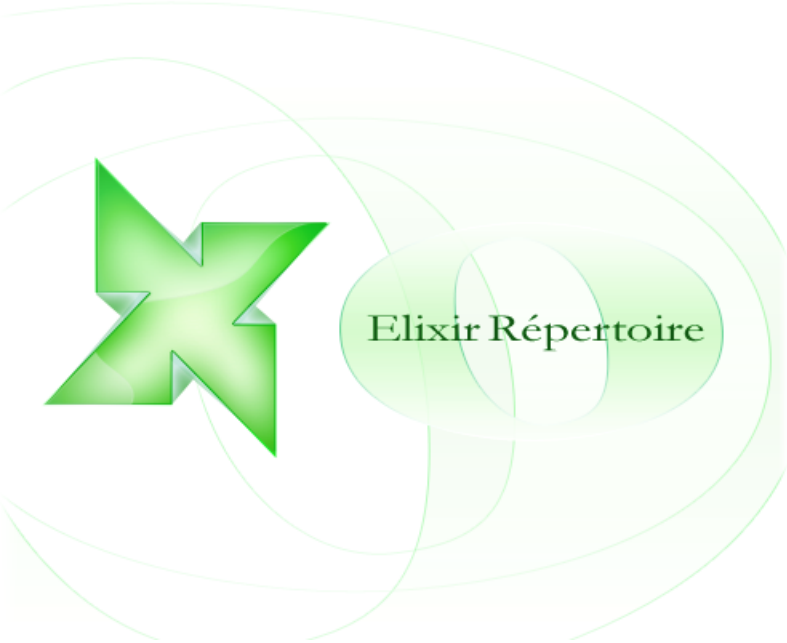
Elixir Schedule Designer User Manual

Release 7.5



Elixir **Technology Pte Ltd**

Elixir Schedule Designer User Manual: Release 7.5

Elixir Technology Pte Ltd

Published 2009

Table of Contents

List of Figures

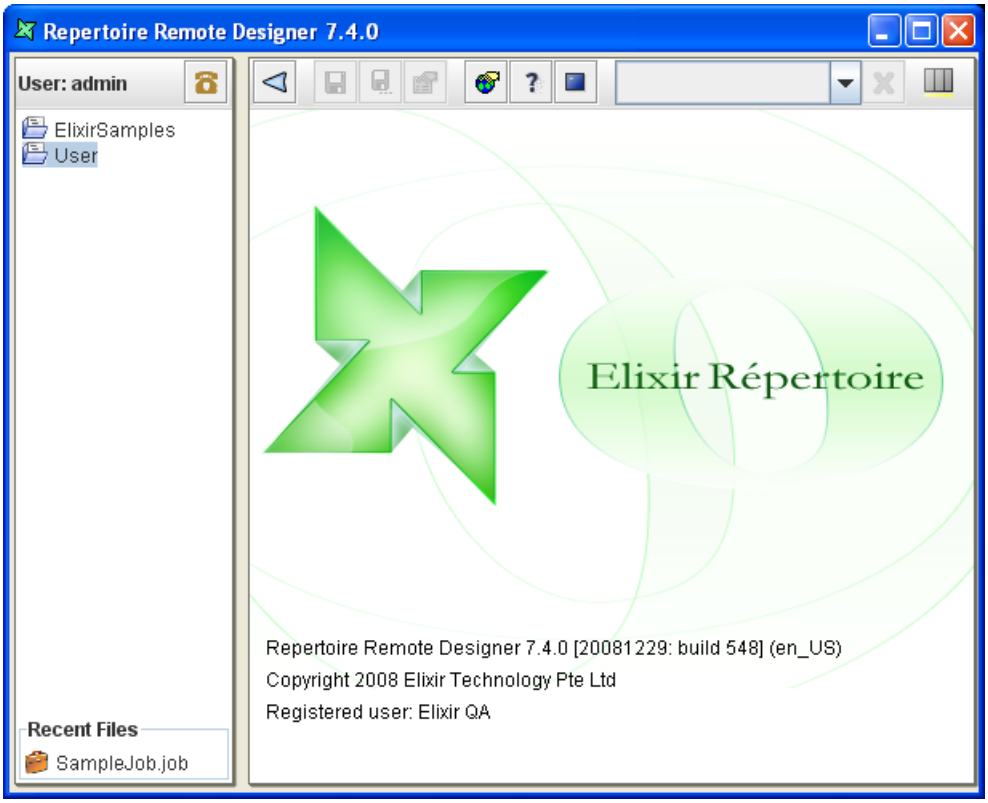
Chapter 1
About Elixir Schedule Designer

Overview

Elixir Schedule Designer provides control over jobs and triggers on the Elixir Server Scheduler. Through the Schedule Designer interface, jobs and triggers may be created, tested, modified and deployed to the server.

The Elixir Repertoire Remote interface is used to host Elixir Schedule Designer as shown in Figure 1.1, “Elixir Schedule Designer”. The only difference to the standalone designer is that the repository tree shown on the left side of the Remote application shows the server repository, rather than the repository on the client machine. All of the options that usually apply to the local repository may be performed remotely.

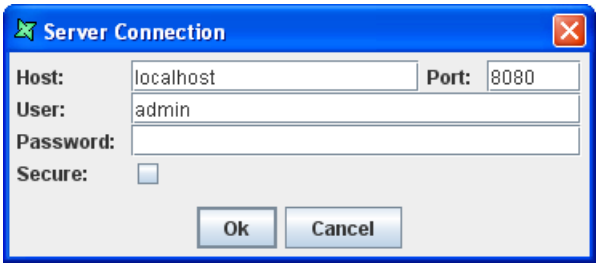
Figure 1.1. Elixir Schedule Designer



Connecting to the Server

In order to use Elixir Schedule Designer, a server connection is needed. Clicking the Connect button above the repository area will show Figure 1.2, “Connection Dialog”.

