RMASConnector 1.0

RMASConnector

RMASConnector 1.0 RMASConnector Edition 1

Author support@certus-tech.com

Welcome to the home site of the Research Management and Administration System (RMAS) Connector project. RMASConnector is a data integration platform developed under the JISC funded Research Management and Administration System project. At the heart of the platform is the Pentaho Open Source Data Integration suite, PDI (also known as Kettle).

RMASConnector is based on an Extract, Transform and Load (ETL) pattern and is compatible with an Enterprise Service Bus (ESB) pattern, in which best practice requires a common message structure and vocabulary to ensure interoperability. The RMASConnector project therefore includes the definition of, and support for, suitable message structures. The chosen standard for these messages is CERIF, which has been refined and developed as necessary to accommodate RMAS requirements.

RMASConnector is available as a download from this site. Also available within the download are load-and-go worked examples of platform usage. These examples are designed to be easily configurable to work with existing data sources.

Overview

1.1. Project Background

The Research Management and Administration System (RMAS) project aims to facilitate the implementation of research management and administration systems within Higher Education Institutes.

As a part of this project, a number of data integration scenarios have been identified by the Universities of Exeter, Kent and Sunderland. These require a flexible tool to perform Extract, Transform and Load operations (ETL) in order to perform data handling and integration, including serialisation to CERIF, a standardised data interchange format used throughout Europe.

1.2. PDI (Pentaho Kettle)

A number of open source ETL tools were trialled and from these, the Pentaho Data Integration Community Edition (PDI CE), also known as Kettle, was selected. NB: For simplicity, the term *PDI* will be used throughout this document to refer to this software.

PDI uses the concept of *Steps* to represent actions performed on data, which can be arranged into *transformations*, allowing sequences of actions to be performed to manipulate data. transformations can be performed in sequence in a *job*, meaning that modular approach can be taken to data manipulation, while maintaining the power of the actions can be performed.

A graphical interface allows the user to see how the data flows as it is handled with jobs and transformations.

1.3. PDI Customisation and Worked Examples

To facilitate the use of PDI in the RMAS project, the concept of the CERIF-backed Common Data Model (CDM) is introduced. This is a model to which data can be mapped and passed between Transformations and can then be efficiently serialised into CERIF. Custom plug-ins have been developed for mapping data to the CDM and serialising the CDM to CERIF, and are provided with the release.

Also included with the release are five worked examples which represent some common data handling scenarios. These examples can be used to gain familiarity with the PDI interface, and can be customised to use other data sources as required. Each example is described in detail later in this document.

Installation

2.1. Overview

This chapter details how to install the RMAS Connector on your system. The installable files are available for *download*[1].

There are three zip files to download:

1. RMASCore.zip
2. RMASWorkedExamples.zip
3. RMASWebServicesExamples.zip

It is recommended that the files are downloaded and installed in this order.

2.2. Installation Procedure

Installation of the RMAS Connector is performed by extracting the zip archives to one of the locations specified below. Each of the RMAS Connector archives extracts into a root RMAS directory, and the archives are structured in such a way that their contents are combined when extracted.

On Linux systems it is recommended to install the RMAS Connector into your home directory, typically at

```
/home/username/RMAS
```

On Windows systems it is recommended to install the RMAS Connector into your user profile folder. To open your profile folder, click *Start -> Run* and enter

```
%USERPROFILE%
```

If your home directory (Linux) or user profile folder (Windows) are located on network storage then it is recommended to create a local directory and to install there instead, as running PDI from network storage can lead to significantly degraded performance. Furthermore, if you are unable to install the RMAS Connector to your home directory or user profile folder for any reason, it is necessary to modify a parameter (project.dir) in the worked examples accordingly.

2.3. RMAS Core

The RMASCore.zip file contains Pentaho PDI (Kettle) and some plugins which have been developed for the RMAS Connector.

To install, unzip the file into an appropriate location, as described in *Section 2.2, “Installation Procedure”*. PDI will be extracted into an RMAS directory at this location.

To run PDI in Linux, run

[1] http://rmas.certus-tech.com

```
RMAS/kettle/spoon.sh
```

or in Windows, run

```
RMAS\kettle\Spoon.bat
```

2.4. Worked Examples

The RMASWorkedExamples.zip file contains five worked examples and some sample services and input data. These examples may be used to familiarise yourself with the PDI interface and may be customised to work with your own data and services.

To install the worked examples, unzip the file into an appropriate location, as described in *Section 2.2, "Installation Procedure"*. The worked examples will be extracted into the RMAS/examplerepository directory.

To use the worked examples, you will need to connect to this repository in PDI. On starting PDI, a dialog box (see *Figure 2.1, "Connect to repository"*) asks you to connect to a repository. Click green + icon (Add) in the top right of the dialog box to create a new repository connection.

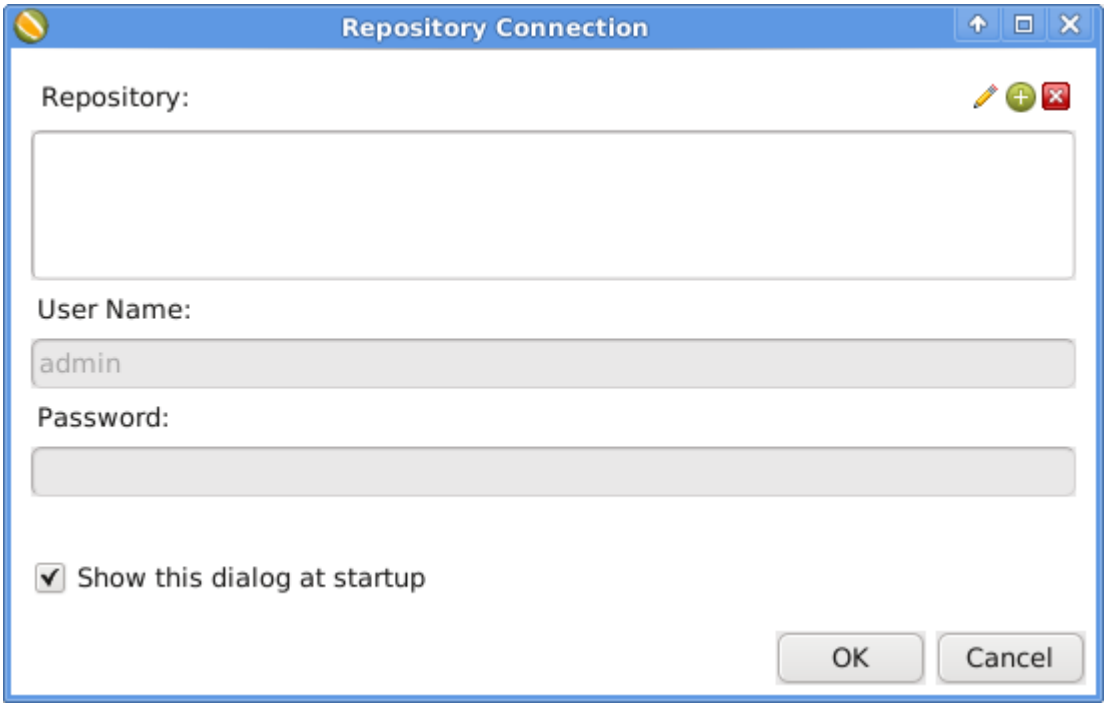

Figure 2.1. Connect to repository

Select *Kettle file repository* and click *OK*. In the next dialog box, select *Browse...* and navigate to and select the *examplerepository* directory. Enter "RMASConnector" in the Name and ID fields (see *Figure 2.2, "Configure repository"*) and click *OK*.

