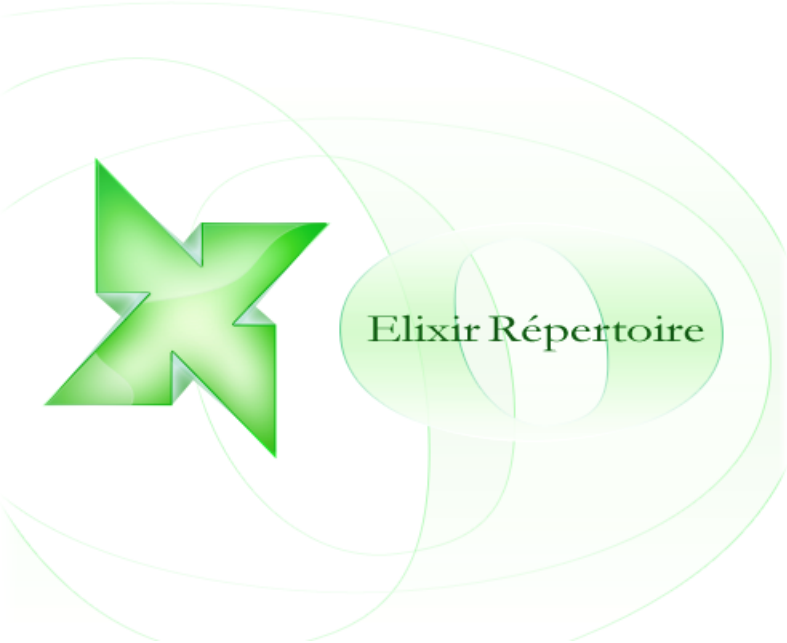
Elixir Repertoire Server Manual

Release 8.8.0



Elixir **Technology Pte Ltd**

Elixir Repertoire Server Manual: Release 8.8.0

Elixir Technology Pte Ltd

Published 2016
Copyright © 2008-2016 Elixir Technology Pte Ltd

All rights reserved.

Solaris, Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. Microsoft and Windows are trademarks of Microsoft Corporation. All other trademarks are registered under their respective owners.

Table of Contents

1. Elixir Repertoire Server 1
    Overview 1
    Features 1
2. Getting Started 2
    Hardware Requirements 2
    Software Requirements 2
    Downloading and installing Elixir Repertoire Server 2
    Starting the Server 3
    Testing the Server 3
    Stopping the Server 3
    Renaming the Server 4
    Running the Server as a System Service 4
3. Overview 6
    Introduction 6
    Main Directories 6
    Core Components 7
4. Configuration 8
    Configuration Overview. 8
    Java Virtual Machine Configuration 8
    Elixir Repertoire Server configuration 9
        Server Configuration 10
        Server Log Control 16
    Administration of Core Services 17
        Configuring the Elixir Repertoire Server 17
        LDAP Secondary Authentication 18
        SMTP Server 20
        Configuring the Logging 20
        Secure Mode 20
    Targets and User Access 21
        Target Parameter Resolution Mechanism 21
        Roles 22
5. Security 23
    Overview 23
    Preparation 23
    Connections and Firewalls 23
    Opening file using URL 23
    Protecting Sensitive Information 24
        Datasource Security 24
        Access Rights 24
    JavaScript Permissions 26
    Case Study 1 - Datasource Security 26
6. Web Interface 29
    Overview 29
    Applet 29
    Repertoire 30
        Search 30
        Manage 31
    User 32
        System 32
        Password 32
    Remote 32
    Administration 33
        Server 33
        Scheduler 34
        Users 34

Groups 35
FileSystems 35
Targets 36
MIME Types 52
Search 53
Data 53
Task 53
Logs 55
Help 55
Logout 55
Case Study 1 - Customized Parameter Entry Page 55
Case Study 2 - Customize Folder Listing 57
7. Elixir Repertoire Server Client 58
Introduction 58
Java Standalone Clients 58
Elixir Repertoire Server Client 58
Using the APIs 58
Code example 59
Java Client usage examples 62
Non-Java Client Connection Library 62
Elixir Repertoire Server Command Client 63
8. Server API 64
Overview 64
REST 64
JDBC 65
Calling HTTP 66
JavaScript Extensions 75
9. Troubleshooting and Common Errors 76
Introduction 76
Server Troubleshooting 76
System Requirements 76
Port Availability 76
Logs 76
Running as a Windows Service 77
Client Troubleshooting 77
JVM Versions 77
Consistent Connection Information 77
Client-Server Troubleshooting 77
Network Access 77
Common Errors 77
Client Errors 77
Report Errors 78
Datasource Errors 78
Printing Errors 78

List of Figures

2.1. Elixir Repertoire Server Logon 3
4.1. Force Password Change 11
4.2. Create a Anonymous User ID 15
4.3. Activity Log 17
5.1. Access Rights Configuration through Web Interface 25
5.2. Access Rights Configuration through Remote Designer 26
5.3. Datasource Security 27
5.4. Row Column Information 27
5.5. Encrypt DataSource 28
6.1. File Target 38
6.2. JDBC Target 39
6.3. JMS Target 41
6.4. Mail Target 42
6.5. sign-position Options 44
6.6. PDF Signer - Sample Signature (self-signed) 44
6.7. PDF Signer Target 45
6.8. Repository Target 47
6.9. Repository User Home Target 48
6.10. SFTP Target 49
6.11. Split Conditions 51
6.12. Split Target 52
6.13. Task Manager 53
6.14. Task Information 54
6.15. Stop Task 54
6.16. Parameters Entry Page Not Customized 56
6.17. Parameters Entry Page Customized 57

List of Tables

4.1. Elixir Repertoire Server Configuration details 18
6.1. Chain Target Configuration 37
6.2. File Target Configurations 37
6.3. JDBC Target Configurations 39
6.4. JMS Target Configurations 40
6.5. JNDI Properties 40
6.6. Mail Target Configurations 41
6.7. PDF Signer Target Configuration 43
6.8. Print Target Configurations 46
6.9. Repository Target Configurations 46
6.10. Repository User Home Target Configurations 47
6.11. SFTP Target Configurations 48
6.12. Socket Target Configurations 49
6.13. Split Target Configurations 50

List of Examples

7.1. Listing the file systems in a server repository. 59
7.2. Listing the reports deployed in a file system. 59
7.3. Generating a report 60
7.4. Request for Data listing. 60
7.5. Trigger for Data Store process. 61
7.6. Using IJobInfo interface to extract job information. 61
7.7. Configuring custom status code and exception message 62
8.1. Retrieve Records From Selected Column 66

Chapter 1
Elixir Repertoire Server

Overview

Elixir Repertoire Server provides a scalable reporting (Elixir Report Designer) and Enterprise Transformation and Loading (Elixir Data Designer) solution that grows with your business needs. It scales from small departmental workgroups to enterprise portal deployments. It may be deployed on small single departmental servers run on lower cost Linux, Windows Servers to large enterprise server such as multi-processors boxes. Its unique architecture allows you to configure and tune the usage of server resources.

It is written in Java language to provide cross platform functionality and will run on a Java 6 (or later) compliant machine. The server provides a web interface, support for the Elixir Repertoire Remote Designer and an HTTP-based API so that you can build applications in the programming language of your choice and call the server for scheduling, reports, data and dashboard functionality.

Elixir Repertoire Server supports repository-based storage and provides secure access to the datasources, reports and dashboards that it manages.

Features

Cross platform: Elixir Repertoire Server run on all hardware platforms that supports Java SE 7 platform and Java SE 8 platform e.g. Windows, Linux, Solaris etc.

Browser interface: For administration, dashboards and report generation and archiving

Multi-threaded: The rendering of the report can be executed concurrently within the same Java Virtual Machine.

Integrated Repository: This allows authenticated access to all resources, datasources, reports and dashboards.

Failover and load balancing capable: Elixir Repertoire Server is based on HTTP, so any HTTP load-balancers, proxies and portal / redirection tools can be used.

Extensible: Given HTTP is language neutral, you are free to implement a custom solution in the language of your choice - Java, C#, Ruby, Python and others.

Scheduling: Elixir Repertoire allows you to schedule your jobs. Elixir Schedule Designer, part of the Elixir Repertoire Remote tool, allows you to remotely connect to the server to schedule your job. Details on scheduling functionality may be found in the Schedule Designer documentation.

Chapter 2
Getting Started

Hardware Requirements

Elixir Repertoire Server will run on any hardware platform that supports Java Standard Edition version 6 or later. As a general rule-of-thumb, any online type of application that needs low turn around time will typically require two or more high-end processors.

The minimum RAM required for installation of Elixir Repertoire Server is 256 MB but the recommended hardware memory is at least 512 MB. The amount depends largely on the data volume, load (i.e. concurrent report generation) and desired turn-around time.

A proper system requirement study is recommended to estimate the proper hardware (CPU), JVM configuration and memory size. You may email the support team at Elixir Technology for information on this.

A total of 300 MB disk space is recommended. This consists of the following:

- 200MB for the Elixir Repertoire Server software, documentations and samples
- Up to 80 MB for the Java runtime (depending on the version)
- 20 MB reserved space for other 3rd party add-on software like database drivers. Intel® Pentium® 4 Processor with 800MHz or more is required

Features such as report, data caching will require additional disk space.

Software Requirements

Elixir Repertoire Server is supported on all Java Standard Edition compliant operating system platforms such as Windows and Unixes such as AIX, Linux and Sun Solaris. The Java Virtual Machine version should be Java SE 6. Please refer to the respective vendors for any specific installation details for the JVM on their platform.

Downloading and installing Elixir Repertoire Server

Elixir Repertoire Server is available for download as .zip file. Once it is downloaded, unpack the archive to a suitable location on your machine. It should all unpack into a single directory named RepertoireServer. You must have Java 6 or later installed on your system before running the server.

A license key is required before you are able to run the server. An evaluation license is available when you download the evaluation copy of our server. A registered user license will be sent to you once you have purchased the product. You will need a specific license to enable Report, Dashboard and ETL features - the evaluation license provides all three for a limited time period. Please contact Elixir Sales if you need further assistance in this area.

The license key is a text file which you can deploy by copying the license key to an appropriate directory. When you are running the Elixir Repertoire Server as in a Windows operating system, put the license in the .\ext directory as there is no meaningful user directory. Otherwise, you can also put the license file in the user directory (if several users have access to the machine). When you are using a Unix-based machine, it is advisable to create a new account for running the server. This provides more security as it limits the files that the server can access. Therefore, on a Unix-based machine, if you create a new account `repertoire` you will put the license text file in /home/repertoire. You should never run the server as root.

Starting the Server

The first step is to start the server. You will find the startup script in the bin directory inside the RepertoireServer directory. To run the server for Windows, use the startServer.bat, for Unix-like operating systems you should use startServer.sh. You should then see log messages from Elixir Repertoire Server components as they are deployed in the server log file (look in the log directory for server.log).

The first time the server runs, it will establish a default database in the db subdirectory. This may take a few minutes. If in future you upgrade your server, the db directory will not be overwritten, so settings there will be preserved. If you want to reset your server to it's original installed condition, simply stop the server and delete the db directory. A new db directory will be generated when the server is restarted.

Testing the Server

By default, the server is configured to run on port 8080. You can edit this in the config file config/ERS2.xml if that port is already in use.

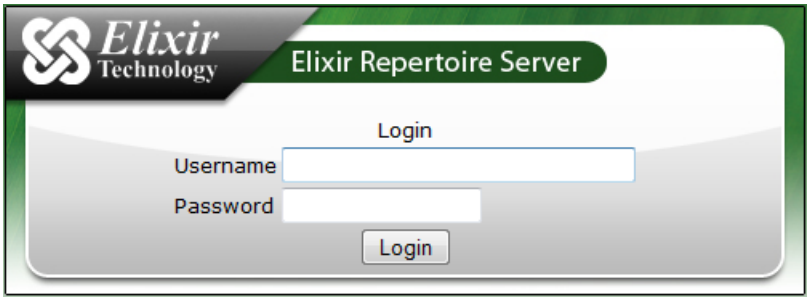
Connect to the server by pointing your web browser to http://localhost:8080/.

Note

You should substitute the server name for localhost, if the server is running on a different machine. Throughout this document we will refer to localhost, but you should use the actual machine name if your server is not on the same machine as your browser.

You should see in the browser the logon screen as shown in Figure 2.1, “Elixir Repertoire Server Logon”. The default administrator name is `admin` with password `sa`. Use these credentials to logon and then you should go to the User menu option to change the default password.

Figure 2.1. Elixir Repertoire Server Logon



Stopping the Server

To stop the server, you can use the Shutdown button on the Administration/Server page in your browser. Alternatively, you can type Ctrl-c at the console where the server is running, or you can run the stop

script in the bin directory (stopServer.bat or stopServer.sh). If you use the script, you must supply the admin username and password, e.g.

```
stopServer --user admin --pass sa
```

Renaming the Server

Renaming the server enables you to differentiate multiple server instances and makes it easier to administrate.

For Windows operating systems, open the bin/startServer.bat file to edit. Under the last line of the batch file, insert the machine name property in the following:

```
%JAVACMD% -mx512M -Delx.machine="ServerMachineName"
-jar RepertoireServer-Launcher.jar
```

For Linux operating systems, open the bin/startServer.sh file to edit. Under the last line of the .sh file, insert the machine name property in the following:

```
java -mx512M -Delx.machine="ServerMachineName"
-jar RepertoireServer-Launcher.jar
```

> **Note**
> The code segments should be added along the same line, connected by a space.

Replace "ServerMachineName" in the code segments with the new name. Save the changes before closing. Execute the startServer.bat or startServer.sh file. View the new name by pointing your web browser to http://localhost:8080/query/machine/.

Alternatively, you can edit this in the config file config/ERS2.xml, as shown in the section called “Server Machine Name”.

Running the Server as a System Service

In a live deployment you would not usually want to stop and start Elixir Repertoire Server manually but will want it to run in the background as a service or daemon when the machine is booted up. The details of how to do this will vary between platforms and will require some system administration knowledge and root privileges.

On Linux or other Unix like systems, you have to modify the startServer.sh file to specific to the Unix platform. On a Windows system, you can use a utility such as the Java Service Wrapper to deploy the server as a Window server service. The bin directory contains an example script. To run Repertoire Server as a Windows service, be sure to copy your server license file into the RepertoireServer\ext directory. For latest information, please read **windows-service-readme.txt** in the RepertoireServer\bin directory. You may refer to http://wrapper.tanukisoftware.org/ for in-depth discussion of this product.

From Elixir Repertoire Server 8.6.0 onwards, you will be able to install Windows service with a specific version of Java, and run Windows service with JMX (Java Management Extensions) remote connections.

Depending on which type of Windows operating system you are using, run Elixir Repertoire Service.exe (for 32-bit) or Elixir Repertoire Service(64).exe (for 64-bit). In the following examples, we will assume the 64-bit version.

- **Install Windows service with a specific version of Java:**

  1. Copy the JRE folder, which can be "jre6", "jre7", or simply "jre". In this example, we copy the jre6 folder from C:\Program Files\Java\jre6. Paste it into the Repertoire Server installation directory (\RepertoireServer). The JRE folder you pasted must be renamed to **jre**.

2. Open a **Command Prompt** window and locate the RepertoireServer\bin directory.
3. To install the service, run the following codes:

   ```
   "Elixir Repertoire Service(64).exe" /install
   ```

   You only need to run the codes above once. The service will be in the stopped state by default.
4. To start the service after installation, run the following codes:

   ```
   "Elixir Repertoire Service(64).exe" /start
   ```

5. Open a browser and connect to http://localhost:8080/
6. On the Repertoire Server login page, enter the correct username and password. Click **Login**.
7. Click **Administration** > **Server** to view the specific Java Version, which displays under the **Configuration** header. In this example, the Java Version shows `Sun Microsystems Inc. 1.6.0_41` instead of the default Java Version.

- **Run Windows service with JMX remote connections:**

1. Copy the 64-bit JRE folder. Paste it into the Repertoire Server installation directory (\RepertoireServer). The JRE folder you pasted must be renamed to **jre**.
2. Under the Repertoire Server installation directory, go into the \bin directory, and open the Elixir Repertoire Service.exe.vmoptions file. To set the port for JMX remote connections, edit this file by adding the following codes:

   ```
   -Dcom.sun.management.jmxremote.port=8004
   -Dcom.sun.management.jmxremote.ssl=false
   -Dcom.sun.management.jmxremote.authenticate=false
   ```

3. Open a **Command Prompt** window and locate the RepertoireServer\bin directory.
4. To install the service, run the following codes:

   ```
   "Elixir Repertoire Service(64).exe" /install
   ```

   You only need to run the codes above once. The service will be in the stopped state by default.
5. To start the service after installation, run the following codes:

   ```
   "Elixir Repertoire Service(64).exe" /start
   ```

6. Execute the jvisualvm.exe file, e.g. from C:\Program Files\Java\jre6\bin directory.
7. Right-click **Local** and select **Add JMX Connection...**
8. Type **localhost:8004** in the **Connection** field, and click **OK**.

   The 64-bit Windows service is working with JMX remote connections. Similarly, you can get 32-bit Windows service running by using the Elixir Repertoire Service.exe file.

Note

Specific details for running the server as a system service are described on the web at http://www.elixirtech.com/. Choose the Support menu option and check for your specific platform in the online documentation.

Chapter 3
Overview

Introduction

Now that you have installed Elixir Repertoire Server and have run the server for the first time, the next thing you will need to know is how the installation is laid out and what goes where. The overview covers the core components of the server, server directory structure, location of the key configuration files, log files, deployment and so on. It is worth familiarizing yourself with the layout at this stage as it will help you understand the server architecture so that you will be able to find your way around when it comes to deploying your own reports, data sources and their support libraries.

Main Directories

The distribution unpacks into a top-level `RepertoireServer` directory. There are thirteen core sub-directories immediately below this:

- *bin*: contains startup and shutdown and other system-specific scripts. We have already seen the run script which starts the server.
- *clients*: contains the client side supporting libraries and application for connecting to the server to generate report and list report templates deployed on the server. The clients/lib sub directory contains Java client core libraries (RepertoireClient.jar). The sub directory clients/bin contains some scripts to test the server and provide a direct invocation examples via command line, this useful for integrating report generation with a operating system like cron job.
- *config*: contains the server configuration files. These are discussed in Chapter 4, *Configuration*.
- *db*: this directory only exists after the server has run for the first time.
- *docs*: contains the server documentations and the APIs used for the server client connection.
- *ext*: is the location to place any third party java libraries such as JDBC drivers or custom Java classes. These will be loaded into the classpath of the server for use by dashboards, data sources and reports.
- *lib*: JAR files which are needed to run the Elixir Repertoire Server. You should not add any of your own JAR files here.
- *license*: Elixir Repertoire Server depends on a number of excellent open source libraries. Licenses for those libraries, along with the Elixir license are included here.
- *log*: contains all the server generated log files.
- *output*: The folder1 and folder2 subdirectories are used as sample targets.
- *plugins*: Plugins for Ad Hoc.
- *samples*: contains default set of sample reports, dashboards and data sources for testing the server.
- *web-pages*: The web pages used to present the browser interface. These often have dynamic content.

- *web-resources*: The web resources used to present the browser interface. These are static content.

Core Components

The Elixir Repertoire Server is made of six main components: the server core, logging, repository for storage of the report templates and data sources, administration and security (access control). The Java Management Service manages part of these services and provides a common access via the management console. None of the changes made through the console are persistent. The original configuration will be reloaded when you restart the server.

The server core is responsible for managing the rendering of reports and controlling the number of report within a queue. It interfaces with the report clients to provide basic services to end users application as repository listing, file object i.e. report template listing and generating of the reports in various format. Communication between the server and client uses a socket-based protocol.

The logging mechanism, based on Log4j, provides both an audit trail of report generation and information about the health of the server. The level of logging may set in the log configuration file to provide fine grain control over the log. This is particularly useful for debugging purposes.

The scheduling mechanism, manages the scheduling of job for triggering report or processing data.

Elixir Repository manages the storage of all report resources, such as templates, images and data source definition. A repository can contain many file systems. Each file system identifies a physical location where the resources are actually stored. Several kinds of filesystem are supported. The most commonly used filesystems are Local File System and Jar archived File System.

The browser interface provides functionality to manage the repository and additional functions may be access via other JMX beans (like shutting down the server, mentioned earlier).

Elixir Repertoire Server has an enhanced security model based on Java security infrastructure. Users can still make use of their application's single logon mechanism if needed.

The server provides both an HTTP and a Java client API for integration into software solutions. The Java client provides the ability to generate data and reports and simple repository browsing. The HTTP interface provides an extensive range of capabilities, including the ability to add, update and delete files in the repository, add, edit and delete users and groups, render reports, generate data, trigger jobs etc. Virtually the full server capabilities are accessible directly through HTTP, which can be called from a wide variety of programming languages and tools.

Chapter 4
Configuration

Configuration Overview.

In the previous chapter, we have seen the overview of the server components. We can now look into the various of configuring the server. Most configurable elements of the servers are located in the configuration directory (./config) except Java Virtual machine (JVM) configuration which is part of the server launch script. All configuration are stored in plain text or in XML format, making it easy to update and version them using a simple text editor.

Java Virtual Machine Configuration

Knowing how to configure the JVM is the most important aspect of configuring Elixir Repertoire Server. This has significant impact on the performance and memory load. It is critical if the server is going to have a heavy load or be deployed in a 24 by 7 environment. The parameters are specified in runnable server script as startServer.bat for Windows or startServer.sh for Unixes.

The default setting of the server are:

```
java -mx512M
 -Djava.security.auth.login.config=../config/auth.conf
 -Djava.security.policy=../config/java2.policy
 RepertoireServer-Launcher.jar
```

where `java` is the JVM executable. The actual configuration may differ between vendor and JVM version. User are advised to be familiar with all these settings. This information is shipped as part of the JVM documentation.

