OpenSyncro 2.2

support@opensyncro.org

February 27, 2008

Contents

1 Introduction

Welcome as a user of Smilehouse OpenSyncro software. This manual introduces the most important features of the software and explains how to perform the most common tasks.
OpenSyncro is a free, lightweight enterprise application integration tool written in Java language. It runs on 100% open source platform, being based on Apache Tomcat application server and storing its configuration data in MySQL database. OpenSyncro was originally developed for connecting Smilehouse Workspace to other enterprise applications, for example ERP, CRM and SCM systems, but it can be used independently of Workspace for integrations between other applications, as well as for various data conversion tasks.

Address
Itälahdenkatu 22 A
00210 Helsinki

Phone&Fax
+358 9 25 122 10
+358 9 25 122 119

WWW
http://opensyncro.org
support@opensyncro.org

2 Installing and updating OpenSyncro

OpenSyncro is delivered as two separate ZIP archives: "install" and "source". For installing, deploying and updating OpenSyncro, only the "install" ZIP archive is needed. It contains the OpenSyncro web application package ("opensyncro.war"), an example configuration file ("opensyncro.properties"), two alternative database dump files ("opensyncro_demodb.sql" or "opensyncro_emptydb.sql") for initializing a new OpenSyncro database, and installation/upgrading instructions.

2.1 Before installation

Before you can start the installation, you need to have the following software installed:

- **Sun Java 2 Standard Edition 5.0 (J2SE/JDK 5.0)** or newer recommended. You can download Java at http://java.sun.com/j2se/downloads.html
- **Apache Tomcat 5.5** or newer recommended. You can download Apache Tomcat at http://tomcat.apache.org/
- **Database** . We recommend **MySQL version 5.0 or 4.0** . You can download MySQL at http://dev.mysql.com/downloads/mysql
- **JDBC driver** for the database. MySQL is compatible with MySQL Connector. You can download this at http://www.mysql.com/downloads/api-jdbc.html

OpenSyncro should also run on servlet standard compliant application servers other than the Apache Tomcat, when the webapp and configuration files are copied to an equivalent path of, for example, JBoss 4.0 or Jetty 6.

2.2 The installation process

This chapter goes through the installation process step-by-step. If you are installing OpenSyncro for the first time, check that the database server is online and start at step 2.2.1. For updating a previous OpenSyncro installation, you should see step 2.2.2 and then proceed to step 2.2.5.
The installation process begins with creating and initializing a dedicated database for OpenSyncro. Then, OpenSyncro needs a couple of parameters for the database connection. These parameters are located in OpenSyncro's configuration file called "opensyncro.properties".

2.2.1 Creating a new OpenSyncro database

In MySQL, enter the database console with sufficient privileges to create a new database, e.g. " **mysql -u root** ". In the database console, enter " **CREATE DATABASE opensyncro_test DEFAULT CHARACTER SET utf8;** " (without the double quotes) if you are running **MySQL 5.0** , or " **CREATE DATABASE opensyncro_test;** " if your MySQL version is **4.0** . Here, **opensyncro_test**

is the name of the database to be created and **utf8** is our recommendation for the character encoding to be used in OpenSyncro's database (MySQL 5.0+ only).

There are no special restrictions regarding the database's name, as long as no other database already exists under the same name. If database was created successfully, MySQL should reply with "Query OK, 1 row affected". Finally, exit the database console with "exit" command.

```
mysql> CREATE DATABASE opensyncro_test DEFAULT CHARACTER SET utf8;
Query OK, 1 row affected (0.01 sec)

mysql> exit
Bye
```

2.2.2 Initializing/updating an OpenSyncro database

For initializing the database created in the previous step, you have a choice of two database dump files: " **opensyncro_emptydb.sql** " and " **opensyncro_demodb.sql** ", both located in the directory " **Installation/mysql/initialization/** ". The dump files contain series of SQL statements to be executed by the database. The first dump file ("opensyncro_emptydb.sql") creates an empty database and an initial user account ('user') to log in with. The second dump file ("opensyncro_demodb.sql") is basically the same as the first one, but adds a couple of example integration processes (Pipes) for testing and learning OpenSyncro.
For the first installation of OpenSyncro, we recommend choosing the "opensyncro_demodb.sql" dump. On **MySQL 5.0** the initialization can be done by running " **mysql -u root –default-character-set=utf8 opensyncro_test <opensyncro_demodb.sql** " from command prompt, in the "Installation/mysql/initialization/" directory. On **MySQL 4.0**

enter " **mysql -u root opensyncro_test <opensyncro_demodb.sql** " instead. If the initialization is successful, mysql command should not output any messages.

```
user@host:~/initialization$ mysql -u root -default-character-set=utf8
opensyncro_test <opensyncro_demodb.sql
user@host:~/initialization$
```

When **updating** OpenSyncro, see

Installation/mysql/updates directory for further instructions for existing OpenSyncro database upgrade procedure.

2.2.3 Setting database connection parameters for OpenSyncro

Open the OpenSyncro configuration file "opensyncro.properties" from "Installation/tomcat5/shared/classes/smilehouse/conf/" in a text editor. The file contains lines with "parameter name = value" pairs accompanied by explanatory comments. When using MySQL, you only need to specify

jdbc.user and **jdbc.password** . jdbc.user specifies the database username and jdbc.password the database user's password to be used by OpenSyncro. jdbc.uri needs to be modified only if MySQL has not been installed to the same PC as Tomcat and OpenSyncro. Also verify that

log.filename points to an existing directory. charset.database should be assigned the same character encoding that you used to create the OpenSyncro database.

```
# OpenSyncro application properties file.
# Shortened version with minimum set of configuration parameters.
```

```
jdbc.driver = com.mysql.jdbc.Driver
jdbc.user = root
jdbc.password =
jdbc.uri = jdbc:mysql://localhost/
charset.web = UTF-8
charset.database = UTF-8
log.filename = /var/log/opensyncro-log.txt
```

Copy the edited " **opensyncro.properties** " file to path " **shared/classes/smilehouse/conf/** " (or " **lib/smilehouse/conf/** " in Tomcat 6) under Tomcat installation directory.
If you are installing OpenSyncro on application server other than Tomcat and the equivalent path is not available by default, there is always an option to copy the "opensyncro.properties" file under the directory extracted from "opensyncro.war" package (see "Installing and updating OpenSyncro web application package"), for example "webapps/opensyncro/WEB-INF/classes/smilehouse/conf/opensyncro.properties".
Note: The "opensyncro.properties" file may additionally contain a list of databases which OpenSyncro should check for pending Pipe execution requests at startup. Information about enabling this feature is provided in chapter "Pipe Execution Queue".

2.2.4 Adding JDBC database driver to Tomcat's Java classpath

Copy MySQL connector JAR file, e.g. " **mysql-connector-java-5.0.4-bin.jar** " to " **shared/lib/** " (or " **lib/** " in Tomcat 6) subdirectory under Tomcat's installation directory.

2.2.5 Installing and updating OpenSyncro web application package

Copy the OpenSyncro web application package file " **opensyncro.war** " located in "Installation/tomcat5/webapps/" to " **webapps/** " subdirectory under Tomcat installation directory. When updating, stop the OpenSyncro webapp in Tomcat first, then replace "webapps/opensyncro.war" with a new version of the file, delete the "webapps/opensyncro/" directory (if it exists) and finally restart the OpenSyncro webapp.
This completes the basic OpenSyncro installation. If you don't have any external OpenSyncro components to install, you are ready to test your installation by following the instructions in chapter 2.3.

2.2.6 Installing and updating external components

OpenSyncro can be extended by adding components to an existing installation. Components for connecting to various ERP systems are available for purchase from Smilehouse Oy. The basic OpenSyncro installation does not require any external components. If you haven't purchased external components, you can skip to chapter 2.3.
External Dynamic Components are Java archives (.jar files), whose location is specified with " **component.dir** " property in the

opensyncro.properties file (see step 2.2.3). To inform OpenSyncro of the external component location, append a new line to

opensyncro.properties :

```
component.dir = /path/to/component/directory
```

Webapp filename	Properties filename
opensyncro.war	opensyncro.properties
anotheropensyncro.war	os_anotheropensyncro.properties
123.war	os_123.properties

Where "/path/to/component/directory" is the full path to the directory containing the component .jar files. After modifying opensyncro.properties, restart OpenSyncro (or Tomcat). Now the new components should be listed on the Component manager page (see chapter 3.2) and are ready for use.
The OpenSyncro installation with external components is now complete. The next chapter provides instructions for testing the installation and some troubleshooting advice.

2.2.7 Restricting OpenSyncro web application's access with a security policy

For added security it is recommended to allow OpenSyncro web application to access only the resources necessary for successful execution of the intended integration tasks. This can be accomplished by running OpenSyncro's servlet container (Tomcat) on an operating system user account with limited privileges. Another, complementary way would be to configure a security policy to the servlet container.
OpenSyncro distribution includes an example Tomcat " **catalina.policy** " file for restricting the web application's access to minimal filesystem and network resources. The file is located in "Installation/tomcat5/conf/" directory.
Before copying or merging the security policy file to "conf/catalina.policy" under Tomcat's installation directory, you will need to edit the file and update references to filesystem paths and network sockets according to your server environment. OpenSyncro related lines in the policy file are marked with comments explaining the purpose of each granted permission.

2.2.8 Running multiple instances of OpenSyncro in the same application server

OpenSyncro supports running multiple instances of the web application on the same server. If you place several copies of OpenSyncro WAR file under Tomcat's **webapps/** directory, the opensyncro.properties filenames are generated as follows:

If the OpenSyncro WAR filename is anything else than the default "opensyncro.war", its properties file name will be the WAR filename prefixed with " **os_** ".