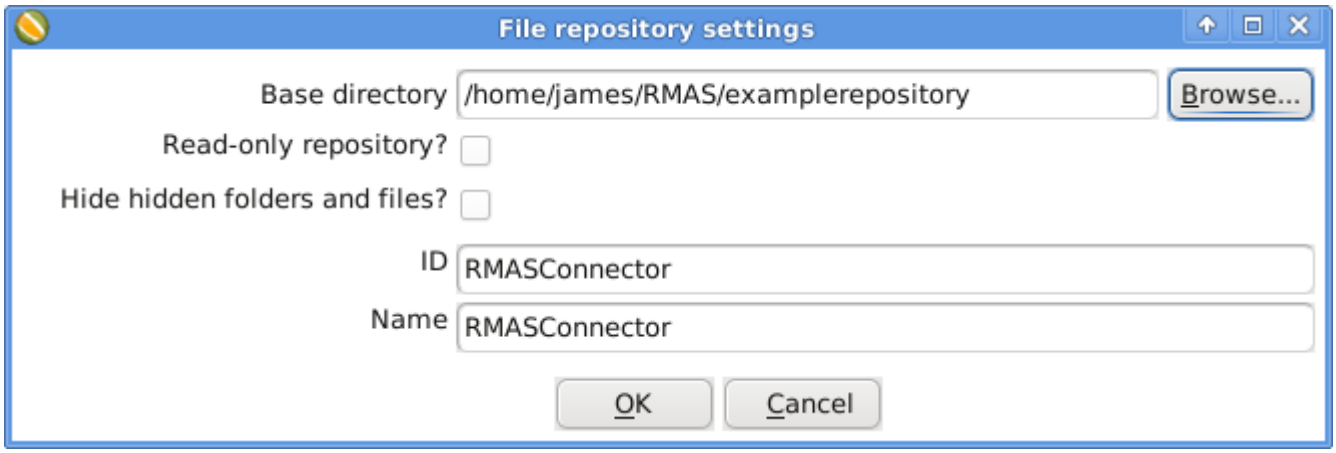

Figure 2.2. Configure repository

The new repository will appear in the Repository Connection list. Highlight it and click *OK* to connect to it. The main window of the PDI application will open, showing the welcome screen. At this point you have successfully connected to the repository, and you will now be able to access the worked example files by selecting *File -> Open...* and navigating through the repository.

2.4.1. Web Services

The RMASWebServicesExamples.zip file contains an Apache Tomcat application container instance and sample services for use with worked example 4. To install the web services bundle, unzip the file into an appropriate location, as described in *Section 2.2, "Installation Procedure"*. The bundle extracts into the RMAS/examplewebservices directory.

Note that while no further configuration is typically required to allow Worked Example 4 to be run, the Tomcat bundle will attempt to listen for HTTP connections on port 8080, and will fail to start up if this port is already in use.

The bundled Apache Tomcat must be running for the WorkedExample4 to operate. On Linux you can start and stop Tomcat by running

```
RMAS/exampleservices/apache-tomcat-7.0.27/bin/startup.sh
RMAS/exampleservices/apache-tomcat-7.0.27/bin/shutdown.sh
```

On Windows systems use

```
RMAS\exampleservices\apache-tomcat-7.0.27\bin\startup.bat
RMAS\exampleservices\apache-tomcat-7.0.27\bin\shutdown.bat
```

2.4.1.1. Repository ID

The web services for worked example 4 expect the PDI repository containing the worked examples to have been given the ID *RMASConnector*. The repository ID will need to be changed in PDI if something else has been used. To change the repository ID, select the repository in the Repository Connection window (shown in *Figure 2.1, "Connect to repository"*), and click on the pencil icon above and to the right. Change the ID to RMASConnector and click OK.

Configuration

3.1. Jobs and Transformations

PDI allows creation of two file types: jobs and transformations. A transformation is used to describe the data flows for an Extract-Transform-Load (ETL) process, such as reading from a source, transforming data and loading it into a target location.

A job is used to coordinate ETL processes. A job might be used, for example, to wait for a file to be created in a directory, trigger an ETL process and finally send an email to communicate success or failure of the process. A job may utilise several transformations to achieve its goal.

3.2. PDI Repository

PDI uses a data repository to store jobs, transformations and metadata. The repository can be either a relational database or a folder on disk. To connect to a repository in PDI go to *Tools->Repository->Connect...* or alternatively press *CTRL+R*, then choose the repository you want to connect to.
For more information on setting up a repository, please refer to the Installation section or the *PDI documentation*[1].

3.3. Steps and Hops

In PDI, transformations and jobs are made up of steps for performing various extract, transform and load operations. They are, in essence, the building blocks of the transformations and jobs. Steps are categorised based on their purpose; for example, there are input steps for reading in data, output steps for outputting data and transform steps for transforming and manipulating the data. See *PDI Steps guide*[2] for a list of all of the available steps.

Figure 3.1. An example of a hop between two steps

Hops are links between steps and describe the flow of the ETL process. You can create a hop between two steps by clicking the step you wish to start the hop from, press and holding *shift*, and draw a line to the corresponding step. By clicking the hop you can disable/enable it - disabled hops appear greyed out. Some hops are conditional and represent the flow of the ETL process under particular circumstances - for example when an error occurs. These hops are normally coloured, or have an icon over them to indicate their purpose.

3.4. Previewing a Step

Some PDI steps allow you to preview the data at that particular stage in the process. To do so, *right-click* on the step you want to preview and select *Preview*. A dialog will appear. Click *Quick Launch* and PDI will attempt to preview the data for you. Note that you can only preview steps within a Transformation, not entries within a Job.

[1] http://infocenter.pentaho.com/help/topic/getting_started_with_pdi/task_pdi_connecting_to_per.html
[2] http://wiki.pentaho.com/display/EAI/Pentaho+Data+Integration+Steps

3.5. Executing a Transformation or Job

Jobs and transformations can be executed within the Spoon interface by clicking the green run arrow shown in *Figure 3.2, "Spoon run controls"*.

Figure 3.2. Spoon run controls

This will bring up a window allowing you to configure the execution before it is run. Here you can override job parameters and set the level of logging to be displayed. Click *Launch* to then execute the job or transformation.

PDI Implementation

4.1. Parameterisation

All of the worked examples make use of parameters to allow the jobs and transformations to be configurable. Each transformation may specify its own parameters with default values, but these may be overridden by defaults specified at the job level. The job level defaults may in turn be overridden at runtime, for example with values specified on the command line.

Parameters can be specified in *Spoon* using the properties editor for a job or transformation. While editing a transformation in Spoon, press *CTRL+T* and while editing a job press *CTRL+J*. Then in the dialog select the *Parameters* tab to access parameter configuration.