- *Java Heap Memory*: The Java heap memory limits the amount of memory that is available to the server. The setting `-mx[XXX]M` determines the maximum memory that the server could use for reporting where -mx is the maximum memory size, XXX is the actual memory size and M is the unit of measurement in Megabytes. Additional parameters -ms can be added predetermine the startup JVM memory size.

  The default configuration ,`-mx512M`. Generally 256 megabytes of memory is good enough for most small ,medium report generation load. The most common indicator when there is insufficient memory to generate the report is that the VM will throws OutOfMemory exceptions. In this situation, the memory can increase accordingly. Load testing is needed to determine the actual setting. It is recommended that 512 MB or more of both heap and ram memory for production server. The optimal maximum heap memory should be set at 75% to 85% range or lesser of the hardware RAM memory. This is to allow enough memory to be reserved for the operating system and other applications.

- *Java security model*: Java System Parameters, -Djava.security.auth.login.config set the URL pointing to the login configuration file (auth.conf) and the -Djava.security.policy URL pointing policy file that handles Principal-based queries, and the default policy implementation supports Principal-based grant entries. Thus, access control can now be based not just on what code is running, but also on who is running it.

Additional JVM settings can be applied. For JVM shipped by Sun additional parameters as -server for server side hotspot optimization and -Djava.awt.headless where it can run without the Window Frame Buffer for Unix platform. The -Duser.home system property determines the location of the license keys location. This may be added to the VM setting to modify the location.

Fine tuning of the Garbage Collection (GC) for performance and load is useful. Different JVM from vendors may provides various setting for optimizing GC like Parallel Throughput Garbage Collector which is useful with large young generation heaps, and lots of CPUs. While a batch report may use Mark-Sweep Garbage Collector. A search in the web may provide the white papers on the various discussion in this area. This optimization schema may vary vendor to vendor of the JVM and version.

Elixir Repertoire Server specific JVM configuration:

- *Server root directory (-Delixir.home)*: This setting is optional.
- *Server configuration directory (-Delixir.config)*: determines where all the configuration files are stored. This setting is optional. The default value is elixir.home/config.
- *Server log configuration (-Delixir.log)*: determines the log4j configuration file. This setting is optional. The default value is elixir.home/config/log-config.xml.
- *Server db configuration (-Delixir.db)*: determines the database location. This setting is optional. The default value is elixir.home/db.

Elixir Repertoire Server configuration

Elixir Repertoire server configuration files are located in the server configuration directory(./config). The files are

- *Server configuration*

  1. *ERS2.xml*: determines loading of the core report server functionality as MBeans. There is also an ERS2-sample.xml, which is a backup file, in case you need to recover from any edits.
  2. *jetty.xml*: determines the configuration of the server, for example, the session timeout interval, JSP, etc.

- *Server log control, log-config.xml*: set up the log configuration of the server.
- *Report Engine configuration, EREngine-config.xml*: determines the setup of the rendering parameters for the report engine and the PDF font mapping location.

  Elixir Repertoire Server 8.5.0 enables you to set Report Timeout to control the timeout interval for report rendering. Besides general rendering tasks, it can also work in conjunction with JDBC DataSource Timeout, as shown in Elixir Data Designer User Manual > Chapter 3. JDBC Datasource > JDBC Options. Report Timeout will start to function after JDBC DataSource Timeout takes its turn. In this file, uncomment the following command line and make necessary changes upon the timeout value to start using Report Timeout:

  ```
  <property name="rml.render.timeout" value="60"/>
  ```

  From Elixir Repertoire 8.6.0 onwards, you can control whether there will be a merged cell at the end of the report after it is rendered into XLS format, by editing the following codes in ERengine-config.xml:

  ```
  <property name="elx.xlspipeline.merge-trailer" value="yes"/>
  </properties>
  ```

The default value is set to "yes", which means there will be a merged cell at the end of the rendered report. Setting the value to "no" will enable you to render a report into an XLS file without merged cell at the end of the report.

Note

Setting the value to "no" might bring consequences to the report in XLS format, e.g. a cut-off of image, while the remaining things in the XLS file are normal.

- Other config files are for internal use or debugging purposes.

Server Configuration

Server Machine Name

The server machine name of the currently running instance can be altered after uncommenting a code segment in *ERS2.xml*.

```
<ers:property name="MachineName">Repertoire Server</ers:property>
```

Replace "Repertoire Server" in the code segment with the new name. View the server machine name by pointing your web browser to http://localhost:8080/query/machine/.

Alternatively, you can edit this in the bin/startServer.bat or bin/startServer.sh file, as shown in the section called “Renaming the Server”.

Password Policy

Password policy can only be enabled after uncommenting a code segment in *ERS2.xml*.

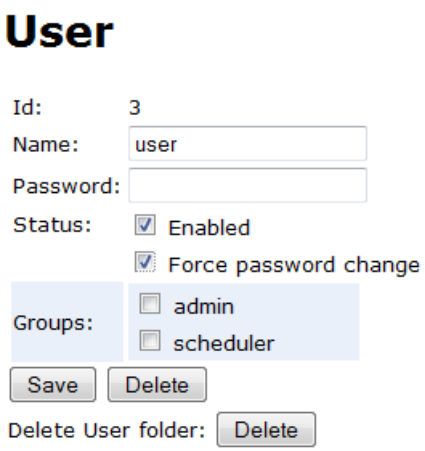
```
<ers:mbean name="ERS2:name=PasswordPolicy"
 class="com.elixirtech.ers2.security.PasswordPolicy">
 <ers:property name="MinLength">4</ers:property>
 <ers:property name="MaxLength">10</ers:property>
 <ers:property name="MustHaveDigit">true</ers:property>
 <ers:property name="MustHaveUpperCase">true</ers:property>
 <ers:property name="MustHaveLowerCase">true</ers:property>
 <ers:property name="MustHaveSymbolSet">!@$%</ers:property>
 <ers:property name="MustNotHaveSymbolSet">\/|</ers:property>
 <ers:property name="NotSameAsLogon">true</ers:property>
 <ers:property name="PasswordMaxExpiresDays">90</ers:property>
 <ers:property name="DifferentPasswordCount">2</ers:property>
 <ers:property name="RetryAttemptLockoutCount">5</ers:property>
 <ers:property name="RetryAttemptDelaySecs">5</ers:property>
 <ers:property name="RetryAttemptDelayMultiplier">2</ers:property>
</ers:mbean>
```

MinLength	Minimum length of password
MaxLength	Maximum length of password
MustHaveDigit	Password must contain at least one numeric character. Boolean type.
MustHaveUpperCase	Password must contain at least one alphabet in upper case. Boolean type.

MustHaveLowerCase	Password must contain at least one alphabet in lower case. Boolean type.
MustHaveSymbolSet	Password must contain at least one symbol. All symbols are acceptable except "<", ">" and "&". If the administrator wants to allow the use of "<", ">" and "&", a specific syntax must be used. Below is an example for wanting to use "&". `<![CDATA[&]]>`
MustNotHaveSymbolSet	Password must not contain any symbols. User can enter symbol(s) when trying to change the password, but it will be rejected when user tries to finalise the change.
NotSameAsLogon	Password must not be the same as the username.
PasswordMaxExpiresDays	Duration before password expires. In number of days.
DifferentPasswordCount	The number of previous passwords to be remembered. By default, the server would remember the 5 different passwords used previously. The maximum count is 5.
RetryAttemptLockoutCount	Number of failed logins before account is locked
RetryAttemptDelaySecs	The delay time before user can retry login
RetryAttemptDelayMultiplier	The multiplier rate for the day. For example, *RetryAttemptDelaySecs*=10, *RetryAttemptDelayMultiplier*=2, 1st login failure= delay login retry is 10 seconds, 2nd login failure=delay login retry is 20 seconds and so on.

If the administrator simply wants a user to change the password upon the next login and no other conditions need to be satisfied, it is unnecessary to use the password policy. Simply check `Force password change` as seen in Figure 4.1, “Force Password Change”. If desired, both the selection and the password policy can work together.

Figure 4.1. Force Password Change



Notification using email or Twitter

The notification service sends an email or post a tweet when the tasks in the queue exceed the maximum percentage limit in the settings. You have the option to encrypt secret values when setting up email or Twitter account details for receiving notification.

The notification service can be enabled by modifying the command line containing "NotifyMaxQueueCountExceeded" and setting the value to `true`.

The maximum task number and percentage can be configured in the following code segment in *ERS2.xml*. For percentage, minus value (<0) means no notification, and positive value (>0) means notifying according to percentage (60 = 60 percent).

```
<ers:mbean name="ERS2:name=Jetty"
class="com.elixirtech.jetty.JettyLauncher">
<ers:property name="MaxRenderCount">4</ers:property>
<ers:property name="MaxQueueCount">100</ers:property>
<ers:property name="NotifyMaxQueueCountExceeded">true</ers:
property>
<ers:property name="NotifyThresholdQueueCountExceededPercentage">
60</ers:property>
```

In this example, when the queueing task number exceeds 100 * 60% + 4 = 64, which means starting from 65, you will receive a notification message.

To make sure you can receive the notification message, uncomment the following code segment in *ERS2.xml*:

```
<ers:mbean name="ERS2:name=NotificationMail"
class="com.elixirtech.ers2.notification.Notification">
<ers:property name="ConfigPath">../config/notification/email.xml
</ers:property>
</ers:mbean>
<ers:mbean name="ERS2:name=NotificationTweet"
class="com.elixirtech.ers2.notification.Notification">
<ers:property name="ConfigPath">../config/notification/tweet.xml
</ers:property>
</ers:mbean>
<ers:mbean name="ERS2:name=NotificationLog"
class="com.elixirtech.ers2.notification.Notification">
<ers:property name="ConfigPath">../config/notification/log.xml
</ers:property>
</ers:mbean>
```

To receive notification via email: Fill your email information into /config/notification/email.xml.

To receive notification via tweet: Sign up for a Twitter account, obtain the keys from dev.twitter.com app, and fill your information into /config/notification/tweet.xml.

To encrypt secret values in email or Twitter account details: Run the *encrypt.sh* file in the RepertoireServer/bin/ folder with the secret value as needed (**smtpPassword**, **consumerSecret**, or **tokenSecret**). Edit the email.xml or tweet.xml file accordingly, as shown in the following table:

File	Places To Edit
email.xml	• Set **encrypted** to `true`. • Enter the encrypted value for **smtpPassword**.
tweet.xml	• Set **encrypted** to `true`. • Enter the encrypted values for **consumerSecret** and **tokenSecret**.

Task Manager timeout

The Task Manager timeout properties can be manually set in the ERS2.xml file.

- **Timeout (Seconds):** This property decides how long Task Manager will wait before canceling the report.
- **Monitor Interval (Seconds):** This property decides how often Task Manager will check if a task has exceeded the timeout interval.
- **Flush Interval (Seconds):** This property determines how long Task Manager will keep the tasks in the task list once they have been completed or have timed out.
- **Type:** This property enables you to set the type of the tasks, whose timeout situation will be monitored by Task Manager. There are three options for this property value: *****, **Report**, or **Data**, where ***** means Report and Data.

```
<ers:mbean name="ERS2:name=TaskMonitorThread"
class="com.elixirtech.ers2.task.TaskMonitorThread">
<ers:property name="Timeout">0</ers:property>
<ers:property name="MonitorInterval">60</ers:property>
<ers:property name="FlushInterval">600</ers:property>
<ers:property name="Type">*</ers:property>
</ers:mbean>
```

Jetty buffer size

Jetty buffer size is configurable in the ERS2.xml file.

In the ERS2.xml file, find the following code:

```
<ers:property name="SocketConnectorHeaderBufferSize">4096
</ers:property>
```

The default Jetty buffer size is set to 4K (4096) bytes. You can enable server to use a bigger value upon your requirement, e.g. 16K (16448) bytes, by changing the value here.

JMS Broker using Apache ActiveMQ

In *ERS2.xml*, look out for the codes at the bottom of the document which suggests about the using of JMS. Replace the name, `TestQueue`, to the sending queue name created in Apache (e.g. SQueue).

Next, copy *activemq-all-5.1.0.jar* from JMS broker and place it in /RepertoireServer/ext then start the server. Start a browser and connect to Repertoire Server Web Interface (http://localhost:8080) and login.

With the JMS broker started, on another browser to go to the JMS broker web interface (e.g. http://localhost:8161/admin) to send a trigger job message. The message `Destination` is set to the sending queue (e.g. SQueue) and `Reply To` is set to the receiving queue (e.g. RQueue). The message content format will be as follows :

```
<trigger>
  <run-as-user auth="YWRtaW46c2E="/>
  <job path="/testing/JMSbroker.job">
  <parameters>
    <parameter name="UserAge">26</parameter>
    <parameter name="UserName">David Tan</parameter>
    <parameter name="UserGender">M</parameter>
  </parameters>
```

```
  </job>
</trigger>
```

Click on the Send button to send the message. (In this example, *admin* is used and is converted to Base64 format.)

On the JMS broker web interface, select *RQueue* and check for the received message. A message should be received after the job has been processed. The message content should be like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<trigger-response>
<result>OK</result>
<details>Job '/testing/JMSbroker.job' completed successfully.
  </details>
<job-log>INFO 2008-05-29 11:33:35,796 Job: Job Start
 INFO 2008-05-29 11:33:39,718 RenderReport:
  render "repository:/testing/report/Report.rml",elapsedTime:
  3547 pageCount: 1 byteCount: 17921 mimeType: application/pdf
  statusCode: 1

  INFO 2008-05-29 11:33:39,718 Job: Job
  End Successfully
 </job-log>
</trigger-response>
```

Back to Repertoire Server web interface, under Logs, click on *server.log*. The following extracted logs should be seen:

```
Thread-51,INFO , jms.JMSTriggerThread - executing Job
  /testing/JMSbroker.job
Thread-51,INFO , Scheduler.Job - Job: Job Start
Thread-51,INFO , target.FileTarget - Opened
  C:\RepertoireServer\bin\..\output \folder1\file.pdf for writing
Thread-51,INFO , target.FileTarget - Closed
  C:\RepertoireServer\bin\..\output \folder1\file.pdf
Thread-51,INFO , target.FileTarget - Closed
  C:\RepertoireServer\bin\..\output \folder1\file.pdf
Thread-51,INFO , Scheduler.Job - RenderReport: render
  "repository:/testing/report/Report.rml",elapsedTime: 3547
  pageCount: 1 byteCount: 17921 mimeType: application/pdf
  statusCode: 1
Thread-51,INFO , Scheduler.Job - Job: Job End Successfully
Thread-51,INFO , jms.JMSTriggerThread - Job
  '/testing/JMSbroker.job' completed successfully.
```

Check for the generated pdf file in /RepertoireServer/output/folder1. The pdf file will contain the values of the parameters set when creating the JMS message.

Session Timeout Interval

In *jetty.xml*, this property sets the amount of time taken before a session timeout. It can be changed by editing the figure in the following code:

```
<session maxInactiveInterval="6000"/>
```

It is measured in seconds and the default setting is 6000 seconds, which is equivalent to 100 minutes.

Anonymous Login

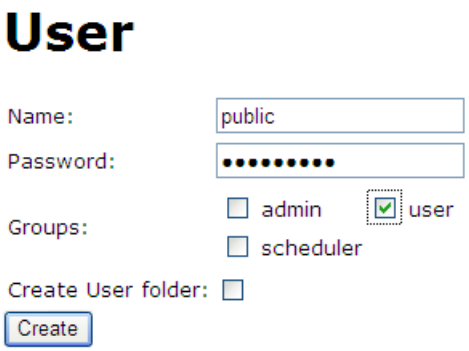
In *jetty.xml*, find the following code:

```
<anonymous enabled="false" user="public" pass="anonymous"/>
```

Change it from `false` to `true` to enable anonymous login.

Create the User ID through the server web interface as shown in Figure 4.2, “Create a Anonymous User ID”. `Name` and `Password` must be in sync with the choice of login parameters stated in *jetty.xml*.

Figure 4.2. Create a Anonymous User ID



Log off from the web interface and restart the server by shutting down the server and starting it again. Then, go to the Repertoire Server web interface once again. The user will be now able to enter to the page with no prompts for User ID and password unless the user visits the login page explicitly.

JSP Support

Before being able to see the JSP verification page when starting up the server, there are a few steps to execute. Firstly, in *jetty.xml*, we will need to change the following code to `true` instead of `false`, which is the default setting:

```
<jsp enabled="false"/>
```

core-3.1.1.jar, *jsp-2.1-jetty-6.1.15.jar*, *jsp-2.1-glassfish-9.1.1.B51.25.p1.jar* and *jsp-api-2.1-glassfish-9.1.1.B51.25.p1.jar* need to be placed in /RepertoireServer/lib/jetty for JSP to work.

Now, start the server by executing *startServer.bat*. Open a browser and enter `http://localhost:8080/test/simple.jsp`. The user will be prompted to login. After a successful login, the user will be able to see the JSP Verification page.

Mime-type Support for Browser

In *jetty.xml*, look out for the following code:

```
<mime-types>
<mime-type ext="doc" type="application/msword"/>
</mime-types>
```

User can add more extensions which are not recognized by the browser, following the same format as the one given. User can simply add the extension(s) before or after the line in *jetty.xml*.

Web Application Archive (WAR)

Before the server is able to deploy a WAR file, the server has to be configured to support JSP. For information on how to do that, refer to the section called “JSP Support”.

Then, place the WAR in the C: directory. In *jetty.xml*, look out for the following line:

```
<war file="C:/hello.war" url="/hello"/>
```

Enable this line by having it outside of the comments. If the file is saved in *C:/Documents and Settings/Admin/ElixirHome*, user can define as:

```
<war file="${elixir.home}/hello.war" url="/hello"/>
```

Save the file and start Repertoire Server. Open a web browser. Enter *http://localhost:8080/hello/* in the URL. The web page will then load the JSP file.

Server Log Control

The administrator can set *log-config.xml* found in the /RepertoireServer/log to display logs from one or more particular users. In order to do that, the administrator will need to add in the following set of codes into *log-config.xml*. The actions of the user(s) will all be captured in a log file named *activity-user.log*.

```
<appender name="Activity-User"
 class="org.apache.log4j.RollingFileAppender">
 <param name="File" value="../log/activity-user.log"/>
 <param name="Append" value="false"/>
 <param name="Encoding" value="UTF-8"/>
 <param name="MaxBackupIndex" value="5"/>
 <param name="MaxFileSize" value="500KB"/>
 <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern"
    value="%d{ISO8601},%-5p,%-10X{username}, %m%n"/>
  </layout>
 <filter class="com.elixirtech.arch.log.UserNameFilter">
  <param name="UserName" value="userABC" />
  <param name="AcceptOnMatch" value="true" />
 </filter>
 <filter class="org.apache.log4j.varia.DenyAllFilter"/>
</appender>
```

If the value set for `UserName` is *userABC*, the logs will be only of *userABC*. If the administrator wants to capture more than one particular user, the administrator will need to have additional *filter* tags with *UserName* and *AcceptOnMatch* attributes.

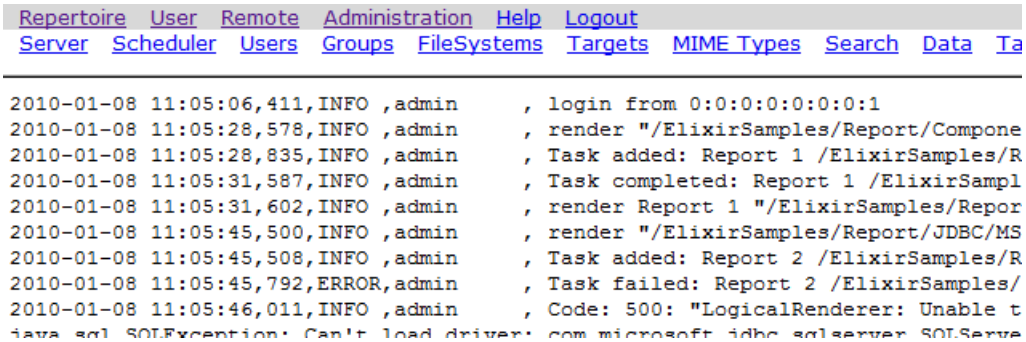
If the administrator do not want any logs from specified user(s), the administrator will only need to edit the *true* value to *false* in *AcceptOnMatch* attribute and remove the following line from the config file :

```
<filter class="org.apache.log4j.varia.DenyAllFilter"/>
```

If the administrator wants to have a unique identity added to a render report task, or data generation task, insert the following in the *ConversionPattern* attribute. This will help users narrow down which report invocation is causing the problem when error occurs. Figure 4.3, “Activity Log” displays a sample of the activity log.