Figure 1.2. Connection Dialog



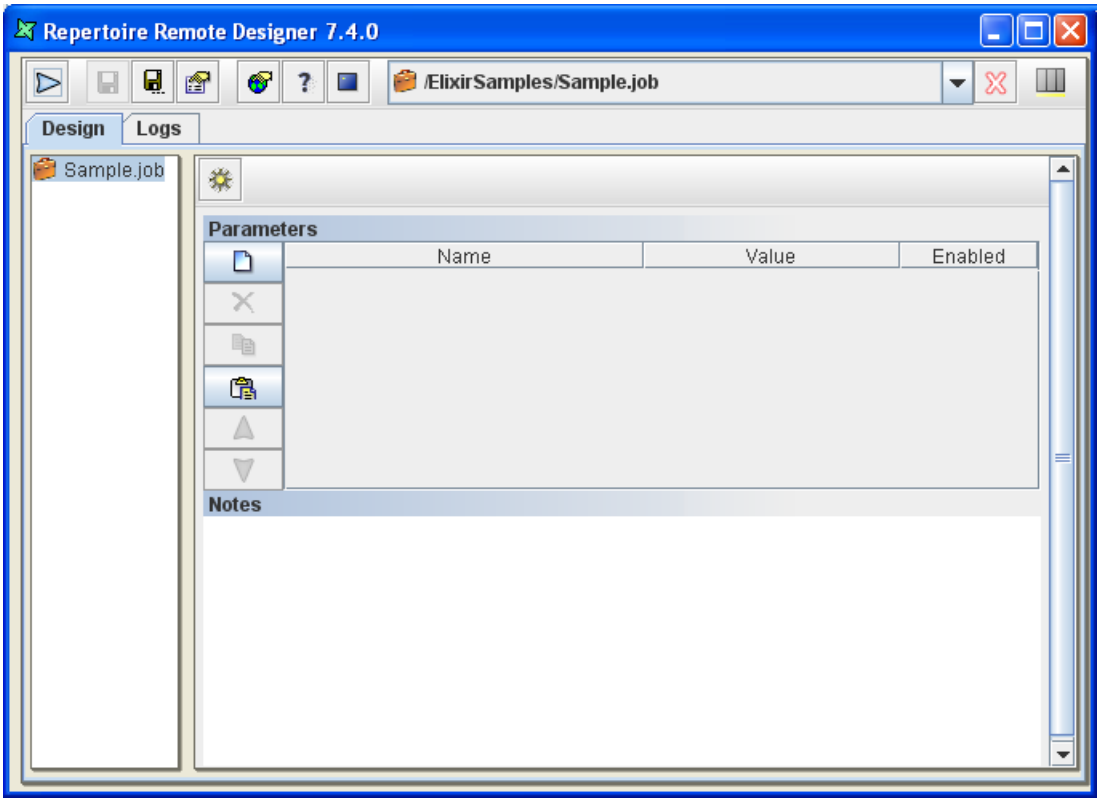
Use the Connection Dialog to enter details for connection to your Elixir Repertoire Server. Once the connection is established, the remote repository should display the filesystems registered with the server. If you have the correct authorization, you will be able to edit and manipulate the files in the remote repository. You cannot add or remove remote filesystems. This can only be done by the Elixir Server administrator through the administration interface.

Chapter 2
Jobs and Tasks

Overview

A Job is a set of tasks that are intended to be executed together. The Job creation interface is shown in Figure 2.1, “Job Designer”

Figure 2.1. Job Designer



When a job is open, there is a task tree on the left of the workspace along with a panel, which shows the details of the currently selected task. When a new job is created, only the job is visible as the root of the tree. Use the popup menu on the Job item in the tree to add new tasks to the job. If you have several tasks in the job you can reorganize them by dragging them around.

Job

Selecting the job itself at the root of the task tree allows parameters to be set. These parameters can be used by all the tasks within the job and it is useful to have a single point of maintenance. Job parameters can either be hard-coded, for example User = Bill, or can be dynamic, for example, User = ${User Name##Bill}. This indicates that the triggering code should supply a value for User, which may be by prompting for "User Name", and that the default value is Bill. See the Elixir Repertoire manual for details of how dynamic parameters can be used throughout the Elixir Repertoire suite. In the case of Elixir Schedule Designer, if the job is triggered manually, then a popup form will prompt

for dynamic parameter values to be entered. However, if the job is triggered by a scheduled Trigger, then it is the Trigger that needs to supply any dynamic parameters.

Each job may define a log file in the repository to which it will write the progress of execution. This is configured on the Logs tab in the job workspace. If no log name is specified, the job will still execute, but you won't be able to review the progress of execution. You can choose the level of logging using the combo box on the right of the Log File entry. If you choose Debug you will get details of the start and stop time of every task as it is executed. Info, Warn and Error provide increasing smaller amounts of logging output, which can make it easier to spot problems that might otherwise get lost in the most verbose log output.

Parameters

Some jobs may require the use of parameter(s) and the necessary parameter(s) should be enabled. If the name of a parameter is highlighted in blue or red, a tooltip will be seen to let the user know what can be done when the cursor rolls over the name of the parameter.

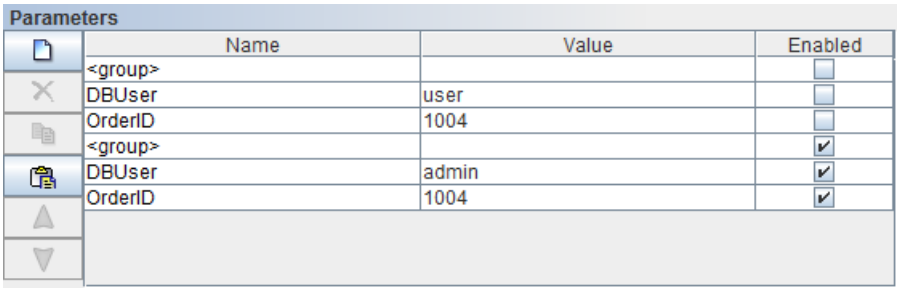
Users can change the value of a parameter by clicking on the Value column of a particular parameter, then edit the value directly or click on the "..." button to bring up a dialog box where the user can choose a value from the choices available. Parameter of different type will have a different dialog box.

- If the parameter is of "choice" type, the dialog box will show a list of available choices.
- If the parameter is of "date" type, the dialog box will show a date chooser.
- If the parameter is of "lookup" type, the dialog box will show a list of available choices loaded from the datasource at runtime.
- If the parameter is of "password" or "string" type, the "..." button will not be present.

