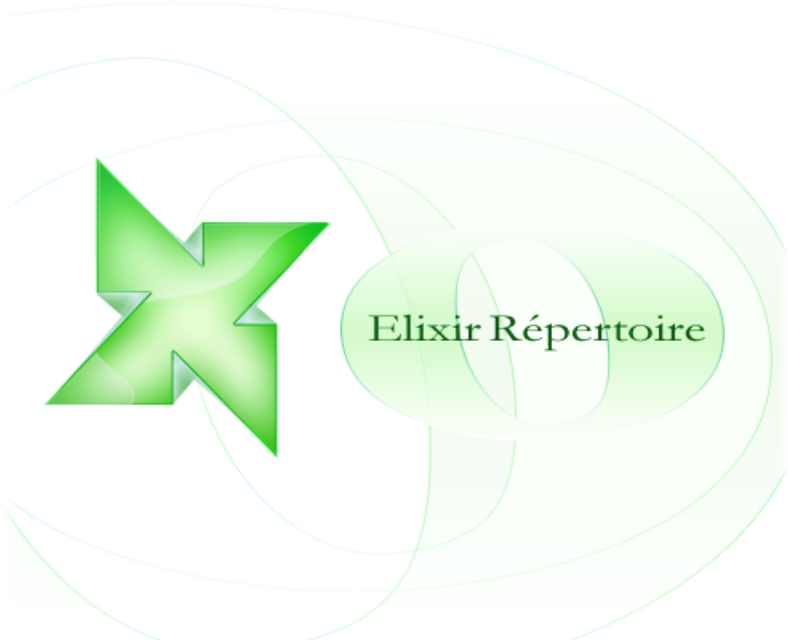
Elixir Repertoire Server Manual

Release 7.3



Elixir **Technology Pte Ltd**

Elixir Repertoire Server Manual: Release 7.3

Elixir Technology Pte Ltd

Published 2008
Copyright © 2008 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

List of Figures

List of Tables

List of Examples

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 multi-processors boxes. It's 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 5 (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 J2SE platform (>=5) 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 2 Standard Edition version 5 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 100 MB disk space is recommended. This consists of the following:

- 15MB for the Elixir Repertoire Server application
- 35MB for the samples
- 50MB for the Java Virtual Machine (depending on the version)

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

Software Requirements

Elixir Repertoire Server is supported on all Java 2 Standard Edition compliant operating system platforms such as Windows and Unixes such as AIX, Linux and Sun Solaris. The minimum Java Virtual Machine version should be J2SE 5. 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 either .zip, or .tar.gz files. The software is the same regardless of download package, so obtain whichever flavor is most convenient for the platform you are running on. 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 5 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 the home directory of the user who will be running the server.

Note

If 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 wil 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 their 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.

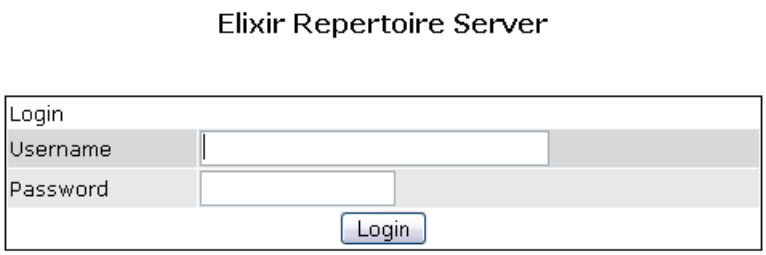
Connect to the server by pointing your web browser to http://localhost:8080/.

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, eg.

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

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. Please read window-service-readme.txt. You may refer to http://wrapper.tanukisoftware.org/ for in-depth discussion of this product.

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 in the config file ERS2Config.xml.
- *samples*: contains default set of sample report and data source 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 2 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, new from version 7.0 provides a much more 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 critial if the server is going 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 2 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, 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.
- *Server log control, log-config.xml*: setup 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.
- Other config files are for internal use or debugging purposes.