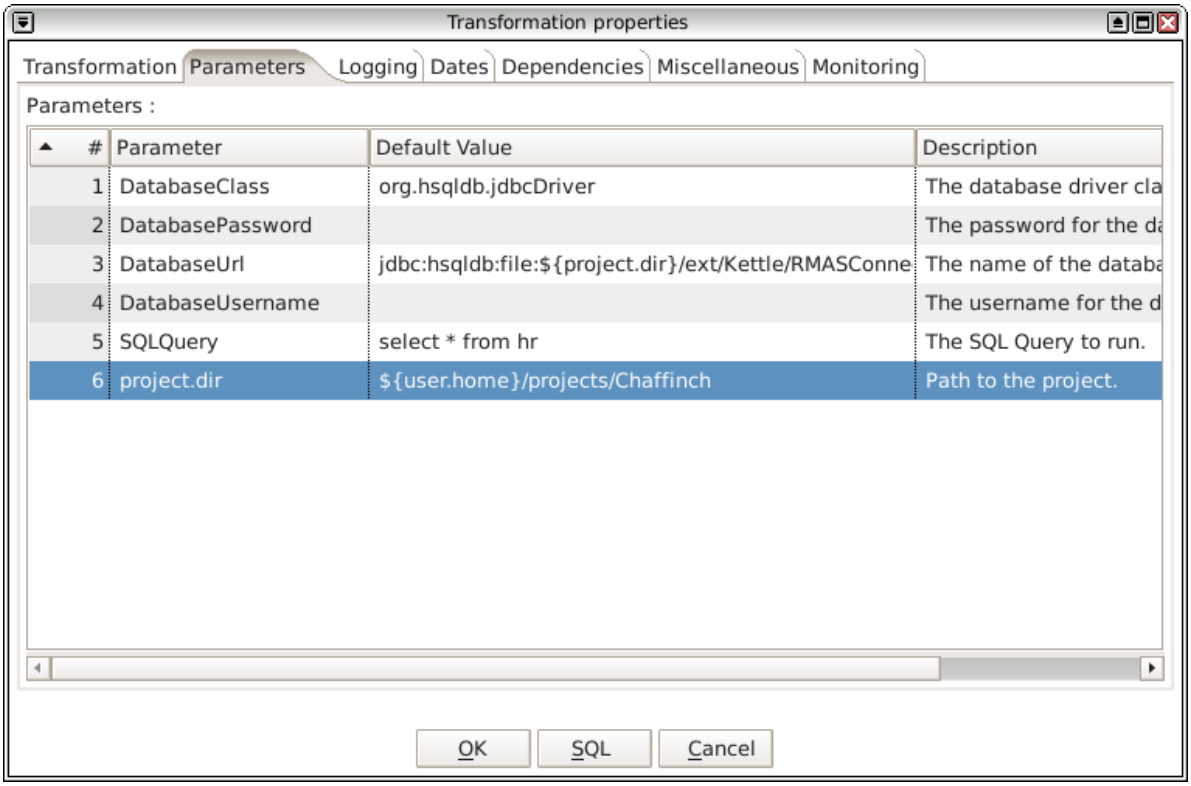

Figure 4.1. The parameters tab

You will see a table showing the parameter name, default value and description. The default values specified in transformation parameter configuration can be overridden by specifying the same parameter at the job level. The parameters specified in the job are by default passed to the transformations which the job uses.

The worked examples make use of parameters to configure their input and output stages, such as values required to set up a database connection, or the location and filename of an output file. Thus if a different input source is required (such as using a different database) it is only necessary to go to the Job parameters and change the relevant parameters.

All of the worked examples define a *project.dir* parameter. This defines the root directory under which all of the other example files may be found or are created. The worked examples all specify input and output filenames relative to project.dir, and should operate 'out of the box' with their default project.dir

setting if the RMAS Connector has been installed into one of the recommended locations identified in *Section 2.2, "Installation Procedure"*.

4.2. CERIF-based Common Data Model Steps

The RMAS Connector includes customised transformation steps for working with the CERIF-based Common Data Model (CDM). These extend the standard functionality provided by PDI.

4.2.1. Map to CDM

This step acts as a mapping step to map from a set of input fields to CDM fields. The step is accessed in the *Design* window by choosing *Transform->Map to CDM*.

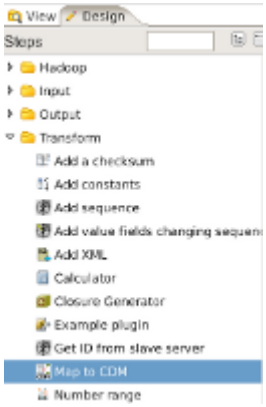

Figure 4.2. The Map to CDM plugin

This step can be configured to map *from* any input fields, but can only map *to* CDM fields. In the step properties there are three tabs: Select & Alter, Remove and Meta-data.

The *Select & Alter* tab allows for fields to be selected for inclusion, and renamed. The *Get fields to select* button on the right-hand-side of the window can be used to automatically determine the available fields from previous steps.

The *Remove* tab is used to remove fields from the stream. This is useful to prevent unwanted fields from an input source being mapped to CDM (for example if there is no appropriate field to map to), or to remove temporary calculation fields from earlier steps.

The *Meta-data* tab allows definition of the mappings from the input fields to the CDM Fields. Select an input field in the *Fieldname* column, and choose a CDM field to map to from the dropdown in the *CDM Field* column.

4.2.2. Get CDM fields

The *Get CDM fields* step is used to retrieve CDM fields made available by an earlier transformation in the same Job.

When it completes, a transformation may export rows of data to the job it executed within. In the worked examples it is typical to have an input transformation which, for example, queries a database. This will then perform any necessary data preparation and use the *Map to CDM* step to produce a stream containing only rows of CDM fields. The input transformation then ends with a *Copy rows to result* step to export these rows. If the transformation is run in a job, the rows may be picked up by any subsequent transformations by using the *Get CDM fields* step.

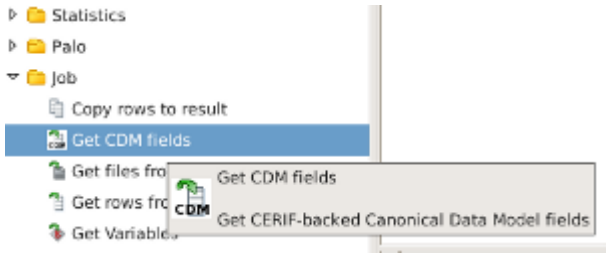

Figure 4.3. The Get CDM fields plugin

This step cannot be configured and only imports CDM fields into the transformation.

4.2.3. Convert CDM to CERIF

Convert CDM to CERIF is used to convert all supported CDM fields to valid CERIF version 1.4. The step is accessed in the *Design* window by choosing *Output->Convert CDM to CERIF*.

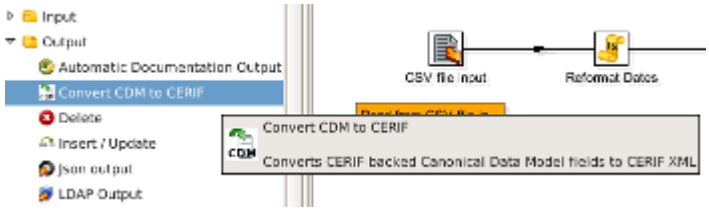

Figure 4.4. The Convert CDM to CERIF plugin

No configuration of this step is necessary and it will automatically serialise all supported CDM fields to valid CERIF XML. This step also performs validation of the resulting document against the CERIF version 1.4 schema. As a result, it is not possible that the resulting XML document contains invalid CERIF if this step completes without reporting any errors.

4.3. Command-Line Invocation

All of the worked examples are intended to be executed in the Spoon graphical environment, but may also be run from the command line. PDI provides a command-line tool called Kitchen for this purpose.

Kitchen utilises the same repository configuration details as Spoon, so once a user has set up the RMAS repository in Spoon they may also run jobs from the command line without further configuration.

Kitchen is found in the root of the PDI installation; in Linux distributions it is located at

```
RMAS/kettle/kitchen.sh
```

or in Windows, at

```
RMAS\kettle\Kitchen.bat
```

The following command lines demonstrate how to execute a worked example on both Linux and Windows. Note that the worked example job files each reside in their own subdirectory within the repository. Also note that the quote marks around the repository name are optional in this case, but they must be included if the repository name contains spaces.