In the case whereby the Schedule Designer is unable to identify the type of parameter, the parameter will be treated as a string type.

Users can group certain parameters such that the parameters can be enabled or disabled as a whole. To be able to do so, add a new parameter with name "<group>" as seen in Figure 2.2, “Grouping of Parameters”

Figure 2.2. Grouping of Parameters



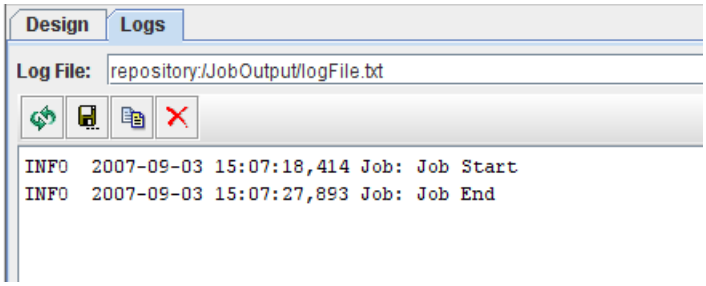
A "<group>" defines a starting of a new group and the ending of the previous group. When the checkbox for a particular "<group>" is checked or unchecked, all the parameters that belong to the group will be enabled or disabled respectively.

Log

A log file is a text file that records the processes that happens in the background when the job is run. The path of the log file must be a valid path in the repository, e.g. /JobOutpt/LogFile.txt. Local directory path like "C:/log/myLog.txt" will not be able to work. In order to get a valid path of the log file, user

can select the log file in the repository panel, right-click and select "Copy URL". Under the "Logs" tab, paste the URL in the field named "Log File". If a job runs successfully, a similar screen like the following figure, Figure 2.3, “Log File”, will be shown.

Figure 2.3. Log File



Tasks

Each task added to a job has a name and a space for notes to be entered. Both of these are optional, and are to allow documentation of the job so that subsequent administrators and maintainers can interpret it in future. The following sections describe each individual task in detail. Some are composite tasks, which means they can also contain child tasks. Composite tasks are useful for loops, optional routines and parallelism. Where text can be entered in a task - that is, in any text field, variable substitutions such as ${Name} can be used. The appropriate parameter values will be substituted when the task is executed.

Ant

The Ant task allows Ant build files, as described by the Apache Foundation, to be launched from within a job. Ant is extremely flexible, allowing file operations, ftp, mail etc. to be coordinated. In order to use the Ant task, you need only define the location of the Ant file (typically called build.xml) that you wish to launch. Optionally, you can specify the name of the target within the build file that should be executed. If no target is named, then the default target will be chosen. Please see the Ant manual at http://ant.apache.org/manual/ for more details on the Ant file format and options.

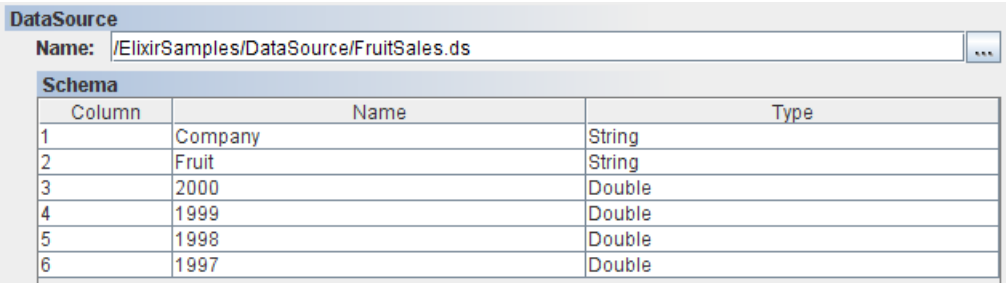
CallJob

The CallJob task allows one job (master) to invoke another (child). The child job is identified by name, and may have parameters passed to it to control its execution. When the CallJob task is executed within the master job, it locates the child job by name and executes it within the scope of the master job. This means all parameters, scripts etc. of the master job are available to the child. In addition, the child job will write it's log to the master job log. This task will only complete when the child job has completed.

DataLoop

The DataLoop task is a composite task that executes each of the child tasks sequentially. The DataLoop has a datasource that supplies records. For each record, the DataLoop will set the execution scope to include the record fields, so they can be accessed by name from the children. The children will then be executed sequentially. The process will repeat until all records in the datasource have been used. In order for users to know that the datasource is valid, the fields of the datasource will be displayed under Schema, as shown in Figure 2.4, “DataLoop”. If not, the text in the Name field will be red in colour.

Figure 2.4. DataLoop



Column	Name	Type
1	Company	String
2	Fruit	String
3	2000	Double
4	1999	Double
5	1998	Double
6	1997	Double

Echo

The Echo task allows information to be written to the log. As with all tasks, text values may include variable substitution strings of the form ${Name} which will be replaced by the appropriate parameter value during execution. This is useful for logging progress of a job and also for debugging. For example, you can echo a message such as

```
Job run by ${User} to render ${Report}
```

FileLoop

The FileLoop task is a composite task that executes each of the child tasks sequentially. The FileLoop requires the name of a folder in the repository. You can also define file criteria that allow the files within the folder to be filtered, for example by extension or modification date. The FileLoop will iterate through all files and execute the child tasks sequentially, once for each file that matches the criteria. While executing the children, the loop makes available the current filename so that it can be used in variable substitutions. The name of the variable is taken from the Parameter Name value defined in the task panel, the default name is FileName. If no criteria are entered, all files in the folder will be used. Selecting the Recursive option repeats the process for all subdirectories under the chosen folder.

GenerateData

The GenerateData task invokes an Elixir Data Designer's Composite DataSource to output to one of its DataStores. The name of the datasource is required, along with the name of the datastore within it that is to receive the records. It is also possible to pass parameters with the invocation which are then accessible within the datasource. Note that only parameters explicitly identified in the GenerateData task will be passed to the datasource - those defined within the scope of the job are not implicitly available to the Ensemble engine.

