A1-D10-3

Implementation: L-DSMS – A Local Data Stream Management System

Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Project number:	IST-2004-506779
Project instrument:	EU FP6 Network of Excellence (NoE)
Project thematic priority:	Priority 2: Information Society Technologies (IST)
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	IST506779/Munich/A1-D10-3/D/PU/a1
Responsible editors:	Christian Hänsel
Reviewers:	H.J. Ohlbach
Contributing participants:	Munich
Contributing workpackages:	A1
Contractual date of deliverable:	29 February 2008
Actual submission date:	14 March 2008

Abstract

L-DSMS is a Local Data Stream Management System. It is a Java Program which can read an XML-file with a description of a network of processing nodes for streaming data. L-DSMS automatically combines all the processing nodes into a single Java program which then processes the data. L-DSMS has a number of predefined nodes, together with an interface for implementing new processing nodes. The generated network can be remotely monitored and reconfigured by a client, Visu L-DSMS. An example application of L-DSMS is the transformation of traffic messages coming via radio in TMC into KML, which, in turn, can be visualised by Google Earth.

Keyword List

datastream management

Project co-funded by the European Commission and the Swiss Federal Office for Education and Science within the Sixth Framework Programme.

Implementation: L-DSMS – A Local Data Stream Management System

Christian Hänsel

Department of Computer Science, University of Munich
Email: ohlbach@ifi.lmu.de

14 March 2008

Abstract
L-DSMS is a Local Data Stream Management System. It is a Java Program which can read an XML-file with a description of a network of processing nodes for streaming data. L-DSMS automatically combines all the processing nodes into a single Java program which then processes the data. L-DSMS has a number of predefined nodes, together with an interface for implementing new processing nodes. The generated network can be remotely monitored and reconfigured by a client, Visu L-DSMS. An example application of L-DSMS is the transformation of traffic messages coming via radio in TMC into KML, which, in turn, can be visualised by Google Earth.

Keyword List
datastream management

Contents

Chapter 1

Concepts

L-DSMS is a Local Data Stream Management System. It is a Java Program which can read an XML-file with a description of a network of processing nodes for streaming data. L-DSMS automatically combines all the processing nodes into a single Java program which then processes the data. L-DSMS has a number of predefined nodes, together with an interface for implementing new processing nodes. The generated network can be remotely monitored and reconfigured by a client, Visu L-DSMS. An example application of L-DSMS is the transformation of traffic messages coming via radio in TMC into KML, which, in turn, can be visualised by Google Earth.

This chapter describes the concepts behind L-DSMS and explains how the system works. It should be read carefully in order to understand, what kind of scenarios L-DSMS can be used for, and how to create own L-DSMS projects.

1.1 Basic Concept

One approach to designing L-DSMS could have been to create a monolithic software with every necessary functionality. Such a software, however, can hardly be implemented because nobody knows every possible functionality somebody may ever need. Hence, a concept is needed, that allows the user to enable or disable functions and allows him to extend L-DSMS easily with new functions, if necessary. A monolithic software doesn't fulfill this conditions and it would be difficult to extend and to maintain it. Users will find it very inefficient to analyze a full software system for creating a new functionality, if the logic of the functionality itself can be implemented in only a few lines of code.

A second approach is to distribute the complexity over different 'micro components'. Every micro component provides only the necessary functionality for subproblems. It is therefore very small, easy to maintain, implemented with a few lines of code and highly specialized. One component alone, however, is quite useless. To realize the required functionality, different components have to work together as a kind of production line (cf. figure 1.1). Building up the required system by combining the necessary components is the basic functionality of L-DSMS itself.

At the very beginning of this production line is the raw input data, that is produced by every kind of data source (e.g. sensor). This input will be processed by each component one after another to be the final output after the last component has done its job. To change this

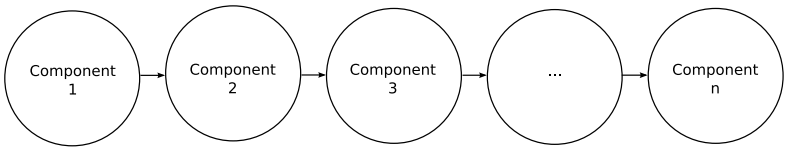


Figure 1.1: components combined to a linear production line

process, components have to be replaced, removed or added, but it isn't necessary to create a total new system. This flexibility allows the user to deactivate unused functions and to run the system with minimal hardware requirements. Unlike in a real processing line, a component can have an arbitrary number of following components, to allow parallel processing paths that can have a tree or even a graph structure. An advantage of configurations with parallel processing paths is the increased stability of the system, because errors in one processing path don't affect other paths.

L-DSMS was designed after the second approach and provides some core functionality to realize the concept of processing lines. The structure is automated build up, based on the informations from a XML configuration document. This allows to change the structure without the need of programming any line of code and the configuration document can be edited by any user with a text editor of its choice. More informations about the configuration file, its layout and how to use it can be found in section A.2.

1.2 Components

As described in the previous section, L-DSMS is based on a core that provides the functionality to combine single components to a network of components. This is done based on the informations from a configuration file. These connected components realize the real functionality and this section describes, what kind of component classes are available, how they work and what they are used for. A detailed description to each component available in the L-DSMS core package is available in the chapter 5.

1.2.1 Source

A source is always at the head of a production path and produces the data that needs to be processed. Each source produces this data in a different way, depending on its implementation. The L-DSMS core package contains sources that read data from files, sockets or external hardware (cf. chapter 5). In most of the cases, a source recieves its data from something external of

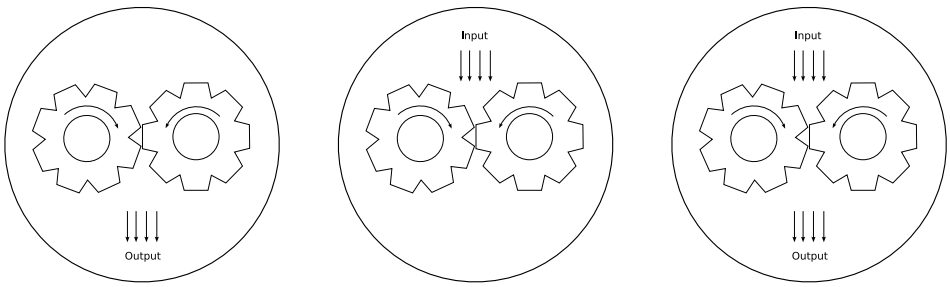
L-DSMS (e.g. a sensor). Nevertheless, within the scope of L-DSMS we are recognising a source as the data source without caring about where this data is originally from. This is the main difference to a node.