2.3 Testing the installation and troubleshooting

Start Tomcat and point your web browser to " **http://localhost:8080/opensyncro/Login** " (assuming that Tomcat is running in port 8080). If installation step 2.2.5 was successful, you should see a OpenSyncro Login page asking for database name, user name and password. Log in to the OpenSyncro database you've created ("opensyncro_test") with username " **user** " and password " **password** ". If "OpenSyncro - All Pipes" page shows up, the installation has been successful.

Figure 1: OpenSyncro Login prompt

In case a "Login failed" dialog is displayed after clicking the "Login" button - or an HTTP error #500 page is displayed, it might be due to one of the following reasons:

- a problem with OpenSyncro's database connection: specified database cannot be found on the server or OpenSyncro is unable to connect to database server via JDBC
- OpenSyncro might be unable to find "opensyncro.properties" file (see step 2.2.3)
- login username and/or password might be wrong (default: "user" / "password")
- On "opensyncro.properties" file's "parameter.name = parameter.value" lines there are **extraneous space characters at the end of the line** , after the parameter value. OpenSyncro includes these tailing spaces into the parameter value and thus may be unable to load the JDBC driver class, connect or log in to the database and so on.

The reason can be found in the OpenSyncro's log file, or in the Tomcat output console if there has been a problem with the log file itself (see step 2.2.3, "log.filename" in "opensyncro.properties"). A list of most common installation problem related log file messages with their explanations follows:

```
Log message: Couldn't get database connection: java.sql.SQLException:
General error message from server: "Unknown database 'opensyncro_test'"
Explanation: Database is not found, see installation step 2.2.1

Log message: OpenSyncro: Cannot find file "opensyncro.properties" in
classpath, startup failed.
Explanation: OpenSyncro is unable to load its settings from
"opensyncro.properties" file, see step 2.2.3

Log message: OpenSyncro: Couldn't get environment information, startup
failed.
Explanation: JDBC database driver can not be found by Tomcat (OpenSyncro),
see step 2.2.4
```

```
Log message:  Couldn't get database connection:  java.sql.SQLException:
No suitable driver
Explanation:  jdbc.uri value in opensyncro.properties might be missing
the last '/' (slash) character, see step 2.2.3
```

3 Getting started with OpenSyncro

Having completed OpenSyncro installation process, point your browser to

http://localhost:8080/opensyncro/Login and log in to your OpenSyncro database. Make sure that Javascript support is enabled in your browser. After login the user is presented with "Pipes / All Pipes" view displaying a list of complete integration processes, i.e. Pipes, stored in OpenSyncro's database. Before proceeding to creating, modifying and executing the Pipes (see chapters 3.3, 3.4, 3.5 and 3.6), it is recommended that the default user's password is changed for security reasons.
This chapter first explains how to add and remove users and edit the user data. Next we have look at Pipe Component management, and then OpenSyncro's Pipe and Component editors will be introduced. After we have familiarized ourselves with OpenSyncro's user interface, a couple of example integration processes are introduced.

3.1 User management

User management interface (" **Users** ") can be found from the " **General** " pulldown menu in the left corner of top menu bar. The Users table view shows all users that have access to the current OpenSyncro database (via OpenSyncro). If you wish to add a new user, click on " **Create** " button in the down-left corner. Users' login name ("Login") and real name ("Name") are displayed with the option to change password, or delete user (" **Delete** " button). User passwords are always hidden. Use the "OK" button to save, or "Cancel" button to abandon changes.

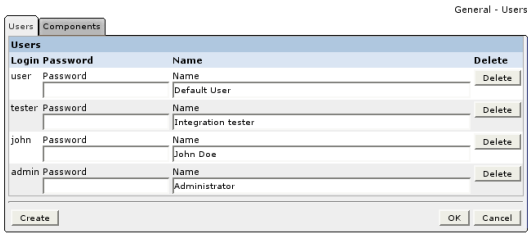

Figure 2: User management

Note: currently any OpenSyncro user may change other users' data, including passwords.

3.2 Installing and removing Pipe Components

Component manager (" **Components** ") is located in the " **General** " pulldown menu, the left corner of top menu bar. It allows to view and (re)load Components, which are the Java based building blocks of OpenSyncro Pipes.
OpenSyncro's Component selection consists of built-in Default Components and external Dynamic Components (.jar files), which are loaded from a directory path specified by " **component.dir** " property in the

opensyncro.properties file. While the Components are initially loaded at OpenSyncro web application start-up, Component manager provides a " **Reload all components** " button which allows to install/remove Dynamic Components without restarting OpenSyncro.

Component manager displays a table of names, descriptions and Component ID's of all Source, Converter and Destination Components currently available to OpenSyncro. External Dynamic Components' information is highlighted with a green background color.

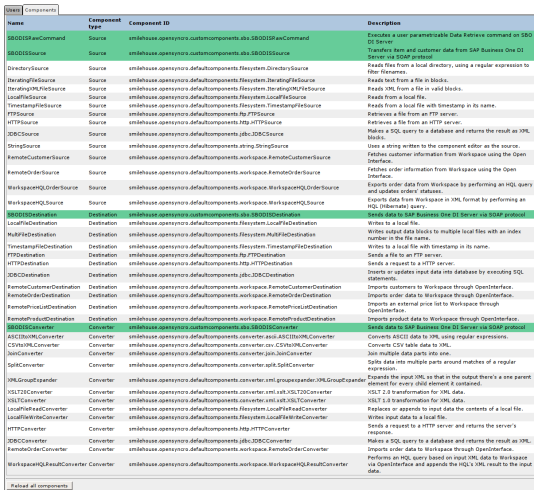

Figure 3: Component manager

*Please note that the SBODISSource, SBODISRawCommand, SBODISConverter and SBODISDestination components in the above screenshot are not included in the free OpenSyncro distribution packages, but are available as an extension called the SAP Business One Connector Pack.

3.3 Pipe list: Adding and removing integration processes

After logging in to OpenSyncro's administration interface, the user is taken to the Pipe list ("All pipes") page which displays the names of all integration processes stored in the current OpenSyncro database. It is possible to create a new Pipe by clicking on the " **Add** " button, edit an existing Pipe (" **Edit** " button next to the Pipe name), start a Pipe (" **Start** " button), clone a Pipe (" **Clone** " button) and remove a Pipe (" **Delete** " button). Cloning a Pipe creates a new Pipe which has exactly the same Components and settings as the original Pipe. Deleting a Pipe also deletes Transfer log messages concerning the Pipe. Pipe names can be edited in the list and the changes saved by hitting the " **OK** " button. The " **Cancel** " button abandons all changes.
Currently running Pipe's rows are highlighted with a light green background color. Information about the last execution of a Pipe is also visible on this page. You will see last execution start and end times, duration, status and the name the OpenSyncro user who started the Pipe. If a Pipe execution was initiated by remote invocation (HttpStart servlet) request, "Remote" is displayed in place of the user name. If a Pipe has never been executed these columns will be empty.

The Pipe list page is always accessible from the Pipes pulldown menu by selecting "All pipes" menu item.

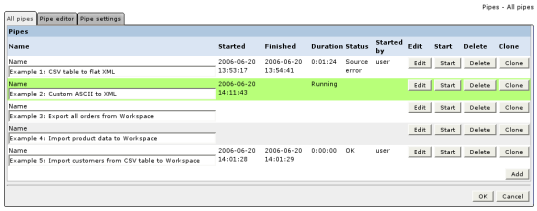

Figure 4: Pipe list

3.4 Pipe editor: Creating, maintaining and testing integration processes

Pipe editor is entered by clicking on an " **Edit** " button in **Pipe list** . Alternatively, the " **Pipes** " pulldown menu in the top menu bar contains a "Pipe editor" item which opens up the Pipe editor for the last Pipe edited during the current session.
Pipe editor displays a Pipe configuration in three columns: Source, Converters and Destination. Source column on the left specifies a single Source Component used to retrieve data to be processed by the Pipe from the source system. Converters column in the middle is list of optional Converter Components. First Converter on the top takes its input from the Source Component and after processing the data passes it to the next Converter on the list. The Destination Component on the right column takes the output from the last Converter - or directly from the Source Component if there are no Converters - and delivers it to the target system.

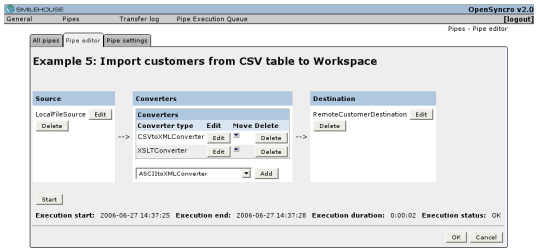

Figure 5: Pipe Editor

Components can be added to the Pipe by selecting the Component name from a pulldown menu and clicking the "Create" button next to the menu. For removing a Component from the Pipe, there is a "Delete" button for each Component. Converter Components' order in the middle column can be changed by clicking the small up/down arrows next to Component's name.
Every OpenSyncro Pipe Component can take parameters. Component parameters are set and modified in Component Editor, which is entered by clicking "Edit" button next to the Component.
OpenSyncro Pipes can be started by pressing the "Start" button. When the execution of the Pipe task is finished, the Pipe Editor page reloads and execution information (start/end times, duration and status) will be displayed. Status information of the Pipe execution is written to "Transfer log", available at the top menu bar. Chapter 3.7 explains the "Transfer log" and its settings.