> **Note**
>
> When the job is running, information regarding the progress of the job can be seen in the `Logs` tab. The starting, processing and the ending of the job will be recorded. The information is saved into the log file specified in `Log File:`. This applies to all jobs.

Loop

The Loop task is a composite task that executes each of the child tasks sequentially. The loop requires a simple repeat count that indicates the number of times the children should be executed. If there are three children, A,B and C, and the repeat count is two, then the sequence of execution will be A,B,C,A,B,C. Loop has two common uses, first, to skip a set of tasks, and second to perform the same tasks repeatedly, for example for performance testing or benchmarking.

Parallel

The Parallel task is a composite task that executes all of the child tasks at the same time. Each child task runs as a separate thread and the parallel task will not complete until all of the child tasks have completed. This task allows a thread count to be specified. If no count is given then each child task runs in a separate thread. Otherwise a pool of threads is created and each child task uses a thread from the pool when it is available. For example, by using a thread count of two, only two of the child tasks would be running simultaneously.

RenderReport

The RenderReport task invokes the Elixir Report engine to render a report on the server. The required information includes the report name, the chosen target on the server and the mime-type that is required. Additional parameters may be passed to the report and to the target. Report parameters would include information needed to control the datasource or rendering process. Target parameters would include information needed by the chosen target. For example, if the target is a directory, a target parameter would usually be used for the filename. Alternatively, if the target is an email, a target parameter might be needed to identify the recipient (this depends on how the target is configured on the server - ie. whether the recipient name has been pre-defined). Note that only parameters explicitly identified in the RenderReport task will be passed to the rendering engine - those defined within the scope of the job are not implicitly available.

When user tries to change to another report, there is a possibility that there will new parameters listed under `Target`. As a result, there will be presence of unnecessary parameters. To resolve this, user can right-click within the list of parameters and an option, `Clear unused parameters`, will help to delete parameters that are not required.

Script

The Script task allows Elixir Schedule Designer to invoke JavaScript and Java codes within the job. Variable substitutions of ${Name} variables is performed before executing the code.

SendMail

After the SendMail task is created, the email panel is empty. You will need to fill up `From` and `To` field in order to send out an email. The rest of the fields are optional.

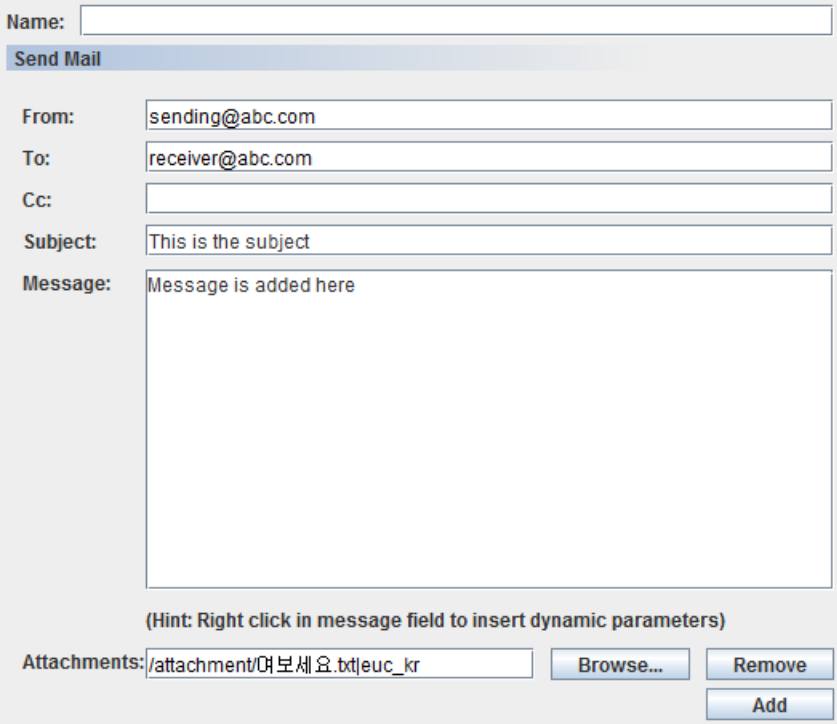
The user can right-click in the `Message` field to find the list of parameters that can be embedded in the email. The parameters are evaluated and substituted with real values on the server side before the email is being sent out. If there are no parameters available, the popup menu will show *No available parameters*. If the the user define some parameters in the task that encloses the SendMail task (such as the job), he will also be able to find the parameters in the popup menu. However, disabled parameters defined in the job will not be shown in the popup menu. Only enabled parameters are shown.

The user can click on the `Add` button in the email panel to add attachments. Add more than one attachment by clicking on `Add` again. To remove the attachement, simple click on the `Remove` button.

Note

SendMail task allows encodings to be included in attachments so as to allow the receiver to see the original text correctly. An example is shown in Figure 2.5, “Encoding in Attachment of SendMail task”.

Figure 2.5. Encoding in Attachment of SendMail task



New SMTP Server

In order to send emails, an SMTP server needs to be configured for Repertoire Server. The configuration is done in *ERS2.xml* as follows :

```
<ers:mbean name="ERS2:name=SMTPServer"
   class="com.elixirtech.ers2.mail.SMTPServer">
 <ers:property name="Host">elixir.aspirin</ers:property>
 <ers:property name="User"></ers:property>
 <ers:property name="Password"></ers:property>
 <!-- Set a DNS host here if default DNS lookup doesn't
   work when using elixir.aspirin SMTP -->
 <ers:property name="DnsResolver"></ers:property>
 <ers:property name="Debug">false</ers:property>
</ers:mbean>
```

elixir.aspirin is a built in SMTP server that comes with Repertoire Server. You can use it if you do not have an external SMTP server to use. However, there are some limitations for Aspirin SMTP server. In most cases, mails sent from as Aspirin server will not be accepted by server implementing SPF mail authorization.

For built-in SMTP server, you do not need to configure the user or password when you usually need to for external SMTP server. You can configure the `DnsResolver` if the default DNS lookup provided by the OS does not work.

For external SMTP server, you can also set `Debug` to *true* such that more debug messages are logged by Repertoire Server.

