” folder. (For your convenience, the compiled “ClassicModelsExample.jar” JAR file is included in the “resources” folder of the “ClassicModelsExample” project.)

Modify the “genReport.bat” file to include the path to the “lib/ClassicModelsExample.jar” file.

%BIRT_HOME%\ReportEngine\lib\ClassicModelsExample.jar;

Open the command window and run the following command to test the Report Runtime Engine and connection to the database. All output from the command will be captured in the file “genReportOutput.txt”. A file named “testConnection.html” will be generated in the same directory as the report design file.

```
genReport.bat -runrender ./ClassicModelsExample/Reports/testConnection.rptdesign >
genReportOutput.txt
```

To run the other test reports, execute the file “runClassicModelReports.bat”. This batch file will run all the reports in the project.

Using the BIRT Engine to Execute Reports

The BIRT Engine can be embedded into your application to execute and render the report designs directly from your custom code. This feature of BIRT is not covered in this book, however, there are other good resources on the web that cover this in detail.

<http://www.eclipse.org/birt/phoenix/deploy/reportEngineAPI.php>

<http://www.birt-exchange.org/org/devshare/deploying-birt-reports/1061-birt-api-examples/>

<http://www.birt-exchange.org/org/devshare/deploying-birt-reports/1059-integrating-birt-ppt/>

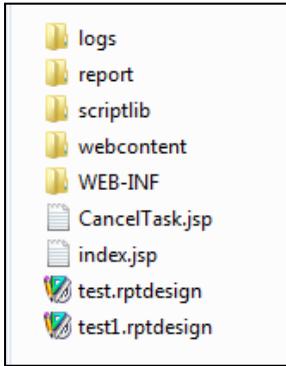
Deploying to the BIRT Viewer

An open source BIRT Viewer is available for download through the Eclipse BIRT web site. This viewer can be deployed to a J2EE application server for the purpose of delivering BIRT reports through the web. Since it is open source, the viewer can also be customized and/or embedded into your web application. Visit

http://www.eclipse.org/downloads/download.php?file=/birt/downloads/drops/R-R1-2_5_2-201002221500/birt-runtime-2_5_2.zip to download the BIRT 2.5.2 Runtime environment. The BIRT Viewer is included in the Runtime Engine download. Follow these instructions to deploy the Classic Models Example project to the BIRT Viewer.

Deploy the Viewer into Apache Tomcat or your application server of choice. See <http://www.eclipse.org/birt/phoenix/deploy/viewerSetup.php> for more information on how to do this.

The directory structure of the “WebViewerExample” folder is below:



Copy and paste the “ClassicModelsExample” folder structure to the “WebViewerExample” folder.

Copy and paste the MySQL driver named “mysql-connector-java-5.1.10-bin.jar” to the following directory in the Runtime Engine.

WebViewerExample\WEB-INF\platform\plugins\org.eclipse.birt.report.data.oda.jdbc_2.5.1.v20090821\drivers

In Eclipse, build the JAR file from the “ClassicModelsExample_Java” project. Copy the JAR file into the “lib” folder. (For your convenience, the compiled “ClassicModelsExample.jar” JAR file is included in the “resources” folder of the “ClassicModelsExample” project.)

Copy and paste the “ClassicModelsExample.jar” to the “/WEB-INF/lib” directory.

Start the application server and navigate to the WebViewerExample project. Run the reports to test in the viewer.

http://localhost:8888/WebViewerExample/frameset?_report=ClassicModelsExample/Reports/testConnection.rptdesign

http://localhost:8888/WebViewerExample/frameset?_report=ClassicModelsExample/Reports/TestThemeReports/customerListThemeSwitcher.rptdesign

More information about the BIRT Viewer is available at:

<http://www.eclipse.org/birt/phoenix/deploy/#viewer>

Deploying to the BIRT Server

Deploying to the BIRT Server is not covered in this book. However, it is covered in detail in the manuals that come with Actuate BIRT and the BIRT Server. In the manual *Using Actuate BIRT*, Part 5 is dedicated to this topic.

Notes