To provide additional informations about the produced data, a source can create optional metainformations for each produced data package. The data together with its optional metadata forms the output of a source. This output would be useless, if no other component consumes it. Therefore, a source has to have at least one child component, that recieves the output. Child components for a source can be an arbitrary number of drains or nodes (cf. figure 1.3). Which drains or nodes are children for a source, is configured in the configuration file (cf. section A.2). To connect a drain or node to a source, you have to consider that the output types (data and meta) of the source are compatible to all child component input types. This means, if Source S_1 produces data of type t_1 and metadata of type t_2, all connected drains and nodes have to accept data of type t_1 and metadata of type t_2. If not, L-DSMS will produce an error and may not run. A drain or node allways accepts data of type t_1 and metdata of type t_2 if its own data input type is t_1 or *java.lang.Object* and its own metadata input type is t_2 or *java.lang.Object*. Detailed descriptions about the input and output types for each component can be found in chapter 5.

The configuration file contains, beside the relation between sources and their child components, some source specific properties. Two properties can be specified for all sources. The first property is the type itself, which is not optional and has to be specified using the attribute `class`. The second property is optional and specifies the name of the source using the attribute `name`. If you specify the name, it has to be unique and no other component is allowed to be named equal because this name is used by drains or nodes to identify sources as additional parents. All other properties are source specific and are listed in the detailed description for each source. The ByteArrayFileSource (cf. section 5.2.6) for example has the additional attributes `file`, `delay` and `repeat`.

1.2.2 Drain

A drain is a component, that consumes data and is always the tail of a production path. Within the scope of L-DSMS, we are recognising a drain as the final reciever of data, because it can't have any child components inside of the system boundaries of L-DSMS. Beside of this,



(a) source (b) drain (c) node

Figure 1.2: schema for the available components

can the data be further processed by systems outside of L-DSMS. If you take for example a SocketDrain, that provides the incomming data to every process connected to this drain via a socket connection. This data doesn't end in the SocketDrain but leaves the system boundaries of L-DSMS. The L-DSMS core package contains drains, that stores the incomming data to files, send them over socket connections or simply print them to the console screen.

Drains can recieve their data from one ore more sources as well as nodes (cf. figure 1.4). These connections between the drain and its parent components are configured in the configuration file and will be managed automated by L-DSMS. As described in the previous section, a source can produce beside the data some corresponding metadata. Consequently, a drain can recieve a data package as well as its corresponding metadata package and both together are the input of the drain. To connect a drain with a source or node, the parent component has to produce output, that can be processed by the drain. This means, that the data output type of the parent has to be compatible to the data input type of the drain and the metadata output type of the parent has to be compatible to the metadata input type of the drain. A drain allways accepts data of type t_1 and metdata of type t_2 if its own data input type is t_1 or *java.lang.Object* and its own metadata input type is t_2 or *java.lang.Object*. For a detailed description about the input and output types for each component take a look at chapter 5.

In the configuration file is not only the relation between the drain and its parents configured, it is also the place to configure drain specific properties. Two properties can be specified for all types. The first property is the type itself which is not optional and has to be specified using the attribute `class`. The second property is optional and specifies the names of additional sources using the attribute `sourcerefs`. If you specify sourcerefs, the names are used to specify the sources or nodes, identified by the names, as parent components for this drain. All other properties are drain specific and are listed in the detailed description for each drain. The ByteArrayFileDrain (cf. section 5.2.5) for example has the additional attribute `file`.

1.2.3 Node

Nodes are a combination of sources and drains and can be located at every possible position within a production line. They are designed to recieve data from one ore more components, to process it and forward the results to an arbitrary number of components following them in the

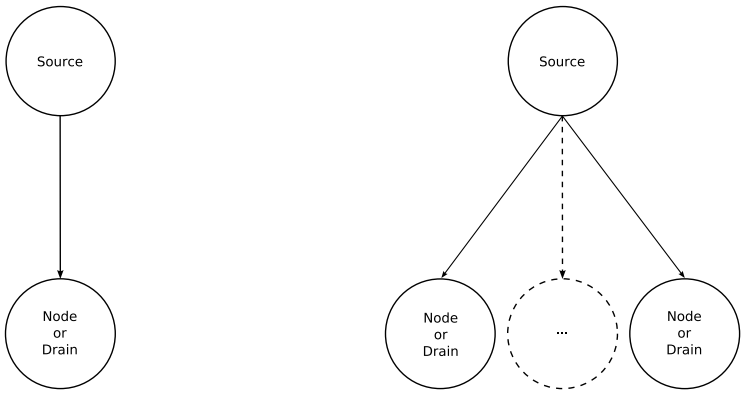


Figure 1.3: source with one or more children

hierarchy. Every single work, that is done between a single source and all drains at the end of the processing line is done by an arbitrary number of nodes. Consequently, because nodes are sources as well as drains, they inherit the properties of both of them. Therefore, they have the same restrictions regarding the process of connecting them with other components. To connect a source (or node) with a node, you have to consider, that the type of the outgoing data as well as the type of the outgoing metadata is compatible to the input types of this node. The same counts, if you want to connect a node with a drain (or node). Nodes are configured the same way as sources and drains. Beyond node specific properties, they all have the non-optional `class` attribute that is used to define the type. They can have a name (like a source), that is specified using the attribute `name` and additional sources (like a drain), that are specified using the `sourcerefs` attribute.

Each node available in the L-DSMS core package processes the data in a different way and therefore exists no general conclusion about the outcome of nodes. Some of them produces data, that has the same type as the input and others produces data of a totaly different type. They even doesn't allways produce output for every input they have recieved but discard the recieved data if it doesn't fulfils their conditions (e.g. a filter).

1.3 Logging

L-DSMS was designed to run as a server process without any need of user interaction. To ensure, that the system is working correct, the system administrator need to monitor it. Therefore L-DSMS can print logging messages to the console screen to inform the system administrator about internal events. These messages are distinguished into the categories:

- INFO
- WARN
- ERROR
- DEBUG

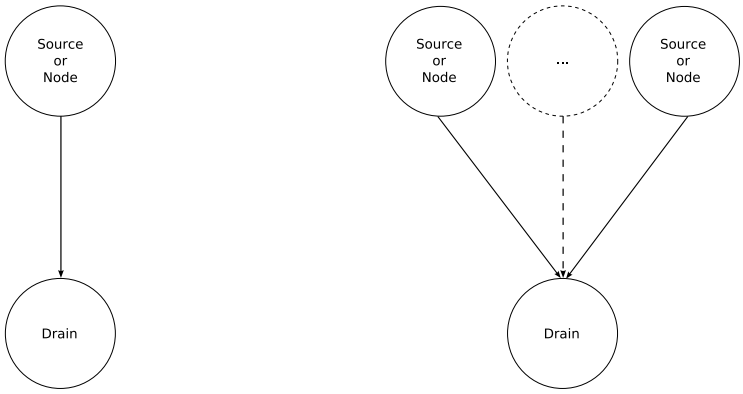


Figure 1.4: drain with one or more parents

These categories are hierarchical ordered, beginning with INFO as the lowest and ends with DEBUG as the highest category. You can assign a logging category to each component separately as well as a default logging level for the system, that will be used for each component, no category was specified for.
INFO is the lowest category and therefore contains no other categories. Info messages are used to inform the system administrator about usual and non-critical events (e.g. system started, component loaded, client connected/disconnected).