OnError

The OnError task can only be the child of a job and cannot be added under other composite tasks such as Loop. It should always be the last top-level task listed under the job. There can only be one OnError

task for each job. When the user try to add another one, he will be prompted to overwrite the existing one. Similarly, when the user copy a OnError task from one job to another that has an existing OnError task, he will be prompted to overwrite the existing one.

OnError task is triggered to run when error occurs during the job execution. The user can configure the number of times to re-try the job when it hits an error. The default value is zero, which means there is no re-trying.

A user can also add tasks under the OnError task, which are executed when the OnError task runs. Typically, it will be those tasks that clean up the leftover from previous job failure such that Repertoire can re-try the job again in a consistent way.

If there are no errors during job execution, the OnError task is skipped.

Chapter 3
Triggers

Overview

A Trigger is a mechanism for deciding when to invoke a job. Most commonly, triggers are time-based - for example invoke the job every Tuesday at 6am. All triggers have a name, an enabled/disabled flag and some specialized fields for identifying when to run.

All triggers have a Start and Stop time and can only fire between those times. It is possible to set the Stop to be `Never`, so that the trigger is always operational.

Trigger

There are five pages in the Trigger Wizard.

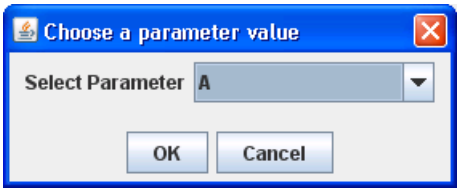
- **Basic Information Page**: Defines the name of trigger, location of the job and parameters required.
- **Job Schedule Page**: Configures the schedule of the trigger.
- **Trigger Active Period**: The start and stop time of the trigger.
- **Trigger Calendar**: The Calendar to refer to. (Calendars are discussed in Chapter 4, *Calendars*)
- **Misfire Configuration**: Chooses what to do when the trigger does not fire as it is supposed to.

Basic Information Page

On the first page of the Trigger Wizard you need to enter the name of the trigger and the location of the job. If the job requires parameters, you will be prompted to enter them here. In the `Parameters` table, the rows might appear in red or blue. Red parameters indicate a value has been supplied that is no longer required by the job, and it can be safely deleted (it will be ignored anyway). Blue parameters are required for the job and so appropriate values should be entered.

Where parameters provide a list of choices for the user to choose, a `[...]` chooser button will be shown. Clicking the button will show the allowed values (as shown in Figure 3.1, “Choose a Parameter Value”).

Figure 3.1. Choose a Parameter Value



Job Schedule Page

On the Job Schedule page of the Trigger Wizard, you can schedule a job to run once only, daily, weekly, monthly, at regular intervals or controlled by a CRON expression.

For weekly events, you need to choose the day(s) of the week when the job should run. For monthly events, you need to choose the day(s) and month(s) to run the job. You can also choose specific day(s) within each month. For jobs that run daily, weekly or monthly, the job will fire at the same time on the scheduled days.

Trigger Active Period

This page of the wizard lets the user configure the absolute time range in which a trigger can fire. The actual firing time itself is governed by the values entered on the previous page - this page only indicates the earliest and latest times that it could possibly fire.

Note

By default, a trigger becomes active upon completion of the wizard and never stops. Also, when the trigger has completed the job run, it will be set to disabled automatically. If user would like to run the job again, the trigger will need to be enabled and make adjustment in *Trigger Active Period*.

Trigger Calendar

If you need to exclude certain periods from the trigger schedule, such as public holidays, you can mark them on a calendar and point the trigger to the calendar. The trigger will not fire during the period(s) marked in the calendar. The calendar options are discussed in Chapter 4, *Calendars*.

Misfire Configuration

This page allows the user to choose the action to be taken when a trigger fails to work the way it was supposed to.

Security

Read-Only: When selected, user will only be able to see the settings done for the trigger and all settings cannot be edited.

Hide Details: This option hides all settings done for the trigger.

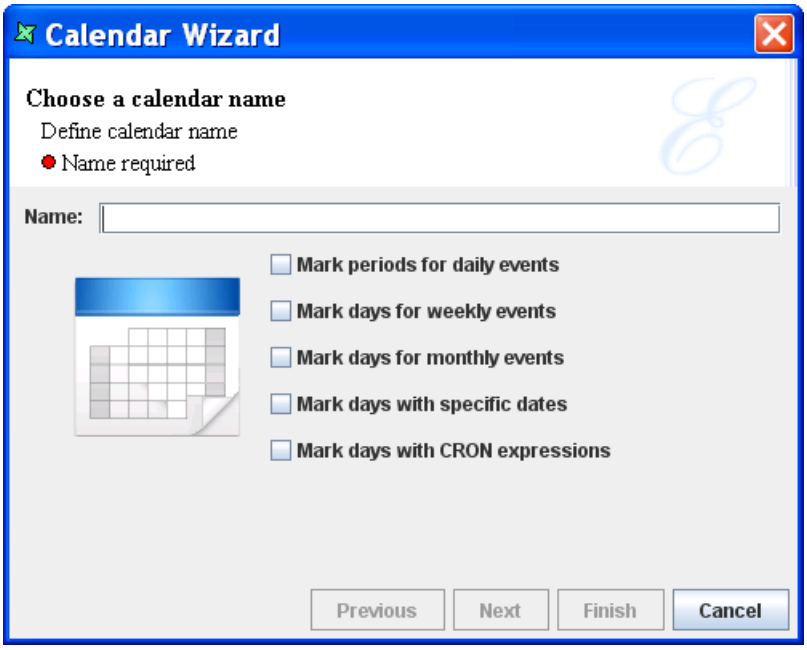
Encrypted: This option is used with either `Read-Only` and/or `Hide Details`. When this option is selected together with any of the above option, a password is required to change the selections in the page.

Chapter 4
Calendars

Overview

Triggers may use calendars to determine special date or times that should be excluded from the firing schedule. Upon adding a Calendar you will see Figure 4.1, “Calendar Wizard”.

Figure 4.1. Calendar Wizard



Mark periods for daily events

You can use this option to mark ranges of the day when the trigger should not fire. For example, if you want to avoid performing an hourly data loading operation between midnight and 6 am, you would set Start Time 12 am End Time 6 am to exclude that range.