Linux:

```
kitchen.sh -rep="RMASConnector" -dir=/WorkedExample2/ -job=WorkedExample2 -param:input.file=/
var/test/data.csv
```

Windows:

```
Kitchen.bat /rep:"RMASConnector" /dir:/WorkedExample2 /job:WorkedExample2 /param:input.file:c:
\tmp\data.csv
```

Worked Example 1: HR database to Symplectic CSV

5.1. Overview

Worked example 1 demonstrates how human resource data can be extracted from one data source ready for import into another: in particular, how to extract data from a Human Resources (HR) database ready for loading it into Symplectic. The data involved contains some personal details and also academic details such as department and employment.

Because the source schemas differ, the data must be mapped from one schema to the other in order to ready the data for import.

5.2. User Story

A project manager wishes to import data from their institution's HR database to their Symplectic database.

The HR database contains personnel information including details about academic status, such as department, school and whether they teach and/or research. They have been asked to import data from the HR database into the institutionâ□□s Symplectic database. The manager must therefore extract the HR data and deliver it in a form suitable for import into Symplectic.

The manager starts Spoon and opens the HR to Symplectic job. They modify the job parameters to specify the details of their HR database and the location of the output CSV file. They run the job and a CSV is produced, containing the entries from the HR database in a Symplectic format. The CSV file also contains an extra required field detailing whether each user is an academic. The manager imports this file into Symplectic.

5.3. Structure of the Example

Worked example 1 consists of three PDI components: one job (WorkedExample1) and two transformations (HRDatabaseInput, SymplecticCSVOutput). These are located in the WorkedExample1 subdirectory of the PDI repository.

Worked example 1 may be used in the context of the *Spoon* graphical front end to PDI, by loading the job and executing it directly. It may also be executed from a command line.

5.3.1. Parameters

The following parameters are defined by the *WorkedExample1* job and used by the transformations:

Parameter	Description
DatabaseClass	The database driver class name.
DatabaseUrl	The URL for the database connection.
DatabaseUsername	The username for accessing the database.
DatabasePassword	The password for accessing the database.

Parameter	Description
SQLQuery	The query to execute on the database.
output.file	The path to the file which is generated by the process. If the file does not exist, it will be created automatically.
project.dir	The root directory for the project.

Table 5.1.

5.4. Running the job from Spoon

This section describes how to configure and/or run the job from the Spoon visual editor.

5.4.1. Job Structure

Figure 5.1. The worked example 1 job

This job comprises two transformation steps. *Read HR database* reads in the data from a specified HR database and maps this to a CERIF-based Common Data Model (CDM). The resulting data is then passed to the second step, *Output CSV for Symplectic*, which maps from CDM to the Symplectic schema and outputs a CSV file ready to be imported into a Symplectic database.

The HR database being used by the job is configured in the job settings. One parameter is used to specify the database configuration; another is used to define the name of the CSV output file.

5.4.2. Read HR database

Reading the HR database is achieved using a transformation which contains three steps:

Figure 5.2. Read HR database transformation

The *Query Database* step performs a query on the configured HR database using parameters specified in the Worked Example 1 job. The *Map to CDM* step maps the fields we want to import to Symplectic, removing any others. Finally, the data is made available to the *Output CSV for Symplectic* transformation.

5.4.3. Output CSV for Symplectic

The output transformation to create the CSV file with valid Symplectic schema data comprises a number of steps.

Figure 5.3. Get CDM fields

First the transformation reads in the input data output by the *Read HR database* transformation using the *Get CDM fields* step.

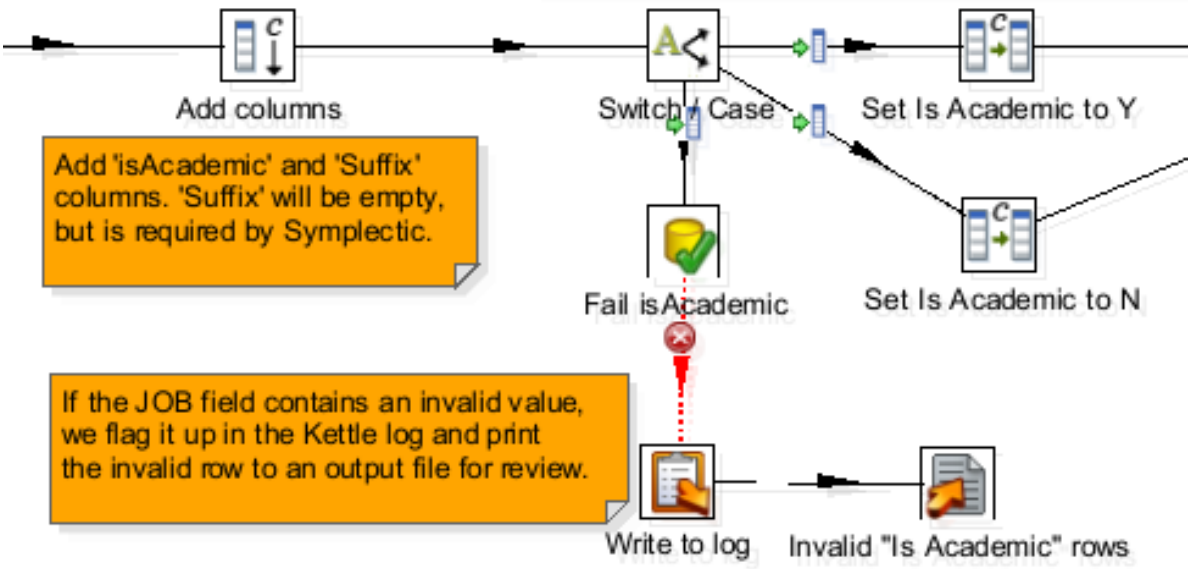

Figure 5.4. Set Is Academic field

Secondly, a new field *Is Academic* is created. Its value is determined by the value of the JOB field. If the value of this field is "Research only", "Teaching only" or "Research and Teaching", then the new *Is Academic* field is set to "Y". If the value is equal to "Not teaching and/or research", then the value is set to "N". Any other value for this field is considered an error and so results in a log entry.

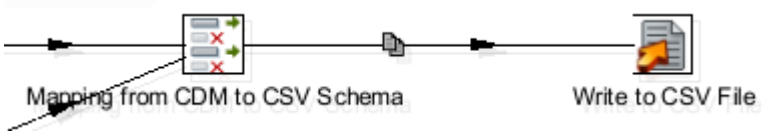

Figure 5.5. Write output to CSV

The *Mapping from CDM to CSV Schema* step then completes the mapping of the data. This is then output as a CSV file by the final step in the transform.

5.5. Running the job from the command line

You can also run the job from the command line, either as the downloaded worked example or after configuring and saving it using Spoon. The following shows how to run the command from both a linux environment and a Windows environment, using the *Kitchen* tool which is provided with PDI.

5.5.1. Running the job from a Linux command line

Navigate to the RMAS/kettle directory:

```
cd RMAS/kettle
```

Ensure that the kitchen.sh script is executable:

```
chmod u+x kitchen.sh
```

Run the job using the following command. Note that the quote marks around the repository name are optional, but they *must* be included if the repository name contains whitespace:

```
./kitchen.sh -rep="RMASConnector" -job=WorkedExample1/WorkedExample1
```

5.5.2. Running the job from a Windows command line

Navigate to the RMAS/kettle directory:

```
cd RMAS\kettle
```

Run the job using the following command. Note that the quote marks around the repository name are optional, but they *must* be included if the repository name contains whitespace:

```
Kitchen.bat /rep:"RMASConnector" /job:WorkedExample1/WorkedExample1
```

5.6. A note on configuring the worked example

Worked example 1 gets its input by running a select query on a Human Resources database. This can be configured by ensuring that all the database parameters are set at the job level. Default values for these parameters have been entered for the sample database which is provided as part of the example (RMAS/exampledata/WorkedExample1/WorkedExample1.script). When configuring your own database, you will need to ensure that the *Map to CDM* step in the *Read HR Database* transformation maps the column names in the database to the appropriate CDM fields.