3.5 Pipe settings: Logging, email notification and remote invocation

Pipe settings is entered by clicking on a " **Pipe settings** " tab or via the " **Pipes** " pulldown menu in the top menu bar.
Pipe settings tab allows you to change the name of the pipe, enable remote HTTP invocation and set transfer log verbosity level. " **Transfer log verbosity level** " determines the types of messages recorded in the log. The possible choices are: ' **Errors** ', ' **Warnings** ', ' **All** ' (debug) and ' **Dynamic** '. Whereas the first three modes log messages equal to and above the set verbosity level (i.e. 'warnings' includes also errors and 'all' also covers warnings and errors), 'dynamic' mode logs all messages only if the Pipe execution fails with an error. If the Pipe is executed successfully, warning messages are written to the Transfer log but the debug messages from 'all' level are deleted.
Email notification requires the " **Outgoing email server** " (SMTP hostname or IP address) and " **Receiver of notification email** " (one or more email addresses separated by a comma or semi-colon character) fields to be filled in. Email subject line will contain the Pipe name, completion status and execution time. The email body consists of Transfer log message lines.
The " **Notification level** " determines the threshold for sending the notification mail, choices being ' **Errors** ', ' **Warnings** ', ' **All** ' and ' **None** '. If the pipe's log does not contain messages at or below the set notification level, no mail is sent. For example, when Notification level is set to 'Warnings', a mail is sent if the log contains warning or error messages. The types of messages that are present in the notification mail is determined by the **"Transfer log verbosity level"** value. Setting notification level to **'None'**

will result in no message being sent, regardless of log contents.

OpenSyncro can be told to send a notification mail when a pipe aborts but there are no errors in the log. For this, check the " **Send notification mail when pipe aborts** " checkbox. Also, make sure that ' **Notification level** ' is not set to ' **None** '

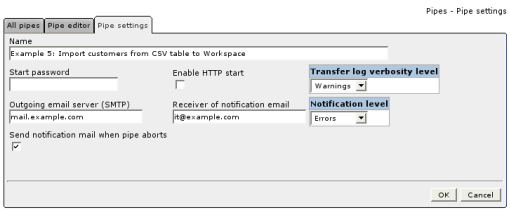

Figure 6: Pipe Settings

When " **Enable HTTP start** " is activated, the Pipe may be started with a single HTTP GET request, loading a URL containing the Pipe name and it's " **Start password** ". With this feature, an OpenSyncro Pipe execution can be easily initiated by a command line tool such as 'wget' or 'curl', scheduled to run for example every 30 minutes. The OpenSyncro's HTTP Start URL format is as follows:
http:// **localhost:8080** /opensyncro/HttpStart?database= **DatabaseName** &pipe= **PipeName** &password= **PipeStartPassword** &mode= **Mode**

The

mode parameter determines whether the HttpStart servlet waits until the Pipe has finished executing (this requires the mode parameter to be set to **sync**) or, if another instance of the Pipe is already executing, adds the request to the Pipe Execution Queue (mode parameter set to

async). HttpStart returns an error message if the HTTP request parameters are incorrect or HTTP start has not been enabled for the specified Pipe. If Pipe execution is **started** successfully, "OK" message is returned if sync mode is used. In async mode the message returned is "QUEUED (queue length: n)", where n is the total number of requests for this Pipe waiting to be executed. **Note** : In sync mode HttpStart responds with "OK" even though the Pipe execution may have resulted in an error.

3.6 Component editor: Setting Pipe Component parameters

Component editor is started **from the Pipe editor** by clicking the " **Edit** " button next to any Source, Converter or Destination Component in the Pipe. Component editor displays all the changeable parameters of the Component and allows user to modify the parameter values. Use the "OK" button to save, or "Cancel" button to abandon changes.

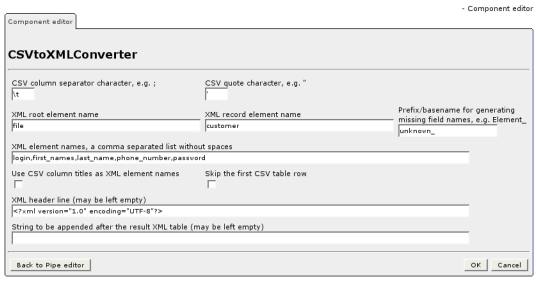

Figure 7: Component Editor view of CSVtoXMLConverter

3.7 Transfer log: Monitoring and debugging integration processes

When a Pipe is executed, status and debug information is written to the " **Transfer log** ", depending on the " **Transfer log verbosity level** " setting in **Pipe editor** . The setting has 3 choices: " **Errors** " logs only error messages that cause the Pipe execution fail, " **Warnings** " includes both error and warning messages and " **All** " writes to log every debug message generated by Pipe components - in addition to the warning and error messages.

The overall status of the finished Pipe execution is displayed in the " **Status** " column. Possible status messages are "OK", "Source error", "Conversion error", "Destination error" and "Aborted". The "Aborted" status means that the Pipe execution was stopped, but no error occurred. For example if a Source Component has been set up with a search query that returns an empty result set, the Pipe is aborted as there is no data with which to continue the Pipe process.
At the bottom left corner on a Transfer log page you'll find a " **Clear Transfer Log** " button, which clears the entire log - all log messages originating from any Pipe execution.

3.8 Pipe Execution Queue: Monitoring Pipe execution request queues

Pipe execution requests OpenSyncro receives via remote invocation (HttpRequest servlet) are queued so that two instances of the same Pipe will not execute simultaneously. This page allows you to view the current statuses of these execution requests. The columns displayed are " **Pipe** ", " **Request queued** "

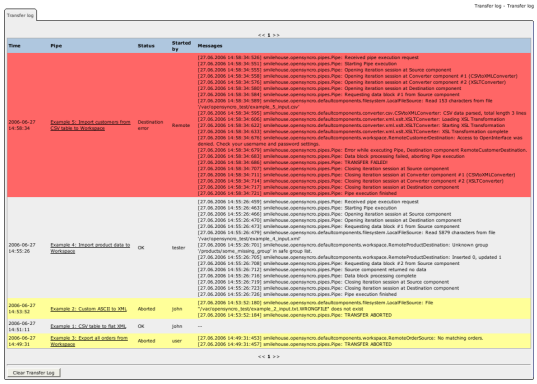

Figure 8: Transfer Log

and " **Pipe started** ". "Request received" shows the time and date when the request for Pipe execution was received. "Pipe" shows the name of the Pipe to be executed. "Pipe started" shows the time and date when Pipe execution started or "Queued" if the request is waiting for another instance of the Pipe to finish execution.

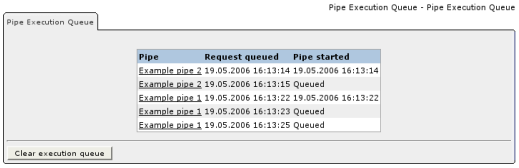

Figure 9: Pipe execution queue

The " **Clear execution queue** " button removes all pipe execution requests from the queue. Note however, that Pipes already executing are not stopped.
When Tomcat is started, OpenSyncro checks the database for unprocessed Pipe execution requests. What it does with found requests depends on a per-database setting. It is possible to set up OpenSyncro to ignore all Pipe execution requests in the database and remove them, or execute all found requests at startup. To specify which route to take you need to modify the

opensyncro.properties file. For each OpenSyncro database two lines need to be added. First, "init1.database= **database name** ", second "init1.resumemode= **resume mode** ". Resume mode should be set to either ' **CLEAR** ', which removes all Pipe execution requests from the specified database, or ' **RUN** ', which executes found requests. The number in the property name (in this case '1' in init1.database) should be incremented with each added database (for example, the second OpenSyncro database in the file should start with "init2" prefix). When using more than one database, make sure that all numbers in the property names are sequential (i.e. no gaps in the numbering).
If you do not use OpenSyncro's remote execution capabilities you can skip adding your database(s) to the opensyncro.properties file. In this case OpenSyncro will not delete requests from the database nor will it execute them.

3.9 Example Pipes of OpenSyncro Demo Database

If you have initialized OpenSyncro's database using the database dump with demonstration material ('opensyncro_demodb.sql', see: "Initializing/updating an OpenSyncro database"), the Pipe list view displays five example Pipes which you can experiment with.

Before starting any of the example Pipes from Pipe editor, you need to visit the Component editor of Source and Destination components of each Pipe and enter the correct arguments, for example **directory paths, URLs and database connection parameters** .

Example 1: CSV table to flat XML Reads a CSV table (" **Installation/examples/example_1_input.csv** ") from filesystem, converts it to flat XML format and writes the result to a file.

Example 2: Custom ASCII to XML Reads a custom ASCII based data file (" **Installation/examples/example_2_input.txt** ") from filesystem, converts it to XML and writes the result to a file.

Example 3: Export all orders from Workspace Connects to a Workspace webshop via Workspace OpenInterface (Web Services), exports all available sales orders in the native OpenInterface XML format and writes the result to a file.

Example 4: Import product data to Workspace Reads an XML file (" **Installation/examples/example_4_input.xml** ") in Workspace OpenInterface product data format and sends the product record to a Workspace webshop.

Example 5: Import customers from CSV table to Workspace Reads a CSV table (" **Installation/examples/example_5_input.csv** ") containing simple customer account data from filesystem and converts the CSV data to flat XML format. The flat XML is then transformed to Workspace OpenInterface customer data format and sent to a Workspace webshop.

For example Pipes 1, 2, 4 and 5 you need to enter the full path of the example input data files provided in the OpenSyncro install ZIP archive, to the LocalFileSource component parameters. For example Pipes 3 - 5 the database connection parameters for a Workspace webshop need to be set to the Source/Destination components. Additionally, for OpenSyncro to perform data import/export functions with Workspace you need to create **a Workspace user with OpenInterface access enabled** . Example Pipes 1 - 3 write their output data to local filesystem, so you have to specify a valid LocalFileDestination path for each pipe.
Having entered the Source and Destination component parameters for an example Pipe, you can test the Pipe by pressing the " **Start** " button in the Pipe editor. The button press will result in OpenSyncro executing the current Pipe and reloading the Pipe editor page, after which you can check the Transfer Log for status information. If there were no errors or warnings, the output data should be either a file created in the directory you specified in the LocalFileDestination's Component editor (example Pipes 1 - 3) or a product/customer record visible in the Workspace webshop's Administration Interface (example Pipes 4 - 5).