WARN messages are used to inform the system administrator about unusual and non-critical events. Using the WARN category includes all messages assigned to the INFO category (e.g. client is nomore reachable, end of file reached).

ERROR messages are used to inform the system administrator about unusual and mayby critical events. Using the ERROR category includes all messages assigned to the INFO and WARN categories (e.g. illegal input data, host unreachable).

DEBUG messages are used especialy by developers. They are used to inform a developer about what's going on and to help him tracing errors and bugs. Using the DEBUG category includes all messages assigned to the INFO, WARN or ERROR category.

1.4 Plug-in

L-DSMS has a plug-in mechanism to enrich the core functionality of L-DSMS without the necessity to change any line of code of L-DSMS. The plug-ins can be hooked into L-DSMS just by adding them into the configuration file. This makes it possible to enable and disable functions as needed.

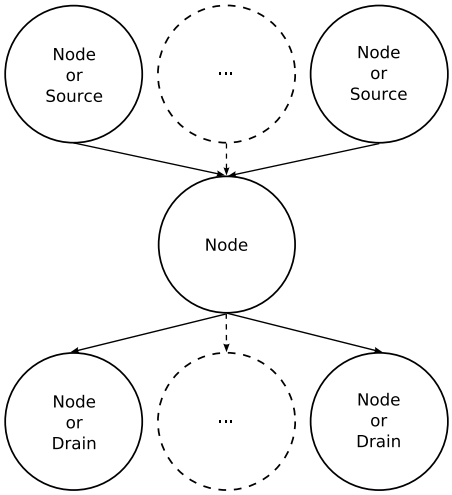


Figure 1.5: node with an arbitrary number of parents and children

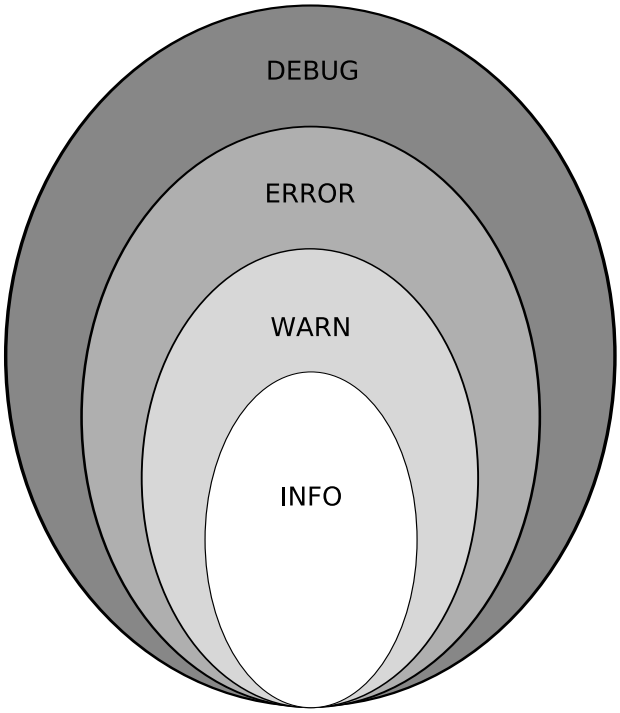


Figure 1.6: hierarchy of the logging categories

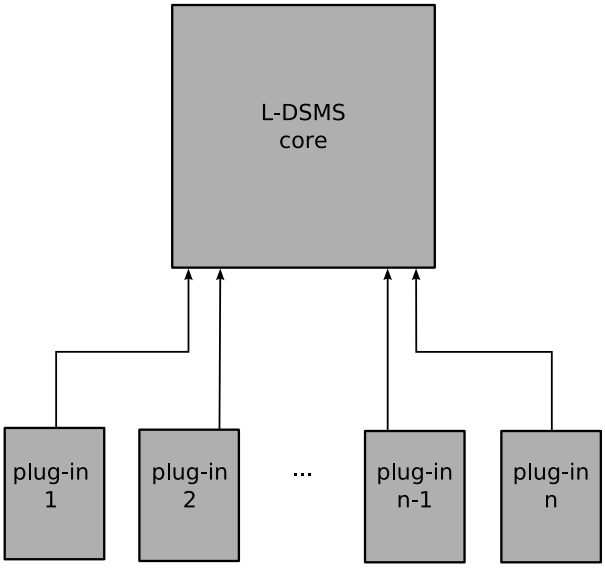


Figure 1.7: plug-ins hooked to the L-DSMS core

Chapter 2

Installation

This chapter explains, how to set up LDSMS on your Linux or Windows system. LDSMS is written in Java. So it needs a working J2SE™ Runtime Environment (JRE) (version $\geq$ 5.0). Setting up a working JRE is described in the first section. In the second section, the installation and configuration process of LDSMS itself will be described.

2.1 Install Java

This section describes, how to set up a running JRE 5.0 at your system. If you'd like to install LDSMS on a Linux system, please proceed with 2.1.1. If you'd like to install LDSMS on a Windows system, please proceed with 2.1.2.

2.1.1 Linux

2.1.1.1 check JRE version

If you think, there is allready a JRE installed, please perform the following steps, to check if the version of the installed JRE fits our needs, otherwise skip this part. To run LDSMS you need a version greater or equal to 1.5 (5.0 is a synonym for 1.5).

1. Open a console window (e.g. press 'Alt'+'F2', type 'xterm' (cf. figure 2.1) and press 'Enter').
2. Type `java -version` in the console window and press `Enter`.

If the result looks like shown in figure 2.2, a JRE is already installed and correct configured. In our case, the current version is 1.5.0_11, but everything $\geq$ 1.5 will fit and you can skip this section and proceed with section 2.2.
If the result looks like shown in figure 2.3, no JRE is installed or your JRE is not configured correctly and you have to go on with the following install instructions.