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"/>
```

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

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	Jetty 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 (eg. you have multiple network cards), you need to set the desired IP address in dotted-byte format (eg. 192.168.1.1). Requests will now only be accepted if sent to this specific local IP.
Server Listener Port	Jetty Port	Elixir Repertoire Server Listener handles all the incoming request from the clients. The default port number is set to 8080.
Allowed Clients	Jetty 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.

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 6.5, “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, eg. 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, eg. ${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 (and MIME type) Configurations Moved to Database

Prior to 7.3 release, target configurations are saved in ERS2-Config.xml. Moving the targets to the database provides the following benefits:

- Any updates to the target configuration can take effect without restarting Repertoire server.
- Target configuration in database can be shared by multiple instances of Repertoire server. This enable us to cluster Repertoire servers in future.

After this change, the ERS2-Config.xml file is deprecated. All Targets are configured through the browser interface and saved in database.

New 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.

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.

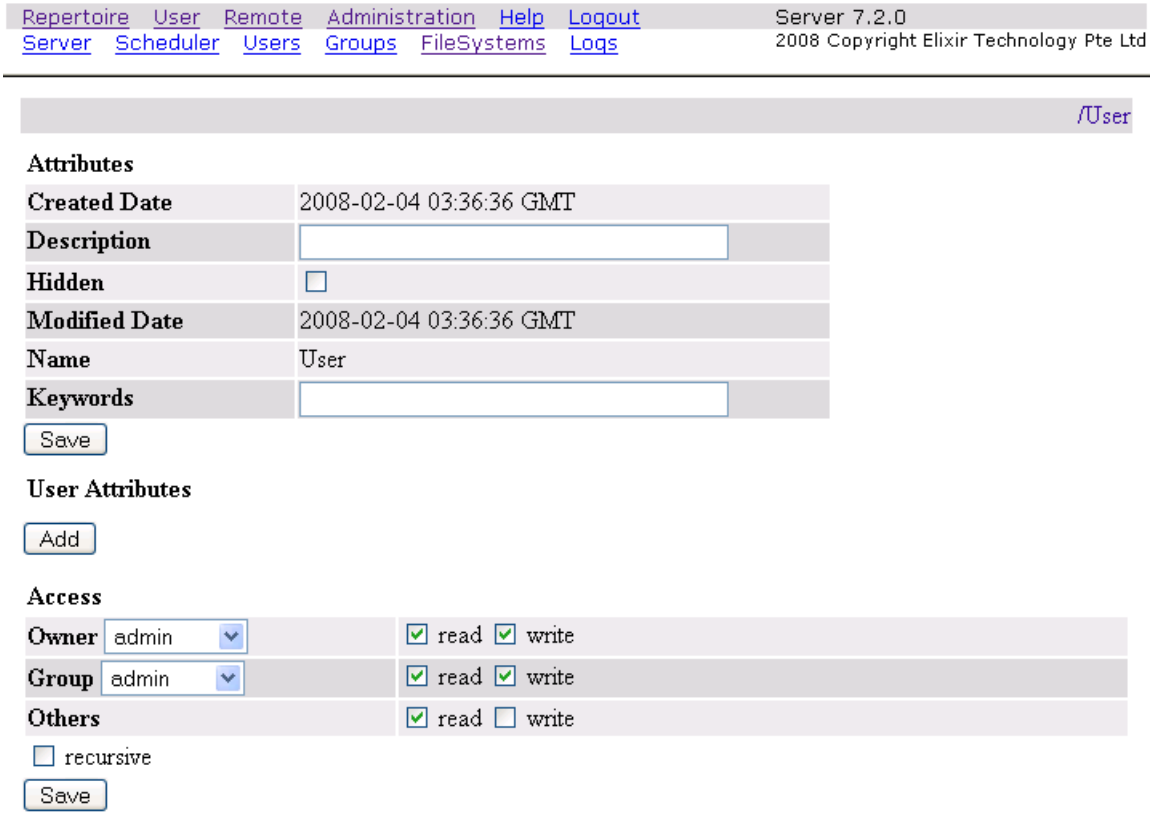
A final alternative is to use Elixir Safe. With this approach, all sensitive information, for datasources, reports, dashboards and jobs, can be kept in a single file and read by supplying the encryption key. This can be a dynamic parameter that the user must enter in order to access the file. Full details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

Access Rights

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

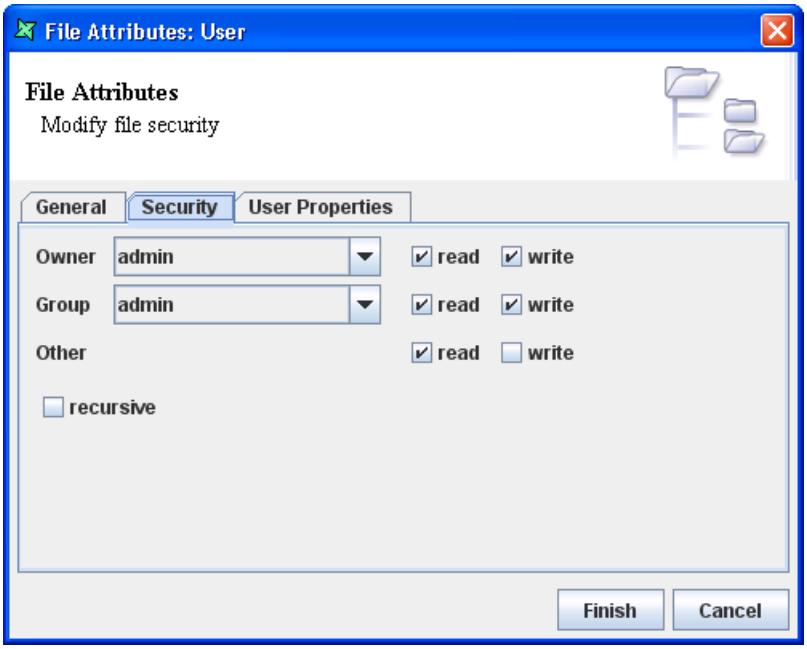
The access rights can be configured in two ways. The first way will be configuring through the web interface. By clicking on the icon on the right of the directory in `Repertoire` page, the page for configuring the access rights for the particular folder will load, as shown in Figure 5.1, “Access Rights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

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 (eg. 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.

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.