```
%X{task}
```

Figure 4.3. Activity Log



In order for any changes to take effect, Repertoire Server will need to be restarted.

Note

Any changes done to any config files must be saved and the server will need to be restarted before the changes will take effect.

Administration of Core Services

Configuring the Elixir Repertoire Server

The initial server configuration can be done by editing the config file ERS2.xml. All day-to-day administration can be conducted through the web interface or the Remote Designer.

Table 4.1. Elixir Repertoire Server Configuration details

Name	Element name	Description
Server Listener IP	Host	By default, Elixir Repertoire Server accepts requests on all local IP addresses. If you wish to restrict the listener to a single IP address (e.g. you have multiple network cards), you need to set the desired IP address in dotted-byte format (e.g. 192.168.1.1). Requests will now only be accepted if sent to this specific local IP.
Server Listener Port	Port	Elixir Repertoire Server Listener handles all the incoming request from the clients. The default port number is set to 8080.
Allowed Clients	Accept	Elixir Repertoire Server can allow or refuse connections based on IP address. The Accept value is a regular expression that will be tested against the dotted-byte IP string of the client. Only those clients with accepted IP addresses will be allowed to connect. By default, this parameter is disabled, so that all clients can connect. Note that this value is a regular expression, so any dots (.) may need to be escaped. For example "192\.168\.1\.1" identifies the client at "192.168.1.1". You can include IP ranges using "192\.168\..*" - the final ".*" means any characters (in this case dots and digits) are allowed. This will allow connection from "192.168.5.20", "192.168.80.1" etc. whatever the value of the last two dotted bytes. You can also enumerate values, for example: "192\.168\.1\.1|192\.168\.1\.5" will allow connections only from the .1 and .5 clients.
Maximum Concurrent Report Render Count	MaxRenderCount	This parameter controls the number of report generation requests that can be processed concurrently. The set size will not exceed what is specified in the license. When the requests is exceeded the count, the requests will wait in the queue. The size dependent on the hardware, shared load with other application, operating system and its capacity. Proper sizing of the hardware is required to determine the optimal size. The general thumb rule for every one unit of CPU, the size can be incremented by 2 to 3 unit i.e. a two CPU system can handles 4 to 6 concurrent report generations at any one time.
Maximum Queue Count	MaxQueueCount	This parameter controls the number of report generation requests that can be kept in the queue. This set size will not exceed the value specified in the license. When the number of requests exceed the maximum queue size, incoming requests will be rejected.
Encrypted connection to client	Secure	When set to true, all data packets sent between the client and server are encrypted. Allowed values: true or false. The default is false. Note that ERSClient must be configured with setSecure(true) if this option is enabled. For more details on configuring secure mode (which requires generation of a server certificate) see the section called “Secure Mode”.

LDAP Secondary Authentication

In order to enable LDAP secondary authentication, the administrator will have to edit ERS2.xml with the necessary LDAP details before starting Elixir Repertoire Server.

During normal logon, the user's user name and password is checked against Elixir user's record. With LDAP secondary authentication, a user with user name or password not found in Elixir user record will be checked using LDAP to verify the user name and password entered. If this user is valid in the

LDAP server, Elixir user and group records will be updated according to the values accepted by the LDAP server. If no such user name was found, a new user will be created. New groups may be created dynamically to match those that the user was assigned to in the LDAP server. After a successful secondary authentication, subsequent logons will be as per normal as the records are already stored in Elixir records. If it still fails, the user will be unable to logon.

The web interface for users to change their password is disabled when LDAP secondary authentication is enabled. This will avoid any confusion as any subsequent entry error will result in the resetting of password back to the LDAP password.

If the LDAP password is changed, the user can logon immediately with the new password due to secondary authentication. However, the user can still keep to the old password until a RESET forces the user to use the new password.

An Administrator is required to log in to trigger the Reset function. In order to reset, in the Web interface, go to Administration, Users. Click on the Reset LDAP Users button.

Alternatively, you can reset the LDAP server using the REST action. Simply paste the following URL in the address bar :

```
http://localhost:8080/tool/admin/users.html?action=ResetLDAPUsers
```

This action will then apply the changes done to the user's password.

To verify that the LDAP user can no longer log in with the old password, go to Elixir Repertoire Server Web interface log in page and enter the LDAP username and the old password. The log in will fail. When the user tries logging in with the LDAP username and new password, the user will be able to log in successfully.

LDAP Multiuser Subtree

Before configuring Elixir Repertoire Server to support LDAP Multiuser Subtree, make sure a multiuser subtree exists in LDAP server and LDAP authentication mode is enabled. This feature is supported in Repertoire Server 8.2 and later versions.

Under the Repertoire Server installation directory, go into the /config directory, open the ERS2.xml file, and look for the following codes:

```
<ers:mbean name="ERS2:name=LDAPUserRoleAuthentication" class=
"com.elixirtech.ers2.security.ldap.LDAPUserRoleAuthentication">
```

Change the statement

```
"com.elixirtech.ers2.security.ldap.LDAPUserRoleAuthentication"
```

to the following statement:

```
"com.elixirtech.ers2.security.ldap.LDAPMultiUserRoleAuthentication"
```

Replace the code

```
<ers:property name="UsersDN">ou=people,dc=elite</ers:property>
```

with the following codes:

```
<ers:properties name="UsersDNList">
<ers:item>ou=users,dc=example,dc=com</ers:item>
<ers:item>ou=altusers,dc=example,dc=com</ers:item>
</ers:properties>
```

Save the ERS2.xml file and start the Repertoire Server. After entering the correct user name and password, the user will log in successfully.

SMTP Server

Elixir Repertoire Server includes an SMTP server named `elixir.aspirin`. The configuration of this server is in ERS2.xml.

You may add additional external SMTP servers and remove the default one too, if required. External SMTP servers require additional information like this:

```
<ers:mbean name="ERS2:name=GmailSMTPServer"
      class="com.elixirtech.ers2.mail.SMTPServer">
    <ers:property name="Host">smtp.gmail.com</ers:property>
    <ers:property name="Port">465</ers:property>
    <ers:property name="User">[user]@gmail.com</ers:property>
    <ers:property name="Password">password</ers:property>
    <ers:property name="ConnectionTimeout">30000</ers:property>
    <ers:property name="TLSEnabled">false</ers:property>
    <ers:property name="SSLEnabled">true</ers:property>
    <ers:property name="Debug">false</ers:property>
</ers:mbean>
```

Note that the mbean name (`ERS2:name=GmailSMTPServer` in this case) must be unique within ERS2.xml. Once the SMTP server is configured, you can reference it by name from a Mail Target (see Figure 6.4, “Mail Target” below, or from a Job (see the Elixir Schedule Designer manual).

Configuring the Logging

Log4j is used for Elixir Repertoire Server logging mechanism. If you're not familiar with the log4j package, you can read the full detail about it at the Jakarta web site. (http://jakarta.apache.org/log4j/).

Logging is controlled from a central log configuration file (config/log-config.xml). This file defines a set of appenders, specifying the log files, what categories of messages should go there, the message format and the level of filtering. By default, Elixir Repertoire Server produces a log file called server.log in the log directory).

There are 4 basic log levels used: DEBUG, INFO, WARN and ERROR. The logging threshold is INFO, which means that you will see informational messages, warning messages and error messages but not general debug messages.

The default server log is set to rotate the log every 500KB and the file is overwritten every time the server is restarted, up to five server logs are generated before the same file name is reused.

Secure Mode

Elixir Repertoire Server provides an HTTP interface. This uses the http:// URL prefix. If you wish to run a secure protocol, then you will need to switch to https://. The secure protocol is configured by following these steps:

1. Edit ERS2.xml to uncomment these lines:

   ```
   <!--ers:property name="Secure">true</ers:property>
   <ers:property name="Port">8443</ers:property>
   <ers:property name="Password">secret</ers:property -->
   ```

2. Edit ERS2.xml to comment out the plain mode alternatives immediately below the secure version.

3. Create a new directory `ssl` inside the `config` directory. You now have to set up a "keystore" that contains a digital certificate. The server uses this to authenticate itself to the clients.

4. Open a command prompt in the new ssl directory and enter:

   ```
   keytool -keystore keystore -alias jetty -genkey -keyalg RSA
   ```

   You will be prompted first for a password. Enter something memorable. Now you need to answer a number of questions. Most are optional, the only question that you must answer is the first one: "What is your first and last name?". Enter the name by which users will access the server, for example www.example.com. You should not include any prefix, e.g. https. You can ignore the other questions if you like. Finally you will be asked for a key password. Just press enter to use the same password entered at the start.

5. You now have a file called keystore in config/ssl. Go back to ERS2.xml and enter the password you chose into the password property. Start the server and you should now be able to connect to the server with https://localhost:8443/ as your new URL (substitute your machine name as appropriate).

When you connect for the first time over https to the server, your browser will ask you if you want to accept the server certificate. You should look at the contents and ensure they match the certificate you created. If you accept, the browser will remember the server, so that you can connect directly in future.

The steps that have just been described, show you how to create a self-signed certificate. If you are intending to allow external users to connect to the server, you may wish to purchase an SSL certificate from a trusted Certification Authority (CA). When connecting to a trusted server, you will not need to accept the certificate the first time the browser connects. There are further implications for using https with a self-signed certificate, which are discussed in Chapter 8, *Server API*.

If you don't want to store the plain text password in the configuration file, you can encrypt it using the encrypt utility in the server /bin directory. Open a command prompt at the bin directory and run the encrypt program with a single parameter - the string you wish to encrypt. The encrypted value will be returned. If your string contains spaces or special characters, be sure to quote it `"like this"` to ensure the encrypt routine sees the whole string as one value.

It is essential to ensure your configuration is working first, before you encrypt the password. Once you have the encrypted value, paste it into the configuration file and mark the property encrypted, like this:

```
<ers:property name="Password"
    encrypted="true">FrRMRoVI36lj3o3drUGqNA==</ers:property>
```

(this example is wrapped to fit on the page - you must not insert spaces and newlines in the encrypted string). You can encrypt any mbean property values in ERS2.xml using the same approach.

Targets and User Access

A target is a destination to which the server can direct output. Targets include physical file locations, either on the local or remote machines, and email addresses.

Users and groups can be granted access to different targets. You can use parameter substitutions, e.g. ${dir} in target parameters, so that users can set the values by passing parameters while rendering. This allows the same mail target to be used to send to different recipients, for example.

Target Parameter Resolution Mechanism

Each target can be configured via a set of properties. Administrators can define parameters in target configuration through the browser interface, if some parameters are to be provided by end users.

Roles

Users and groups are defined and maintained using the server's web administration interface. See Chapter 6, *Web Interface* for more details.

You may want to restrict certain users or groups from using certain targets - for example, you don't want everyone to be able to email confidential reports. Using the section called “Targets”, you can define which users and groups should have access to which targets.

Chapter 5
Security

Overview

In this chapter we will review security features and recommend best practices to ensure Elixir Repertoire Server is used securely.

Preparation

It is recommended that you create a new user account and use that to run the server. This ensures you can limit access by the server to any restricted files and programs. You should not run the server using a root or administrator account, as this will typically give the program (and depending on your security configuration, all user scripts) full access to the machine.

If your server is running on an Intranet, then you can use the default http: protocol. User names and passwords are sent to the server using HTTP Basic Authentication, which is obfuscated, but not encrypted. It is possible, using packet snooping tools, to extract the user name and password from this data stream. On an intranet, these packets will not be accessible outside your network. However, when running over the Internet, these packets will be visible to external programs. Therefore, if your server is running on the Internet, you should consider switching to https: protocol (secure mode), which will encrypt all data, including the HTTP Basic Authentication user name and password. See the section called “Secure Mode” for details on how to configure secure mode.

Connections and Firewalls

Elixir Repertoire Server uses one of two ports by default. Either port 8080 is used, for regular http: connections, or port 8443 is used, for secure mode https: connections. You should ensure that your clients are allowed access by permitting TCP connections to the desired port to pass through your firewall. You can change the operating port by editing the server configuration file config/ERS2.xml. If your client program is unable to connect to the server, try using your browser to connect instead, by entering the server URL, something like http://myserver:8080/. If the browser can't establish a connection either, then probably there is a firewall preventing connections.

You can further restrict access to the server by entering the allowed client IP addresses in the Accept regular expression in config/ERS2.xml. See the section called “Administration of Core Services” for more details.

Opening file using URL

User can open a file using paste the direct URL in the address bar of a web page. User will be prompted for username and password before being able to view to desired file. This is to check if the user has access rights. For dashboard, use the following syntax

```
http://localhost:8080/tool/dashboard/<dashboard directory>/xxx.pml
```

For report, use

```
http://localhost:8080/tool/report?report=/<report directory>/
 xxx.rml
```

For datasource, use

```
http://localhost:8080/tool/data?datasource=/
 <datasource directory>/xxx.ds
```

Protecting Sensitive Information

When connecting to databases you often need to supply a user name and password. There are a few ways that you can protect this information. Of course, the ultimate protection is to not store the password and instead require the user to enter it each time the database is accessed. This can be done with dynamic parameters. More commonly though, the password is entered into the the datasource file. It is protected from casual view by showing marks such as ***** instead of the actual password characters.

However, by opening the .ds file (an XML file) with a text editor, all of the information is available in plain text (This is a good thing - Elixir does not lock you in to proprietary binary file formats). There are a few ways to prevent the file being read. Firstly, the last page of each datasource wizard lets you hide the datasource details. The user and password fields will no longer appear on the GUI. Next, you can encrypt the datasource - this prevents the password being read using text editors. By selecting Hide Details and Encrypted you have protected this sensitive information. Note that you must remember the password you used when encrypting the file if you ever need to unencrypt it again.

Often several datasources read from the same database. Rather than enter the password in each one, you can define a Connection Pool to hold the common connection information. In this case, you only need to apply Hide Details and Encryption to the connection pool. The individual datasources will delegate to the connection pool and do not hold any sensitive information.

A final alternative is to use Elixir Safe. With this approach, all sensitive information, for datasources, reports, dashboards and jobs, can be kept in a single file and read by supplying the encryption key. This can be a dynamic parameter that the user must enter in order to access the file. Full details on Elixir Safe, along with sample code fragments are provided in the Elixir Repertoire User Manual.

Datasource Security

Specific row(s) and/or column(s) can be restricted to specific user or group. In this way, sensitive information in the row(s) and/or column(s) will not be seen by the specified user.

Before configuring the data that are to be restricted, a little amendment needs to be done in ERS2.xml. In ERS2.xml, ensure that the following code segment is not commented.

```
<ers:mbean name="ERS2:name=DSMFactory"
 class="com.elixirtech.data2.DataSourceManagerFactory">
 <ers:property name="DataSourceManagerClassName">
  com.elixirtech.ers2.data.SecureRowColDataSourceManager
 </ers:property>
</ers:mbean>
```

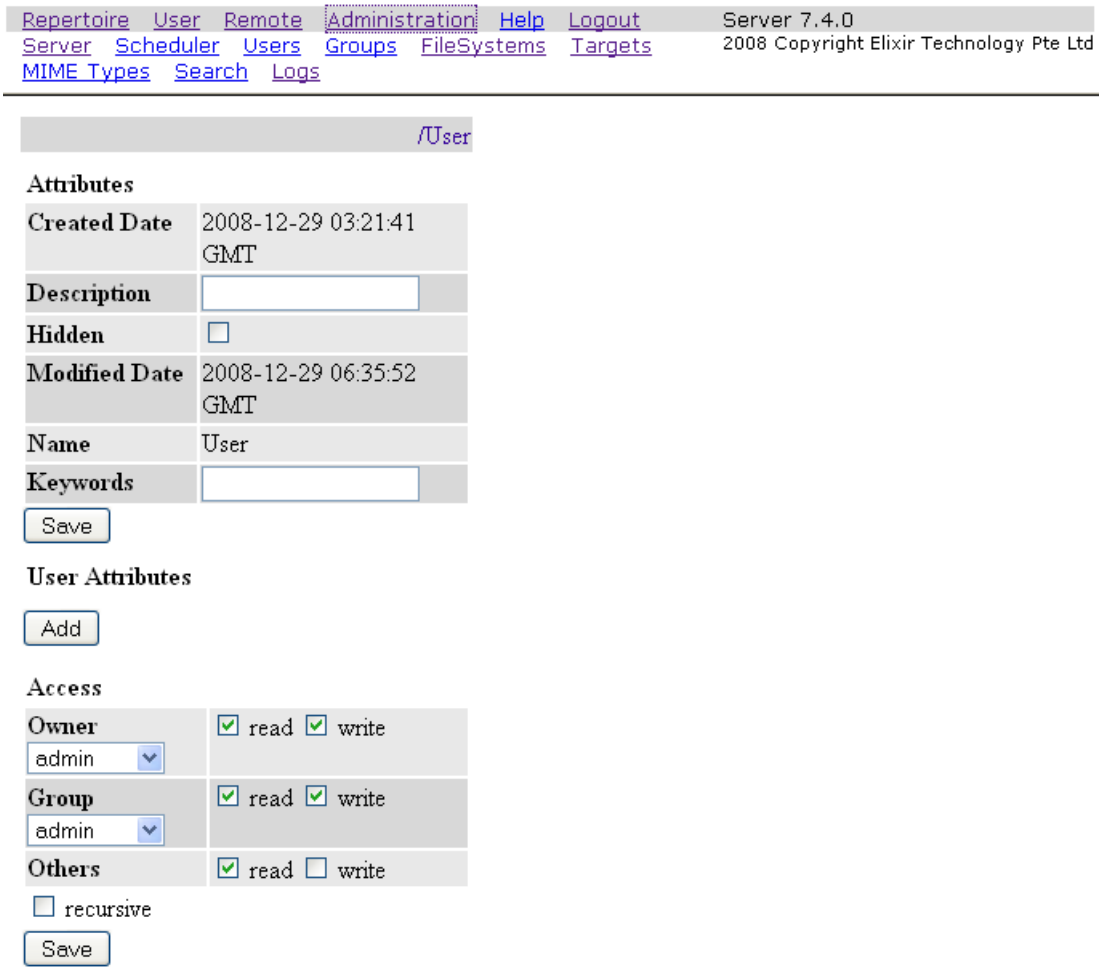
To start the configuration, go to *Administration*, then to *Data*. Clicking on the Add icon will allow user to add the desire restrictions. More information on datasource security can be found in the section called “Case Study 1 - Datasource Security”.

Access Rights

Access rights configurations are based on Unix Users and administrators will have the privilege to set the access rights to directories, provided they have the necessary permissions to do so.

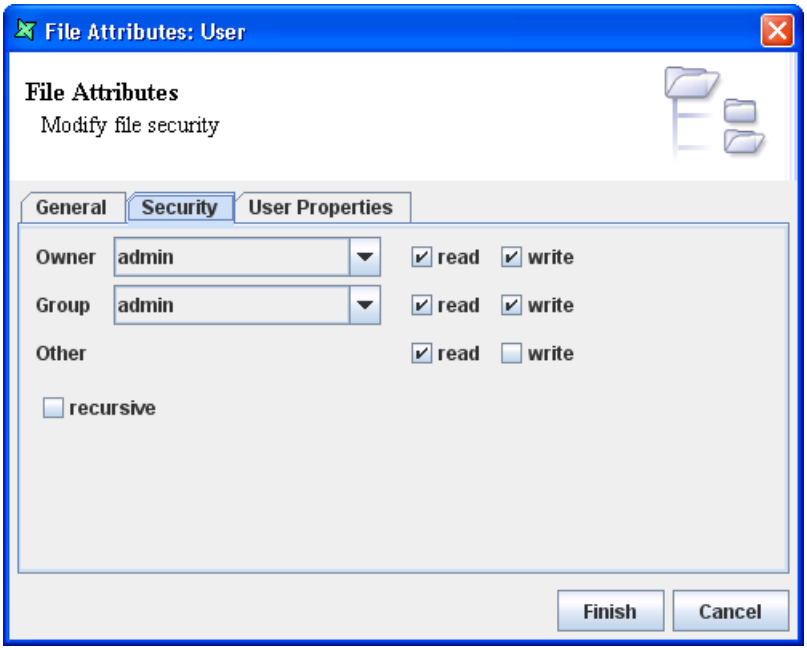
The access rights can be configured in two ways. The first way will be configuring through the web interface. By clicking on the icon on the right of the directory in `Repertoire` page, the page for configuring the access rights for the particular folder will load, as shown in Figure 5.1, “Access Rights Configuration through Web Interface”, allowing configurations to be done.

Figure 5.1. Access Rights Configuration through Web Interface



The second way will be doing it using Remote Designer. By right-clicking on the filesystem, select `Edit File Properties...` and select `Security` tab. The page for setting the configurations, as seen in Figure 5.2, “Access Rights Configuration through Remote Designer”, is similar to the one setting through the Web Interface.

Figure 5.2. Access Rights Configuration through Remote Designer



JavaScript Permissions

When Elixir Repertoire Server is running, it is often executing JavaScript code on behalf of the user. It is important that this code is prevented from performing any dangerous operations, such as reading sensitive files or maliciously deleting information. In order to do this, the Server restricts the operations that can be performed through JavaScript, for example unrestricted access to files. By default the server restricts potentially unsafe operations. You can relax the constraints if necessary by modifying the security policy file. For more details please see the JavaScript chapter in the Elixir Report Designer Manual.

Case Study 1 - Datasource Security

This case study briefly demonstrates the way to enforce datasource security.

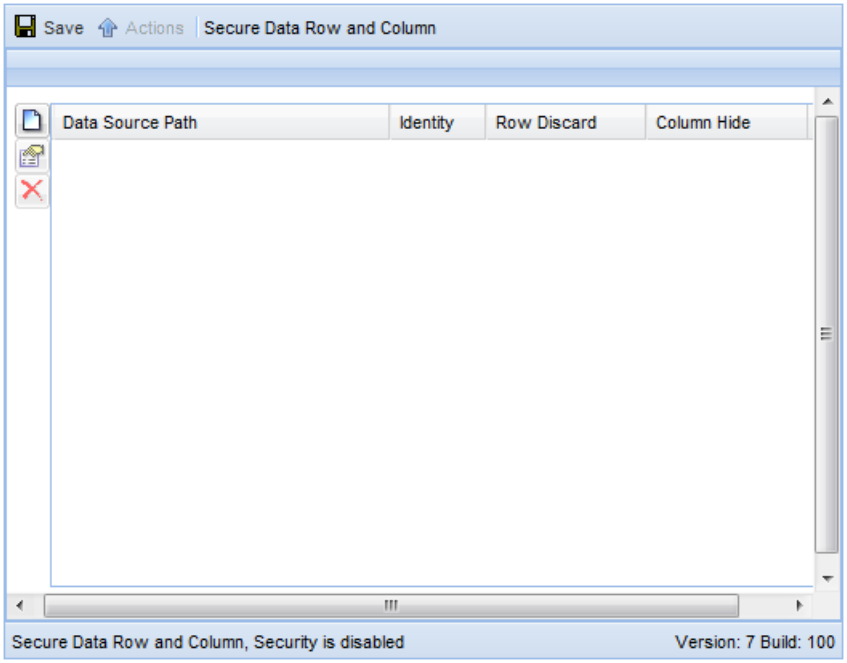
1. As mentioned in the section called “Datasource Security”, to enable datasource security, a segment of codes in ERS2.xml must be uncommented. The segment of codes is as follows:

```
<ers:mbean name="ERS2:name=DSMFactory"
 class="com.elixirtech.data2.DataSourceManagerFactory">
 <ers:property name="DataSourceManagerClassName">
  com.elixirtech.ers2.data.SecureRowColDataSourceManager
 </ers:property>
</ers:mbean>
```

2. Start the server. Information about starting up the server can be found in the section called “Starting the Server”. If the server is already started before the editing of ERS2.xml, a restart of server is required.

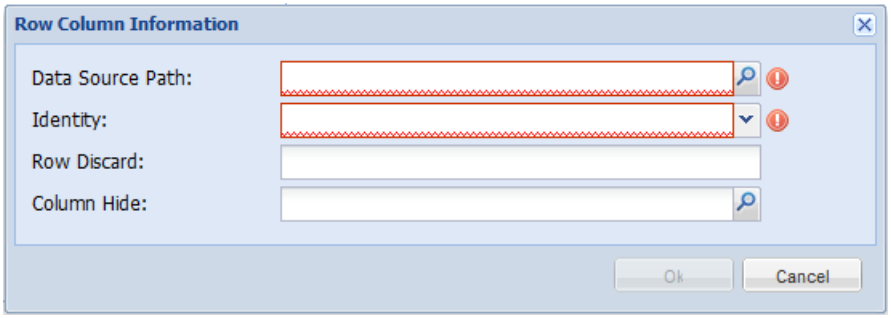
3. After log in, go to *Administration*, then *Data*. A screen as seen in Figure 5.3, “Datasource Security” will be loaded. If the codes are still commented in ERS2.xml, or a the server is not restarted after the editing, at the bottom of the page, it will say *Secure Data Row and Column, Security is disabled*.

Figure 5.3. Datasource Security



4. Click on the Add icon to add a new restriction. A window like in Figure 5.4, “Row Column Information” will appear.

Figure 5.4. Row Column Information



DataSource Path — User enters or selects the path of the datasource. This field is compulsory.

Identity — Specifies the user or group that is being restricted. Similar to *DataSource Path*, this field is compulsory.

Row Discard — Discards a particular row. Filter rules such as "<", ">", "<=", ">=", "=", "!=", "~" and "!~" are supported. For example, using *FruitSales.ds* to demonstrate, to discard row(s) that contains `Apple` in the `Fruit` category, enter the following in the field:

```
Fruit=Apple
```

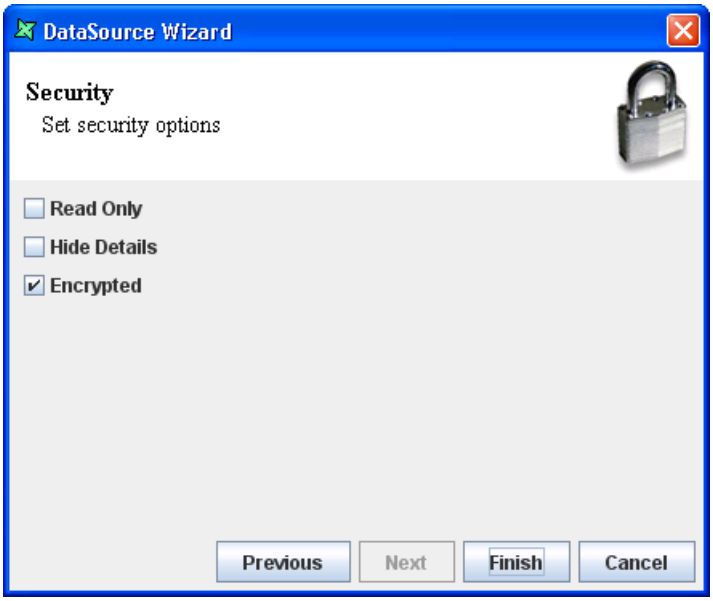
When the restricted user or group views this datasource, any row(s) that contains `Apple` will not be loaded.

Column Hide — Hides a particular column from the selected user

5. Click on `Ok` to confirm the configurations. Click on `Save` to save all configurations done, else all editing will not be saved.

6. It is recommended that a datasource is encrypted if it contains sensitive data. This is because the restrictions would not apply if a datasource is being copied from one directory to another. On the other hand, an encrypted file retains the encryption properties even after being copied to another directory. This can be done in the last page of the DataSource Wizard by checking `Encrypted` as seen in Figure 5.5, “Encrypt DataSource”.

Figure 5.5. Encrypt DataSource



Chapter 6
Web Interface

Overview

Elixir Repertoire Server provides a web interface that allows authenticated users to generate data, render reports, view dashboards and run jobs as well as viewing archived information, such as PDF reports and Excel files. In addition, the web interface provides a set of administration tools that handle the day-to-day administration requirements: creating users, reviewing logs, backing up the database, etc.

Once you have logged on to the server, by pointing your browser at the server location (e.g. http://localhost:8080/), you will see a menu bar of options and the list of current filesystems below, if you want to start immediately navigating the repository. The server web pages have been designed so that you can make full use of your browser. You can open links in new tabs, you can bookmark all links - including those within the repository - and can reopen them at any time (provided you are authenticated). The only exception to this is individual views within a dashboard, the entire dashboard needs to remain together in order for views to update each other. However, the dashboard can be launched separately from the main menu and bookmarked directly, as described previously.

Note

Advanced administrators can even modify the web pages themselves - adding new links, adding styles and company logos etc.

The rest of this chapter describes the individual pages of the web interface.

Applet

Before running Remote Designer as an Applet, make sure the Java version on your machine is up to date.

Complete the following steps to configure Repertoire Server to run Remote Designer as an Applet:

1. Locate the Repertoire Server 8.6.0 install directory. Open the **web-resources/jnlp** directory.
2. Extract **RepertoireRemote.jar.pack.gz** into the **web-resources/applet** directory using the following command line:

   ```
   unpack200 RepertoireRemote.jar.pack.gz RepertoireRemote.jar
   ```

3. Open the **bin** directory and execute the **startServer.bat** or **startServer.sh** file.
4. Open a browser and visit **http://localhost:8080/applet**. Remote Designer will be successfully launched as an Applet.

Note

Before Remote Designer gets started, a warning message may pop up, asking if you want to run this application - Elixir Repertoire Remote Designer. Select the **"I accept the risk and want to run this application"** checkbox and click **Run**.

Complete the following steps to configure Repertoire Server to show Applet on top menu:

1. Locate the Repertoire Server 8.6.0 install directory. Open the **web-pages** directory and open the **top-menu.html** file.

2. Find the correct section to add the codes for Applet by looking for the following codes:

```
<div class="topmenu">
```

3. Add the following codes:

```
<div class="menu1"><a href="/applet" target="main">Applet</a>
</div>
```

4. Open the **bin** directory and execute the **startServer.bat** or **startServer.sh** file.

5. Start a browser and visit **http://localhost:8080/**.

6. Under the Server login page, enter the correct username and password to log in.

7. Click **Login**. You will be able to see **Applet** as one of the options in the top menu. Click the **Applet** option to start Remote Designer.

Repertoire

The Repertoire page lists the repository filesystems. Clicking on a filesystem and subsequent folders and files allows you to navigate through viewing of datasources, templates, dashboard and executing jobs. Other files such as pdf, text file, xml files etc can also be viewed through here. The structure of the repository is left entirely to the administrator, archived reports could be placed in separate filesystem from the report templates used to generate them, so that access rights can be easily administered.

If you click on a datasource, you will be prompted for any parameters the datasource requires and the data will then be generated into the browser. If no parameters are needed, then the data will be generated directly upon choosing the datasource file.

If you click on a report, you will be prompted for any parameters the report requires and also will be asked to choose the output type for the report. The default output type is HTML, which will show directly in your browser. Other formats, such as Glint, PDF and Excel are also available. Some of these may appear in your browser (depending on installed plugins), or you may be prompted to download the file. The Simple HTML option produces just a single streamed HTML output, rather than the paged output you get from the default HTML choice. This produces only a single file, with no dependencies. Therefore, this format is only suitable for reports that have no generated images (it is ok to include images that are accessible by URL).

If you click on a dashboard, you will be prompted for any parameters the dashboard requires and the dashboard view will then be generated into the browser. If no parameters are needed, then the dashboard will be generated directly upon choosing the dashboard PML file.

For other file types, the data will be sent to the browser. How the browser handles the data depends on how it has been configured and what plugins are available. For example, with Adobe Acrobat installed on Windows, choosing a PDF file from the repository will open it directly in your browser. However, if you don't have Adobe Acrobat, you will be prompted to save the file instead.

Search

The repository search allow you to locate files in the system based on pre-defined criteria as file name, path or extension. You can do range search based on file modified date.

You can customized your search using Lucene query criteria with Lucene syntax. Some of criteria you can use are as follow.

File Name	File name in the repository
File Extension	File extension in the repository. For example, you can enter "ds", "rml" or "pml" to search for files in a certain type.
File Last Modified Date	Date that file was last saved after changes were made
File Path	File path in the repository
File Keywords	Keywords of file in the repository
DataSource embedded in file (path)	Datasource contained in file in the repository
Report embedded in file (path)	Report nested in a file in the repository
Lucene Query	Customize search. For example, you can use the syntax [field]:[query] `file.name:E* AND file.ext:ds` This is to locate any files starting with E and has an extension of ds. The details of the syntax is located at Apache Lucene website [http://lucene.apache.org/java/docs/index.html] under Query Parser Syntax.

The repository search indexing is updated once on the server startup. You may manually trigger a fresh or schedule the indexing of the repository. The configuration detail is found later in this chapter.

Manage

This option makes organizing and maintaining of files and folders in the repository easier and faster with buttons clearly labelled for each respective task.

- *Copy*: copying of files and folder between filesystem and folder
- *Move*: this option is only available to files and folders. Option will be available when file(s)/folder(s) are checked. Selected file(s)/folder(s) from one panel will be moved to the directory of the other panel.
- *Rename*: renames a selected file/folder. Similar to the *Move* option, this option is only available when a file/folder is selected.
- *Delete*: deletes selected file(s)/folder(s)
- *New Folder*: creates a new folder in the current directory
- *Upload*: uploads file into the current directory
- *Download Folder*: downloads a folder in ZIP file format to a location as configured in the browser settings. The entire folder will be downloaded, disregarding which file(s)/subfolder(s) are checked.
- *Continue*: brings user back to current directory in the Repertoire. When user clicks on the `Continue` button on the right/left panel, it will bring the user bring the user back to directory that the respective panel is in.

User

System

The filesystem structure is cached for efficiency. If a file has been recently added, it may be necessary to refresh the filesystem to see the changes. Users can also refresh all the filesystems in the repository at one go, to ensure all new files will be listed.

The dashboard views remember their state, for example what rows are selected, even if you switch to do something else. If you return to the dashboard within the same logon session, you can continue from where you left off. There is an option on the System page to force the dashboard to reset the views to their initial conditions. Note that only the latest dashboard state is remembered to save server memory. If you switch to a different dashboard, then the previous dashboard state is automatically reclaimed.

Password

User can change their own password. The existing password is required, followed by new password, which must then be re-entered for confirmation. If the existing password is correct and both the newly entered passwords match, then the change is accepted. The new password will be required next time the user logs on.

Remote

In addition to the stand-alone Repertoire Designer, there is a Repertoire Remote Designer, that provides a client GUI, but the repository, rendering and generation remains on the server. This menu option takes you to a page where you can launch the Remote Designer, using Java WebStart. You must have Java version 6 or later installed on your client machine in order to use this feature.

Note

Once the Remote tool has been downloaded, which might take a minute or two the first time, depending on the speed of your network connection, it will be cached on your local machine, so subsequent launches will be much faster. If the Remote tool is updated on the server, then a new version will be automatically downloaded to your client next time it is executed.

Launching of the Remote tool depends on the availability of RepertoireRemote.jar, which is placed in web-resources/jnlp. To speed downloading the jar is compressed using Sun's pack200 format, so the file is actually called RepertoireRemote.jar.pack.gz. You can uncompress it manually by running

```
unpack200 RepertoireRemote.jar.pack.gz RepertoireRemote.jar
```

The unpack200 program is in your Java bin directory. You don't need to uncompress the file for use with WebStart, but you can unpack and copy this jar to the client and launch it directly from there. In this case, you won't benefit from the WebStart functionality of automatically downloading updates. To launch the jar directly, either double-click on the jar (assuming the jar file type is registered by Java) or by running:

```
java -jar RepertoireRemote.jar
```

When the Remote tool is launched from the Server by an authenticated user, it immediately opens a connection to the server. However, if the tool is accessed by an unauthenticated user, or run directly from the client machine, the user will be prompted for their user name, password and server details before being allowed to access the server resources.

When you launch the tool through WebStart, you are connecting the the URL /remote/remote.jnlp on your server. The response is constructed from remote.jnlp.template in the config directory. You can customize this to control advanced WebStart settings. Certain parameters can be passed to the remote URL and will be embedded in the jnlp response.

initialFile — Determines the initial file to load when the Remote tool opens.

maxWorkspace — If set to true, this option maximizes the workspace by collapsing the repository panel.

You can craft specific URLs to launch Remote with these parameters. For example,

```
/remote/remote.jnlp
    ?initialFile=/ElixirSamples/Dashboard/Tutorial/
    dashboard/SampleDashboard.pml
    &maxWorkspace=true
```

(all on one line) will load the SampleDashboard and maximize the workspace.

Note

The initialFile parameter will only work if the user is already authenticated with the server.

Administration

This option is only available to the admin user, or users belonging to the admin group. The administrator can perform day-to-day configuration, maintenance and monitoring through, this interface.

Server

The Server screen is classified into different sections as described below:

Configuration

The server configuration details, such as the Java version used, the Elixir directory structure and also the processor information. The directory locations are controlled by command line options, such as -Delixir.home which are discussed in the section called “Java Virtual Machine Configuration”.

License

The server license indicates what options the server will support. The license is typically stored in the user home directory. If the license file is changed then the server will need to be restarted to read the new settings.

Classpath

The classpath shows all directories and jar files which are usable by the server. The list includes checksums for the jars for easy version comparison, which also aids in determining whether the Elixir Repertoire Server has been modified, or is using inconsistent or mismatched resources. Some server functionality, for example JDBC access, is dependent on having the right 3rd party jar files installed for your specific database. Often the database vendors produce several variants of the jar, often with the same name, which can make it tricky to troubleshoot JDBC drivers errors.

Database

Elixir Repertoire Server contains a database which holds user and group information, along with filesystem support. You can backup the database without stopping the server by choosing the backup

option and identifying a directory where the backup should be stored. It is not possible to restore the database without stopping the server, as this could affect ongoing requests. In order to restore the database, you need to stop the server and copy the backup directory to the elixir.db directory (which is shown in the Configuration section at the top of this web page). Note that if you want to reset to a default database, you can stop the server and delete the elixir.db directory (which is called `db` by default) and a fresh database will be created next time the server is started.

The pre-requisite for this option is that the user must login with Administrator privileges.

Backup DB

By entering a valid path directory in the field provided beneath *Backup to:* and clicking on the `Backup` button, the backup copy of the database will be located at the directory the user defined.

Export DB

To export the DB to a directory, the user will need to define a valid path directory in the text field. On clicking of `Export`, exported schema files can then be found in the directory specified. The schema files consists of *dbfs.schema*, *quartz.schema* and *system.schema*.

Compare DB

- Reference schema in current release: On clicking of `Compare`, *Database Schema Comparison Result* page will load. If there are any changes in the schema, the affected keys will be displayed in a table format. If not, a message *No changes in schema* would be displayed under the specific schema file name.

- Folder that contains expected DB schema: Click on the `Compare` with a valid path directory which contains the schema files in the text input field. If there are any differences in the schema, it will be displayed in a table format in *Database Schema Comparison Result* page. Else, *No changes in schema* will be displayed under the specific schema file name.

Scheduler

The status of the system scheduler is shown on this web page. In particular, you can view any jobs that are in progress. You can also review what triggers are loaded by the scheduler and when they last fired and will next fire. You can also force the scheduler to reload the triggers, though this is only necessary if the trigger files have been manually edited. If you have edited the trigger files using the Schedule Designer (which is included in the Remote Designer package) then the triggers will be automatically reloaded by the scheduler when you save them.

Users

Every user of Elixir Repertoire Server should have a unique logon name and password. This gives access to the Server via the web interface (through a browser), through Elixir Repertoire Remote and through the variety of client tools that Elixir provides. Use of specific filesystems and files is granted based on user name and/or group.

To create new user, click on the `Create User` at the bottom of the Elixir Repertoire Users list. User name and password are required in order to create a Repertoire user successfully. You should usually also assign the user to one or more groups. You can edit a user by clicking on the user name in the list. You can change the name, password, groups etc. If you leave the password field blank, then the password will not be altered. You cannot change a name that would result in duplicate names. You can change the name of the admin user to anything you like, and even then change another user to be called admin. However, internally all users and groups are identified by id, and user id = 1 is the admin user, regardless of the actual user name. For this reason, you can delete any user, except user id = 1.

Repertoire also supports anonymous login where no login authentication is required. This is discussed in the section called “Anonymous Login”.

Groups

Rather than assign access rights at the user level, it is often more convenient to combine users into groups and assign rights to the group. Each user can belong to any number of groups. Any user who is assigned to the admin group will have the ability to administrate the server. Just like with users, group names can be changed, so group id = 1 is the admin group, regardless of what the group is actually called. For this reason, you can't delete group id = 1.

There is another special internal group "*" which has id = 2. This group is used to represent all users. Every user created automatically belong to this group. When describing access rights, for example in filesystems or secure .access files (described in the next section) you can use an asterisk (*) rather than enumerate every single user and/or group. The "*" group cannot be viewed or edited directly as it is managed by the system.

FileSystems

A filesystem is a collections of files such as datasource, report templates and dashboards that users can view over the web. An administrator has the option to create, edit, compact and delete filesystems, and also to grant access to certain groups. Compact will clean up the directory by removing all the backup files.

To create a filesystem, select the `Create filesystem` link at the bottom of the Repertoire Filesystem list. Display Name and Configuration are required. Every filesystems on the Elixir Repertoire Server must have unique name. Administrator can assign the group(s) to have access to the filesystem.

There are five types of filesystem available: Dbfs, Jar, Jdbc, Local and Secure.

Note

Even if no privilege is being assigned to *admin* during the creation of a DBFS filesystem, *admin* will still have access to them.

Dbfs
: The Database filesystem (Dbfs) stores files in an SQL database within the server. The files in Dbfs support properties and you can manage user access to individual files or folders. The files and folders have three sets of permissions, for the owner, the owning group, and others. Each permission set may contain Read (r) and/or Write (w). When a file or folder is created it will inherit the access rights of the parent.

  The configuration field contains the name of the store to use as store=[name] i.e. store=local. The configuration can be left empty, in which case the default store is local. A Dbfs `/User` filesystem is automatically created when the server is first initialized and is given appropriate permissions for each user to store their personal files. You can create additional filesystems of type Dbfs as necessary.

Jar
: A Jar filesystem stores all files and folders in a Java archive format, that is typically a file with a .jar or .zip extension. Files in a jar filesystem are read-only, so this is a useful way of deploying reports and dashboards so that you are certain they cannot be modified. The configuration field for a Jar filetype should provide the full path to the archive file. Remember this file must be on the server, so it is a server path name you are providing.

Local
: A Local filesystem stores all files in a directory tree on the server. This is often the easiest filesystem to use for single-server solutions as the storage mechanism is completely transparent. You can use all your regular file-handling tools for manipulating the directory tree. The configuration field for a Local filesystem should provide the full path to the root of the directory tree. Remember this tree must be on the server, so it is a server path name

you are providing. Also, ensure you are running the server with a restricted set of permissions, (e.g. by creating a new user called repertoire and running the server from that user account) to prevent access to the entire server filesystem.

Secure
: A Secure filesystem is an extension to the local filesystem and used when the file access needs to be limited access to specific users and groups. A secure filesystem required a special file called .access to be placed inside each restricted directory of the filesystem, which allows the administrator to give different access privileges to named users and groups.

  A .access file has this format:

```
#This is a comment
read:finance,sales
write:elixir
write:admin
```

  Each file can have multiple read and write statements, one per line, in any order. Granting write access automatically grants read access. You can use either user names or group names and access is only allowed if explicitly granted. You can also use the special group "*" to allow access to everyone.

Jdbc
: The Jdbc filesystem is deprecated as it is replaced by Dbfs, which has additional access rights control. You can still access the files stored in any Jdbc filesystem but will no longer be able to create new filesystems of this type.

Targets

Target Constants

If you find yourself typing a string repetitively when configuring targets, you can define that string as a constant. Then you can refer to that constant in target property values like ${constant-name}. Target constants can be enabled/disabled. Only enabled constants can be used in target configurations and they will not appear in Repertoire Remote Designer. You can define multiple constants with the same name, but only one of them can be enabled at any time.

Target List

- Target Creation/Update/Deletion
  All manipulations can be done through the browser interface, under Administration->Targets. For most of the targets, configuration is simple. You need to provide a name, enable the target and provide values for required target properties.

- Status
  Targets can also be enabled/disabled. Only enabled targets can be used in RenderReport task. You can define multiple targets with the same name, but only one of them can be enabled. If you make one target enabled, the rest targets with the same name will be disabled automatically.

- Target Properties and Parameters
  Each target requires certain properties such as name of the report, mime-type of the report (A list of mime-types can be found through the browser interface under Administration->MIME Types), etc.

  When editing a target, the web page lists all required properties for the target. An administrator has several options when configuring those property values:

  - Provision of an exact string value for some properties.

  - Reference to target constants.

- Definition of parameters in some properties if they should be provided by end users. For example, the parameter in property "filename" can be defined as "${file#report}_${date}". End users should provide values for those parameters when invoking targets.

Create New Target

When you create a new target, you will notice that all the properties are filled automatically with parameters. If you don't change them, those parameters should be provided by end users. In addition, nested parameters are not supported. For example, you cannot specify a parameter like ${file##${reportname}}.

The following table, Table 6.2, “File Target Configurations”, are the individual targets configuration. Under "**Properties**", it shows the parameters entered through the web browser. The accompanying image, Figure 6.1, “File Target” is a screenshot of the Repertoire Remote Designer that shows the result of your configuration.

Chain Target

A Chain Target links a PDF Signer Target, together with other targets to form a series of events.

Table 6.1. Chain Target Configuration

Name	value
chained-source	${source##PdfSigner_Target}
chained-targets	${targets##PrintTarget, Mail_Target}
delay-closed	${delay-closed##true}
mime-type	${mime-type}
chained-mode-onerror	chained-onerror-fail

chained-onerror-fail discontinues the forwarding if source target contains error. *chained-onerror-continue* goes on with the forwarding regardless of the error in the source target.

File Target

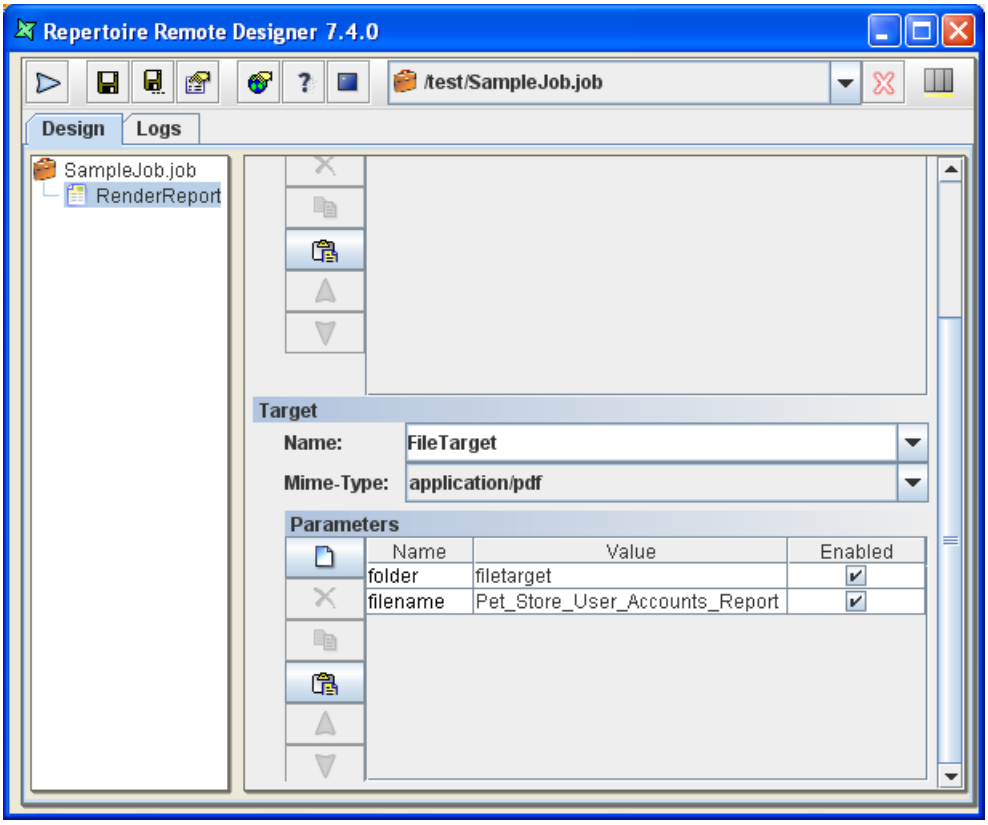
A File Target represents a directory on the server where reports can be stored. You can define as many file targets as you need.

Table 6.2. File Target Configurations

Name	Value
dir	${output.dir}/${folder##filetarget}
overwrite	true
filename	${filename##Pet_Store_User_Accounts_Report}
mime-type	${mime-type##application/pdf}

`${output.dir}` is the target constant created and it will not appear in the Repertoire Remote Designer.

Figure 6.1. File Target



JDBC Target

A JDBC Target allows reports to be written directly into a database. This is useful if you have some subsequent program to pick them up or otherwise act on them - for example a document management system. Each report is written as a record into a specific table in the database. The report data itself is stored as a BLOB. Before you can use the JDBC target, you need to set up a database with a table that has the correct schema to accept a report file. An example as shown:

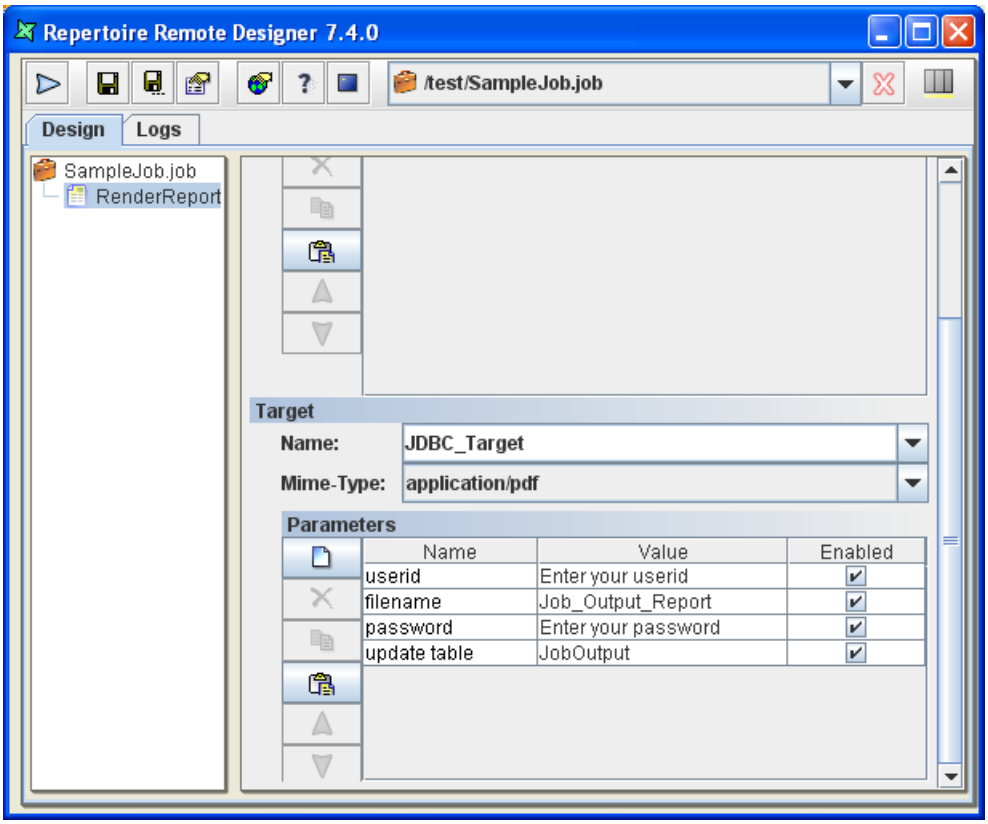
```
CREATE TABLE JOBOUTPUT (
  id INTEGER NOT NULL GENERATED ALWAYS AS
   IDENTITY (START WITH 1, INCREMENT BY 1),
  name VARCHAR(256) NOT NULL,
  lastModified BIGINT NOT NULL,
  content BLOB NOT NULL,
  CONSTRAINT JOBOUTPUT_PK PRIMARY KEY(id) )
```

Once the database is setup, Figure 6.2, “JDBC Target” shows an example of the configuration through the browser interface that will write into a table called JobOutput in the Derby database that is built into the Elixir Repertoire Server:

Table 6.3. JDBC Target Configurations

Name	Value
driver	org.apache.derby.jdbc.EmbeddedDriver
url	jdbc:derby:/home/ers/jdbctarget
table	${update table##JobOutput}
user	${userid##Enter your userid}
password	${password##Enter your password}
name	${filename##Job_Output_Report}
overwrite	true
mime-type	${mime-type##application/pdf}

Figure 6.2. JDBC Target



- *user* The user name to use when logging on to the database.
- *password* The password to use when logging on to the database.

JMS Target

A JMS Target can be used for asynchronous messaging. JMS applications can use job messages as a form of managed request/response processing, to give remote feedback to the users on the outcome of their send operations and the fate of their messages. Examples of job messages are Exception, Expiration, Confirm on arrival (COA), Confirm on delivery (COD), etc.

You need to provide settings of the JNDI server. According to JMS specification, all JMS connection factory and destinations are hosted on a JNDI server and clients need to know how to connect to the server before it can connect to the JMS broker. You need to configure setting in ERS2.xml as you need to trigger job execution by sending messages to JMS brokers. Uncomment the block of codes for

`mbean name="ERS2:name=JMSTrigger"` and edit the values that fit your JNDI provider and JMS broker configuration.

For most of the JNDI servers, you need to provide property values like:

- java.naming.provider.url, which is the URL for JNDI server.
- java.naming.factory.initial, which is the context factory implementation provided by the JNDI server.

Different JNDI servers may also require some customized property values. You need to check JNDI servers documentation to find out what properties they support.

Configuration also needs to be done through the browser interface.

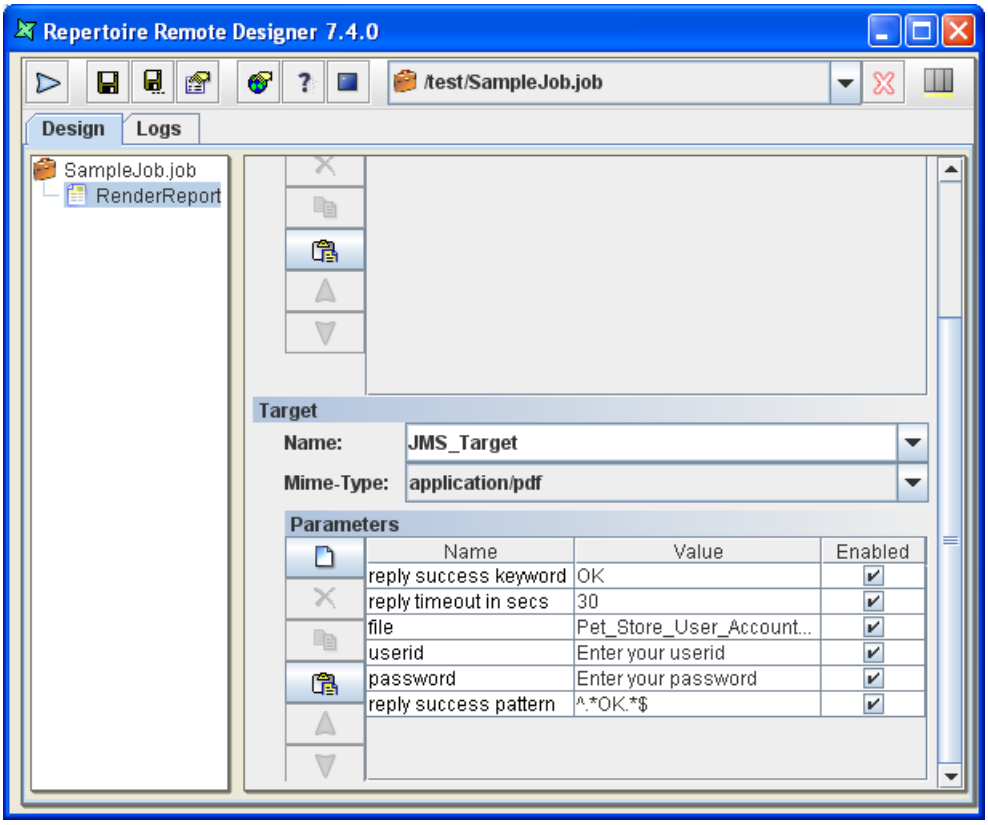
Table 6.4. JMS Target Configurations

Name	Value
jms.destination	RQueue
jms.connection.factory	ConnectionFactory
jms.user	${userid##Enter your userid}
jms.password	${password##Enter your password}
jms.reply.required	true
jms.reply.timeout.seconds	${reply timeout in secs##30}
jms.reply.success.keyword	${reply success keyword##OK}
jms.reply.success.pattern	${reply success pattern##^.*OK.*$}
filename	${filename##Pet_Store_User_Accounts_Report}
mime-type	${mime-type##application/pdf}

Table 6.5. JNDI Properties

Name	Value
queue.name	RQueue
java.naming.factory.initial	org.apache.activemq.jndi.ActiveMQInitialContextFactory
java.naming.provider.url	tcp://localhost:61616

Figure 6.3. JMS Target



Mail Target

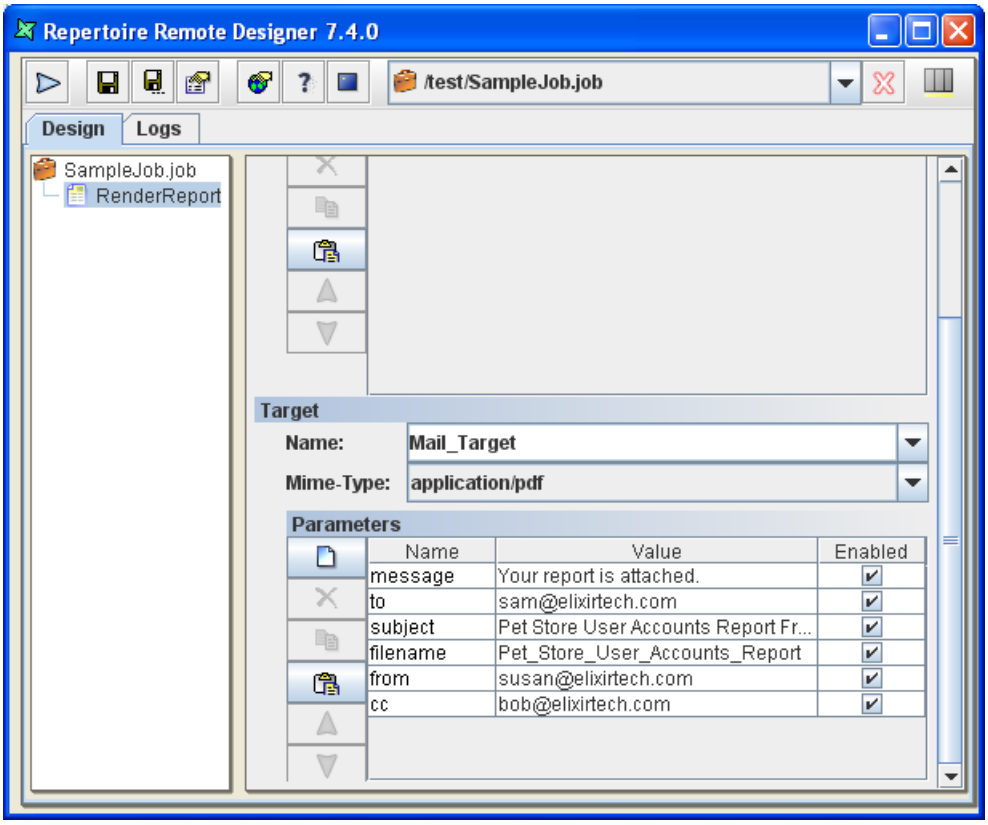
A Mail Target allows the output to be sent by email. Before using this, ensure that the appropriate SMTP Server is configured in ERS2.xml. The default SMTP server elixir.aspirin is built in to Elixir Repertoire Server, so you only need to change it if you wish to use an external SMTP Server. The smtp.host parameter can either be the name of the SMTP Server mbean (we used `ERS2:name=GmailSMTPServer` when we discussed this in the section called “SMTP Server”) or the matching smtp.host value from the mbean that reference to other SMTP Server of your choice.

There are a number of parameters to specify, but remember that you can use substitutions to avoid hard-coding those that you decide need to be flexible. The report will be sent as an attachment by email, so you can choose the render format you prefer.

Table 6.6. Mail Target Configurations

Name	Value
smtp.host	elixir.aspirin
from	${from##susan@elixirtech.com}
to	${to##sam@elixirtech.com}
cc	${cc##bob@elixirtech.com}
subject	${subject##Pet Store User Accounts Report From Elixir Server}
message	${message##Your report is attached.}
filename	${filename##Pet_Store_User_Accounts_Report}
mime-type	${mime-type##application/pdf}}

Figure 6.4. Mail Target



PDF Signer Target

PDF Signer Target is used when a PDF output needs to be signed digitally. It prints a "signature" in a PDF file when the file is rendered.

Table 6.7. PDF Signer Target Configuration

Name	Value
signer-appearance	${signer-appearance##self-signed}
keystore-type	${keystore-type}
keystore-url	${key.url##C:/Java/jre6/bin}
key-alias	${key-alias}
key-password	${key-password##keyStorePassword}
key-password-encrypted	true
sign-page	sign-last-page
sign-position	${signature-position##bottom-right}
sign-width	${sign-width##100}
sign-height	${sign-height##60}
signer-image-url	${signature-image-url##D:/image/logo.png}
output-mode	${output-mode##output-mode-file}
output-file-location	${output-file-location##../output/folder1/signed-pdf-document.pdf}
info-reason	${info-reason##Reason for signature}
info-location	${info-location##bottom-left}
certificate-level	${certificate-level##certificate-no-changes}

PDF Signer Target Properties

signer-appearance

There are two types of signing mode. They are *self-sign* and *wincer-sign*.

self-sign signer can be generated using Java key generator (keytool.exe). An example of a signature command is as follows :