2.1.1.2 Download the JRE offline installation file

Go to `http://java.sun.com/javase/downloads/index_jdk5.jsp` to download the latest release of JRE 5.0. To download the latest release of JRE instead, go to

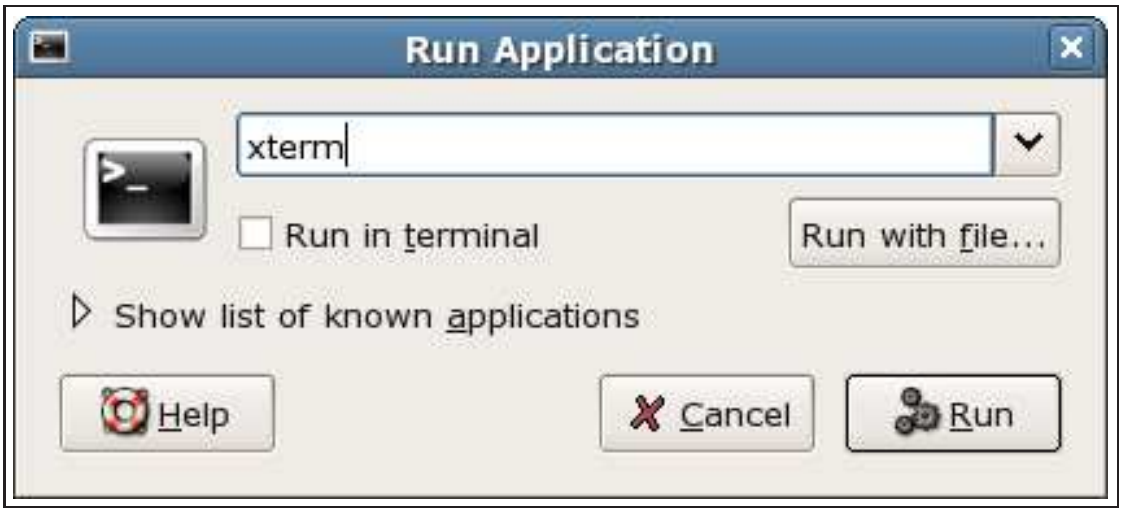


Figure 2.1: run application

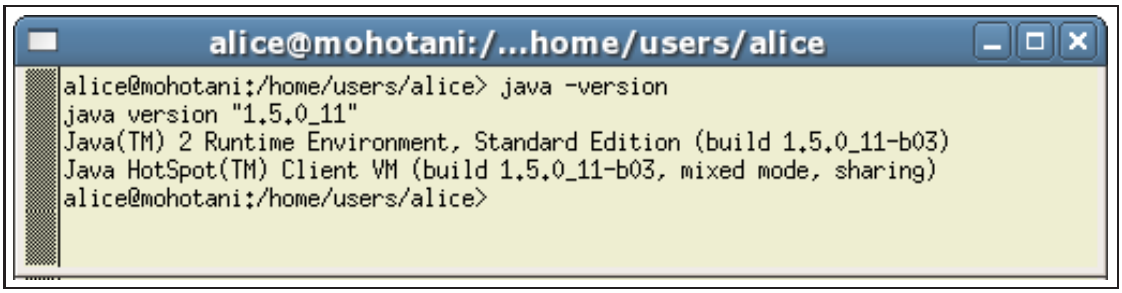


Figure 2.2: java -version (successful)

Figure 2.3: java -version (unsuccessful)

`http://java.sun.com/javase/downloads/index.jsp`. Press the download button for *Java Runtime Environment (JRE) 5.0* (cf. figure 2.4) and follow the download instructions to save the file (e.g. at `/Downloads/Java/`).

Figure 2.4: Download JRE 5.0

2.1.1.3 Run the installer

After downloading the installation package, open the folder where the file is located. Execute the installation package and follow the installation instructions.

2.1.1.4 Configure path environment

To use the JRE, you need to tell your system, where to find the JRE. If you run a command within the console (e.g. `java -version`), the system will try to find this executable in the current folder. If the executable isn't located in the current folder, the system will try to find the executable in every folder, that is listed in the `PATH` variable. So we need to add the JRE installation folder to the `Path` variable. To set the PATH variable, follow these steps:

1. Open a console window (e.g. press 'Alt'+'F2', type 'xterm' (cf. figure 2.1) and press 'Enter').
2. Type "vi ~/.bashrc" and press 'Enter'.
3. Press 'i' to change to the 'Insert' mode.
4. Insert "export JAVA_HOME=<install folder>" where <install folder> is your JRE installation folter (e.g '/Program Files/Java/jre1.5.0_11') in a new line.
5. Search for the line, that starts with "export PATH" and append ":JAVA_HOME" to this line.
6. Press 'Esc' to stop the 'Insert' mode.
7. Type ':' to switch to the 'Command' mode.
8. Press 'x' and press 'Enter' to save the changes and close the file.
9. Type '. ~/.bashrc' to load the changes.

2.1.2 Windows (2000/XP)

2.1.2.1 check JRE version

If you think, there is allready a JRE installed, please perform the following steps, to check if the version of the installed JRE fits our needs, otherwise skip this part. To run LDSMS you need a version greater or equal to 1.5 (5.0 is a synonym for 1.5).

1. Click the start button (cf. figure 2.5).
2. Select "Run..." from the menu.
3. Type `cmd` in the window (cf. figure 2.6) and click OK. This opens a command prompt window.
4. Type `java -version` in the command prompt window and press `Enter`.

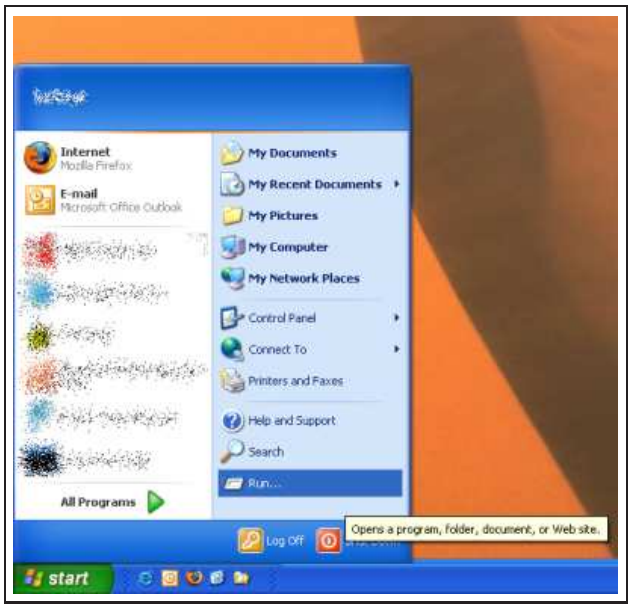


Figure 2.5: Start

If the result looks like shown in figure 2.7, a JRE is already installed and correct configured. In our case, the current version is 1.5.0_11, but everything ≥ 1.5 will fit and you can skip this section and proceed with section 2.2.
If the result looks like shown in figure 2.8, no JRE is installed or your JRE is not configured correctly and you have to go on with the following install instructions.