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 5 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 disccused 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.

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

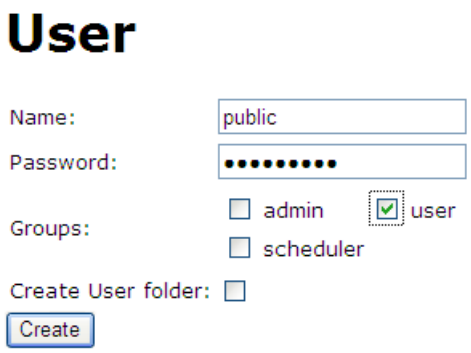
Every user of Elixir Repertoire Server should have a unique logon name and password. This gives access to the Server via the web interface (through a browser), through Elixir Repertoire Remote and through the variety of client tools that Elixir provides. Use of specific filesystems and files is granted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

- Go to RepertoireServer->Config directory and open the jetty.xml file.
- From the code `anonymous enabled="false" user="public" pass="anonymous"`, change the "false" to "true" to enable anonymous login.
- Create the user id through the browser interface as shown. "User" and "Password" parameters must be in sync with your choice of logon parameters in the jetty.xml.

Figure 6.1. Create a Anonymous User ID



- Log off from the browser interface and close the Repertoire Remote Designer and "startServer" file if they are running.
- Launch the "startServer" file again from RepertoireServer->Bin directory.
- Start a browser and connect to your Repertoire Server web interface. You will be able to access with no prompt to enter a user id unless you explicitly visit the logon page.

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.

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.

Jdbc
: The Jdbc filesystem is deprecated in 7.2 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.

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, (eg. 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.

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 are the individual targets configuration. Under "**Properties**", it shows the parameters entered through the web browser. The accompanying image is a screenshot of the Repertoire Remote Designer that shows the result of your configuration.

File Target

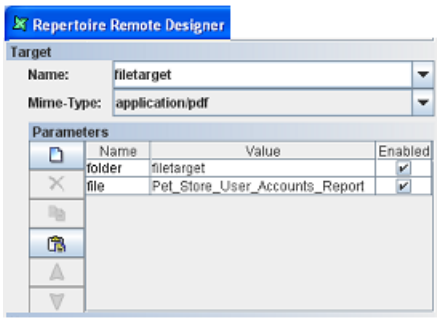
A File Target represents a directory on the server where reports can be stored. You can define as many file targets as you need.

Properties

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

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

Figure 6.2. 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:

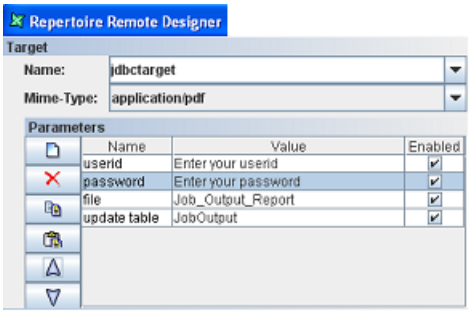
```
CREATE TABLE JOBOUTPUT (
  id INTEGER NOT NULL GENERATED ALWAYS AS
   IDENTITY (START WITH 1, INCREMENT BY 1),
  name VARCHAR(256) NOT NULL,
  lastModified BIGINT NOT NULL,
  content BLOB NOT NULL,