```
keytool -genkey -keyalg RSA -alias QAkey -keypass mypassword
  -keystore keystore.ks -dname "cn=My Key Name, c=SG"
```

wincer-sign is recommended if higher security is required. Certificate obtained is installed to the web browser and needs to be exported to be used in PDF signing.

keystore-type

For *self-sign*, no keystore type is required. For *wincer-sign*, the keystore type will depend on the respective vendor. For example, a VeriSign keystore type will be `pks12`.

keystore-url

The directory of the keystore is entered here. *repository* is prohibited for this parameter.

key-alias

This parameter is optional. It is to define the alias used by the keystore file.

key-password

key-password is the password entered when creating the keystore file. The password can either be in encrypted or unencrypted form.

key-password-encrypted

This parameter holds a boolean value, `true` if the key password is encrypted and `false` if otherwise.

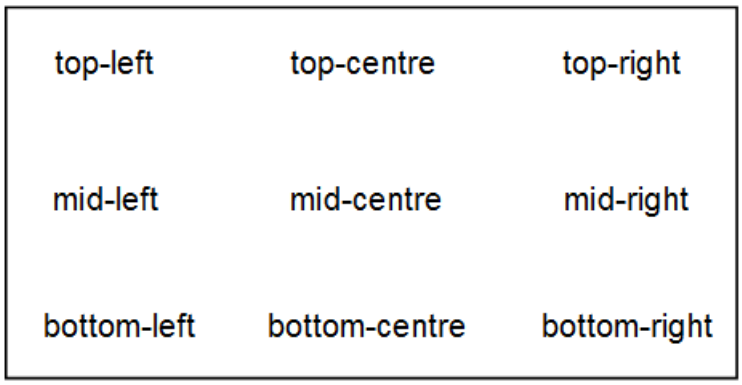
sign-page

This parameter is to specify the page that the signature is placed. *sign-no-page* will mean that the signature will not be visible. *sign-first-page* and *sign-last-page* implies that the signature will be placed on the first and last page of the PDF file respectively. If you wish to place the signature on a specific page, enter the page number as the value for the parameter. For example, *${sign-page##5}*, for placing the signature on the 5th page of the PDF document.

sign-position

The signature can be placed in different location of a document. Figure 6.5, “sign-position Options” displays the options available for placing a signature.

Figure 6.5. sign-position Options



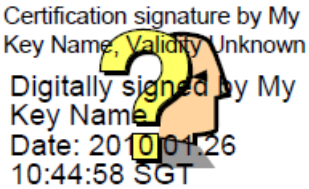
sign-width And sign-height

These two parameters are for user to specify the height and width of the signature rectangle size.

signer-image-url

The directory of the image that is to be used with the signature. When no value is entered for this parameter, the signature will appear like the in Figure 6.6, “PDF Signer - Sample Signature (self-signed)”.

Figure 6.6. PDF Signer - Sample Signature (self-signed)



output-mode

There are three output mode options.

1. *output-mode-file* drops the signed PDF file in the directory specified in `output-file-location`.

2. *output-mode-streamed-memory* keeps the signed PDF file in memory and is to be used with Chain Target.

3. *output-mode-streamed-cached* keeps the signed PDF document in a temporary file and it helps to keep the memory small when processing large files.

output-file-location

The directory to place the PDF output is entered here.

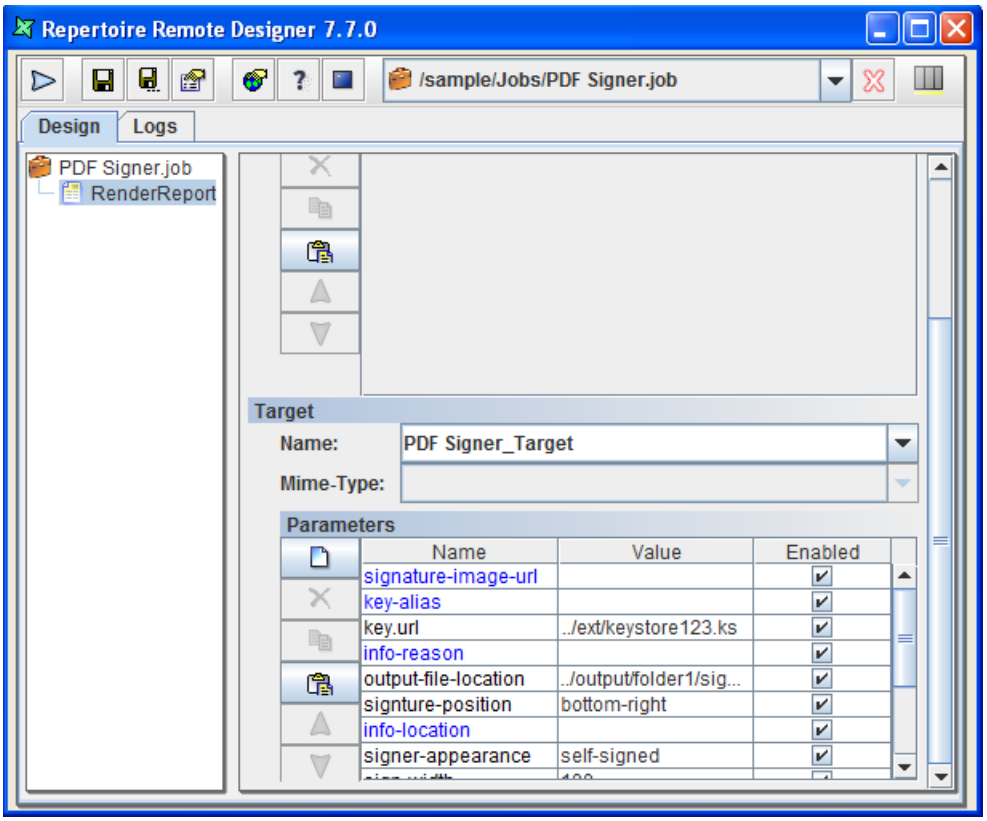
info-reason And info-location

The reason for placing a signature in the PDF document and the position of the reason.

certificate-level

certificate-not	has no visible signature in the PDF document. The document is digitally counter signed.
certificate-no-changes	displays the signature in the PDF document and no changes can be made to the document.
certificate-form-filling	also displays the signature in the PDF document, but only the filling in of forms, signing and page adding is allowed for the document.
certificate-form-filling-annotation	displays the signature in the PDF document, but only commenting, form fill-in, signing and page adding is allowed.

Figure 6.7. PDF Signer Target



Print Target

A Print Target allows you to send a report to a named printer. The only option is the name of the printer. If you have multiple alternate printers, you could use a separate target for each printer so that you could control access by different groups. In most cases, the printer names will be fixed and not

include substitutions. You can also leave the printer name blank as it will route automatically to the default printer defined on the server.

Table 6.8. Print Target Configurations

Name	Value
printer-name	Canon iR C3220 PCL5c

Repository Target

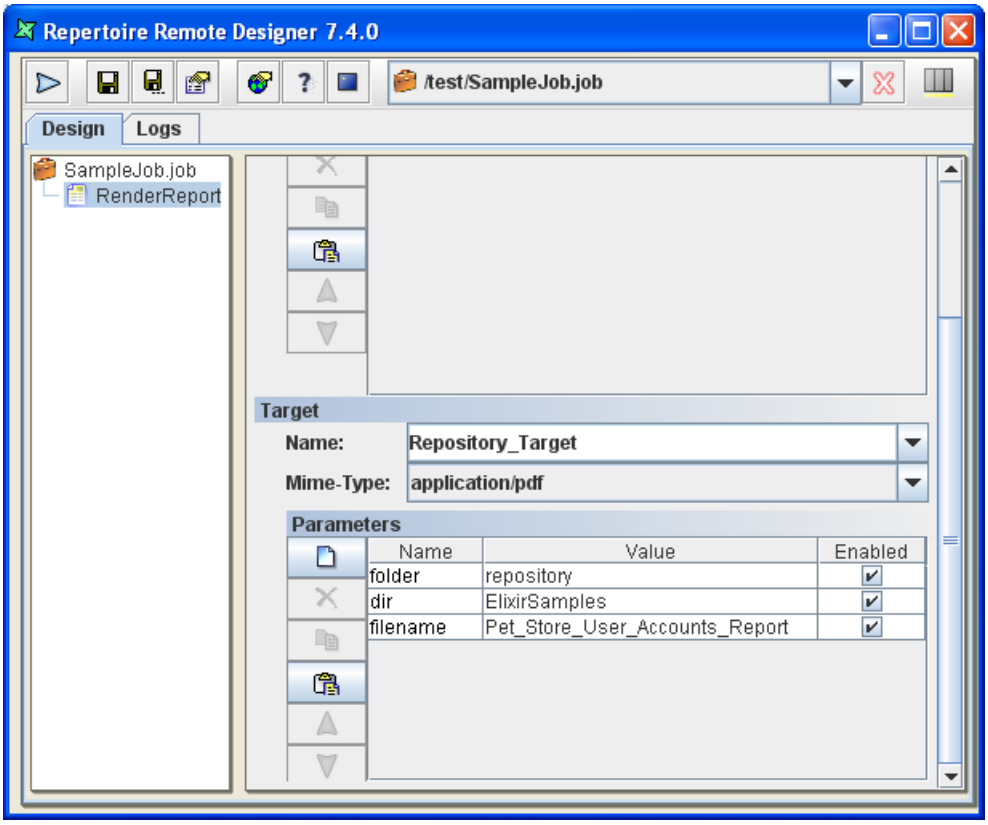
A Repository Target writes the report to the filesystems in the Repertoire Remote Designer. You can identify a target folder in the repository and provided it is writable, files will be written there. This works regardless of whether the target filesystem is of type local, secure or dbfs. You should use Repository Targets when you want to allow users to view the reports through their browser as the repository will automatically update to show the latest files. If you use a File Target where the report is written to a target filesystem not in the Repertoire Remote Designer, users will not be able to access the report output using the Repertoire interface.

During the configuration, the "folder" property must refer the "dir##" parameter to an existing target filesystem in the repository. The "folder##" parameter need not as it will create a new folder after its name if it is not found in the repository.

Table 6.9. Repository Target Configurations

Name	Value
folder	${dir##ElixirSamples}/${folder##repository}
filename	${filename##Pet_Store_User_Accounts_Report}
overwrite	true
mime-type	${mime-type##application/pdf}

Figure 6.8. Repository Target



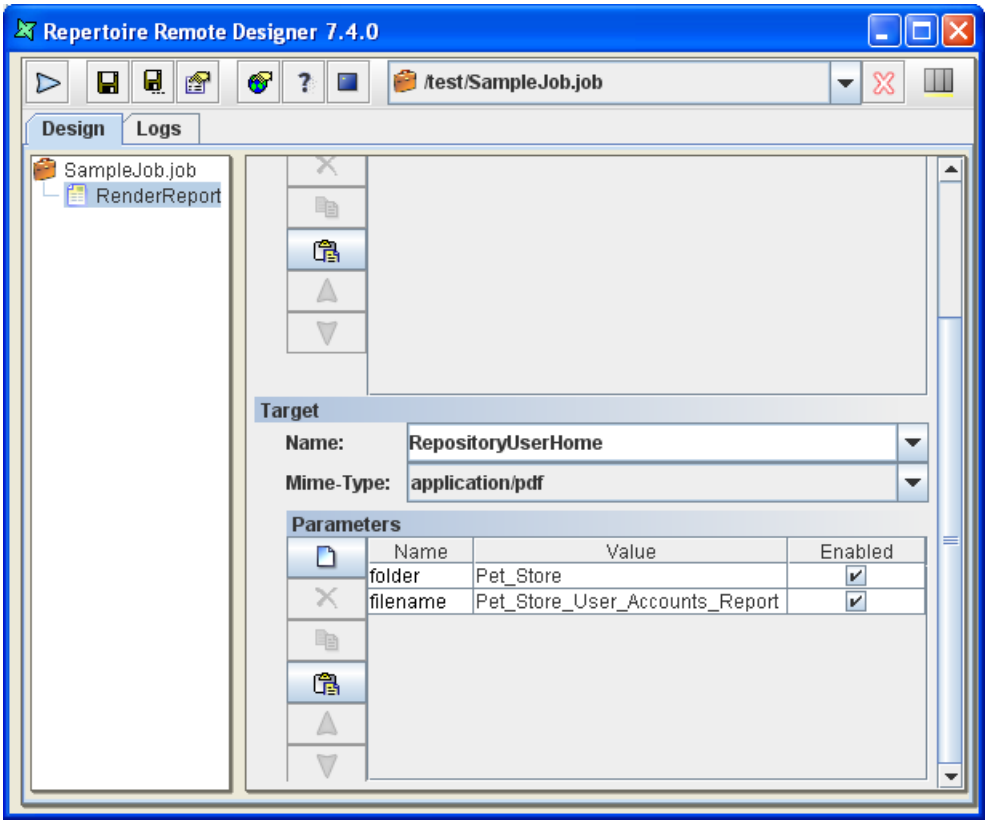
Repository User Home Target

A Repository User Home Target writes the report to the invoking user's folder in the repository. Users can share jobs and they can keep separate output without overwriting one another. In the example, once the user "Jon" signs in to the repository and runs the job, the report will be written to /User/Jon/Pet_Store.

Table 6.10. Repository User Home Target Configurations

Name	Value
folder	${folder##Pet_Store}
filename	${filename##Pet_Store_User_Accounts_Report}
overwrite	true
mime-type	${mime-type##application/pdf}

Figure 6.9. Repository User Home Target



SFTP Target

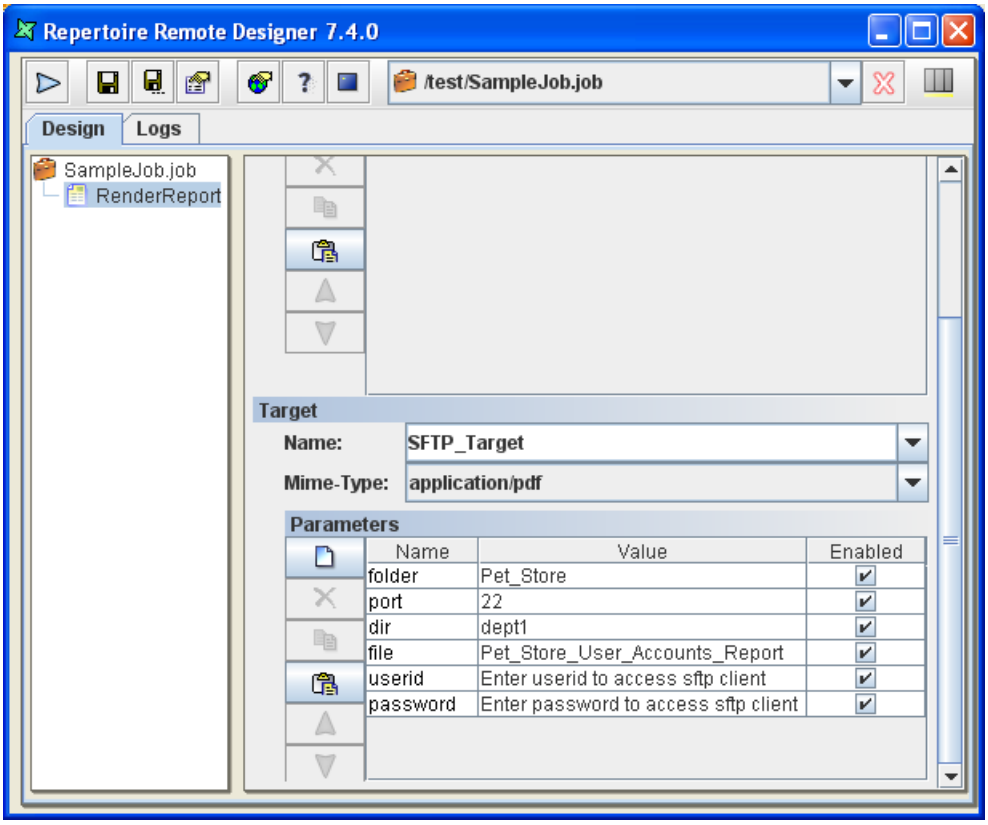
A SFTP Target allows the report to be transferred to a user's secured FTP Server. The available parameters are user, password, host, port, dir and filename. The port is optional and will default to 22, which is the default SFTP port, if not specified.

The parameter in "dir" property must be an existing directory found in the target ftp server.

Table 6.11. SFTP Target Configurations

Name	Value
host	${sftp host##domain_name.com}
port	${port##22}
user	${userid##Enter userid to access sftp client}
password	${password##Enter password to access sftp client}
dir	${dir##dept1}/${folder##Pet_Store}
filename	${filename##Pet_Store_User_Accounts_Report}
mime-type	${mime-type##application/pdf}

Figure 6.10. SFTP Target



Socket Target

A Socket Target sends the report to a program which is listening, typically on another machine. For example, a program can be written that listens on a company.com port 6000 and writes any data it receives to a database, or to a fax etc. It is up to the receiving program what it does with the data. The server opens a connection to the listening program, using the host and port information required by the socket target and streams the data across the network to the listening socket.

Table 6.12. Socket Target Configurations

Name	Value
host	company.com
port	6000

Split Target

A Split Target allows you to split a single report into different sets of pages (sometimes called "Report Bursting"), you can configure a split target to describe how the report will be split and which pages will be forwarded to which targets.

You need to set up the report in the repository first, in order to run the job. An example as shown :

- Create a report template with four Sections of different contents.
- Make sure the Render Sequence table includes all four Sections.
- Go to the Section Header of each Section and choose Table of Contents from the property sheet.

- For Section 1, enter a literal string: first. Section 2 as second. Same applies for the third and fourth section. Make sure the TOC Enabled checkbox is ticked.

Please note that the TOC doesn't need to be four different sections, it could be four groupings of data (group header-details-group footer) in a single section itself.

Once the report is setup, proceed with the configuration from the browser interface.

When the "split" property is indicated as "1", it means to break at the top level splits. You might choose to break at any tree depth.

For each split condition, you need to configure a patten match string that matches certain title in table of contents. When a match happens, the matching portion of the report is sent to the targets configured in the split condition.

You can provide parameter values for the targets referred in split conditions. For example, you may want to send a portion of a report to "printer" target and the target required one parameter "file". So you can provide value for the "file" parameter as "${0}.glint". Note that the string contains another parameter ${0}, which is provided by Repertoire automatically when the split condition matches title of a report. (If you are familiar with regular expression, the parameter ${0}, ${1}, etc..actually refers to groups in the result of string pattern matching.)

Although the UI allows you to add more than one targets to a split condition, the matching report is only sent to the first target if the condition matches.

Table 6.13. Split Target Configurations

Name	Value
split	1

Figure 6.11. Split Conditions

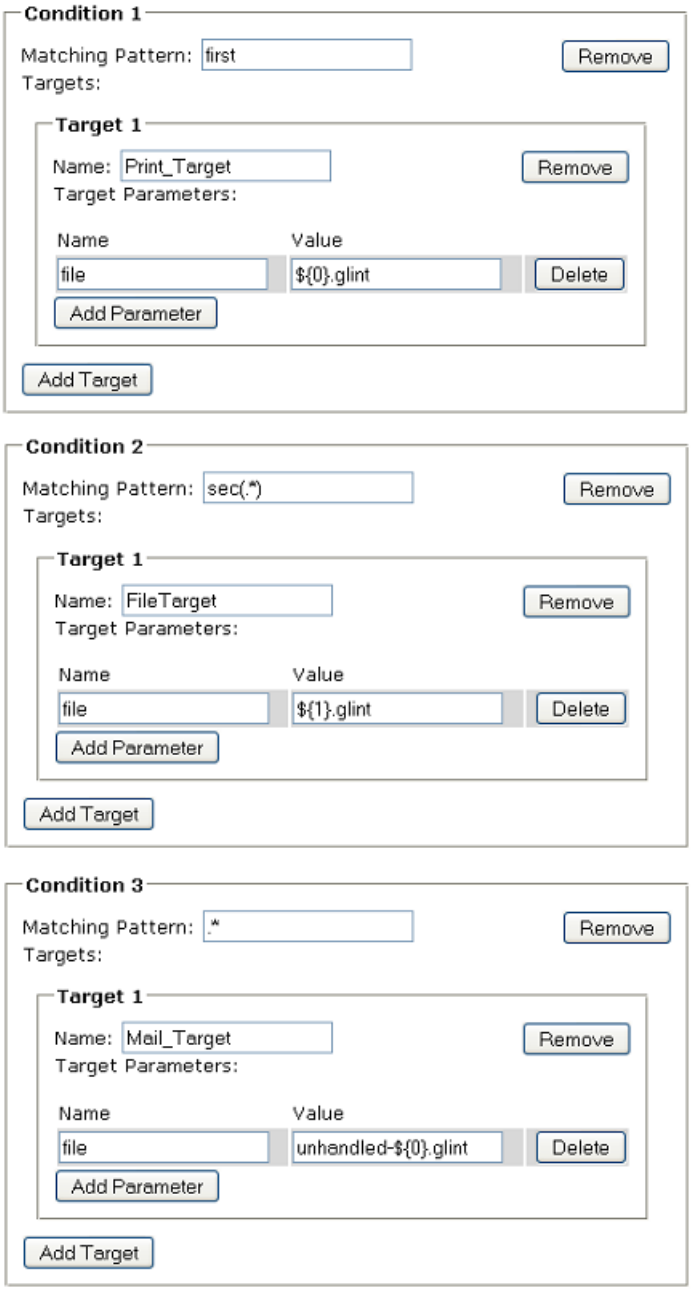
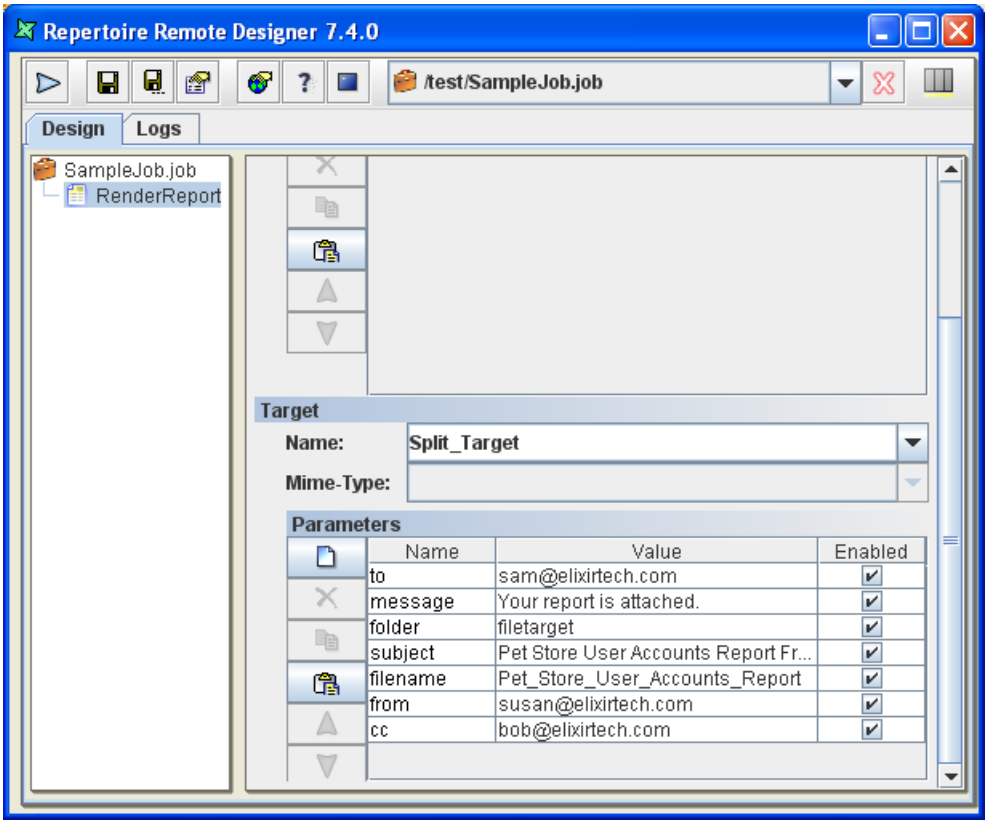


Figure 6.12. Split Target



Target Access Control

- User Access Control
  There is access control support on targets. For each user/group, administrators can specify which target he can access. For example, if a job is run under privilege of user "Jon", the reports rendered in the job can only be sent to targets that Jon can access. Note that there is a wild card "*" that represents any user/group/target. So you can specify access control rules for all users easily.

Import Target Configuration

If you upgrade Repertoire server from a previous release, you need to manually import your target configurations from a ERS2-Config.xml file. You can do this task through the browser interface, under Administration->Targets ->Import target configuration.

During the import process, there are some migration work for each target to make them backward compatible with current job configuration.

For example, a FileTarget requires "filename" property, and that property may be provided in previous releases. To make this work after the target is imported to database, the parameter for "filename" property needs to be changed to ${filename##default-file-name}. So if a parameter for "filename" is provided in RenderReport panel, that parameter will be used. Otherwise, the default parameter in "filename" property will be used.

MIME Types

On clicking of `MIME Types`, a list of mime types for the Remote Designer will be listed. 2 options are available. They are `Disable` and `Delete`. By disabling the mime type, that particular option will not be seen in the Render Wizard of the Remote Designer, therefore the user will be unable to render in that particular mime type. To get the option back, simply enable the option.

If a mime type is deleted, the option will also be removed from the list in Render Wizard. However, in order to get it back, the user will need to add the mime type back and that can be done at the bottom of the screen.

Search

The configuration of the search can be found in the Search administration page. You can enable or disable search service to the users. You can manually trigger or schedule the building of search index from the administration page.

You can do a full search index or by incremental build. An incremental build is where only a modified file is indexed. Increment index may decrease the time needed to index the repository. Schedule of indexing is recommended to reduce frequency of rebuild index. A recommended approach is you can do a full index once a day and trigger incremental index every one or two hours.

The index scheduler uses cron to control when the job is triggered, example 0 0 0 * * ? fires job every midnight. You should avoid triggering index job during peak hours or decrease the frequency during that time. For more information on the cron syntax see