2.1.2.2 Download the JRE offline installation file

Go to `http://java.sun.com/javase/downloads/index_jdk5.jsp` to download the latest release of JRE 5.0. To download the latest release of JRE instead, go to

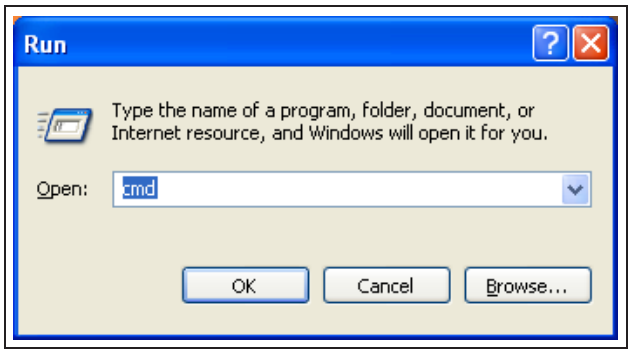


Figure 2.6: Run

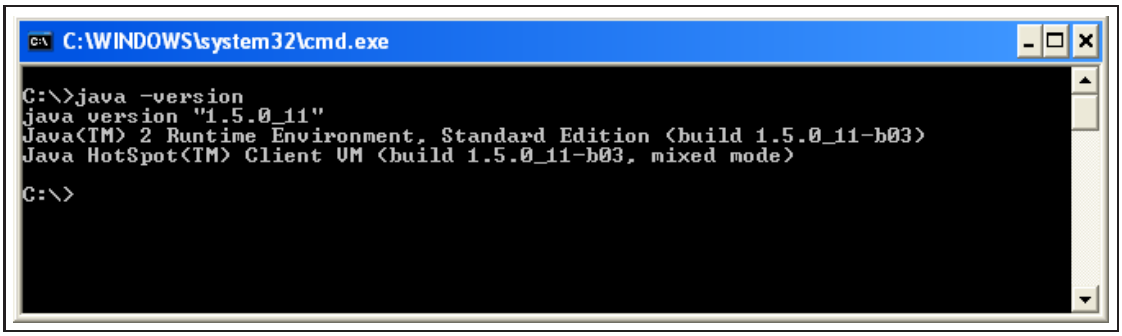


Figure 2.7: Version

`http://java.sun.com/javase/downloads/index.jsp`. Press the download button for *Java Runtime Environment (JRE) 5.0* (cf. figure 2.9) and follow the download instructions to save the file (e.g. at `C:\Downloads\Java\`).

2.1.2.3 Run the installer

After downloading the installation package, open the folder where the file is located. Execute the installation package and follow the installation instructions.

2.1.2.4 Configure path environment

To use the JRE, you need to tell your system, where to find the JRE. If you run a command within a command prompt window (e.g. `java -version`), the system will try to find this executable in the current folder. If the executable isn't located in the current folder, the system will try to find the executable in every folder, that is listed in the `Path` variable. So we need to add the JRE installation folder to the `PATH` variable. To set the PATH variable, follow these steps:

1. Click the start button.
2. Open the 'Control Panel'.

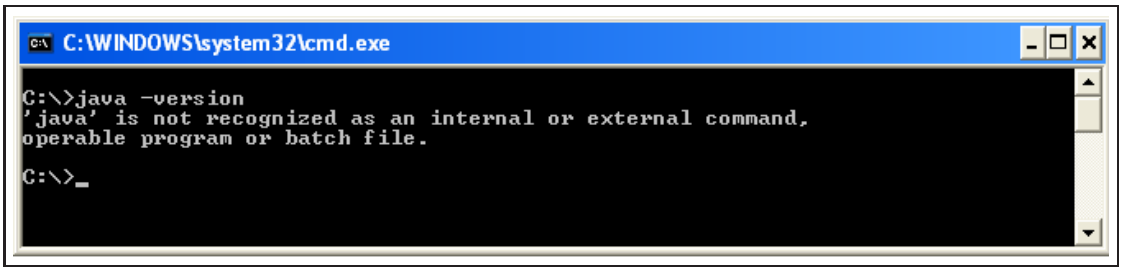


Figure 2.8: Error

Figure 2.9: Download JRE 5.0

3. Go to System, select the 'Advanced' tab and click the 'Environment Variables' button (cf. figure 2.10).
4. In the region 'System Variables' click the 'New' button.
5. Set 'Variable Name' to 'JAVA_HOME' and 'Variable Value' to your JRE installation folter (e.g "C:\Program Files\Java\jre1.5.0_11", cf. figure 2.11) and click the 'OK' button (cf. figure 2.13).
6. In the region 'System Variables' select "Path" and click the 'Edit' button.
7. Add ";%JAVA_HOME%\bin" at the end of the 'Variable value' field (cf. figure 2.12) and click the 'OK' button (cf. figure 2.14).

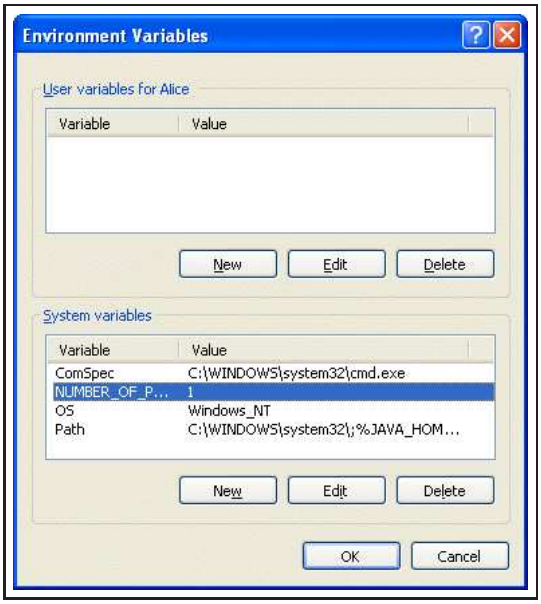


Figure 2.10: configure 'Environment Variables'