The output is a single CSV document, which is created at the location specified in the output.file parameter (the default is tmp/Example1CSVOutput.csv). The CSV will contain data which is compliant with the Symplectic schema.

Worked Example 2: Publication details - Symplectic to Converis

6.1. Overview

Worked example 2 demonstrates how data from a Symplectic CSV export, containing details of publications, can be mapped to CERIF 1.4 XML. The goal might be to integrate with CERIF XML data from other sources, using PDI or some other CERIF-compliant integration tool such as Converis.

6.2. User Story

A technical manager maintains a Symplectic system populated with details of publications including names and unique identifiers of authors, publication titles, dates of publication, page numbers, etc. The manager also maintains a Converis system for marshalling the university's research management data, and wishes to import the publication details into it. To import into the Converis system it is necessary to produce a valid CERIF XML representation of their Symplectic data.

Having first exported the required information from Symplectic to a CSV file, the manager starts PDI and opens the Symplectic to Converis job. They ensure that the job is configured to access their Symplectic export file. They then run the job, which generates a CERIF XML file containing the Symplectic data.

6.3. Structure of the Example

Worked example 2 consists of three PDI components: one job (WorkedExample2.kjb) and two transformations (SymplecticCSVInput.ktr, CERIFXMLOutput.ktr). These are located in the WorkedExample2 subdirectory of the PDI repository.

Worked example 2 may be used in the context of the *Spoon* graphical front end to PDI, by loading the job and executing it directly. It may also be executed from a command line.

6.3.1. Parameters

The following parameters are defined by the *WorkedExample2* job and used by the transformations:

Parameter	Description
input.file	The path, including the filename, to the Symplectic CSV export which is read by the CSV input transformation.
output.file	The path to the file which is generated by the process. If the file does not exist, it will be created automatically.
project.dir	The root directory of the project.

Table 6.1.

6.4. Running the job from Spoon

This section describes how to configure and/or run the job from the Spoon visual editor. Worked example 2 may be used in the context of *Spoon* by loading the job and executing it directly. As distributed, it is configured with default parameter values that will allow it to run without modification from a sample Symplectic CSV file.

6.4.1. Job Structure

Figure 6.1. The worked example 2 job

Worked example 2 consists simply of reading an input file into the CERIF-based Common Data Model (CDM) and serialising to CERIF XML. No intermediate processing is required. There are therefore just two entries in the job: *Read Symplectic CSV* and *Write CERIF XML*.

6.4.2. Read Symplectic CSV

Reading Symplectic CSV is achieved with the following transformation:

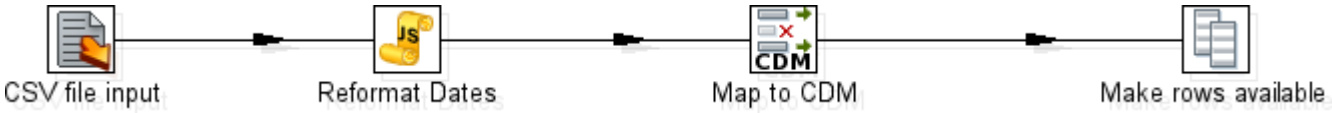

Figure 6.2. The Read Symplectic CSV transformation

The first step reads from the CSV file, and introduces the configured fields into the transformation. As CSV is intrinsically similar to the way data is handled in transformations - as a series of rows with a defined set of fields (columns) - no special handling is required, and a standard PDI *CSV Input* step is used. Although all rows and columns in the input data are read by this step, only the field names defined in the CSV Input step are added to the stream.

Next, a JavaScript step is used to convert any dates in a dd/MM/yyyy format into the xs:dateTime format which is required by CERIF (yyyy-MM-dd). Dates with times are also converted into a CERIF-compatible format. These converted dates and dateTimes are added as new fields to each row.

A *Map to CDM* step then maps from the local fields to CDM fields. In worked example 4 the *Map to CDM* step is configured to:

- Remove the now spurious original date fields loaded from the CSV file. This is configured via the field listing on the *Remove* tab.
- Map the remaining fields to *CDM Fields* via the *Meta-data tab*.

The *Make rows available* step allows the *Post CERIF to Web Service* transformation to access the data produced by this transformation.

6.4.3. CERIF XML Output

The content of the CDM is serialised to a valid CERIF 1.4 XML document with the following transformation:

Figure 6.3. The CERIF XML Output transformation

The *Get CDM Fields* step makes the CDM data from the *Read Symplectic CSV* transformation available to this transformation. The *Convert CDM to CERIF* step serialises the CDM data to CERIF XML and validates it. This step produces a single row with a single field, containing the entire XML document. The final step in the transform writes the content of this field into a plain text file.

6.5. Running the job from Spoon

You can also run the job from the command line, either as the downloaded worked example or after configuring and saving it using Spoon. The following shows how to run the command from both a linux environment and a Windows environment, using the *Kitchen* tool which is provided with PDI.

6.5.1. Running the job from a Linux command line

Navigate to the RMAS/kettle directory:

```
cd RMAS/kettle
```

Ensure that the kitchen.sh script is executable:

```
chmod u+x kitchen.sh
```

Run the job using the following command. Note that the quote marks around the repository name are optional, but they *must* be included if the repository name contains whitespace:

```
./kitchen.sh -rep="RMASConnector" -job=WorkedExample2/WorkedExample2
```

6.5.2. Running the job from a Windows command line

Navigate to the RMAS/kettle directory:

```
cd RMAS\kettle
```

Run the job using the following command. Note that the quote marks around the repository name are optional, but they *must* be included if the repository name contains whitespace:

```
Kitchen.bat /rep:"RMASConnector" /job:WorkedExample2/WorkedExample2
```

6.6. A note on configuring the worked example

Worked example 2 ingests an export from Symplectic, in the form of a CSV file. For details of the columns which are expected in the CSV, refer to the sample data file at RMAS/exampledata/WorkedExample2/SymplecticCSV.csv.

CSV files with a different schema may be substituted. Doing so requires the following modifications to the input transformation:

- Update the field names as appropriate on the CSV Input step - note that the field names defined here do not have to match the column names in the CSV file; in Symplectic exports it is the position of the columns which is important.
- Update or remove the date reformatting step, as appropriate. If date conversion is required then the input and output formats may be altered by changing the variables at the top of the script. If no reformatting is required, the step may be removed entirely or routed around.
  - To route around a step, redirect the input hop to target the subsequent step and then disable the output hop from the step.

The output from the worked example is a single XML document, which by default is produced at RMAS/output/Example2CERIFOutput.xml. The XML document will contain CERIF 1.4 data which has been validated against the CERIF 1.4 schema.

Worked Example 3: Key mapping across multiple tools

7.1. Overview

Worked example 3 demonstrates how data from multiple sources can be joined even when sources maintain different identifying fields (keys). For example, one may wish to aggregate personnel across Human Resources (HR) and project management databases.

Assuming that the keys for data entries in one source of interest are different from those in another, this example demonstrates how a key mapping file can be utilised. Such a file maintains relationship between keys in different data sets.

7.2. User Story

A project manager wishes to find the contact details of staff who have released publications in the last twelve months.

They have been provided with a Symplectic export file (a CSV file) which lists publications from the last twelve months, where the authors are identified by keys which are unique to Symplectic. They also have access to the HR database, where each member of staff is identified by a unique HR key. Finally, they have a mapping file (again in the form of a CSV file), which identifies which key in the HR database relates to map which key in Symplectic.