Mark periods for weekly events

This option allows you to choose which days of the week to exclude. For example, you could select Saturday and Sunday to prevent associated triggers from firing at the weekend.

Mark periods for montly events

This option allows you to choose which days of the month to exclude. For example, you could select all but 1 and 15 in order to have the trigger restricted to fire on only the first and fifteenth of each month.

Mark days with specific dates

To prevent triggers being fired on public holidays, you can identify them using this calendar option. You can also choose whether the dates apply only for one year (some holidays change date each year) or whether the same date should be excluded every year. For example, to exclude triggers from firing on Christmas Day you would set the Date to Dec 25th and tick "Repeat every year".

Mark days with CRON expressions

The CRON expression option is the most powerful, but requires the most configuration. Remember that the expressions chosen are used to exclude not include times from the firing of a trigger. In most cases a combination of the previous options will be easier to maintain. If a trigger is scheduled to fire every hour, eg. 00:00:00, 01:00:00, 02:00:00 etc. then you can use a CRON expression

```
0 0 /2 ? 0,6 *
```

to restrict the trigger to fire only once every two hours on weekends. The CRON sample does this by excluding 00:00:00, 02:00:00, 04:00:00 etc. for days 0 and 6 (Sunday and Saturday).

Note

You can attach more than one trigger to a job, so an alternative strategy for regular restrictions such as this would be to define a one hourly trigger for weekdays and a second two-hourly trigger for weekends. This would avoid any exclusion ranges and would perhaps be more flexible to maintain.

Chapter 5
Cookbook

Overview

This cookbook describes a number of typical uses for Elixir Schedule Designer.

Job Recipes

Render All Reports In A Folder

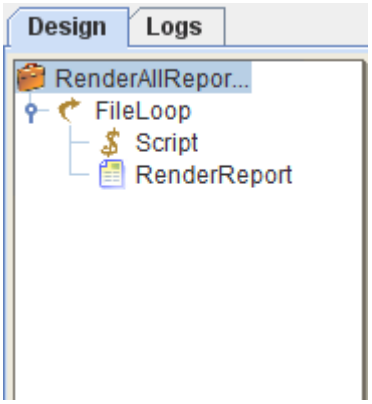
This section will show the user the steps to rendering all the reports in a specified folder into the desired output format. PDF will be the chosen output format in this case.

1. In a repository or folder, right-click and select `Add` and `Job`. Enter a unique name for the job. For instance, *RenderAllReport*, then click `Finish`. The job will be created successfully.

2. In `Design` tab, right-click on the job, select `Add`, then `FileLoop`.

3. Right-click on `FileLoop` and select `Script`. Then, add `RenderAllReport`. The structure will similar to Figure 5.1, “FileLoop Tree”.

   Figure 5.1. FileLoop Tree

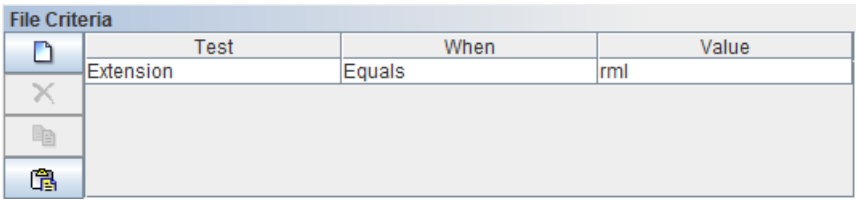
   

4. Click on *FileLoop*. In the `FileLoop` panel,

5. Click on *FileLoop*. In the `FileLoop` panel, enter the folder directory in in the `Folder:` field. User can also click on `...` to select the folder. Check `Recursive` if the files are contained within the folders of folders.

6. In the `File Criteria` panel, add a new criteria. In `Test`, choose *Extension* in the dropdown list. For `When` condition, select *Equals*. As for `Value`, enter *rml* like in Figure 5.2, “File Criteria”.

Figure 5.2. File Criteria



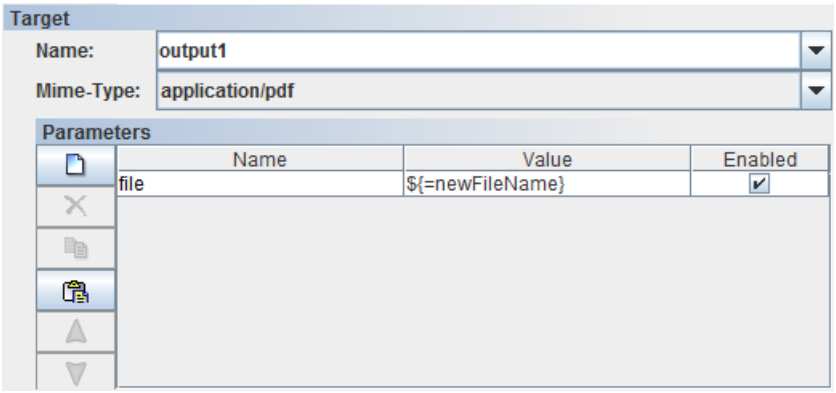
7. In `Script`, enter the following. This is to retain the directory structure:

```
newFileName = FileName.replace(/.rml/g,".pdf");
```

8. In the `Render Report` job, enter *${FileName}* for the `Name:` field in the `Report` panel. This is so that the name of the PDF generated will take on the name of the rml file.

9. For the `Target` panel, select `output1` and `application/pdf` for the output directory name and output mime-type from the dropdown list respectively. Inside the `Parameters` panel, edit the parameter value as seen in Figure 5.3, “Target Parameter”.

Figure 5.3. Target Parameter



10. If user wants to see the progress of file processed, user can add enter the path of a log file in the `Log File:` field under the `Log` tab.

11. Now, run the job. After the job has finished running, the generated PDF files can be found in /RepertoireServer/output/folder1, and a folder with the name of the repository will contain the generated PDF files.