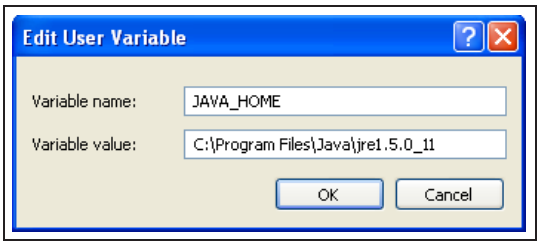


Figure 2.11: system variable JAVA_HOME

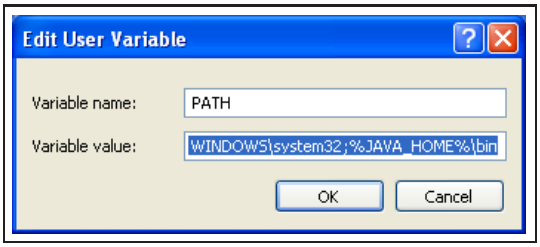


Figure 2.12: system variable PATH

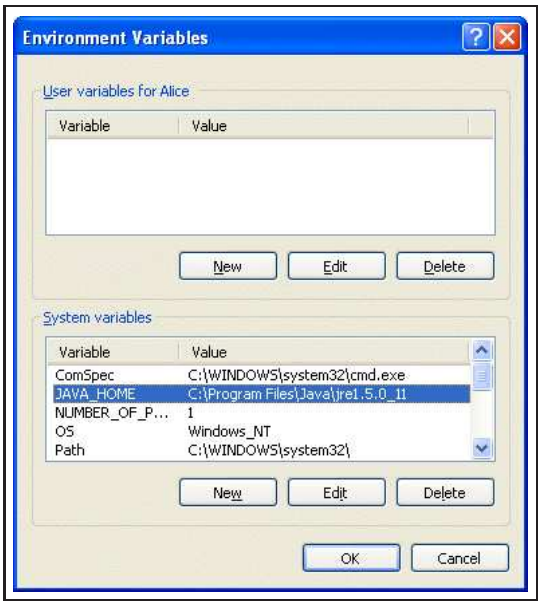


Figure 2.13: JAVA_HOME configured

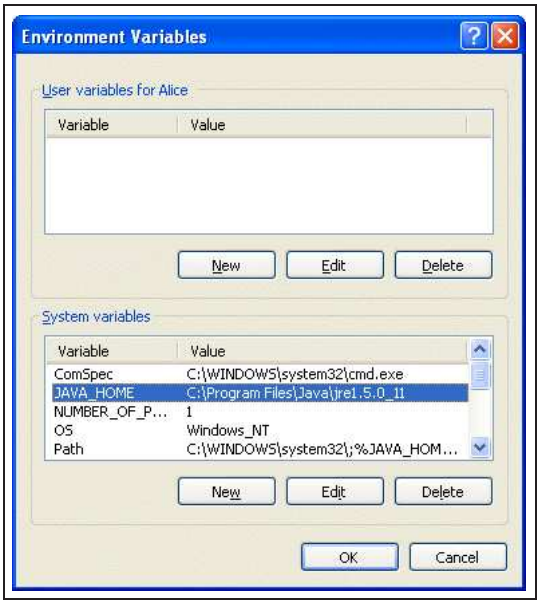


Figure 2.14: JAVA_HOME and PATH configured

2.2 L-DSMS

This section describes step-by-step, how to set up L-DSMS at your system assuming that a working JS2E^{TM} Runtime Environment is allready available.

1. Download the latest release from the LDSMS project page `http://www.pms.ifi.lmu.de/rewerse-wga1/ldsms/`.
2. Unzip the file to your program folder (e.g. /Program Files/ or C:\Program Files\).
3. You now should have a new folder 'ldsms' in your program folder, we will refer to this folder using the <LDSMS_HOME> as a symbolic representation. This folder should contain
   - config (folder)
   - lib (folder)
   - ldsms.jar (file)
4. Download all additional libraries, listed in section A.1 and save them in <LDSMS_HOME>/lib.
5. Test your installation with the Hello World example (section 3.2).

Chapter 3

Examples

3.1 General

This section contains general informations for you, that are useful in the following examples. To increase the clearness of the example descriptions, these informations are collected in the current subsections and will be referenced within the example descriptions as needed.

3.1.1 Path Informations

All paths are given in the Linux syntax (with a slash '/' as separator). Windows user need to replace the slash ('/') by a backslash ('\') and add a hard drive label (e.g. C:\). The first listing shows a path in Linux syntax.

Listing 3.1: Path for Linux users

```
/Program Files/ldsms/
```

The second listing shows the (semantic) same path for Windows users.

Listing 3.2: Path for Windows users

```
C:\\Program Files\ldsms\
```

3.1.2 Install Folder

In every example, the paths are relative to the LDSMS installation folder (see section 2.2). Instead of writing the full path, we use '<LDSMS_HOME>' as a symbolic representation for the installation folder.
Assume, you installed LDSMS in '/Program Files/ldsms/'. The path

Listing 3.3: absolute path

```
/Program Files/ldsms/examples/hello_world/
```

will be written as

Listing 3.4: relative path

```
<LDSMS_HOME>/examples/hello_world/
```

3.1.3 Open a Console

3.1.3.1 Linux

There are many ways to open a console in Linux, depending on different criteria like the installed window manager or the installed software. The following descriptions are only an extract of these numerous ways:

- general solution: press 'Alt+F2'
- for Gnome users: click on 'Applications' at your menu bar and select 'Run Application...'
- for KDE users: click on the KDE logo at your menu bar and select 'Run command'

This opens a text prompt, that asks for the name of the programm to execute. Type either 'xterm' or 'konsole' and press 'Run'.

3.1.3.2 Windows

1. Click the start button (cf. figure 3.1).
2. Select "Run..." from the menu.
3. Type `cmd` in the window (cf. figure 3.2) and click OK. This opens a command prompt window (console)

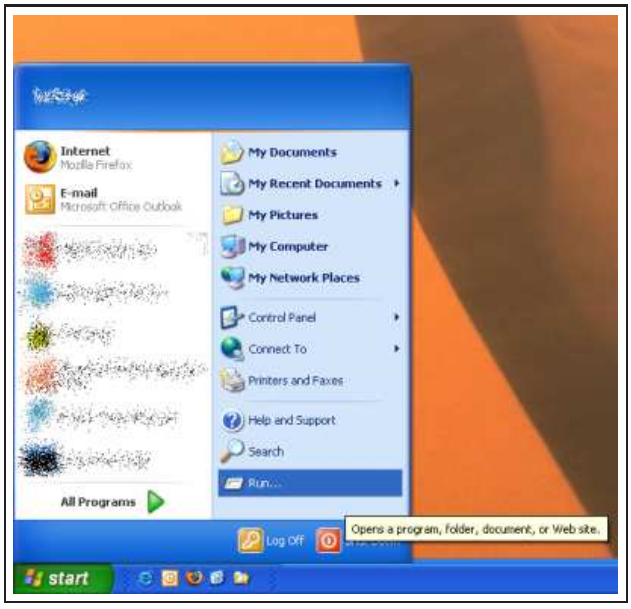


Figure 3.1: Start

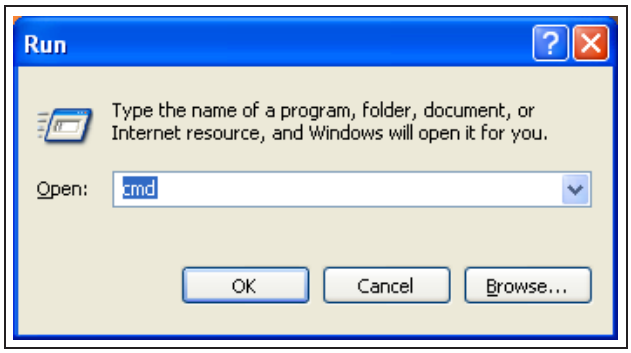
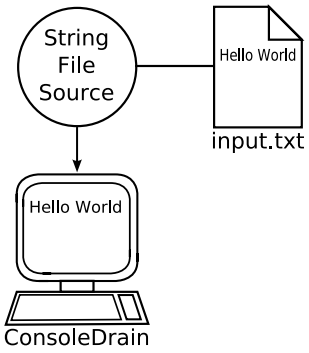


Figure 3.2: Run

3.2 Hello World



This example illustrates a very basic configuration, that prints “Hello World” onto the screen. Please follow the below-mentioned steps, to set up the ‘Hello World’ example.