4 Pipe Component Reference Guide

This chapter contains a reference guide to OpenSyncro's default Pipe Components. Along with a brief description of component's function, component's parameters are explained.

4.1 Source Components

4.1.1 DirectorySource

Reads files from a directory in the local file system. Files can be selected by specifying an optional file name filter regular expression. By default, DirectorySource reads all files in the directory. Subdirectories are

not scanned for files.
DirectorySource component works in iteration mode; files are sent one-by-one through the Pipe, instead of concatenating their contents to a single data block.

Directory Full path to the directory - without trailing slash ('/') character.

File name filter regular expression Regular expression for selecting which files from the specified directory will be read. If left empty, all files are read. (**Optional**)

Charset Character encoding of the file(s). This information is required for conversion to OpenSyncro's internal 16-bit Unicode character set.

4.1.2 FTPSource

Retrieves data from a file located on an FTP server.

Host Host name or IP address of the FTP server.

Port Port number of the FTP server.

User Username to log in with.

Password Password for the User.

Beginning of the file name Full path to the file and the filename itself (for example "/var/test/sourcefile.txt").

Format of the date included in the file name If the filename ends with the current date, the date format can be specified here according to Java SimpleDateFormat syntax (for example "yyyyMMdd"). This parameter may be left empty. (**Optional**)

File extension Filename suffix (for example "txt"), without the leading dot. This parameter is intended to be used together with "Format of the date included in the file name" option and may be left empty. (**Optional**)

File type Chooses the FTP file transfer mode to be used, either Binary (recommended) or ASCII.

Add/subtract days from the current date Amount of days to add to the current date. Use e.g.

-1 for yesterday and **1** for tomorrow.

Charset Character encoding of the file to get.

4.1.3 HTTPSource

Sends a request to an HTTP server and returns the server response body. It can be used for example to simulate web form submission (GET and POST methods).

Host Host name or IP number of the HTTP server.

Port Port number of HTTP server (default: 80).

Directory Directory path for the file to be retrieved. With a leading slash ('/'). (**Optional**)

Filename/Query Name of the file to be retrieved. HTTP GET query parameters can appended to the filename.

User User name to be used for protected sites. (**Optional**)

Password Password to be used for protected sites. (**Optional**)

Request method Request method to use. POST and GET methods are available.

Request protocol Request protocol to use: either HTTP or HTTPS.

Request parameters Request parameters to be used in the request. (**Optional**)

Choose charset to use in reading the response Check to activate the " **Charset used to read the response** " setting. If not activated, the HTTP(S) response is assumed to be in **ISO-8859-1** charset.

Charset used to read the response Character set to decode the HTTP(S) response with. Requires the checkbox " **Choose charset to use in reading the response** " to be selected - otherwise this setting is ignored.

Note: Parameters specified in the **Request parameter** field have to be name value pairs, where names and values are separated by "=" and pairs themselves are separated by a newline. For example:

```
lang=en
req=orders
```

would result (in case GET method is used) in a request:
http://host:port/directory/query?lang=en&req=orders.

4.1.4 IteratingFileSource

Reads contents of a file from the local file system. The file is divided to components that are sent through the Pipe one-by-one.
IteratingFileSource can be used for example to read a CSV table file, one row at time. CSV table rows spanning multiple text lines are supported.

Directory Full path to the directory containing the file - without trailing slash ('/') character.

File name Name of the file (without path).

Delimiter The delimiter that is used to divide the file into data components. For example to split a CSV table file to individual rows, you can use **\n** .

Quote character If the quote character is defined, delimiters inside quoted regions are ignored. For example **"** . (**Optional**)

Escape character If the escape character is defined, quote chars preceded by an (unescaped) escape char are ignored. For example \ . (**Optional**)

Block size Block size tells how many pieces one iteration block contains (at most). Must be a positive integer, at least **1** .

Charset Character encoding of the file. This information is required for conversion to OpenSyncro's internal 16-bit Unicode character set.

4.1.5 IteratingXMLFileSource

Reads contents of an XML file from the local file system and sends it through the Pipe in user definable fragments. Each fragment contains one or more elements extracted at a certain level in the XML element hierarchy. IteratingXMLFileSource can be set to read the immediate child elements of the root tag, one by one, enabling processing of arbitrarily large XML files. Only the current XML fragment needs to fit into the main memory.
IteratingXMLFileSource outputs well-formed XML fragments, preserving all parent elements' start and end tags. Each fragment begins with XML declaration from the input file.
The name of the XML file may contain an optional timestamp between the file name prefix and extension.

Directory Full path to the directory containing the file - without trailing slash ('/') character.

File name (prefix) Name of the file (without path) or a file name prefix if Timestamp and File extension parameters are given.

Date format If specified, a timestamp will be appended to the File name prefix. The date format must conform to Java SimpleDateFormat syntax (for example "yyyyMMdd"). (**Optional**)

File extension File name suffix, which is appended to File name prefix and the optional timestamp. (**Optional**)

Block size Block size tells how many XML elements to read at each iteration step. Must be a positive integer, at least **1** .

Chop depth Specifies the hierarchy level at which XML elements are extracted. Must be an integer, at least **0** . 0 iterates over root level elements, 1 selects the immediate children of the root element(s) and so on.

Charset Character encoding of the file. This information is required for conversion to OpenSyncro's internal 16-bit Unicode character set.
Note: The **encoding** attribute of the XML declaration **is not read** from the file. Character encoding of the XML file must match with this parameter.

4.1.6 JDBCSource

Sends an SQL select query to database and returns the query results in XML. The SQL query is read as a parameter from the **input form** . The input form may contain **only one** SQL select query.
The XML results may be returned in blocks. Number of results per block is set as a parameter. If set to

0, all results are returned at once.
It is possible to choose whether to include the XML declaration line at the beginning of each returned result block or not.

JDBC URL Full JDBC URL (consult your JDBC driver's documentation), including the database's name and type. MySQL example: jdbc:mysql://localhost/testdatabase

User name Name of the user to login with

Password Password of the user

JDBC driver class name Java class name of the JDBC driver to be used for connecting. For example for MySQL Connector/J try **com.mysql.jdbc.Driver** . The JDBC driver must be included in the Java classpath of OpenSyncro webapp.

Charset Character encoding for result XML.

Number of results to return at one iteration step Number of query results JDBCSource returns at each iteration step. If set to 0, all results are returned at once (=iteration disabled).

Output XML declaration line at each iteration Should each result start with the XML declaration line or not.

SQL select query A free SQL select query to be executed to retrieve data from the database. Note: only one Select statement is allowed.

4.1.7 LocalFileSource

Reads contents of a file from the local file system.

Directory Full path to the directory containing the file - without trailing slash ('/') character.

File name Name of the file (without path).

Charset Character encoding of the file. This information is required for conversion to OpenSyncro's internal 16-bit Unicode character set.

4.1.8 RemoteCustomerSource

RemoteCustomerSource exports customers from Smilehouse Workspace in WS OpenInterface XML format. If no customers matching the selection criteria are found, Pipe execution is **aborted** .

Open Interface URL Complete URL path to the Workspace installation, e.g. http://localhost:8080 . No trailing slash ('/') character should be added. The component will append "/workspace.admin/openinterfaceaddress" to this OI URL in order to locate OpenInterface Web-Service.

Database Name of the Workspace shop's database.

User name Name of the Workspace user as which OpenSyncro will log in to WS. Note: the WS user must have OpenInterface access enabled.

Password Password of the Workspace user (see above).

Customer ID Match customer with this customer ID number. (**Optional**)

Customer ID greater than Match customers with ID number greater than the specified integer. (**Optional**)

Customer ID less than Match customers with ID number smaller than the specified integer. (**Optional**)

Customer ID in Retrieve customers with ID matching one of the integers in a comma separated list. (**Optional**)

Primary customer group Match customers with primary customer group name equal to this string. (**Optional**)

Customer group Match customers with customer group name equal to this string. (**Optional**)

Customer created after Match customers created after the specified date (format: dd.MM.yyyy HH:mm:ss). (**Optional**)

Customer created before Match customers created before the specified date (format: dd.MM.yyyy HH:mm:ss). (**Optional**)

Last visit after Match customers with last visit after the specified date (format: dd.MM.yyyy HH:mm:ss). (**Optional**)

Last visit before Match customers with last visit before the specified date (format: dd.MM.yyyy HH:mm:ss). (**Optional**)

Customer modified after Match customers which have been modified after specified time (format: dd.MM.yyyy HH:mm:ss). (**Optional**)

Customer modified before Match customers which have been modified before specified time (format: dd.MM.yyyy HH:mm:ss). (**Optional**)

Admin modified after Match customers which have been modified by administrator after specified time (format: dd.MM.yyyy HH:mm:ss). (**Optional**)

Admin modified before Match customers which have been modified by administrator before specified time (format: dd.MM.yyyy HH:mm:ss). (**Optional**)

Operation between customer and admin modified range Operation defines how customer and admin modified ranges (after-before) are related to each other ('OR','AND'). If there is not enough information on ranges then this operation will be ignored. By default 'OR' is selected.

Note: if **Database** parameter is incorrect, OpenSyncro may write "Failed to get the O.I. Endpoint address" error to the Transfer log - even if the **Open Interface URL** would be correct.

4.1.9 RemoteOrderSource

RemoteOrderSource exports purchase orders from Smilehouse Workspace in WS OpenInterface XML format. If no orders matching the selection criteria are found, Pipe execution is **aborted** .

Open Interface URL Complete URL path to the Workspace installation, e.g. http://localhost:8080 . No trailing slash ('/') character should be added. The component will append "/workspace.admin/openinterfaceaddress" to this OI URL in order to locate OpenInterface Web-Service.

Database Name of the Workspace shop's database.

User name Name of the Workspace user as which OpenSyncro will log in to WS. Note: the WS user must have OpenInterface access enabled.