The project manager starts PDI and opens the Key Mapping job. They modify the job and transformation parameters to specify the details of the HR database, the locations of their Symplectic and key mapping CSV files, and the location of an output file. They run the job, which joins the publication data with HR data, generating .a CSV file with the required information.

7.3. Structure of the Example

Worked example 3 consists of two PDI components: a job (WorkedExample3.kjb) and a transformation (MapAndAggregateData.ktr). These are located in the WorkedExample3 subdirectory of the PDI repository.

Worked example 3 may be used in the context of the Spoon graphical front end to PDI, by loading the job and executing it directly. It may also be executed on the command line.

7.3.1. Parameters

The following parameters are defined in the *WorkedExample3* job and used by the transformations:

Parameter	Description
CSVFile	The path to the CSV file containing publication data.
DatabaseClass	The database driver class name.
DatabaseUrl	The URL for the database connection.

Parameter	Description
DatabaseUsername	The username for accessing the database.
DatabasePassword	The password for accessing the database.
SQLQuery	The query to execute on the database.
mapping.file	The path to the CSV file containing key mappings.
output.file	The path to the file which is generated by the process. If the file does not exist, it will be created automatically.
project.dir	The root directory for the project.

Table 7.1.

7.4. Running the job from Spoon

This section describes how to configure and/or run the job from the *Spoon* visual editor.

7.4.1. Job Structure

Figure 7.1. The worked example 3 job

The Worked example 3 job uses a single transformation, *MapAndAggregateData*, to perform the key mapping and join.

7.4.2. Aggregate Data

The *MapAndAggregateData* transformation uses two streams. The top stream performs a query on the HR database, and then removes fields which are not required, while mapping others to a CERIF-backed Common Data Model (CDM). The rows are then sorted ready for joining:

Figure 7.2. The MapAndAggregateData transformation top stream

As an alternative, the fields removed by the *Map To CDM* step could have been omitted by altering the SQLQuery parameter.

The bottom stream extracts data from a Symplectic CSV file. It removes unwanted fields and maps the remainder to the CDM. A lookup step is then performed, where the key for each row in the Symplectic data is looked up in the mapping CSV file, and the corresponding HR key is added to the row:

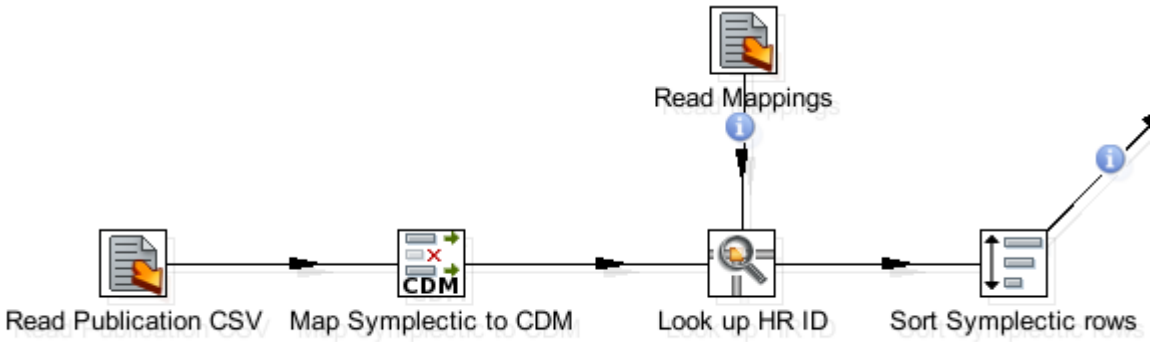

Figure 7.3. The MapAndAggregateData transformation bottom stream

Prior to the join, the data in the stream is again sorted on the HR key. This is necessary in order to correctly perform a merge join on the HR key:

Figure 7.4. The MapAndAggregateData transformation join

The joined data is then saved to a CSV file.

7.4.3. A note on scalability

The previous section describes how data can be aggregated from two sources. In order for this transformation to aggregate data from additional sources, a new stream with a mapping step must be added for each additional source. Because a merge step only allows two inputs, new merge steps need to be added to merge the data from each new stream with the merged data from the inital two streams. As such, the complexity of this transformation increases for each additional source.

7.5. Running the job from the command line

You can also run the job from the command line, either as the downloaded worked example or after configuring and saving it using Spoon. The following shows how to run the command from both a linux environment and a Windows environment, using the *Kitchen* tool which is provided with PDI.

7.5.1. Running the job from a Linux command line

Navigate to the RMAS/kettle directory:

```
cd RMAS/kettle
```

Ensure that the kitchen.sh script is executable:

```
chmod u+x kitchen.sh
```

Run the job using the following command. Note that the quote marks around the repository name are optional, but they *must* be included if the repository name contains whitespace:

```
./kitchen.sh -rep="RMASConnector" -job=WorkedExample3/WorkedExample3
```

7.5.2. Running the job from a Windows command line

Navigate to the RMAS/kettle directory:

```
cd RMAS\kettle
```

Run the job using the following command. Note that the quote marks around the repository name are optional, but they *must* be included if the repository name contains whitespace:

```
Kitchen.bat /rep:"RMASConnector" /job:WorkedExample3/WorkedExample3
```

7.6. A note on configuring the worked example

This example requires a connection to a Human Resources database. This can be configured by ensuring that all the database parameters are set at the job level. Default values for these parameters have been entered for the sample database which is provided as part of the example (RMAS/exampledata/WorkedExample3/WorkedExample3.script). When configuring your own database, you will need to ensure that the *Map to CDM* step in the *MapAndAggregateData* transformation maps the column names in the database to the appropriate CDM fields.

A CSV file of the required publication must be supplied. A sample has been provided as a part of the example and can be found at RMAS/exampledata/WorkedExample3/SymplecticInput.csv. The path to the files must be set as the value of the CSVFile parameter at the job level. As with the database, you should ensure that the *Map Symplectic to CDM* step maps the publication data to the appropriate fields.

Finally, a CSV mapping file is required. This identifies which keys in the publication data map to which in the HR database. The column headers used for this are “mapping_key” and “HR_key” respectively. An example file is provided at RMAS/exampledata/WorkedExample3/SymplecticKeyMapping.csv.

On running the job, the joined data is output to a file in the location specified by the output.file parameter. An outer join is performed, so if there is Symplectic data with no corresponding HR data, or vice versa, it will still appear in the output.

Worked Example 4: Event data SOAP wrapper to XML payload

8.1. Overview

Worked example 4 demonstrates job execution in response to a trigger from a third party. To start the process, a transformation is run which posts data into a web service. The web service acts as a listener endpoint, such as may be attached to an Enterprise Service Bus (ESB). It invokes a job to convert the data to CERIF XML and post it onwards to a second web service, via a SOAP interface.

8.2. User Story

An information systems manager maintains an Enterprise Service Bus (ESB) to facilitate the integration of services. The manager already has a service which may be configured to respond to important activities - such as the insertion of a new record in the HR database - and wishes to trigger data integration by invoking a remote web service. The remote web service ingests CERIF XML, and may be invoked by posting a SOAP message via HTTP.

The user first configures their local service to post a message containing the newly inserted data to the ESB. The ESB is then configured to forward the message content to a RESTful web service endpoint.

The web service receives the message content and invokes the RMAS Connector, which handles conversion of the data to CERIF XML and invokes the remote web service via SOAP.

8.3. Structure of the Example

Worked example 4 consists of a transformation which posts some content to the 'listener' web service endpoint (*PostFileToTriggerService*), and a main job which performs the conversion to CERIF XML and posts the result to the final web service (*WorkedExample4*). The *WorkedExample4* job comprises two transformations: *SymplecticCSVInput* and *CERIFXMLToSOAPOutput*.