  CONSTRAINT JOBOUTPUT_PK PRIMARY KEY(id) )
```

Once the database is setup, Figure 6.3, “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:

Properties

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

Figure 6.3. 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.
- *query-id* A special property to indicate how to retrieve identifiers from the database. If this is not included, or set to "false", then the JDBC driver will be asked to return generated keys. This doesn't work on Oracle, which doesn't support this JDBC feature. If query-ids is set to "true" then a slightly slower creation process is used, which queries the table to get the new identifier. This approach should be used if your database does not fully implement this aspect of JDBC.

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.

Properties

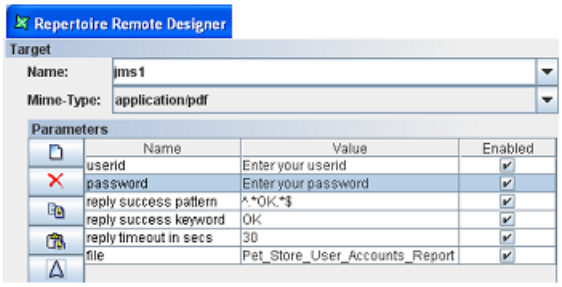
Name	Value
jms.reply.success.pattern	`${reply success pattern##^.*OK.*$}`
jms.reply.success.keyword	`${reply success keyword##OK}`
jms.password	`${password##Enter your password}`
jms.connection.factory	`ConnectionFactory`
jms.reply.required	`true`

filename	`${file##Pet_Store_User_Accounts_Report}`
jms.user	`${userid##Enter your userid}`
mime-type	`${mime-type##application/pdf}`
jms.reply.timeout.seconds	`${reply timeout in secs##30}`
jms.destination	`RQueue`

JNDI Properties

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

Figure 6.4. JMS Target



Mail Target

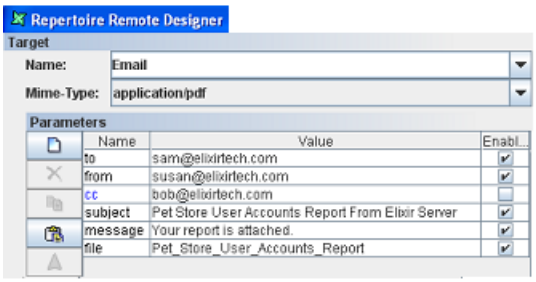
A Mail Target allows the output to be sent by email. Before using this, ensure that the appropriate SMTP Server is configured in ERS2.xml. The default SMTP server elixir.aspirin is built in to Elixir Repertoire Server, so you only need to change it if you wish to use an external SMTP Server. The smtp.host 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.

Properties

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

Figure 6.5. Mail 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 printers defined in the invoking users' computers.

Properties

Name	Value
printer-name	`Canon iR C3220 PCL5c`

Repository Target

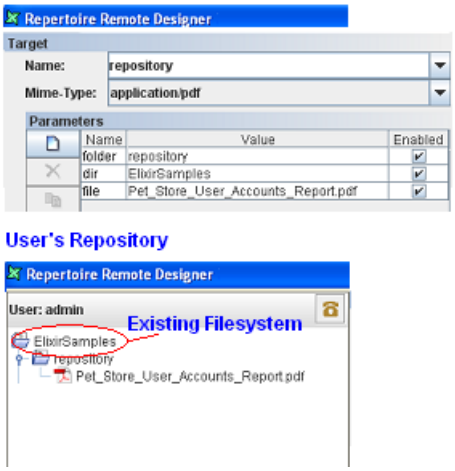
A Repository Target writes the report to the filesystems in the Repertoire Remote Designer. You can identify a target folder in the repository and provided it is writable, files will be written there. This works regardless of whether the target filesystem is of type local, secure or db. You should use 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.

Properties

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

Figure 6.6. Repository Target



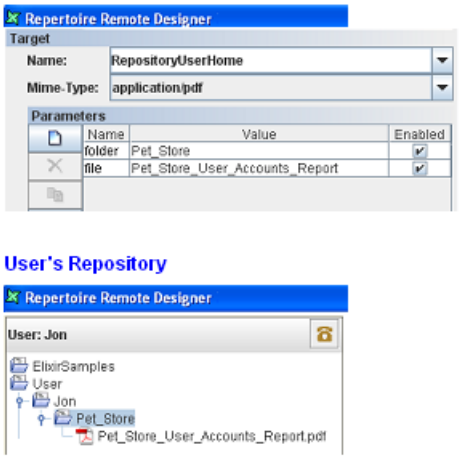