Password Password of the Workspace user (see above).

Order ID greater than Match orders with ID number greater than the specified integer. (**Optional**)

Order ID less than Match orders with ID number smaller than the specified integer. (**Optional**)

Order ID in Retrieve orders with ID matching one of the integers in a comma separated list. (**Optional**)

Sum greater than Match orders with total sum of items greater than the specified number. (**Optional**)

Sum less than Match orders with total sum of items smaller than the specified number. (**Optional**)

Handling status in Match orders with handling status matching one of the comma separated list of names. (**Optional**)

Payment status in Match orders with payment status matching one of the comma separated list of names. (**Optional**)

New handling status Change the handling status of retrieved orders in WS to the specified name. This **status name must be defined in WS** , otherwise the status change will have no effect. New status is set only after the exported orders have been processed through the Pipe without errors. (**Optional**)

Customer ID in Match orders with customer ID matching one of the comma separated list of integers. (**Optional**)

After Match orders created after the specified date (format: dd.MM.yyyy HH:mm:ss). (**Optional**)

Before Match orders created before the specified date (format: dd.MM.yyyy HH:mm:ss). (**Optional**)

Note: if **Database** parameter is incorrect, OpenSyncro may write "Failed to get the O.I. Endpoint address" error to the Transfer log - even if the **Open Interface URL** would be correct.

4.1.10 StringSource

Simply sends a user defined static text data to the Pipe.

Source data for pipe String (text data) to output.

4.1.11 TimestampFileSource

Reads contents of a local file with timestamp in the filename. The timestamp pattern should conform to java.text.SimpleDateFormat.

Directory Full path to the **directory** containing the file - without trailing slash ('/') character.

File name prefix Beginning of the filename, before the automatically generated Date.

Date format Date format according to Java SimpleDateFormat syntax (for example "yyyyMMdd").

File extension Filename suffix (for example "txt") **without** the leading **dot** . This parameter may be left empty.

Charset Character encoding of the file. This information is required for conversion to OpenSyncro's internal 16-bit Unicode character set.

4.1.12 WorkspaceHQLOrderSource

Exports **order** data from a Smilehouse Workspace webshop by performing a Hibernate v3 (http://hibernate.org) Select query. The query results are returned in OpenInterface XML format, inside a <result/> root element.
WorkspaceHQLOrderSource extends WorkspaceHQLSource by adding order handling status and payment status updating options. The HQL query should **select full order objects** for the status updating functionality to work. The exported XML data is scanned for order ID numbers with an XPath query "//order/@id" and these extracted order ID's are used to create order status update requests back to Workspace. If you need to export other XML objects than orders, we recommend the generic WorkspaceHQLSource component.
This component supports iteration and it can be set to return any amount of query results at once. Iteration can also be disabled by telling WorkspaceHQLOrderSource to return all results at the first iteration step (see parameter "Number of results to return at one iteration step").
If the HQL query returns empty result, the Pipe execution is **aborted** .

Open Interface URL Complete URL path to the Workspace installation, e.g. http://localhost:8080 . No trailing slash ('/') character should be added. The component will append "/workspace.admin/openinterfaceaddress" to this OI URL in order to locate OpenInterface Web-Service.

Database Name of the Workspace shop's database.

User name Name of the Workspace user as which OpenSyncro will log in to WS. Note: the WS user must have OpenInterface access enabled.

Password Password of the Workspace user (see above).

HQL Select query for Workspace database A free HQL query to be executed by Workspace's OpenInterface to retrieve order data from the webshop. Note: the query **must not** end in semicolon (';') character, otherwise an error will occur.

Timeout for the query iterator Maximum time in milliseconds OpenInterface will keep an inactive iterator connection open between OpenSyncro and Workspace. The connection timeout value should be large enough for all components in the Pipe to finish execution.
Also if the "New order handling status" or "New order payment status" options are set, the timeout should include the amount time needed by Workspace to update the order status information (this is due to a limitation of OpenInterface: updating order statuses must be done outside the HQL iterator connection).
The smaller the timeout value the less system resources Workspace's OpenInterface uses. For example, start with 10000 (ten seconds) and increase it if you get "HQL iterator has been closed" errors.

Number of results to return at one iteration step Number of query results Workspace OpenInterface returns at each iteration step. If set to 0, all results are returned at once (=iteration disabled).

Output XML declaration line at each iteration Should each result start with the XML declaration line or not.

New order handling status Order handling status that will be set to exported orders in Workspace. A new order history entry is also created. New status is set only after the exported orders have been processed through the Pipe without errors. (**Optional**)

New order payment status Order handling status that will be set to exported orders in Workspace. A new order history entry is also created. New status is set only after the exported orders have been processed through the Pipe without errors. (**Optional**)

Note: if **Database** parameter is incorrect, OpenSyncro may write "Failed to get the O.I. Endpoint address" error to the Transfer log - even if the **Open Interface URL** would be correct.
Creating HQL queries for Workspace requires expert knowledge of OpenInterface and Workspace's data model. A few example queries with explanations are listed here:

from Order Retrieves all orders.

from Order o where o.handlingStatus.name = 'Received' and o.paymentStatus.name = 'Order paid' Retrieves orders which have handling status set to 'Received' and payment status 'Order paid'.

from Order o where timestamp(o.creationTime) >= timestampadd(day,-7,now()) Retrieves orders which have been created during the last 7 days (or today). Note: Uses timestamp functions specific to MySQL 5.0+.

select o from Order o inner join o.answers as oa where oa.version.question.typeId=2 and oa.value='12345' Retrieves orders with postal code 12345 in the address. Please refer to Customer XML schema in Workspace OpenInterface documentation for questionType numbers.

4.1.13 WorkspaceHQLSource

Exports **product** , **customer** and

order data from a Smilehouse Workspace webshop by performing a Hibernate v3 (http://hibernate.org) Select query. The query results are returned in OpenInterface XML format, inside a <result/> root element.
This component supports iteration and it can be set to return a certain amount of query results at once.

Iteration can also be disabled by telling WorkspaceHQLSource to return all results at the first iteration step (see parameter "Number of results to return at one iteration step").
If the HQL query returns empty result, the Pipe execution is **aborted** .

Open Interface URL Complete URL path to the Workspace installation, e.g. http://localhost:8080 . No trailing slash ('/') character should be added. The component will append "/workspace.admin/openinterfaceaddress" to this OI URL in order to locate OpenInterface Web-Service.

Database Name of the Workspace shop's database.

User name Name of the Workspace user as which OpenSyncro will log in to WS. Note: the WS user must have OpenInterface access enabled.

Password Password of the Workspace user (see above).

HQL Select query for Workspace database A free HQL query to be executed by Workspace's Open-Interface to retrieve data from the webshop. Product, Customer and Order objects are supported by Workspace as of v1.10. Note: the query **must not** end in semicolon (';') character, otherwise an error will occur.

Timeout for the query iterator Maximum time in milliseconds OpenInterface will keep an inactive iterator connection open between OpenSyncro and Workspace. The connection timeout value should be large enough for all components in the Pipe to finish execution. On the other hand, the smaller the timeout value the less system resources Workspace's OpenInterface uses. For example 10000 (ten seconds).

Number of results to return at one iteration step Number of query results Workspace OpenInterface returns at each iteration step. If set to 0, all results are returned at once (=iteration disabled).

Output XML declaration line at each iteration Should each result start with the XML declaration line or not.

Note: if **Database** parameter is incorrect, OpenSyncro may write "Failed to get the O.I. Endpoint address" error to the Transfer log - even if the **Open Interface URL** would be correct.
Creating HQL queries for Workspace requires expert knowledge of OpenInterface and Workspace's data model. A few example queries with explanations are listed here:

from Order Retrieves all orders.

from Product Retrieves all products.

from Customer Retrieves all customers.

from Order o where timestamp(o.creationTime) >= timestampadd(day,-7,now()) Retrieves orders which have been created during the last 7 days (or today). Note: Uses timestamp functions specific to MySQL 5.0+.

from Product p where timestamp(p.lastModified) >= timestampadd(day,-3,now()) Retrieves products which have been modified during the last 3 days (or today). Note: Uses timestamp functions specific to MySQL 5.0+.

select o from Order o inner join o.answers as oa where oa.version.question.typeId=2 and oa.value='12345' Retrieves orders with postal code 12345 in the address. Please refer to Customer XML schema in Workspace OpenInterface documentation for questionType numbers.

select p from Basket b, Product p where b.itemCode = p.itemCode and not b.parent is null Retrieves all products that have been ordered from the shop. We choose only those Basket objects which are linked to an existing Order (Basket.parent is not null).

from Product p where p.inventory.amount < p.inventory.alarmLimit Retrieves products with free stock amount below the product's stock alarm limit.

select p from Product p inner join p.parentsHibernate as ph inner join ph.productGroup as pg where p.visible is true and pg.name = 'Software' Retrieves all products with visibility status set to true from a product group called Software.

select p from Product p inner join p.parentsHibernate as ph inner join ph.productGroup as pg where p.visible is false and pg.groupCode in ('PG1234', 'XYZ1000') Retrieves all hidden products from product groups with groupcode either PG1234 or XYZ1000.

select c from Customer c inner join c.groups as cg where cg.name = 'b-to-c new customers' and c.primaryGroup.name = 'b-to-c new customers' Retrieves customers which belong to customer group "b-to-c new customers", which is also set as the primary customer group of the customer.

select c from Customer c inner join c.answers as ca where ca.question.typeId = 12 and ca.value = 'John' Retrieves customers whose first name is "John". Please refer to Customer XML schema in Workspace OpenInterface documentation for questionType numbers.

from Customer c where timestamp(c.modifiedByCustomer) >= timestampadd(hour,-24,now()) or timestamp(c.modifiedByAdmin) >= timestampadd(hour,-24,now()) Retrieves customers whose data has been modified during the last 24 hours either by the customer herself or by the webshop administrator. Note: Uses timestamp functions specific to MySQL 5.0+.