```
http://www.quartz-scheduler.org/documentation/quartz-1.x/
tutorials/crontrigger
```

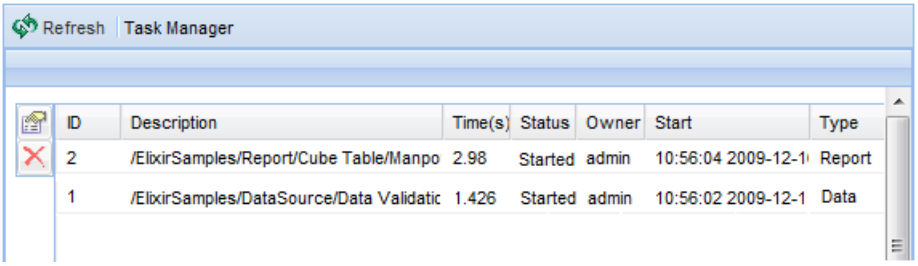
Data

This is for enforcing security on datasource(s). More information can be found in the section called “Datasource Security”.

Task

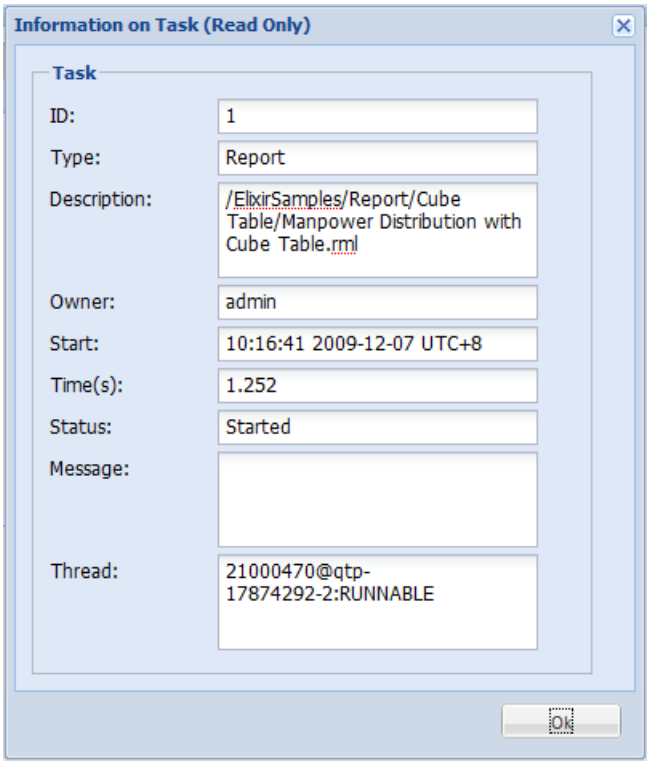
Task Manager can be accessed from `Administration` then `Task`. Only ongoing tasks would be reflected in the table, as seen in Figure 6.13, “Task Manager”.

Figure 6.13. Task Manager



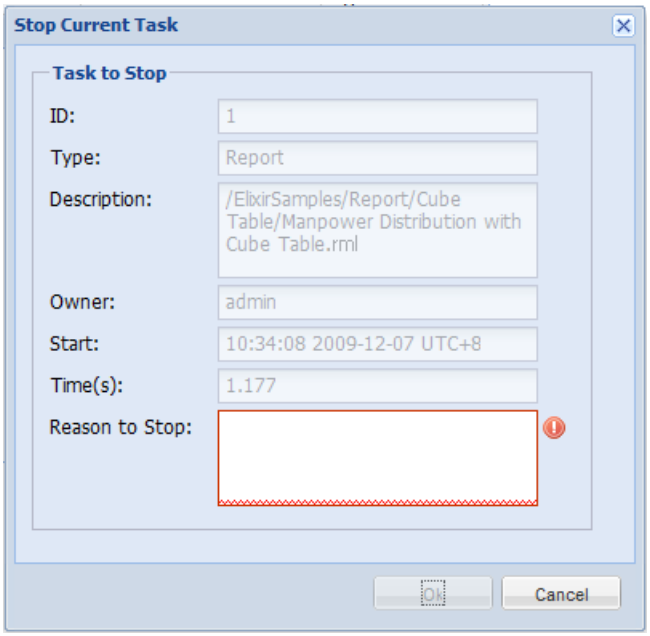
Clicking on a task in the Task Manager then on the `View Information` icon will open a window like in Figure 6.14, “Task Information”. No editing can be done.

Figure 6.14. Task Information



Clicking on a task, then the `Stop` icon, will open a window like in Figure 6.15, “Stop Task”. A reason needs to be specified before the job can be stopped.

Figure 6.15. Stop Task



Completed, failed, cancelled and timeout task(s) will be removed from the table. A new log file, *task.log* will be added to `Logs`, where the log files are placed. This log file keeps track of all activities appearing under Task Manager.

Note

Task Manager does not refresh the page automatically. A manual refresh is required.

Logs

All logs from the server log directory are listed and can be inspected through the browser interface. The primary log is the activity log, which lists all session activities (logon and logoff) as well as generating and rendering actions. More detail is provided in the server log; this provides a fine-grain view of the system, which is primarily for debugging. The jetty log file lists all file accesses.

Help

The Help menu option provides links to all Elixir Repertoire documentation (including this one). Documentation shown over the web is in HTML format, but you will also find the PDF version of this document in the server docs directory. PDF versions of the designer manuals are included with the Elixir Repertoire Designer release.

Logout

User session is terminated when Logout is selected and the browser returns to the logon page.

Case Study 1 - Customized Parameter Entry Page

In this case study, we will be showing the user the steps to execute in order to have a customized HTML page for user to enter the parameter(s) when rendering a report.

1. Create a report template that requires the use of parameter(s). It is not necessary that the parameter(s) require the use of user input. In this case study, we will choose to have the user to enter the values for the parameters. This will show that the values entered will overwrite the default values defined in the report template.

2. In /ElixirSamples/Report/Components/Parameters.rml, the default values defined can be seen in the Report tab, Parameters panel. We will be using this report template in this case study.

3. Create a HTML file with the same name as the report template and place it in the same folder where the report template is saved. In this case, the HTML page will be named as *Parameters.rml.html*.

4. In the HTML file, enter the following:

```
<html>
  <head><title>Custom Parameter Entry Page</title></head>
  <body>
    <h1>Custom Parameter Entry Page</h1>
    <form method="get" action="/report/ElixirSamples/Report/
      Components/Parameters.rml?mode=parameters">
      <table cellspacing="2" cellpadding="2" border="0">
        <tr>
          <td align="right">Date</td>
          <td><input type="text" name="Date"></td>
        </tr>
        <tr>
          <td align="right">Password</td>
```

```
          <td><input type="password" name="Password"/></td>
        </tr>
        <tr>
          <td align="right">Text</td>
          <td><input type="text" name="Text"></td>
        </tr>
        <tr>
          <td align="right">Choice</td>
          <td><input type="text" name="Choice"></td>
        </tr>
        <tr>
          <td align="right"></td>
          <td><input type="submit" value="Submit"></td>
          <input type="hidden" name="mime-type"
            value="application/pdf"/>
        </tr>
      </table>
    </form>
  </body>
</html>
```

5. This will allow the user to enter the parameters. If the user do not want the end-user to enter text for a particular parameter, the user can edit accordingly the HTML file to do that. In this case study, we will be rendering the report into PDF. This can be changed as well.

6. Go to the Web Interface. Under `Administration` then `FileSystems`. From there, refresh the filesystem.

7. When done, go to `Repertoire`, the filesystems available will be listed. In /ElixirSamples/Report/Components, find *Parameter.rml*. When clicked, end-user will be led to the HTML page created as seen in Figure 6.17, “Parameters Entry Page Customized”. If the parameters entry is not customized, it will look similar to Figure 6.16, “Parameters Entry Page Not Customized”.

Figure 6.16. Parameters Entry Page Not Customized

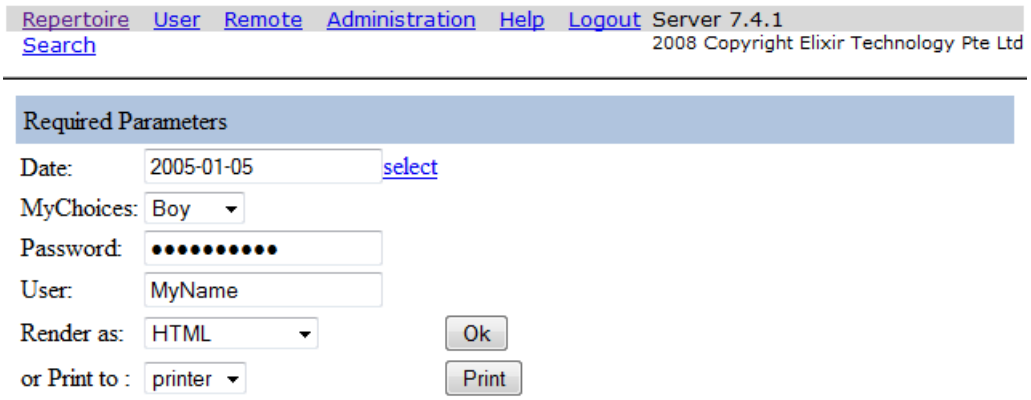
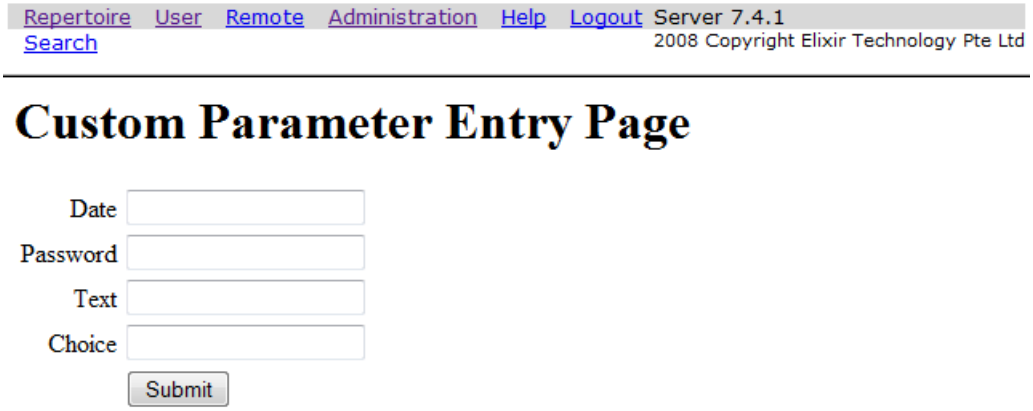


Figure 6.17. Parameters Entry Page Customized



8. Enter values for the respective parameters. Click on the `Submit` button. At the bottom of the rendered PDF, the values for the respective fields will be displayed according to the text entered by the end-user.

9. If user wishes to use JScript script file or CSS to design the HTML page, simply add the following lines to the HTML page. The JScript script file and CSS file will have to be in the same folder as the report template.

```
<script language="JavaScript" src="JScriptFile.js"
       type="text/javascript"></script>
<link type="text/css" rel="stylesheet" href="CSSFile.css" />
```

Case Study 2 - Customize Folder Listing

In this case study, we will demonstrate the way to have a HTML page before listing the files in the filesystem. This can be used to display some reminders or notes, before the user enter into the folder.

1. For example, we have a folder named *testing*. In the folder, there are some report templates, datasources, etc.

2. In the folder, create a HTML file and name it *folder.html* with the following:

```
See <a href="?folder.html=ignore">Show Files Under
  testing Folder</a>
```

3. When a user clicks into the folder, the user will not see the usual listing of files. Instead, the user will see the HTML page created.

4. On clicking `Show Files Under testing Folder`, the user will see the page listing all files in the folder.

Chapter 7
Elixir Repertoire Server Client

Introduction

Elixir Repertoire server provides a set of Java standalone client APIs for connection to the server. This API allows you to trigger all the report-related or ETL functionalities. Access to the functionality is controlled by the Server license. The report-related functionality allows listing of reports deployed on the server repository, extracting dynamic parameters in the report and generating reports. The ETL functionality allow you to access data generation and DataStore functions. This client library may be deployed as part of a J2EE solution. The detailed version of the Java API documentation is shipped with the server.

The alternative to the Java Client API is the Server API described in Chapter 8, *Server API*. This is a much more powerful and complete API, that is independent of programming languages, but requires more programming ability and doesn't provide the same high level of abstraction. Therefore, you need to choose between the simplicity of the Java Client API and the power of the Server API. If you are using Java and can accomplish your tasks within the simple API provided, stick to the Client API described in this chapter, otherwise continue on to the next chapter where the Server API is explored.

Java Standalone Clients

The ERSClient is a Java class that provides the core API for all reporting-related functions. It is used with Java applications like servlets, JSP or even a simple standalone Java client. The ERSClient-Command is a wrapper class of ERSClient, this provides script based integration for command line invocation of reports.

The supporting Java libraries are placed in the RepertoireServer\clients\lib directory. RepertoireClient.jar is the core library containing the API. The other support libraries are log4j for logging mechanism and Glint.jar for ui components.

Elixir Repertoire Server Client

Elixir Repertoire Server Client provides a light weight interface to connect to the Repertoire Server. The basic features are listing file system, querying reports deployed in the report server repository, querying dynamics parameters in the report and generating report.

Using the APIs

The full list of the APIs for RepertoireClient is found in RepertoireServer\web-resources\help\api The basic steps to make use of the functions are as follows:

- Determine the Report server IP or host name, Port number, user name and password.
- Create a new instance of com.elixirtech.ers2.client.ERSClient class. If the server has been configured in secure mode, setSecure(true) must be called on the ERSClient instance. Failure to set this to match the server mode will result in encryption/decryption errors.
- Select the API to use to generate report, list filesystem etc. These basic functions are:

```
// retrieve the list of file systems in the repository.
String[] getFileSystems();

// retrieve a file system from the repository.
IFileSystem getFileSystem(String filesystemName);

// retrieve the list of the reports in a filesystem.
String[] getReports(String filesystemName);

// retrieve the list of the parameters in a report.
Parameter[] getParameters(String report);

// generate a report into an output stream.
IJobInfo renderReport(String report, String mimeType,
      OutputStream os, Properties properties);
```

Code example

Below are some code examples to interface with the report server from a java client.

Example 7.1. Listing the file systems in a server repository.

File systems are used for storage. They could be file directories or jar files the implementation is transparent to the user. The getFileSystems API will list the file systems.

```
import com.elixirtech.report2.runtime.IFileSystem;
public void listReports( )
{
    String[] filesystem = client.getFileSystems();
}
```

Example 7.2. Listing the reports deployed in a file system.

Once you have obtained the file system name (String), you can list the report names in that file system.

```
public void listReports( )
{
    ERSClient client = new
        ERSClient(localhost, 8080, username, MyPassword);

    String[]  reports = client.getReports("myfilesystemname");
}
```

Example 7.3. Generating a report

This illustrates report generation where "outputstream" is the java io stream that the report output will be written to. Job Information can be retrieve via the interface `IJobInfo`.

```
import java.io.OutputStream;
import com.elixirtech.ers2.client.ERSClient;
import com.elixirtech.report2.runtime.IJobInfo;
...
public void generateReport( OutputStream outputstream)
{
    ERSClient client = new
      ERSClient(localhost,8080, username, MyPassword);

    IJobInfo job =  client.renderReport(
      "/myrepository/myreport.rml",
      "application/pdf",
      outputstream,
      properties);
}
```

Example 7.4. Request for Data listing.

This illustrates how you can query for the data source for a list of records. The output will be written on the stream. Depending on the mime type, the format of returning dataset may in the format of excel (application/vnd.ms-excel), comma separated file (text/csv) or default data source xml format (text/xml).

```
public void generateDataSource()
{
 String dsFS = "/ElixirSamples/DataSource/ChartData.ds";
 File f = new File("ChartData.ds");
 if (f.exists())
  f.delete();
 FileOutputStream fos = null;
 try
 {
  fos = new FileOutputStream(f);
  Properties properties = new Properties();
  //mime-type format needed to return the data
  properties.put("mime-type", "text/xml");
  m_ERSClient.generateData(dsFS, fos,properties);
  if (fos != null)
   try
   {
    fos.close();
   } catch (IOException e)
   {
   }
  } catch (Exception ex)
  {
   System.err.println("Error: " + ex.toString());
  }
 }
```

Example 7.5. Trigger for Data Store process.

Data Store allows the user to trigger a section of actions to process data and finally load the final dataset to specific location i.e. Database table, file etc. The API ,generateData, is used but add parameter key "datastore" and the datastore name is needed to identify which data store to activate.

```
public void generate_DataStore()
{
 String dstoreFS="/ElixirSamples/DataSource/CompositeEmployee.ds";
 try
 {
  Properties properties = new Properties();
  //datastore property requirement to identify which
  // data store to push the data to.
     properties.put("datastore", "CSV");
  m_ERSClient.generateData(dstoreFS, outputstream, properties);
 }
 catch (Exception ex)
 {
  // ...
 }
}
```

Example 7.6. Using IJobInfo interface to extract job information.

When rendering reports or generating data, the methods return an IJobInfo interface. You can use this interface to extract information about a particular job. An example is as listed below:

```
import com.elixirtech.report2.runtime.IJobInfo;

...

private static double diffSecs(double start, double end)
{
    return (end-start)/1000d;
}

public void displayReportGenerateJobInfo(IJobInfo myJob)
{
    SimpleDateFormat df = new SimpleDateFormat("yyyyMMdd-Hmm");
    m_DateFormat = df.format(new Date());

    long timeReceived = job.getLong(IJobInfo.JOB_RECEIVED);
    long timeStarted = job.getLong(IJobInfo.JOB_STARTED);
    long timeEnded = job.getLong(IJobInfo.JOB_ENDED);

    double totalQueueTime = diffSecs(timeReceived,timeStarted);
    double totalJobProcessTime = diffSecs(timeStarted,timeEnded);
    double totalJobTime = diffSecs(timeReceived,timeEnded);

    int pageCount = job.getInteger(IJobInfo.PAGE_COUNT);
    int recordCount = job.getInteger(IJobInfo.RECORD_COUNT);
    String mimeType = job.getString(IJobInfo.MIME_TYPE);
    long sizeBytes = job.getLong(IJobInfo.BYTE_SIZE);

    // display or log these values as you choose
}
```

Example 7.7. Configuring custom status code and exception message

In this example, the user will receive a custom message, which will display in third-party API software.

```
public void rendersReport(String reportName, Properties prop)
{

  ERSClient ersClient = new ERSClient(localhost, 8080, username,
  MyPassword);

  try {
   FileOutputStream fos = new FileOutputStream(
    new File("sample.pdf"));

   IJobInfo ji = ersClient.renderReport(reportName,
   "application/pdf", fos, prop);

   System.out.println(

        "Job Received: "
       ......
       + ji.getString(IJobInfo.NAME) + ", "
       +  "Status Code: "
       + ji.getString(IJobInfo.STATUS_CODE) + ", "
       +  "Exception Message: "
       + ji.getString(IJobInfo.JOB_EXCEPTION)


   );


  } catch (FileNotFoundException e) {
   e.printStackTrace();
  } catch (NetException e) {
   e.printStackTrace();
  } finally {
   ersClient.close();
  }
}

public static void main(String[] args) {
  System.out.println(">>>1 Generate Report using File Output
  Stream");
  testClient.rendersReport("/Path/sample.rml",params)
}
```

Java Client usage examples

In the RepertoireServer\clients\demo\JavaClientApp directory, there a Java Client demo with source code provided to show the how a Java Swing Client view can be built with the Server Client's API. The full demonstration of a web based report application is now available from our web site.

Non-Java Client Connection Library

Non-Java Clients should use the Server API, which is described in Chapter 8, *Server API*. The Server API is also applicable to Java users who want more comprehensive functionality.

Elixir Repertoire Server Command Client

Elixir Repertoire Server Command client allows you to integrate report generation with a script based command line invocation. This can be use with a system scheduler like a cron job as a batch script.

Here is an example of the script,

```
java
  -classpath RepertoireClient.jar
  com.elixirtech.ers2.client.ERSClientCommand --host localhost
  --port 8080 --report "/Reports/myreporttemplate.rml" --mimetype
  application/pdf --output "./myreports/Test.pdf" --user user
  -password pass
```

where request is submitted to the server to return a pdf formatted report name Test.pdf of the name "/Reports/myreporttemplate.rml". For an updated list of parameters to be used with the command interface, please use

```
java -classpath RepertoireClient.jar
    com.elixirtech.ers2.client.ERSClientCommand --help.
```

Chapter 8
Server API

Overview

Elixir Repertoire Server is accessible and manageable through the Hypertext Transport Protocol (HTTP), the kind of network connection used by a browser when accessing an http: or https: web site. We've already seen the browser interface provided by the server in Chapter 6, *Web Interface*, in this chapter we will look at how the server URLs can be used to interact with the tool through programs.

The HTTP protocol specifies Uniform Resource Locators, like http://www.elixirtech.com/ and operations GET, POST, PUT and DELETE. Browsers typically use GET for reading web pages and POST for submitting data, such as entry forms. Each operation has characteristics defined by the HTTP standard, so that all software that uses HTTP can follow the same rules.

GET
: Getting a resource can be performed at any time, and as many times as necessary. It has no side-effects, so calling GET again is perfectly safe - just like keep hitting refresh on your browser. It is like coding PRINT X. X won't change, however many times you call it. Many tools have a limit on the size of a URL, so a GET with a long parameter string can occasionally cause an error. Therefore some Report and Data services that should logically be GET services are also provided as POST services because POST has no artificial limit on parameter size. These services will be noted below as "GET or POST" - you will find the GET version simpler to test from a browser, but might need to use the POST version in the rare case that the total size of your parameters exceeds a few kB.

POST
: Posting should be used with care as these methods do have side-effects. For example on a website when you book a flight, you shouldn't submit (POST) twice, or you will probably get billed twice. It is like coding X = X + 1. Every time you call it X ends up with a different value.

PUT
: Putting a resource can be performed at any time. This method created or updates the value of a resource. You can repeat the operation and there are no side-effects. It is like saying X = 5. You can say it as many times as you want. After the first time, it has no extra effect.

DELETE
: Deleting a resource can be performed at any time. You can repeat the operation and there are no side-effects. This is because you can safely delete something that isn't there - it isn't an error, the result is, it still isn't there. Following our coding analogy, this is like coding X = NULL.

HTTP provides the basic data interchange protocols, but there are a variety of mechanisms used on top of these four primitive operations. Elixir Repertoire Server uses Representational State Transfer (REST).

REST

The key concept in REST is the idea that each resource should have a unique global identifier and that operations can be applied to the resource through a set of well-defined operations (GET, POST, PUT and DELETE). In Elixir Repertoire, this means that each repository file and folder, user, group, report, datasource, target, even log, all have distinct identifiers - URLs - that allow direct interaction. For example, creating a new user called bill can be done through the Web Interface with your browser, as described previously, but can just as easily be done by sending a PUT request to the URL /user/bill.

Similarly, accessing a server log is as simple as sending a GET request to /log/file/server.log. You can integrate that log into a portal if you choose, or maybe write a utility to scan the log for specific tasks you are monitoring.

Using REST, you can compact a filesystem even without the access to the administration console. Type the following scripts in the command line to compact all filesystems:

```
curl -X POST --basic -u admin:sa http://localhost:8080/repository?
mode=compact
```

All .bak files will be removed.

You can also compact only one filesystem at a time by adding its Display Name after `/repository` in the scripts. For example, when the Display Name is `ElixirSamples`, use the following scripts:

```
curl -X POST --basic -u admin:sa http://localhost:8080/repository/
ElixirSamples?mode=compact
```

All .bak files in /ElixirSamples will be removed.

JDBC

JDBC API supports the retrieving of records from datasource by using SQL query. In order to do that, some basic jar files are required in the project. The files required are:

- RepertoireJDBC.jar
- commons-codec-1.3.jar
- commons-httpclient-3.0.1.jar
- commons-logging.jar
- log4j.jar

Example 8.1. Retrieve Records From Selected Column

When trying to retrieve records from a selected column of a datasource, you can use the following example:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;

public class SelectSpecificColumn {
  public static void main(String[] args) throws Exception
    {
      Class<?> cls = Class.forName("com.elixirtech.jdbc.Driver");

      Properties props = new Properties();
      props.put("user","admin");
      props.put("password","sa");
      Connection conn = DriverManager.getConnection
  ("jdbc:repertoire://localhost:8080/",props);
      Statement s = conn.createStatement();
      ResultSet rs = s.executeQuery("SELECT Company, 2000 FROM
  /ElixirSamples/DataSource/FruitSales.ds");
      while (rs.next())
      {
        String company = rs.getString("Company");
        float val2000 = rs.getFloat("2000");
        System.out.println("-> " + company + " " + val2000);
      }
      s.close();
      conn.close();
 }
}
```

If the server is not running on a local machine, change *localhost* to the IP address of the machine that is running the server. The API allows other SQL query commands like `WHERE`, `AND` and many others.