Repository User Home Target

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.

Properties

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

Figure 6.7. Repository User Home Target



SFTP Target

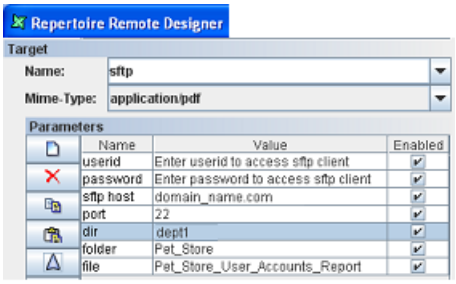
A SFTP Target allows the report to be transferred to a user's secured FTP Server. The available parameters are user, password, host, port, dir and filename. The port is optional and will default to 22, which is the default SFTP port, if not specified.

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

Properties

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

Figure 6.8. 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.

Properties

Name	Value
port	`6000`
host	`company.com`

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 setup 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.

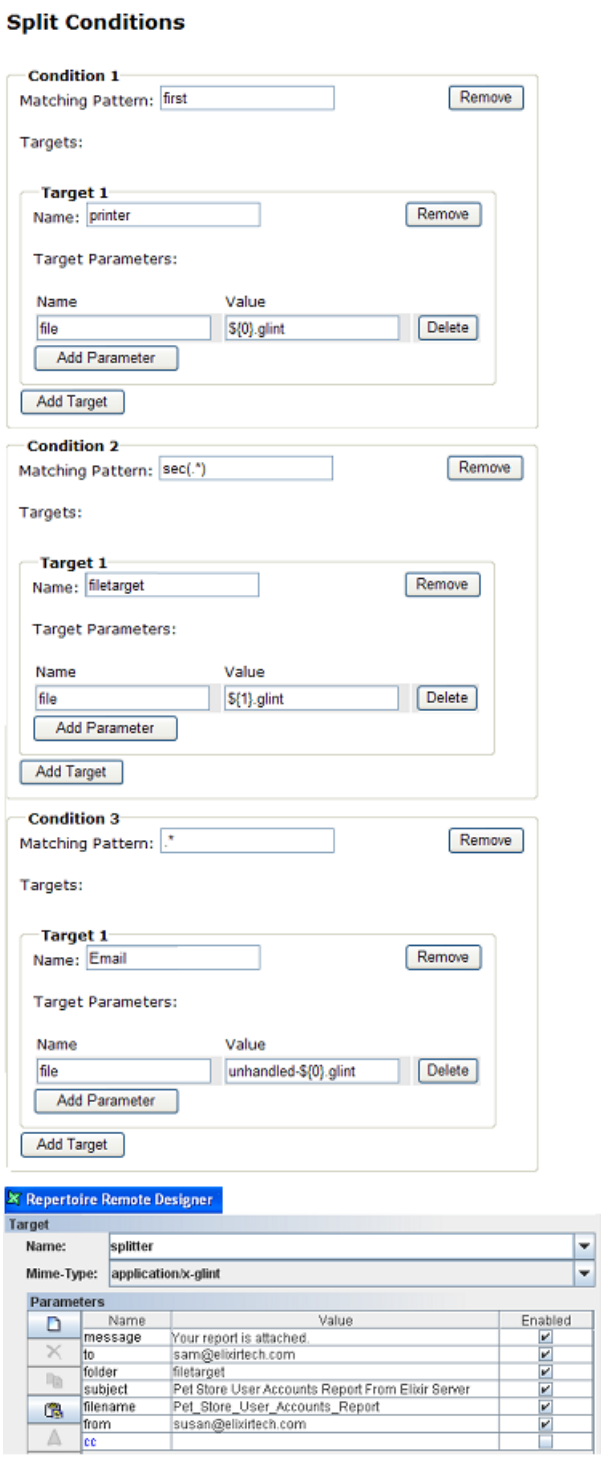
You can provide parameter values for the targets referred in split conditions. For example, you may want to send a portion of a report to "printer" target and the target required one parameter "file". So you can provide value for the "file" parameter as "${0}.glint". Note that the string contains another 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.

Properties

Name	Value
split	1

Figure 6.9. Split Target



Target Access Control

- User Access Contol
  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.

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.

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
}
```

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.

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 (eg. 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/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.

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 - ie. 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: any report parameters (optional)

Render the report into the requested mime-type. 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 if you choose a mime-type application/x-html-zip, or text/html etc. that is understood by your browser. (Actually application/x-html-zip is not understood by the browser, but the zip is disassembled on the server so the browser sees the text/html pages contained within.) The POST version should be used if sending more than about 4kB of 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: 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:

```
```

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,XLS,XML	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, Ref	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.