- Create a folder <LDSMS_HOME>/examples/hello_world/ and open it.
- Create a file input.txt within this folder and open it with your text editor (e.g. vi for Linux, Notepad for Windows).
- Write “HELLO WORLD” into this file. Save and close it.
- Create a file config.xml within this folder and open it with your text editor.
- Insert the content from listing 3.5 (without the line numbers) into this file. Save and close it.
- Open a console and change to <LDSMS_HOME> (cf. section 3.1.3).
- Type “java -jar ldsms.jar examples/hello_world/config.xml” and press ‘ENTER’ (cf. listing 3.6).
- Press ‘Strg’+‘C’ to stop LDSMS, after ‘HELLO WORLD’ has been printed.

Listing 3.5: examples/hello_world/config.xml

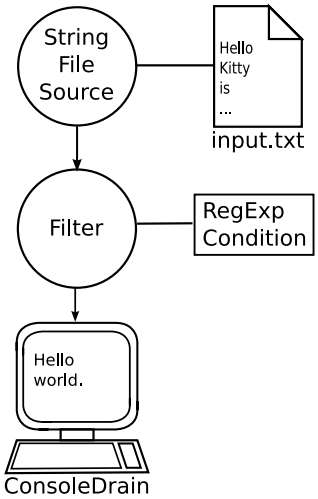
```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2
3  <server>
4    <logging level="INFO" />
5    <services>
6      <network>
7        <source class="generics.StringFileSource" file="examples/
             hello_world/input.txt">
8          <drain class="generics.ConsoleDrain" />
9        </source>
10     </network>
11   </services>
12 </server>
```

Listing 3.6: run the 'Hello World' example

```
java -jar ldsms.jar examples/hello_world/config.xml
1 [main] INFO de.lmu.ifi.pms.ldsms.network.Server  - Configuring Server...
572 [main] INFO de.lmu.ifi.pms.ldsms.network.Server  - Initializing Server
    ...
573 [main] INFO de.lmu.ifi.pms.ldsms.network.Server  - Starting Server...
HELLO WORLD
```

Explanation: In the first two steps, you created a text file as input for L-DSMS. In the third step, you created the configuration file for L-DSMS. In line 7, you specified a StringFileSource (see section 5.2.28). This FileSource will read every data from the text file, specified by the *file* attribute of the source element ('examples/hello_world/input.txt' in this case). In line 8, you specified a ConsoleDrain (see section 5.2.10). A ConsoleDrain will print every input to the console. Because the ConsoleDrain is specified as a child element of StringFileSource, every output of the StringFileSource (here, the data from the text file) will be passed to the input of the ConsoleDrain. With this simple example, you are able to print the content of the text file directly to the console.

3.3 Filtering Strings



This example illustrates a configuration, that uses a regular expression, to filter "Hello World" from a text file. Please follow the below-mentioned steps, to set up this example.

- Create a folder <LDSMS_HOME>/examples/regexp/ and open it.
- Create a file input.txt within this folder and open it with your text editor (e.g. vi for Linux, Notepad for Windows).
- Insert the content from listing 3.7 (with the line breaks) into this file. Save and close it.
- Create a file config.xml within this folder and open it with your text editor.
- Insert the content from listing 3.8 (without the line numbers) into this file. Save and close it.
- Open a console and change to <LDSMS_HOME> (cf. section 3.1.3).
- Type "java -jar ldsms.jar examples/regexp/config.xml" and press 'ENTER' (cf. listing 3.6).
- Press 'Strg'+'C' to stop LDSMS, after 'Hello world.' has been printed.

Listing 3.7: examples/regexp/input.txt

```
Hello
Kitty
is
loved
by
people
all
over
the
world.
```

Listing 3.8: examples/regexp/config.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2
3  <server>
4    <logging level="INFO" />
5    <services>
6      <network>
7        <source class="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
             file="examples/filter/input.txt">
8          <node class="de.lmu.ifi.pms.ldsms.generics.Filter">
9            <condition class="de.lmu.ifi.pms.ldsms.generics.
                 RegExpCondition" data="Hello|world.*" />
10           <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"/
                 >
11         </node>
12       </source>
13     </network>
14   </services>
15 </server>
```

Listing 3.9: run the 'RegExp' example

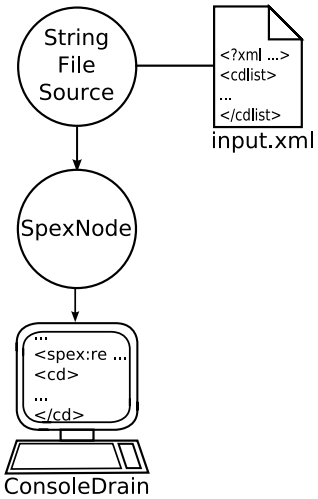
```
java -jar ldsms.jar examples/hello_world/config.xml0 [main] INFO de.lmu.
    ifi.pms.ldsms.network.Server - Configuring Server...
238 [main] INFO de.lmu.ifi.pms.ldsms.network.Server - Initializing Server
    ...
238 [main] INFO de.lmu.ifi.pms.ldsms.network.Server - Starting Server...
Hello
world.

1081 [Server stopp thread] INFO de.lmu.ifi.pms.ldsms.network.Server -
    Stopping Server...
```

Explanation: The first step was to create a text file as input for L-DSMS. The next step, was to create a configuration file. At line 7 in this configuration file is a StringFileSource (section 5.2.28) specified. This StringFileSource will read every data from 'examples/regexp/input.txt' (specified by the *file* attribute).

At line 8, a Filter is specified. This Filter has a RegExpCondition that checks, if the incomming data matches the specified regular expression (specified by the *data* attribute).

At line 9, a ConsoleDrain (see section 5.2.10) is specified. A ConsoleDrain will print every input to the console. Because the ConsoleDrain is specified as a child element of Filter, every output of it (the filtered data) will be passed to the input of the ConsoleDrain.

3.4 Filtering XML stream



This example illustrates a configuration, that filters XML data using SpexNode.

- Create a folder <LDSMS_HOME>/examples/xml/ and open it.
- Create a file input.xml within this folder and open it with your text editor (e.g. vi for Linux, Notepad for Windows).
- Insert the content from listing 3.10 (without the line numbers) into this file. Save and close it.
- Create a file config.xml within this folder and open it with your text editor.
- Insert the content from listing 3.11 (without the line numbers) into this file. Save and close it.
- Open a console and change to <LDSMS_HOME> (cf. section 3.1.3).
- Type "java -jar ldsms.jar examples/xml/config.xml" and press 'ENTER' (cf. listing 3.12).
- Press 'Strg'+'C' to stop LDSMS, after the result has been printed.

Listing 3.10: examples/xml/input.xml