Calling HTTP

HTTP was chosen as the underlying protocol because there are many libraries written for many different programming languages that can use it. It is beyond the scope of this manual to teach you how to program in all of these languages, but examples in common languages, such as Java, C#, Ruby and Python are provided as resources on the Elixir Technology web site. However, as a short example, here's how to call the query mime-types service (described in the reference below) using Ruby:

```
require 'net/http'

Net::HTTP.start('localhost',8080){
|http|
  req = Net::HTTP::Get.new('/query/mime-types')
  req.basic_auth 'admin', 'sa'
  response = http.request(req)
  print response.body
}
```

That wasn't so hard, was it? Every language will vary, but through use of the right abstractions and libraries (e.g. HttpClient for Java) you can write true Service-Oriented Applications (SOA) that utilise

all of the services that Elixir Repertoire Server provides. As an example, Elixir Repertoire Remote is written using exactly the same network API that is exposed to you.

Logon

Service: POST /logon.html

- Parameter: username
- Parameter: password
- Parameter: return (optional)

Explicit logon to the server, just like a user with a browser would do. API users should probably use the BASIC AUTH method of connection instead as this means the credentials are associated with each request and there will be no chance of a session timeout. A timeout would result in a redirect to the logon page, just like in the browser. Use return=SomeURL to redirect to a specific page if successful.

Service: GET /logout.html

Calling this URL invalidates the current session. Even if you don't explicitly logout, your session will expire within a fixed interval (default 100 mins). You will need to logon or supply BASIC AUTH credentials to continue using services.

Service: GET /authenticate-session

- Parameter: session
- Parameter: return (optional)

This URL supports the Single Sign-On (SSO) mechanism. The controlling server should logon to the Elixir Server using either /logon.html or BASIC AUTH and obtain a session cookie. The controlling server then redirects the user to this service, passing in the session id as a parameter. If the session id is valid, the authentication details will be cloned into a new session cookie returned directly to the user. The user may now continue to use the services without the intervention of the controlling server. If the controlling server logs off (/logout.html) providing the original session id, then any user session that was authenticated based on that session id will also be terminated.

Repository

Service: GET /repository

- Parameter: mode (optional)

Provides a repository browser if no mode is supplied. The browser shows a single level of the tree with the ability to navigate through the children or return to the parent folder. Selecting a file will "open" the file - that means if it is text, html or pdf it will probably be streamed to your browser. If it is an rml, ds or pml you will probably be prompted to save it. Note that this provides access to the raw files - you can't render or generate from here. Other services provide that ability. If mode is "xml" then the filesystems are returned as XML. If mode is "tree" then the entire repository tree is returned in XML. The modified attributes indicate the last modified times of the files represented as the difference, measured in milliseconds, between the file time and midnight, January 1, 1970 UTC.

Response:

```
<!-- output when mode=xml -->
<filesystems>
  <filesystem displayName="ElixirSamples"/>
  <filesystem displayName="Scheduler"/>
```

```
</filesystems>

<!-- output when mode=tree -->
<repository>
  <folder name="ElixirSamples"
      modified="1179946379812" access="rw">
    <folder name="Dashboard"
        modified="1179946365312" access="rw">
      <folder name="Border Catalog"
          modified="1180009266296" access="rw">
        <file name="All Borders.ds"
            modified="1179946257296" access="rw"/>
        <file name="All Borders.rml"
            modified="1179946300875" access="rw"/>
        ...
```

Service: GET /repository/ElixirSamples/Report

- Parameter: mode (optional)

This is a continuation of the previous service, as you navigate through the HTML, the service URL changes to mirror the repository file structure. You can use the same mode options, "xml" and "tree", which give you the corresponding output, but rooted on the repository path identified in the URL.

Service: GET /repository/ElixirSamples/Report/CustomerListing.rml

This is a continuation of the previous two services and illustrates the retrieval of a file. In this case, mode is not used as the file will always be returned using its original mime-type. Again, note that this will download the CustomerListing report template, it won't render the report. See the Report and Target services below for rendering options.

Service: PUT /repository/ElixirSamples

- Parameter: action
- Parameter: mode

Update the filesystem. By entering "update" and "filesystem" for the parameters named "action" and "mode" respectively will update the filesystem specified in the java file.

Service: PUT /repository/ElixirSamples/Report/CustomerListing.rml

Set the file contents. If the name doesn't exist, then a new file will be created, or a new folder if the pathname ends with '/'.

Service: DELETE /repository/ElixirSamples/Report/CustomerListing.rml

Delete the file or folder (along with any child files and folder).

Service: POST /repository/ElixirSamples/Report/CustomerListing.rml

- Parameter: action
- Parameter: to (optional)

This service provides utility options for repository management. When the path points to a file or folder and the action is "rename" and to is another filename, the file or folder will be renamed (retaining the same parent). When the path points to a filesystem and the action is "refresh", then that filesystem is refreshed. When the path points to /repository and the action is "refresh" then all filesystems are refreshed.

Task

Service: GET /tool/admin/tasks.html

Displays the loading status of reports and datasources. When loading completes, the task is removed from the list.

Query

Service: GET /query/alive

Tests if the server is alive and responding to requests.

Response:

```
200 (Ok) if the server is alive
```

Service: GET /query/mime-types

Get the mime-types supported for rendering.

Response:

```
<mime-types>
  <mime-type name="application/pdf"/>
  <mime-type name="application/postscript"/>
  <mime-type name="application/rtf"/>
  ..
```

Service: GET /query/targets

Get the available targets for rendering to, along with the default target properties that you can override.

Response:

```
<targets>
  <target name="mail">
    <property name="message">Your report is attached.</property>
    <property name="to">elided...alice@example.com</property>
    <property name="subject">Report from Elixir Server</property>
    <property name="smtp.host">elixir.aspirin</property>
    <property name="filename">report</property>
    <property name="from">tom@example.com</property>
    <property name="cc">elided...susan@example.com</property>
  </target>
  <target name="db">
    <property name="overwrite">yes</property>
    <property name="driver">elided...EmbeddedDriver</property>
    <property name="table">JobOutput</property>
    <property name="url">jdbc:derby:C:/Temp/elxdb/fs</property>
  </target>
  ...
```

Service: GET /query/filesystems

Get the filesystems visible to the user.

Response:

```
<filesystems>
  <filesystem display-name="ElixirSamples"
    configuration="C:\RepertoireServer\samples"
    read-only="false" type="local"/>
  <filesystem display-name="Scheduler"
    configuration="Scheduler"
    read-only="false" type="jdbc"/>
</filesystems>
```

Service: GET /query/jdbc-drivers

Get the JDBC drivers available on the server. This service will only return available known drivers - i.e. those with known classnames that are provided by the GUI as suggestions and marked with a green icon indicating they are available.

Response:

```
<drivers>
  <driver name="Derby Embedded"
    class="org.apache.derby.jdbc.EmbeddedDriver"
    url="jdbc:derby:database"/>
  <driver name="JDBC/ODBC_Bridge (Sun JVM)"
    class="sun.jdbc.odbc.JdbcOdbcDriver"
    url="jdbc:odbc:Sample"/>
  <driver name="MySQL (com.mysql)"
    class="com.mysql.jdbc.Driver"
    url="jdbc:mysql://<host>/dbname"/>
  <driver name="MySQL (mm.mysql)"
    class="org.gjt.mm.mysql.Driver"
    url="jdbc:mysql://<host>/dbname"/>
</drivers>
```

Service: GET /query/repository/ElixirSamples/Report/CustomerListing.rml?mode=params

Gets the parameters for reports, datasources, dashboards and jobs

Response:

```
<!-- in this case there aren't any -->
<parameters report="/ElixirSamples/Report/CustomerListing.rml"/>

<!-- in this case there are -->
<parameters report="/ElixirSamples/Report/NewsToday.rml">
  <parameter>
    <key>Language</key>
      <type>choice(Mixed,Korean,Tamil,Arabic)</type>
      <value>Mixed</value>
   </parameter>
</parameters>
```

Service: GET /query/user

Get the current user information (id, name, groups).

Response:

```
<user name="admin" id="1" is-admin="Yes">
  <group id="1" name="admin"/>
  <group id="2" name="*"/>
</user>
```

Tool

Service: GET /tool/repository

Get the clickable repository tree (breadcrumbs flat HTML) that will open chosen files. This is different from the /repository URL in that it will launch the right engine to handle the Elixir file types - for example selecting a report will render it, selecting a datasource will generate the data and selecting a dashboard will open it for viewing. This is the URL that the shows below the menu bar when you log on to the Web Interface.

Remote

Service: GET /remote/license

Get the Remote license for the current (logged on) user. This license indicates what features of the Remote tool the user has access to.

Report

Service: GET or POST /report/ElixirSamples/Report/CustomerListing.rml

- Parameter: mime-type
- Parameter: elx.disposition
- Parameter: any report parameters (optional)

Render the report into CSV, DocX, Excel, Glint, IML, PDF, PPT and XML mime-types. The report will be streamed back to the client. Because this is a simple GET method, you can try this directly from your browser or embed a link into any web page. The data will be streamed back to your browser, which will show it directly in the specified mime-type. For "elx.disposition", it is specifically for the PDF mime-type, and there are only two options available. They are "inline" and "attachment". For "inline", the PDF will open in the window. As for "attachment", the user will be prompted to open or save the file. The POST version should be used if sending more than about 4kB of parameters.

Dashboard

Service: GET /dashboard/ElixirSamples/Dashboard/Operational Dashboard/Operational Dashboard.pml

- Parameter: mode
- Parameter: card
- Parameter: any dashboard or datasource parameters (optional)

Returns the dashboard at the specified card with parameters entered if any. The options available for mode are "init" and "parameters".

Data

Service: GET or POST /data/ElixirSamples/DataSource/CustomerListing.ds

- Parameter: mime-type
- Parameter: any data source parameters (optional)
- Parameter: xslt=respository:/some/file.xslt (optional)

Generate the data into the requested mime-type. The available mime-types are: application/vnd.ms-excel (Excel), text/xml (XML) and text/csv (CSV). If you choose XML mode, you may also pass an xslt parameter which names a stylesheet in the repository (full repository url required) which will be used to transform the XML while generating. In all cases, the resulting data will be streamed back to the client. The POST version should be used if sending more than about 4kB of parameters.

Service: GET /data/ElxirSamples/DataSource/FruitSales.ds

- Parameter: any data source parameters (optional)
- Parameter: mode

By entering "time" for the "mode" parameter, total number of records and elapsed time will be displayed.

Service: POST /data/ElixirSamples/DataSource/CustomerListing.ds

- Parameter: datastore
- Parameter: any data source parameters (optional)

Generate the data from the Composite DataSource into the named datastore. Note that this service is a POST, unlike the GET version above, because the server is modified by this operation - the data is written onto the server instead of being streamed back to the client. You will recall from the earlier discussion that POST operations may have side-effects - for example the datastore may append records to a JDBC database. You wouldn't want the same records appended twice! The GET version above is stateless, because the data is returned to the user - the server state doesn't change.

Glint

Service: GET /glint/ElixirSamples/Resources/sample.glint

- Parameter: page (optional)
- Parameter: mode (optional)

Returns the specified page of the glint, or the first page if no page parameter is supplied. If the mode value is "page-count" then a plain text number is returned indicating the number of pages available.

Job

Service: POST /job/ElixirSamples/Job/JobSample.job

- Parameter: any job parameters (optional)

Executes the job and returns the job log as a text stream.

Target

Service: POST /target/TargetName

Renders a report to the named target (TargetName). This service requires an XML structure to be sent as the request, detailing the specific report along with any rendering and target parameters that are required. You should always provide the mime-type target parameter unless the target is a PrintTarget. Other target parameters can be identified from the /query/targets service.

Request:

```
<request report="/ElixirSamples/Report/CustomerListing.rml">
  <report-parameters>
```

```
    <param name="paramName">value</param>
  </report-parameters>
  <render-details> <!-- optional -->
  </render-details>
  <target-parameters>
    <param name="mime-type">application/pdf</param>
  </target-parameters>
</request>
```

Log

Service: GET /log/file/activity.log

(Admin only)

Returns the contents of any log file from the server /log directory. Substitute activity.log for any other log name to have it streamed back as plain text.

User

Service: GET /user

(Admin only)

Lists all users and their groups

Response:

```
<users>
  <user id="1" name="admin" enabled="yes">
    <group id="1" name="admin"/>
  </user>
  <user id="2" name="scheduler" enabled="yes">
    <group id="3" name="scheduler"/>
  </user>
  <user id="3" name="user" enabled="yes">
  </user>
</users>
```

Service: GET /user/bill

(Admin only)

Lists a named user and their groups

Response:

```
<user id="1" name="admin" enabled="yes">
 <group id="1" name="admin"/>
</user>
```

Service: PUT /user/bill

(Admin only)

Adds or edits user bill. When creating, don't specify the name attribute, because it is part of the URL. The enabled state defaults to true if not specified. You can specify a name while editing to rename the user. When editing, leaving out the name, password or enabled attributes indicates that those values are unchanged. Similarly, leaving out the group children indicates that the groups remain unchanged.

If you want to remove all groups, you should specify a single child group with no id (this overrides the no groups means no change inference).

Request:

```
<user name="bill" password="XXX" enabled="yes">
  <group id="1">
  <group id="2">
</user>
```

Service: DELETE /user/bill

(Admin only)

Deletes the named user

Group

Service: GET /group

(Admin only)

Lists all groups and their users

Response:

```
<groups>
  <group id="2" name="*"/>
  <group id="1" name="admin">
   <user id="1" name="admin"/>
  </group>
  <group id="3" name="scheduler">
   <user id="2" name="scheduler"/>
  </group>
</groups>
```

Service: GET /group/scheduler

(Admin only)

Lists the group called scheduler and its users

Response:

```
<group id="3" name="scheduler">
  <user id="2" name="scheduler"/>
</group>
```

Service: PUT /group/testing

(Admin only)

Adds or edits the group called testing. When creating, don't specify the name attribute as it is part of the URL. You can specify a name while editing to rename the group. Leaving out the user children indicates that the users remain unchanged. If you want to remove all users from the group, you should specify a single child user with no id (this overrides the no users means no change inference).

Request:

```
<group name="testing">
 <user id="1">
 <user id="2">
</group>
```

Service: DELETE /group/testing

(Admin only)

Deletes the named group

JavaScript Extensions

One additional object is accessible through JavaScript or substitution when running on the Server. It provides a few utility functions related to user access, for example so you can include the name of the user who generated a report as part of the report itself.

The object is called `Server` and provides the following functions:

int getUserId()	Returns the unique id of the current user
String getUserName()	Returns the name of the current user
int[] getGroupIds()	Returns the ids of the groups to which the current user belongs
String[] getGroupNames()	Returns the names of the groups to which the current user belongs
boolean isAdmin()	Returns true if the current user belongs to the admin group
String getRepositoryUserHome()	Returns the name of the current user's repository home directory

The Server object can be used in any server-side JavaScript codes but will be undefined if used in the standalone Designer. Substitutions can use the Server object by invoking JavaScript through an "=" prefix. For example

```
${=Server.getUserName()}
```

will insert the current user name into the output.

Chapter 9
Troubleshooting and Common Errors

Introduction

When the Repertoire Server and Client components appear unable to function together, there are a few things you can try to diagnose the problem. This chapter describes some ways to check the system is running correctly and some errors that you might encounter.

Server Troubleshooting

System Requirements

Ensure your system meets the minimum requirements for running Elixir Repertoire Server. In particular, check the available RAM and ensure that the memory allocated to the server (defined by startServer.bat and startServer.sh) does not exceed the available RAM. The default values are 512MB, set using the option `-mx512M`.

Port Availability

Ensure that port 8080 is not used by other software. These values are set in the config file ERS2.xml, so you can change them in case of any conflict. Of course, if you change the values, clients must connect to the modified port numbers. By default the server will exit automatically with a message if the client connection port is not available.

Logs

The log directory contains a server.log file that will indicate any errors that occurred while the server was running. By default you will see info messages as well as warn and error messages. You can control the level of messages output by editing log-config.xml in the config directory. Look down the file for a <root> element and change the priority value "info" to "debug" for more information, or "warn" or "error" for less information. For optimization, when the server is working properly, you can set it to warn or error.

```
<root>
  <!-- note this value affects minimum job logging level
       error, info or debug. Increase message logged
       default should keep to info.
  -->
  <priority value="info"/>
  <appender-ref ref="Server" />
  <appender-ref ref="STDOUT_ERROR" />
</root>
```

Running as a Windows Service

When a program runs as a Windows service, it has no access to user environment variables. You should also avoid assuming a current directory and use absolute paths. Before attempting to run the server as a Windows Service you should ensure it runs as a standalone server. Once this has been verified, you can start as a service and review the Windows Event Viewer and the server log directory for any error messages.

Client Troubleshooting

JVM Versions

For testing it is advisable to ensure that both client and server are running the same JVM version. The client and server are likely to operate correctly with mixed versions, but fixing on one reduces the chance of errors.

Consistent Connection Information

The default Repertoire Server port is 8080. If you have changed this, then you need to ensure the client connects on the modified port. Similarly, if you have configured the server in secure mode (the default is false), then clients connecting to the server must also connect in secure mode (usually over a different port).

Client-Server Troubleshooting

Network Access

If the client is unable to connect to the server, try running a browser on the client machine if that fails, on the server machine. If your server doesn't have a GUI, you can use a tool like wget to read a web page from a command line interface.

If you are able to connect to the server from the same machine, then there is probably a firewall preventing access to the server port from your original client machine. If you are not able to access the server from the same machine, then you will need to review the server logs for likely causes.

Common Errors

Client Errors

There are three categories of client-side error:

Communication	Communication errors occur because the client is unable to connect to the server, either due to firewall restrictions, the server port not being available, the server not being ready to accept connections, or due to protocol differences - one side is running secure mode, and the other side plain.
Authorization	After successfully connecting, the client attempts to authenticate itself with the server. If your client program fails to connect, then try to connect using a browser and check that you can authenticate manually. Review the activity and server logs for information about logon failures.

Timeout or Connection Lost	This error occurs if the server fails to respond to the client within a meaningful time, or the connection is broken. Check the server log to identify the cause of any timeout, or check the network to resolve why the connection was terminated. This error may be due to a server failure - for example if the server doesn't have enough memory to complete the set of tasks it is currently handling. You might choose the reduce the number of concurrent tasks that you allow the server to perform by editing ERS2.xml.

Report Errors

Report errors are often due to missing information. For example, the report template and all necessary datasources (and any data files) are not deployed to the server, or are not deployed in a consistent location (remember in particular that the Repository is case-sensitive). You should also check that the dynamic parameters passed in to the render engine are correct. It is usually advisable to include default values for the parameters, so that it is easy to verify the report by not including any explicit dynamic parameters.

After rendering a template for the first time or after any change, you should check the server log for any JavaScript errors while running the report, to ensure that all information is rendered correctly and efficiently.

Datasource Errors

Datasource errors are different for each kind of DataSource:

JDBC	ClassNotFound: Missing JDBC driver. You need to put the correct JDBC driver jar file from your database vendor into the server ext directory and restart the server, so that it will be loaded. You should ask your database vendor for the appropriate jar file for your chosen database and version.
ARFF,Properties,Text,Excel,XML, DBF	These datasources depend on external data files, so these need to be available on the server. It is usually best to place these data files within the repository and refer to them with repository: URLs so that if the repository moves in future the files will still be accessible. Further, each data file needs to conform to the correct format - an invalid XML file cannot be parsed by the XML datasource, for example.
Composite, Reference, Cache	These datasources depend on other datasource files, so these need to be available in the server repository.
Object	Object datasources need access to the class files or jar files that contain the compiled Java code for the classes that are used by the object datasource. If you use the classpath functionality within the object datasource, then you need to ensure that the same classpath applies on the server. You might want to use repository: URLs for your classpath, so the jars get deployed onto the server at the same time as the datasources that use them.

Printing Errors

Once the report has been spooled to the operating system, Repertoire Server has no more control over the printing process. Therefore, you need to look at the print queue management software provided by your operating system to see how print errors are reported and resolved.