4.2 Converter Components

4.2.1 ASCIItoXMLConverter

Converts ASCII data to XML, based on a "conversion script" consisting of XML element names and Java regular expressions used to capture data into the XML elements. Each script line has 3 columns separated by a single space character:
Third (rightmost) column specifies the regular expression which should contain at least one capture group (i.e. a part of the regular expression in parentheses). The capture groups will save data matched by the regular expression so that it can be assigned an XML element name in the second column.
The second column contains a comma separated (no spaces) list of XML element names for each capture group, in the same order the capture groups appear in the regular expression.
First column specifies the parent XML element name which will contain all the elements listed in the second column. From here it is possible to refer to any of the second column XML element names on the preceding lines, in order to parse already extracted capture group value using another regular expression.

```
# This is a comment line.
```

```
record name,address regexp_that_captures_(name)_and_(address)
name firstname,lastname regexp_for_(firstname)_and_(lastname)
```

Conversion script The script with XML element names and regular expressions.

XML root element name Name of the XML element that will contain the actual result data.

XML header line XML declaration line. This text will be inserted to the beginning of result data, before the root element. (**Optional**)

4.2.2 CSVtoXMLConverter

Converts CSV table data to XML.

CSV column separator character Character that appears between the table columns, separating the column contents. For example: **;** . Escape sequences \t, \n, \r, \0, \b and \f are supported.

CSV quote character CSV table column values may be contained in quotes. Specify the quote character here, e.g. **"** . Escape sequences \t, \n, \r, \0, \b and \f are supported. (**Optional**)

XML root element name Name for the XML element which will contain all data from the CSV table.

XML record element name Name for the XML element which will contain a single CSV table **row** . For example: **record** .

Prefix/basename for generating missing field names In case a CSV table row contains more columns that have been assigned a name (in "XML element names"), CSVtoXMLConverter will generate missing column names on the fly by appending the column number to a basename. This basename is specified here. For example: **column_** .

XML element names A comma separated list (no spaces) of XML element names for CSV table columns. Set or prefix an element name with a '-' (dash) character to exclude the column from output XML.

Use CSV column titles as XML element names If the checkbox is checked, CSVtoXMLConverter will attempt to use column values of the first CSV table row as XML element names. If you choose this option, you will likely need to use also the "Skip the first CSV table row" option - otherwise the column title row will also appear as column contents in the output XML.

Skip the first CSV table row If the checkbox is checked, CSVtoXMLConverter leaves out the first CSV table row contents from the output.

XML header line XML declaration to start the output XML with. May also contain XML elements or data. (**Optional**)

XML footer line Data to be appended after the XML output. (**Optional**)

String to be appended after the result XML table "Footer" text to insert at the end of the XML output. May contain XML elements or data. (**Optional**)

4.2.3 HTTPConverter

Sends a request to an HTTP server and returns the server response body. It can be used for example to simulate web form submission (GET and POST methods), to upload a file (PUT method) or to send a SOAP message. The component's functionality and parameters are identical to those of the HTTPDestination component.

Host Host name or IP number of the HTTP server.

Port Port number of HTTP server (default: 80).

Path Directory path for the file to be retrieved. With a leading slash ('/'). (**Optional**)

Query/Filename Name of the file to be retrieved. HTTP GET query parameters can appended to the filename.

User User name to be used for protected sites. (**Optional**)

Password Password to be used for protected sites. (**Optional)**

Request method Request method to use. POST, GET, PUT and SOAP methods are available. SOAP method posts the component's input data to the server as Content-Type: text/xml.

Request protocol Request protocol to use: either HTTP or HTTPS.

SOAPAction header SOAPAction header to send with a SOAP request. (**Optional**)

Accept self-signed certificates (HTTPS) Check to allow connections to servers using a self-signed SSL certificate. By default self-signed certificates are not trusted and the connection is aborted. This setting is ignored if **Request protocol** is not set to HTTPS.

Content type Content-Type value for POST and SOAP (=POST) type requests. If left empty, " **text/xml** " is used by default. Note: character encoding information ("; charset= **Charset** ") is automatically appended to the value given here, so the charset info should not be written to this field.

Request parameters Parameters to be used in the request. (**Optional**)

Disable response status code check Check to ignore error codes in the HTTP response and pass the response body normally to the next Pipe component.

Write server response to application log Check to log server responses to requests in OpenSyncro's log file. (**Optional**)

Charset Character encoding to send the HTTP request in. Only affects the SOAP **Request method** .

Choose charset to use in reading the response Check to activate the " **Charset used to read the response** " setting. If not activated, the HTTP(S) response is assumed to be in **ISO-8859-1** charset.

Charset used to read the response Character set to decode the HTTP(S) response with. Requires the checkbox " **Choose charset to use in reading the response** " to be selected - otherwise this setting is ignored.

Request parameters can be specified in two ways. When using

GET or **POST** methods, input data from previous components must be name value pairs separated by "=" and "&". For example:
`id=500&encoding=UTF-8`
Parameters specified in the **Request parameter** field have to be name value pairs, where names and values are separated by "=" and pairs themselves are separated by a newline. For example:
`lang=en`
`req=orders`
Two supplied sets of parameters will be combined (in case GET method is used) in a request:
http://host:port/directory/query?id=500&encoding=UTF-8&lang=en&req=orders.
If

PUT method is used the **Request parameter**

field must be left empty. Input data from previous components is sent as the body in the PUT request.

Using **SOAP** method allows you to send component's input data in the body of the request and the data does not need to be name=value pairs. To send SOAP messages, the component's input data has to be a **complete SOAP XML message including the env:Envelope and env:Body elements** (where xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"). The component simply HTTP POSTs the data as-is without performing XML validation, well-formedness or any other checks.
You can specify additional request parameters in the **Request parameter** field. The

SOAPAction header field can be used to specify the SOAPAction header when sending SOAP requests. Responses to requests can be logged to OpenSyncro's log file. For this the **Write server response to application log** checkbox should be checked. Both the response's HTTP headers and reponse body will be logged.

4.2.4 JDBCConverter

Sends an SQL select query to database and returns the query results in XML. The SQL query is read from the **input data** . The input data may contain **only one** SQL select query.
Results for an example query "select Name, StreetAddress, Phone from SomeCustomerTable;" might look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<Results>
<Row>
<Name>John Row</Name>
<StreetAddress>7th Avenue</StreetAddress>
<Phone>555-12345678</Phone>
</Row>
<Row>
<Name>Jane Row</Name>
<StreetAddress>&lt;unknown&gt;</StreetAddress>
<Phone>555-87654321</Phone>
</Row>
<!- ...  ->
</Results>
```

Host Host name or IP address of the database server

Port Number of the port database server accepts connections to

Username Name of the user to login with

Password Password of the user

Database name Name of the database to login at the server

JDBC driver class name Java class name of the JDBC driver to be used for connecting. For example for MySQL Connector/J try **com.mysql.jdbc.Driver** . The JDBC driver must be included in the Java classpath of OpenSyncro webapp.

Database type Type of the database, e.g. **mysql** .

Charset Character encoding for result XML.

4.2.5 JoinConverter

Joins multiple data parts into one. To be used after components that output multiple strings, such as the SplitConverter.

Delimiter string to be inserted between joined strings String to be appended after each data part when joining the parts into one string. This parameter can be left empty. (**Optional**)

4.2.6 LocalFileReadConverter

Reads contents of a file from the local file system. To be used for replacing or appending input data in the middle of a Pipe.

Directory Full path to the directory containing the file - without trailing slash ('/') character.

File name Name of the file (without path).

Charset Character encoding of the file. This information is required for conversion to OpenSyncro's internal 16-bit Unicode character set.

Append to input data If true, contents of the file are appended to the component's input data.

4.2.7 LocalFileWriteConverter

Writes the data to a file in the local file system. Can be used to capture data at different points in a Pipe.

Directory Full path to the directory the file will be created (or overwritten). No trailing slash ('/') character.

File name Name of the file (without path).

Charset Character encoding to be used in the file.

File write type Output write method used to create file. There are three options available: Always append, overwrite at first iteration, otherwise append, always overwrite. By default always overwrite is selected.

4.2.8 RemoteOrderConverter

Imports XML sales order data to Workspace through OpenInterface. Refer to Workspace OpenInterface API documentation for XML schema and data examples. The component will always return an empty string as a result. The component's functionality and parameters are identical to those of the RemoteOrderDestination component.

Open Interface URL Complete URL path to the Workspace installation, e.g. http://localhost:8080 . No trailing slash ('/') character should be added. The component will append "/workspace.admin/openinterfaceaddress" to this OI URL in order to locate OpenInterface WebService.

Database Name of the Workspace shop's database.

User name Name of the Workspace user as which OpenSyncro will log in to WS. Note: the WS user must have OpenInterface access enabled.

Password Password of the Workspace user (see above).

Import mode Choose one of the following Workspace OpenInterface import methods:

Insert or update - Insert if the order does not exist in the database or update if it does exist.

Only insert - Do not update any existing orders in the database, only insert new orders.

Only update - Update if the order exists in the database, otherwise do nothing.

Invoke 'Received' events for inserted orders If this checkbox is checked, each new order inserted to Workspace will launch actions associated with the 'Received' event.

Additive answer update If this checkbox is checked, the existing list of customer answers is **not cleared** when updating an order. New answers will be appended to the list.

Additive basket update If this checkbox is checked, the existing list of baskets (=order rows, products) is **not cleared** when updating an order. New baskets will be appended to the list.

Note: if **Database** parameter is incorrect, OpenSyncro may write "Failed to get the O.I. Endpoint address" error to the Transfer log - even if the **Open Interface URL** would be correct.

4.2.9 SplitConverter

Splits data into multiple parts around matches of a regular expression.
Note: **SplitConverter outputs**

multiple data parts . All Converter Components after SplitConverter as well as the Destination Component will be executed **once for each data part** .

Regular Expression The input data will be splitted at each occurrence of a pattern matching this regular expression.

String to be inserted at the beginning of each part Insert this string (" **header** ") at the beginning of each splitted data part. (**Optional**)

String to be appended to the end of each part Append this string (" **footer** ") at the end of each splitted data part. (**Optional**)

Don't output empty parts If the input data contains a) more than one Regular Expression match in a sequence without any other data in-between, or b) Regular Expression match immediately at the beginning of the input, SplitConverter would output empty parts (consisting of only the optional header & footer strings) by default. Check this checkbox to disable outputting empty parts.

Don't prefix the first part Check this checkbox to disable inserting the "header" string at the beginning of the first data part that will be output.

4.2.10 WorkspaceHQLResultConverter

Extracts a list of values from the input XML by running an XPath 1.0 query and then proceeds to export all Smilehouse Workspace webshop's (Product or Order) HQL objects that match one of the extracted, unique values. The XPath query, the HQL object name (e.g. "Product") and the object's property name (e.g. "itemCode") to match against the extracted values are given as parameters. From these an HQL query of the following form is generated and sent to Workspace's OpenInterface:

```
from <hqlObjectName> s1 where s1.<hqlPropertyName> in
( <extractedValueList> )
```

References to values through value collections are also supported in the form ":hqlProperty1:hqlProperty2.hqlProperty3", where hqlProperty1 is a collection type property directly under the queried HQL object, hqlProperty2 is a collection type property of hqlProperty1 and hqlProperty3 is a regular (non-collection type) property of hqlProperty2.

If a property name part begins with a colon character, it will be "inner joined" either with the selected HQL object, or with preceding property name part(s). For example a webshop product with a certain item code assigned to one of its product options will be found with HQL object name "Product" and HQL object's property name field value ":options:choices.itemCodeChange". This generates an HQL query with form:

```
select s1 from Product s1 inner join s1.options as s2 inner join
s2.choices as s3 where s3.itemCodeChange in ( <extractedValueList> )
```

Another example, to select products by their inventory shelf name, would be property name "inventory.shelf". No colon characters in this one, since in Workspace there is only one "inventory" connected to "Product" object and in the inventory record there is only one "shelf". Generated HQL query will be simply

```
from Product s1 where s1.inventory.shelf in ( <extractedValueList> )
```

It should be noted that creating HQL queries for Workspace requires expert knowledge of Workspace's data model.

The HQL query results are returned in OpenInterface XML format. WorkspaceHQLResultConverter passes through its Input XML data and appends the HQL query results to the result XML. The final output has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>

<!- The input XML data is passed through here ->

<result>

<!- OpenInterface query results here (Product or Order elements) ->

</result>

</root>
```

Open Interface URL Complete URL path to the Workspace installation, e.g. http://localhost:8080 . No trailing slash ('/') character should be added. The component will append "/workspace.admin/openinterfaceaddress" to this OI URL in order to locate OpenInterface Web-Service.

Database Name of the Workspace shop's database.

User name Name of the Workspace user as which OpenSyncro will log in to WS. Note: the WS user must have OpenInterface access enabled.

Password Password of the Workspace user (see above).

Timeout for the query iterator Maximum time in milliseconds OpenInterface will keep an inactive iterator connection open between OpenSyncro and Workspace. For example 10000 (=ten seconds). **Does not affect the result data.**

Number of results to request from Workspace at once Number of query results the component asks from Workspace OpenInterface at once. **Does not affect the result data.**

XPath 1.0 query for selecting HQL query values XPath 1.0 query to extract values from input XML. The query's result set is filtered to contain **unique values** .

HQL object name to query Workspace's HQL object name to query. For example

Product or **Order** .

HQL object's property to match against the value list returned by XPath query HQL object's property to match the extracted values against. With Product object try e.g. **itemCode** .

Name of the XML root element to contain input XML and HQL query result Name of the root element to contain result XML. For example

root . (**Optional**)

Note: if **Database** parameter is incorrect, OpenSyncro may write "Failed to get the O.I. Endpoint address" error to the Transfer log - even if the **Open Interface URL** would be correct.

4.2.11 XMLGroupExpander

Expands the input XML so that in the output there's one copy of Parent element for every occurrence of a "Child element".

Parent element Name of the XML element which will be output once with each Child element. Parent element will contain the Child element.

Grouping element Name of the element that contains the Child elements in the input XML.

Child element Name of the Child elements to be output with the Parent element.

Retain grouping elements Check this checkbox to pass the Grouping elements from input data to output. By default the Grouping elements are filtered out.

XMLGroupExpander is used to create copies of an XML element and its child elements, so that one child element value is different in each copy. For example, the following input XML:

```
<parentElement name="George">
<someElement>
<anotherElement>some text</anotherElement>
</someElement>
<groupingElement>
<childElement name="Frederik"/>
<childElement name="Bill"/>
</groupingElement>
<someElement>
<anotherElement>some text</anotherElement>
</someElement>
</parentElement>
```

... produces XML output like this:

```
<parentElement name="George">
<someElement>
<anotherElement>some text</anotherElement>
</someElement>
<groupingElement>
<childElement name="Frederik"/>
</groupingElement>
<someElement>
<anotherElement>some text</anotherElement>
</someElement>
</parentElement>
<parentElement name="George">
<someElement>
<anotherElement>some text</anotherElement>
</someElement>
<groupingElement>
<childElement name="Bill"/>
</groupingElement>
<someElement>
```

```
<anotherElement>some text</anotherElement>
</someElement>
</parentElement>
```

Parameter settings for the above example were:

- Parent element = **parentElement**
- Grouping element = **groupingElement**
- Child element = **childElement**
- Retain grouping elements = **true** (checked)

4.2.12 XSLTConverter

Performs an XSL Transformation for XML data using **Xalan**

Java XSLT 1.0 processor.

XSL transformation XSLT 1.0 compatible XSL transformation script to be executed.

4.2.13 XSLT20Converter

Performs an XSL Transformation for XML data using **Saxon**

Java XSLT 2.0 processor. XSLT20Converter includes the Saxon SQL Extension , which allows an XSLT script to perform SQL select queries and other standard operations in JDBC compatible databases.
All xsl:message texts from the XSLT script are redirected to OpenSyncro's Transfer log as debug level messages (verbosity level = "All"). It is also possible for the XSLT script to stop the Pipe execution by throwing an error message as follows:

```
<xsl:message terminate="yes">Some error message here</xsl:message>
```

XSL 2.0 transformation XSLT 2.0 compatible XSL transformation script to be executed.

4.3 Destination Components

4.3.1 FTPDestination

Uploads the data as a file to an FTP server.

Host Host name or IP address of the FTP server.

Port Port number of the FTP server.

User Username to log in with.

Password Password for the User.

Beginning of the file name Full path to the file and the filename itself (for example: "/var/test/destinationfile.txt").

Format of the date included in the file name If you wish to append the current date to the filename, the date format can be specified here according to Java SimpleDateFormat syntax (for example "yyyyMMdd"). This parameter may be left empty. (**Optional**)

File extension Filename suffix (for example "txt"), without the leading dot. This parameter is intended to be used together with "Format of the date included in the file name" option and may be left empty. (**Optional**)

File type Chooses the FTP file transfer mode to be used, either Binary (recommended) or ASCII.

Charset Character encoding of the file to send.

4.3.2 HTTPDestination

Sends a request to an HTTP server. It can be used for example to simulate web form submission (GET and POST methods), to upload a file (PUT method) or to send a SOAP message. The component's functionality and parameters are identical to those of the HTTPConverter component.

Host Host name or IP number of the HTTP server.

Port Port number of HTTP server (default: 80).

Path Directory path for the file to be retrieved. With a leading slash ('/'). (**Optional**)

Query/Filename Name of the file to be retrieved. HTTP GET query parameters can appended to the filename.

User User name to be used for protected sites. (**Optional**)

Password Password to be used for protected sites. (**Optional)**

Request method Request method to use. POST, GET, PUT and SOAP methods are available. SOAP method posts the component's input data to the server as Content-Type: text/xml.

Request protocol Request protocol to use: either HTTP or HTTPS.

SOAPAction header SOAPAction header to send with a SOAP request. (**Optional**)

Accept self-signed certificates (HTTPS) Check to allow connections to servers using a self-signed SSL certificate. By default self-signed certificates are not trusted and the connection is aborted. This setting is ignored if **Request protocol** is not set to HTTPS.

Content type Content-Type value for POST and SOAP (=POST) type requests. If left empty, " **text/xml** " is used by default. Note: character encoding information ("; charset= **Charset** ") is automatically appended to the value given here, so the charset info should not be written to this field.

Request parameters Parameters to be used in the request. (**Optional**)

Disable response status code check Check to ignore error codes in the HTTP response and pass the response body normally to the next Pipe component.

Write server response to application log Check to log server responses to requests in OpenSyncro's log file. (**Optional**)

Charset Character encoding to send the HTTP request in. Only affects the SOAP **Request method** .

Choose charset to use in reading the response Check to activate the " **Charset used to read the response** " setting. If not activated, the HTTP(S) response is assumed to be in **ISO-8859-1** charset.

Charset used to read the response Character set to decode the HTTP(S) response with. Requires the checkbox " **Choose charset to use in reading the response** " to be selected - otherwise this setting is ignored.

Request parameters can be specified in two ways. When using

GET or **POST** methods, input data from previous components must be name value pairs separated by "=" and "&". For example:
`id=500&encoding=UTF-8`
Parameters specified in the **Request parameter** field have to be name value pairs, where names and values are separated by "=" and pairs themselves are separated by a newline. For example:
`lang=en`
`req=orders`
Two supplied sets of parameters will be combined (in case GET method is used) in a request:
http://host:port/directory/query?id=500&encoding=UTF-8&lang=en&req=orders.
If

PUT method is used the **Request parameter**

field must be left empty. Input data from previous components is sent as the body in the PUT request.