Figure 8.1, "Process overview of worked example 4" aims to clarify the structure of the worked example. It indicates the four major steps which occur, the means by which they are connected, and the data which passes between them.

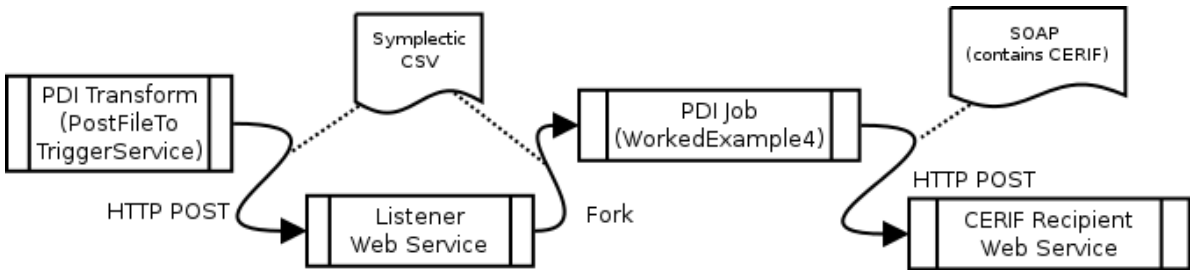

Figure 8.1. Process overview of worked example 4

8.3.1. Parameters

The *PostFileToTriggerService* transformation has the following parameters:

Parameter	Description
datafile	The path, including the filename, to the file which is posted to the listener web service. This should be a Symplectic CSV publication data export, or compatible.
service.url	The URL to post the file to. This is the URL of the listener service.
project.dir	The root directory for the project.

Table 8.1.

The *WorkedExample4* job has the following parameters:

Parameter	Description
input.file	The path, including the filename, to the Symplectic CSV export which is read by the CSV input Transformation. When the Job is invoked by the listener web service, this parameter is overridden on the command line.
soap.template	The path, including the filename, to the XML document containing the complete SOAP message template. The template may contain the placeholder *CERIF_DATA* at the point at which the CERIF XML serialisation should be inserted.
webservice.url	URL to the web service which the SOAP message is posted to.
webservice.action	URI of the web service action to invoke.
project.dir	The root directory of the project.

Table 8.2.

8.4. Running the job from Spoon

This section describes how to configure and/or run the job from the Spoon visual editor.

8.4.1. Triggering Transformation

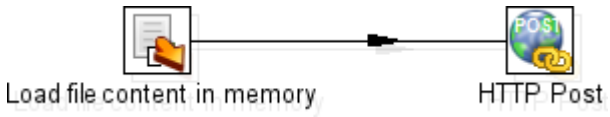

Figure 8.2. Overview of the triggering transformation

The first step reads a file from disk and puts its contents into a single field. This step requires minimal configuration: simply the file(s) to load.

The next step performs an HTTP post, and is configured to post the entire file read as the HTTP request body. The service URL it posts to - the 'listener' service - expects to receive data in this way - no headers or other data are required.

8.4.2. Listener Web Service

The listener web service is a simple REST-like service which receives content directly in the body of an HTTP POST. An example has been included in the RMAS Web Services Examples package and is available whenever the associated Apache Tomcat instance is running.

Note for Windows users: although the Transformation described in *Section 8.4.1, "Triggering Transformation"* can invoke this service, the service is unable to launch the job described below as a process, i.e. the job cannot be triggered externally in Windows. However, the job can be run manually in both Windows and Linux environments, and the following sections apply to both environments.

8.4.3. Main Job

The main job comprises two transformation entries:

Figure 8.3. The worked example 4 job

8.4.4. Read Symplectic CSV

The first transformation is a CSV input transformation which loads CSV data and maps the fields to the CERIF-based Common Data Model (CDM):

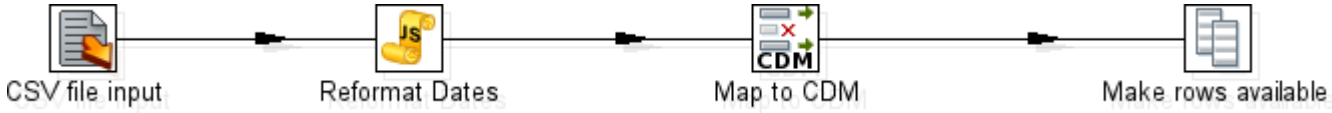

Figure 8.4. The Read Symplectic CSV transformation

The first step reads from the CSV file, and introduces the configured fields into the transformation. As CSV is intrinsically similar to the way data is handled in transformations - as a series of rows with a defined set of fields (columns) - no special handling is required, and a standard PDI *CSV Input* step is used. Although all rows and columns in the input data are read by this step, only the field names defined in the CSV Input step are added to the stream.

Next, a JavaScript step is used to convert any dates in a dd/MM/yyyy format into the xs:dateTime format which is required by CERIF (yyyy-MM-dd). Dates with times are also converted into a CERIF-compatible format. These converted dates and dateTimes are added as new fields to each row.

A *Map to CDM* step then maps from the local fields to CDM fields. In worked example 4 the *Map to CDM* step is configured to:

- Remove the now spurious original date fields loaded from the CSV file. This is configured via the field listing on the *Remove* tab.
- Map the remaining fields to *CDM Fields* via the *Meta-data tab*.

The *Make rows available* step allows the *Post CERIF to Web Service* transformation to access the data produced by this transformation.

8.4.5. Post CERIF to Web Service

The *Post CERIF to Web Service* transformation performs the bulk of the work. The CERIF XML file is wrapped in a SOAP message and posted to a web service.

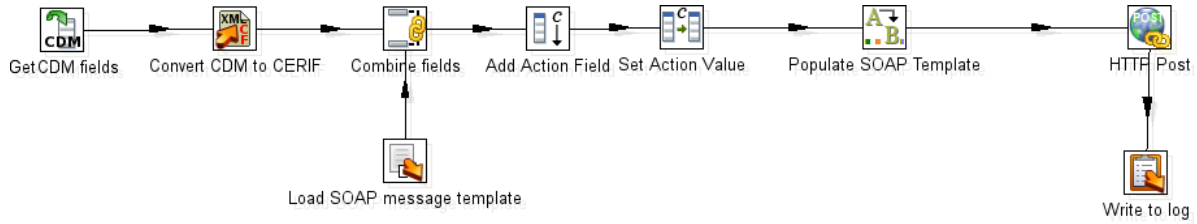

Figure 8.5. The worked example 4 job

Forming and posting the SOAP message to the remote web service requires several steps. The first of these is to load a SOAP message template from the filesystem. The message template must contain the placeholder value *CERIF_DATA* to indicate the point at which the CERIF XML document is to be inserted.

The Combine Fields step is used to merge the stream containing the CERIF XML data and the stream containing the SOAP message template into a single stream.

The Add Action Field and Set Action Value steps introduce the SOAP Action as a new column. The SOAP Action ultimately defines the service method to invoke on the remote endpoint. The action is parameterised, so adding this value requires two steps: firstly to add the new field to the stream, and then to set its value from the transformation parameter.

The SOAP template is then populated with a string replacement, substituting the *CERIF_DATA* placeholder with the actual data.

An HTTP POST is performed to send the request to the server. The HTTP Post step is configured to post the SOAP message as the request body, and the SOAP Action is passed as an HTTP header. This step is also configured to capture the HTTP response code and body as fields.

Finally, a *Write to log* step logs a message containing the HTTP status code and response body.

8.4.6. SOAP Web Service

The download contains an example web service that the SOAP message in this worked example is tailored to. It may be found in the RMAS Web Services Example package.