Generate Data And Then Report On It

Trigger Recipes

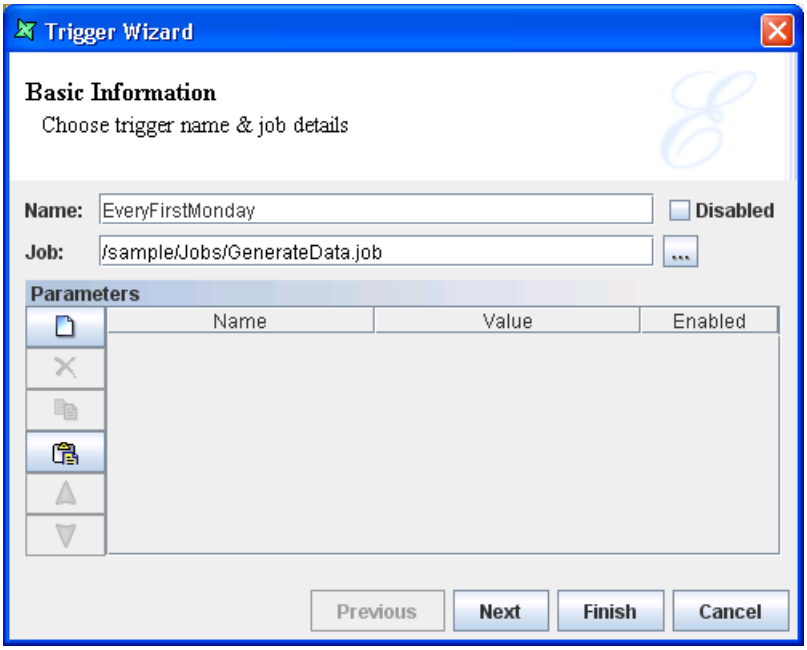
Trigger On The First Monday Of The Month

1. Before creating a trigger, user will need to have an existing job that is working perfectly.

2. When a working job is available, create a trigger by right-clicking on a filesystem or folder,select `Add` then `Trigger`.

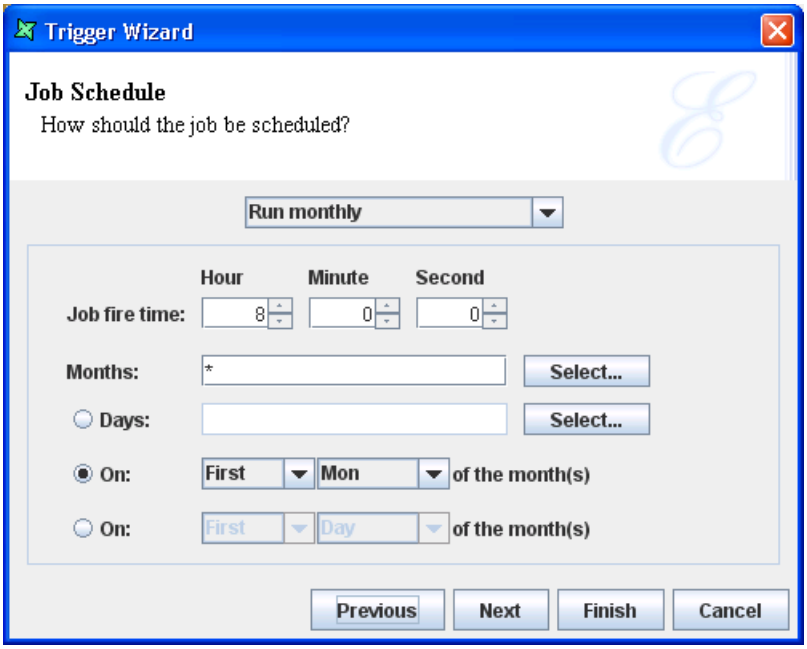
3. In the Trigger Wizard, enter a unique name for the trigger and select the job to be run as seen in Figure 5.4, “Trigger Wizard”. If the job requires parameters to be entered, it will be done in this page of the wizard.

Figure 5.4. Trigger Wizard



4. The next page of the Trigger Wizard is used to configure the schedule of the trigger, setting the time and frequency the trigger will run, like in Figure 5.5, “Job Schedule”. To run the trigger on every monday of the month, select `monthly` from the drop down list. For `Job fire time:`, enter the time the trigger should run on every monday of the month. For `Months`, enter * to imply all months. Else, click on `Select` and select the months respectively. Then, select the second radio button and select `First` for the first drop down list and `Mon` for the second drop down list.

Figure 5.5. Job Schedule



5. The following page of the wizard allows the user to configure the active period of the trigger. If the trigger needs to run forever, select the checkboxes at `Now` and `Never` at . Else, select configure the starting and ending time respectively.

6. The Trigger Calendar page is for users to load the calendar file, explicitly excluding particular dates. However, this is not compulsory. Users can continue without defining a calendar file.

7. Misfire Configuration page is on the next page. This is to let the trigger know what to do when a misfire occurs.

8. Last but not least, the Security page which is exactly the same for all elements in Repertoire Designer and Repertoire Remote Designer.

9. Clicking on `Finish` will complete the creation process.

10. This trigger will now run at 8a.m. on every first Monday of each month, never ending.

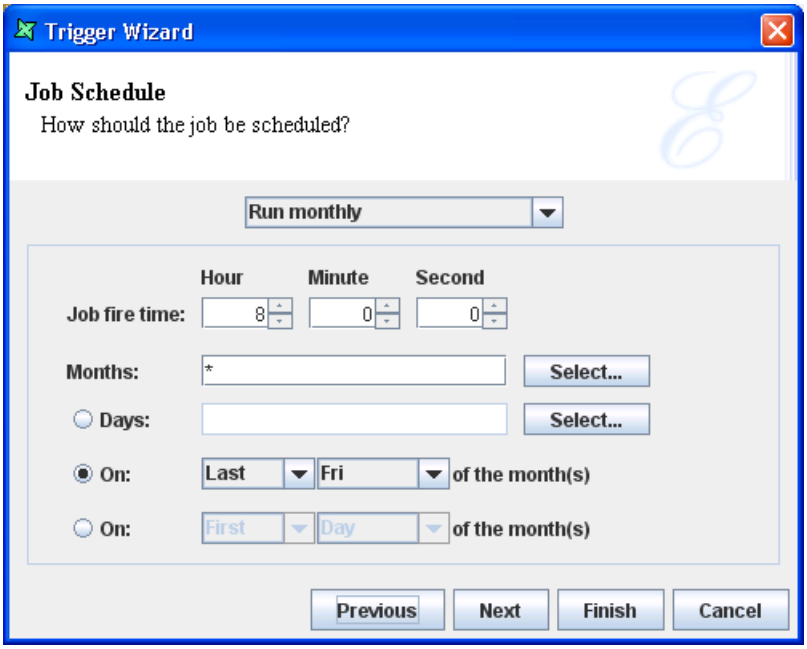
Trigger On The Last Friday Of The Month

This trigger is very similar to the one in the section called “Trigger On The First Monday Of The Month”.