Using **SOAP** method allows you to send component's input data in the body of the request and the data does not need to be name=value pairs. To send SOAP messages, the component's input data has to be a **complete SOAP XML message including the env:Envelope and env:Body elements** (where xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"). The component simply HTTP POSTs the data as-is without performing XML validation, well-formedness or any other checks.
You can specify additional request parameters in the **Request parameter** field. The

SOAPAction header field can be used to specify the SOAPAction header when sending SOAP requests. Responses to requests can be logged to OpenSyncro's log file. For this the **Write server response to application log** checkbox should be checked. Both the response's HTTP headers and reponse body will be logged.

4.3.3 JDBCDestination

Sends SQL Insert, Update and Delete statements read from input data to a database.

JDBC URL Full JDBC URL (consult your JDBC driver's documentation), including the database's name and type. MySQL example: jdbc:mysql://localhost/testdatabase

Username Name of the user to login with

Password Password of the user

JDBC driver class name Java class name of the JDBC driver to be used for connecting. For example for MySQL Connector/J try **com.mysql.jdbc.Driver** . The JDBC driver must be included in the Java classpath of OpenSyncro webapp.

4.3.4 LocalFileDestination

Writes the data to a file in local file system.

Directory Full path to the directory the file will be created (or overwritten). No trailing slash ('/') character.

File name Name of the file (without path).

String to add at beginning of file A String that will be added to the start of the file.

String to add at end of file A String that will be added to the end of the file.

Charset Character encoding to be used in the file.

4.3.5 MultiFileDestination

Writes the data to multiple files in local file system. Each data block will be written to a different file, whose names consist of a prefix, auto-incrementing index number and a file extension.

Directory Full path to the **directory** containing the file - without trailing slash ('/') character.

File name prefix Beginning of the filename, before the automatically generated Date.

Index number start value The auto-incrementing counter between File name prefix and File extension starts with this (integer) value

File extension Filename suffix (for example "txt") **without** the leading **dot** . This parameter may be left empty.

Charset Character encoding of the file(s). This information is required for conversion to OpenSyncro's internal 16-bit Unicode character set.

4.3.6 RemoteCustomerDestination

Imports XML customer data to Workspace through OpenInterface. Refer to Workspace OpenInterface API documentation for XML schema and data examples.

Open Interface URL Complete URL path to the Workspace installation, e.g. http://localhost:8080 . No trailing slash ('/') character should be added. The component will append "/workspace.admin/openinterfaceaddress" to this OI URL in order to locate OpenInterface Web-Service.

Database Name of the Workspace shop's database.

User name Name of the Workspace user as which OpenSyncro will log in to WS. Note: the WS user must have OpenInterface access enabled.

Password Password of the Workspace user (see above).

Import mode Choose one of the following Workspace OpenInterface import methods:

Insert or update - Insert if the customer does not exist in the database or update if it does exist.

Only insert - Do not update any existing customers in the database, only insert new customers.

Only update - Update if the customer exists in the database, otherwise do nothing.

Insert as new - Does not try to match existing customers, inserts all customers as new records.

Replace all - **Clears the entire customer database** before starting to import (insert) customers. Note: This mode can not be currently used in iterating Pipes, as it will clear the database before importing every block of data, leaving only the last data block contents in the database.

Create customer groups If this checkbox is checked, all non-existent customer groups referenced by the imported data will be created.

4.3.7 RemoteOrderDestination

Imports XML sales order data to Workspace through OpenInterface. Refer to Workspace OpenInterface API documentation for XML schema and data examples.

Open Interface URL Complete URL path to the Workspace installation, e.g. http://localhost:8080 . No trailing slash ('/') character should be added. The component will append "/workspace.admin/openinterfaceaddress" to this OI URL in order to locate OpenInterface Web-Service.

Database Name of the Workspace shop's database.

User name Name of the Workspace user as which OpenSyncro will log in to WS. Note: the WS user must have OpenInterface access enabled.

Password Password of the Workspace user (see above).

Import mode Choose one of the following Workspace OpenInterface import methods:

Insert or update - Insert if the order does not exist in the database or update if it does exist.

Only insert - Do not update any existing orders in the database, only insert new orders.

Only update - Update if the order exists in the database, otherwise do nothing.

Invoke 'Received' events for inserted orders If this checkbox is checked, each new order inserted to Workspace will launch actions associated with the 'Received' event.

Additive answer update If this checkbox is checked, the existing list of customer answers is **not cleared** when updating an order. New answers will be appended to the list.

Additive basket update If this checkbox is checked, the existing list of baskets (=order rows, products) is **not cleared** when updating an order. New baskets will be appended to the list.

Note: if **Database** parameter is incorrect, OpenSyncro may write "Failed to get the O.I. Endpoint address" error to the Transfer log - even if the **Open Interface URL** would be correct.

4.3.8 RemotePriceListDestination

Imports XML contract pricing data to Workspace through OpenInterface. Refer to Workspace OpenInterface API documentation for XML schema and data examples.

Open Interface URL Complete URL path to the Workspace installation, e.g. http://localhost:8080 . No trailing slash ('/') character should be added. The component will append "/workspace.admin/openinterfaceaddress" to this OI URL in order to locate OpenInterface Web-Service.

Database Name of the Workspace shop's database.

User name Name of the Workspace user as which OpenSyncro will log in to WS. Note: the WS user must have OpenInterface access enabled.

Password Password of the Workspace user (see above).

Import mode Choose one of the following Workspace OpenInterface import methods:

Insert or update - Insert if the pricelist entry does not exist in the database or update the entry if it does exist.

Only insert - Do not update any existing pricelist entries in the database, only insert new entries.

Only update - Update if the pricelist entry exists in the database, otherwise do nothing.

Replace all - **Clears the entire pricelist** before starting to import (insert) pricelist entries. Note: This mode can not be currently used in iterating Pipes, as it will clear the database before importing every block of data, leaving only the last data block contents in the database.

Note: if **Database** parameter is incorrect, OpenSyncro may write "Failed to get the O.I. Endpoint address" error to the Transfer log - even if the **Open Interface URL** would be correct.

4.3.9 RemoteProductDestination

Imports XML product data to Workspace through OpenInterface. Refer to Workspace OpenInterface API documentation for XML schema and data examples.

Open Interface URL Complete URL path to the Workspace installation, e.g. http://localhost:8080 . No trailing slash ('/') character should be added. The component will append "/workspace.admin/openinterfaceaddress" to this OI URL in order to locate OpenInterface Web-Service.

Database Name of the Workspace shop's database.

User name Name of the Workspace user as which OpenSyncro will log in to WS. Note: the WS user must have OpenInterface access enabled.

Password Password of the Workspace user (see above).

Import mode Choose one of the following Workspace OpenInterface import methods:

Insert or update - Insert if the product does not exist in the database or update if it does exist.

Only insert - Do not update any existing products in the database, only insert new products.

Only update - Update if the product exists in the database, otherwise do nothing.

Replace all

- **Clears the entire product database** before starting to import (insert) products. Note: This mode can not be currently used in iterating Pipes, as it will clear the database before importing every block of data, leaving only the last data block contents in the database.

Create product groups If this checkbox is checked, all non-existent product groups referenced by the imported data will be created.

Additive product group update If this checkbox is checked, an updated product will be only added to new product groups and not removed from any existing groups. If the checkbox is not checked (default), the product group(s) from the XML will replace the product's current group associations in database.

Additive product option update If this checkbox is checked, only new product options can be only added to an updated product and existing product options will not be removed. If the checkbox is not checked (default), product option list from the XML will replace the current options of a product in database.

Protected groups A list of product groups (one full product group path per line) to protect from changes.

Note: if **Database** parameter is incorrect, OpenSyncro may write "Failed to get the O.I. Endpoint address" error to the Transfer log - even if the **Open Interface URL** would be correct.

4.3.10 TimestampFileDestination

Writes data to a local file with timestamp in the filename. The timestamp pattern should conform to java.text.SimpleDateFormat.

Directory Full path to the **directory** containing the file - without trailing slash ('/') character.

File name prefix Beginning of the filename, before the automatically generated Date.

Date format Date format according to Java SimpleDateFormat syntax (for example "yyyyMMdd").

File extension Filename suffix (for example "txt") **without** the leading **dot** . This parameter may be left empty.

Charset Character encoding of the file. This information is required for conversion to OpenSyncro's internal 16-bit Unicode character set.

5 Use of Connector Pack

This chapter contains a general guide to activate Connector Pack.
The Connector Packs use XSLT to convert data and to perform/apply business logic. Modifying XSLT enables building custom transformations to any desired format. The Converter Components of OpenSyncro makes it possible to convert other formats to XML, for example by using ASCIItoXML or CSVtoXML converters.
Usually it takes 3-4 working days to configure and test data transfers depending on complexity of interfaces and transferred data.
Activation of different Connector Packs involves different configurations.
The following general tasks are usually needed:

- Setting up source and destination systems.
- Setting up OpenSyncro Pipes.

5.1 Setting up source and destination systems

The first step is usually to synchronize the values of both systems. It may contain setting up parameters like product properties, customer groups, shipping methods, payment methods, prices, VAT codes and so on.
The second step is to make the ends recognize that information. So mapping is needed. It connects parameters with different names.
The third step is to set up locations and destinations of different protocols like FTP, HTTP and JDBC. Also special conditions can be defined, for example what happens if product group is unknown.

5.2 Setting up OpenSyncro Pipes

First step to do is to create new OpenSyncro Pipes (see: Pipe list).
Second step is to set the configurations of Pipes and test the integration proses (see: Pipe editor and Transfer log).
The third step is to build up automation by setting up a schedule of Pipes to import/export data (see: Pipe editor).