8.5. Running the job from the command line

You can also run the job from the command line, either as the downloaded worked example or after configuring and saving it using Spoon. The following shows how to run the command from both a linux environment and a Windows environment, using the *Kitchen* tool which is provided with PDI.

8.5.1. Running the job from a Linux command line

Navigate to the RMAS/kettle directory:

```
cd RMAS/kettle
```

Ensure that the kitchen.sh and pan.sh scripts are executable:

```
chmod u+x kitchen.sh pan.sh
```

Run the job using the following command. Note that the quote marks around the repository name are optional, but they *must* be included if the repository name contains whitespace:

```
./kitchen.sh -rep="RMASConnector" -job=WorkedExample4/WorkedExample4
```

You can also run the entire example by starting the PostFileToTriggerService transformation

```
./pan.sh -rep="RMASConnector" -trans=WorkedExample4/PostFileToTriggerService
```

8.5.2. Running the job from a Windows command line

Navigate to the RMAS/kettle directory:

```
cd RMAS\kettle
```

Run the job using the following command. Note that the quote marks around the repository name are optional, but they *must* be included if the repository name contains whitespace:

```
Kitchen.bat /rep:"RMASConnector" /job:WorkedExample4/WorkedExample4
```

You can also run the entire example by starting the PostFileToTriggerService transformation

```
pan.bat /rep:"RMASConnector" /trans:WorkedExample4/PostFileToTriggerService
```

8.6. A note on configuring the worked example

The *PostFileToTriggerService* transformation reads an export from Symplectic, in the form of a CSV file. A sample file is located at RMAS/exampledata/WorkedExample4/SymplecticCSV.csv.

The input transformation may be substituted by any transformation which makes available rows of CDM data, such as via the PDI *Copy rows to result* step. The type of file that the input transformation reads defines the type of file that should be posted to the listener service, and hence what sort of file the *PostFileToTriggerService* should post. The WorkedExample4 job will pass the parameter *input.file* to the input transformation, to indicate where data should be read from. When WorkedExample4 is invoked by the listener web service this parameter indicates the location of the temporary file containing the data that has been received by the trigger service.

The *CERIFXMLOutputToSOAP* transformation reads a SOAP message template from the file system. This is a complete XML document containing the entire message body for an HTTP POST. A sample file is located at RMAS/exampledata/WorkedExample4/SOAPRequestTemplate.xml. Note that as the entire CERIF XML document is included verbatim in the SOAP template, it is necessary to ensure that the SOAP document remains well-formed. For this reason the sample template escapes the CERIF XML by wrapping the placeholder value in a *CDATA* block.

The *CERIFXMLOutputToSOAP* transformation captures the HTTP response code and body into two fields. These are logged for convenience, but may be used for additional processing - for example to flag up HTTP 500 errors due to the web service being unavailable.

Worked Example 5: HR to CERIF XML for project costings

9.1. Overview

Worked example 5 demonstrates how data from a Human Resources (HR) database can be mapped to CERIF XML, with the intention that it can then be joined with data from pFACT in order to calculate project costings.

9.2. User Story

A project manager wishes to estimate the cost of an upcoming project. They have used pFACT to determine who will be working on the project and the project timespan. This information has already been exported as CERIF XML. They now need to obtain salary information from HR in order to calculate the costings. Generating this information in CERIF XML format facilitates its merge with the pFACT data.

The project manager starts PDI and opens the HR To CERIF XML job. They ensure that the job is configured to access their HR database, and write a query to run against that database to obtain the required fields. They ensure that the extracted fields are named as required. They then run the job, which generates a CERIF XML file containing their HR data.

9.3. Structure of the Example

Worked example 5 consists of three PDI components: a job (WorkedExample5.kjb) and two transformations (HRDatabaseInput.ktr and CERIFXMLOutput.ktr). These are located in the WorkedExample5 subdirectory of the PDI repository.

Worked example 5 may be used in the context of the *Spoon* graphical front end to PDI, by loading the job and executing it directly. It may also be executed from a command line.

9.3.1. Parameters

The following parameters are used to configure the *WorkedExample5* job. They can be set in order to tailor the example to be specific to your local HR:

Parameter	Description
DatabaseClass	The database driver class name.
DatabaseUrl	The URL for the database connection.
DatabaseUsername	The username for accessing the database.
DatabasePassword	The password for accessing the database.
SQLQuery	The query to execute on the database.
output.file	The path to the file which is generated by the process. If the file does not exist, it will be created automatically.
project.dir	The root directory for the project.

Table 9.1.

9.4. Running the job from Spoon

This section describes how to configure and/or run the job from the Spoon visual editor.

9.4.1. Running the job from Spoon

Figure 9.1. The worked example 5 job

Worked example 5 uses a *Read HR Database* transformation to extract data from the HR database and map it to a CERIF-based Common Data Model (CDM). The CDM data is then passed to a further transformation, *Write to CERIF XML File*, which serialises it to CERIF XML format and validates it against the CERIF schema. We consider each transformation below.

9.4.2. Read HR Database

Figure 9.2. The read HR database transformation

This transformation runs a query against a database which is specified by setting the required parameters at the job level (see above). A JavaScript step is used to convert any dates from a dd/MM/yyyy format into the xs:datetime format required by CERIF. The *Map to CDM* step is then used to remove any irrelevant fields and to map the remaining ones to their equivalent CDM fields. Finally, the data is made available to the *Write to CERIF XML* transformation step.

9.4.3. Write to CERIF XML

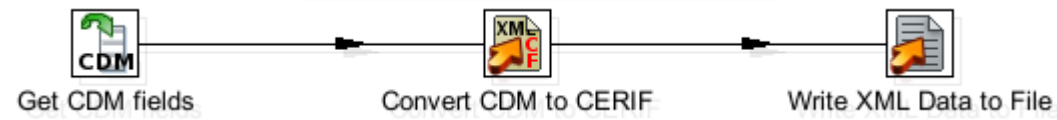

Figure 9.3. The write to CERIF XML transformation

This transformation takes the data made available from the *Read HR Database* transformation and serialises it to CERIF XML. The *Convert CDM to CERIF* step also validates the generated XML against the CERIF schema.

9.5. Running the job from the command line

You can also run the job from the command line, either as the downloaded worked example or after configuring and saving it using Spoon. The following shows how to run the command from both a linux environment and a Windows environment, using the *Kitchen* tool which is provided with PDI.

9.5.1. Running the job from a Linux command line

Navigate to the RMAS/kettle directory:

```
cd RMAS/kettle
```

Ensure that the kitchen.sh script is executable:

```
chmod u+x kitchen.sh
```

Run the job using the following command. Note that the quote marks around the repository name are optional, but they *must* be included if the repository name contains whitespace:

```
./kitchen.sh -rep="RMASConnector" -job=WorkedExample5/WorkedExample5
```

9.5.2. Running the job from a Windows command line

Navigate to the RMAS/kettle directory:

```
cd RMAS\kettle
```

Run the job using the following command. Note that the quote marks around the repository name are optional, but they *must* be included if the repository name contains whitespace:

```
Kitchen.bat /rep:"RMASConnector" /job:WorkedExample5/WorkedExample5
```

9.6. A note on configuring the worked example

This example requires a connection to a Human Resources database. This can be configured by ensuring that all the database parameters are set at the job level. Default values for these parameters have been entered for the sample database which is provided as part of the example (RMAS/exampledata/WorkedExample5/WorkedExample5.script). When configuring your own database, you will need to ensure that the *Map to CDM* step in the *Read HR Database* transformation maps the column names in the database to the appropriate CDM fields.

On running this job, a CERIF XML representation of the data extracted from the database is created in the location specified by the output.file parameter.