1. Follow the instructions from step 1 to step 3 in the section called “Trigger On The First Monday Of The Month”.

2. In step 4 of the section called “Trigger On The First Monday Of The Month”, the configurations are done for every first Monday of the day. For now, it needs to be changed to every last Friday of the month. Simply, change *First* to *Last* and *Mon* to *Fri* in the dropdown list respectively, like in Figure 5.6, “Every Last Friday Of The Month”.

Figure 5.6. Every Last Friday Of The Month



3. The rest of the steps will be similar to the ones mentioned in the section called “Trigger On The First Monday Of The Month”, from step 5 onwards.

4. This trigger will now run at 8a.m. on every last Friday of the month, forever.

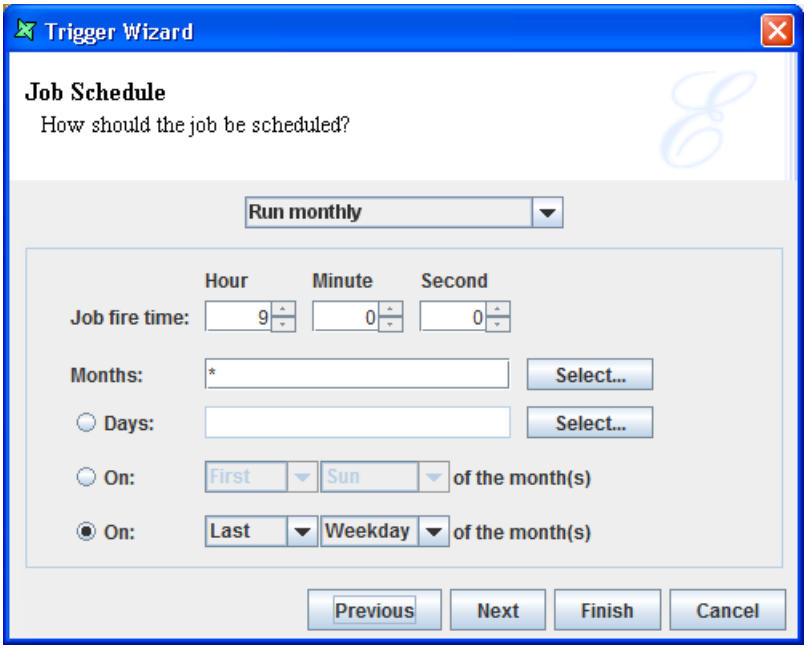
Trigger On The Last Working Day Of The Month

In this example, it is assumed that there are 5 working days in a week, which implies Monday to Friday.

1. Follow the instructions from step 1 to step 3 in the section called “Trigger On The First Monday Of The Month”.

2. In order to configure the trigger to run on every last working day of the month, change the settings to according to Figure 5.7, “Every Last Working Day Of The Month”.

Figure 5.7. Every Last Working Day Of The Month



3. Follow step 5 to 9 mentioned in the section called “Trigger On The First Monday Of The Month”.

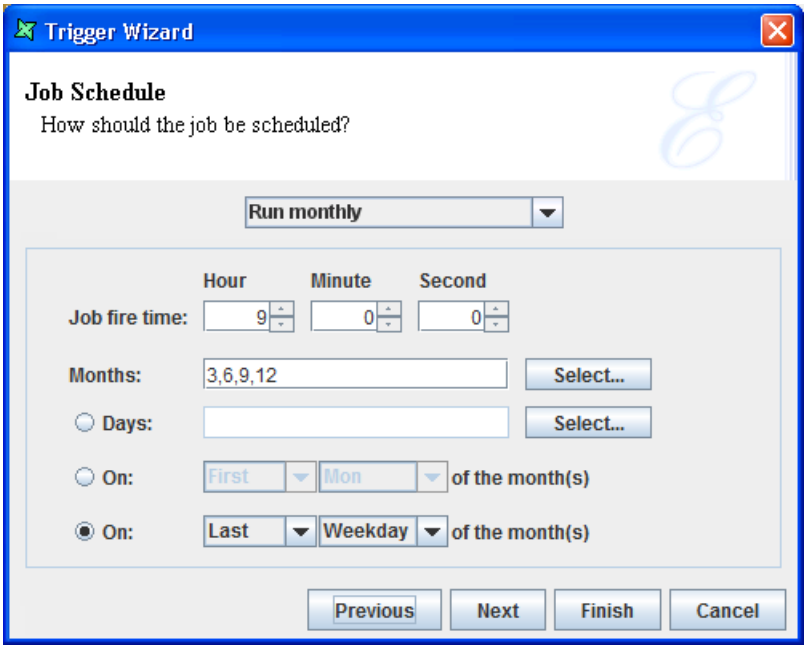
4. When created successfully, the trigger will run on every last working day of the month forever. If user decides not to run this trigger forever, it can be configured in the Trigger wizard.

Trigger On The Last Working Day Of The Quarter

1. Follow the instructions from step 1 to step 3 in the section called “Trigger On The First Monday Of The Month”.

2. In order to configure the trigger to run on the last working day of each quarter, change the settings as shown in Figure 5.8, “Every Last Working Day Of The Quarter”.

Figure 5.8. Every Last Working Day Of The Quarter



3. Follow step 5 to 9 mentioned in the section called “Trigger On The First Monday Of The Month”.

4. When the configurations are done correctly and saved, this trigger will run on every last working day of each quarter.