```
1 <?xml version="1.0" encoding="ISO-8859-1" ?>
2
3 <cdlist>
4   <cd>
5     <name>Yello Submarine</name>
6     <artist>The Beatles</artist>
7     <tracklist>
8       <track id="1">Yellow Submarine</track>
9       <track id="2">Love You Too</track>
```

```
10        <track id="3">Eleanor Rigby</track>
11        <track id="4">Lucy In The Sky With Diamons</track>
12        <track id="5">Hey Buldog</track>
13        <track id="6">Think For Yourself</track>
14        <track id="7">All Together Now</track>
15        <track id="8">Sargent Pepper Lonely Hearts Club Band</track>
16        <track id="9">With A Little Help For My Friends</track>
17        <track id="10">Baby Your Are A Richman</track>
18        <track id="11">Only A Nothern Song</track>
19        <track id="12">All You Need Is Love</track>
20        <track id="13">When I'm Sixty Four</track>
21        <track id="14">Nowhere Man</track>
22        <track id="15">It's All Too Much</track>
23      </tracklist>
24    </cd>
25    <cd>
26      <name>Greatest Hits</name>
27      <artist>The Police</artist>
28      <tracklist>
29        <track id="1">Roxanne</track>
30        <track id="2">Can't Stand Losing You</track>
31        <track id="3">So Lonely</track>
32        <track id="4">Message In A Bottle</track>
33        <track id="5">Walking On The Moon</track>
34        <track id="6">The Bed Is Too Big Without You</track>
35        <track id="7">Don't Stand So Close To Me</track>
36        <track id="8">De Do Do Do De Da Da Da</track>
37        <track id="9">Every Little Thing She Does Is Magic</track>
38        <track id="10">Invisible Sun</track>
39        <track id="11">Spirits In The Material World</track>
40        <track id="12">Synchronicity II</track>
41        <track id="13">Every Breath You Take</track>
42        <track id="14">King Of Pain</track>
43        <track id="15">Wrapped Around Your Finger</track>
44        <track id="16">Tea In The Sahara</track>
45      </tracklist>
46    </cd>
47    ...
48  </cdlist>
```

Listing 3.11: examples/xml/config.xml

```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2
3  <server>
4    <logging level="INFO" />
5    <services>
6      <network>
```

```
7       <source class="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
            file="examples/xml/input.xml" repeat="true">
8         <node class="de.lmu.ifi.pms.ldsms.xml.SpexNode" xpath="/
              descendant::cd[child::artist/child::text()='The Police']"
              >
9           <drain class="generics.ConsoleDrain" />
10        </node>
11      </source>
12    </network>
13  </services>
14 </server>
```

Listing 3.12: run the 'SpexNode' example

```
java -jar ldsms.jar examples/hello_world/config.xml0 [main] INFO de.lmu.
    ifi.pms.ldsms.network.Server  - Configuring Server...
238 [main] INFO de.lmu.ifi.pms.ldsms.network.Server  - Initializing Server
    ...
238 [main] INFO de.lmu.ifi.pms.ldsms.network.Server  - Starting Server...
<spex:results xmlns:spex="http://www.pms.ifi.lmu.de/spex/">
  <spex:result>
    <cd>
      <name>Greatest Hits</name>
      <artist>The Police</artist>
      <tracklist>
        <track id="1">Roxanne</track>
        <track id="2">Can't Stand Losing You</track>
        <track id="3">So Lonely</track>
        <track id="4">Message In A Bottle</track>
        <track id="5">Walking On The Moon</track>
        <track id="6">The Bed Is Too Big Without You</track>
        <track id="7">Don't Stand So Close To Me</track>
        <track id="8">De Do Do Do De Da Da Da</track>
        <track id="9">Every Little Thing She Does Is Magic</track>
        <track id="10">Invisible Sun</track>
        <track id="11">Spirits In The Material World</track>
        <track id="12">Synchronicity II</track>
        <track id="13">Every Breath You Take</track>
        <track id="14">King Of Pain</track>
        <track id="15">Wrapped Around Your Finger</track>
        <track id="16">Tea In The Sahara</track>
      </tracklist>
    </cd>
  </spex:result>

22927 [Server stopp thread] INFO de.lmu.ifi.pms.ldsms.network.Server  -
    Stopping Server...
```

Explanation: The first step was to create a XML document as input for L-DSMS. The next step, was to create a configuration file for L-DSMS. In this configuration file at line 7 is a

StringFileSource (see section 5.2.28) specified. This StringFileSource will read every data from the XML document 'examples/xml/input.txt' (specified by the *file* attribute).

At line 8, a SpexNode is specified. This SpexNode filters the incomming XML data, using the XPath expression from the *xpath* attribute.

At line 9, a ConsoleDrain (see section 5.2.10) is specified. A ConsoleDrain will print every input to the console. Because the ConsoleDrain is specified as a child element of SpexNode, every output of it (the filtered XML data) will be passed to the input of the ConsoleDrain.

Chapter 4

Managing L-DSMS with VISU-L-DSMS

VISU-L-DSMS is the graphic user interface for L-DSMS that was developed to ease the management of L-DSMS. It can manage instances of L-DSMS located at the same host as VISU-L-DSMS as well as instances located on remote hosts. This allows to manage running instances of L-DSMS from a single management workstation, using the available network infrastructure. The following paragraphs explain how to use VISU-L-DSMS to manage running L-DSMS instances.

4.0.0.3 Install VISU-L-DSMS

VISU-L-DSMS is based on L-DSMS, so there has to be a fully functional L-DSMS installation on the system first (cf. chapter 2), the rest is as easy as installing L-DSMS. To install VISU-L-DSMS

- go to the L-DSMS project webpage (`http://www.pms.ifi.lmu.de/rewerse-wga1/ldsms`),
- download the latest visu-l-dsms.zip and
- extract this archive to your L-DSMS installation folder.

Advice:
To manage a L-DSMS instance with VISU-L-DSMS, it needs to be started with the RMI-Plugin (cf. section 5.4) activated.

4.0.0.4 Start VISU-L-DSMS

Linux as well as Windows users can start VISU-L-DSMS by simply excecuting a script that is shiped with VISU-L-DSMS. This script will excecute all necessary instructions and the user don't has to care about the details.

Linux user: go to <LDSMS_HOME> and excecute `visu-l-dsms.sh`

Windows user: go to <LDSMS_HOME> and excecute `visu-l-dsms.cmd`

Excecuting the script opens the VISU-L-DSMS main window (cf. figure 4.1). This main window

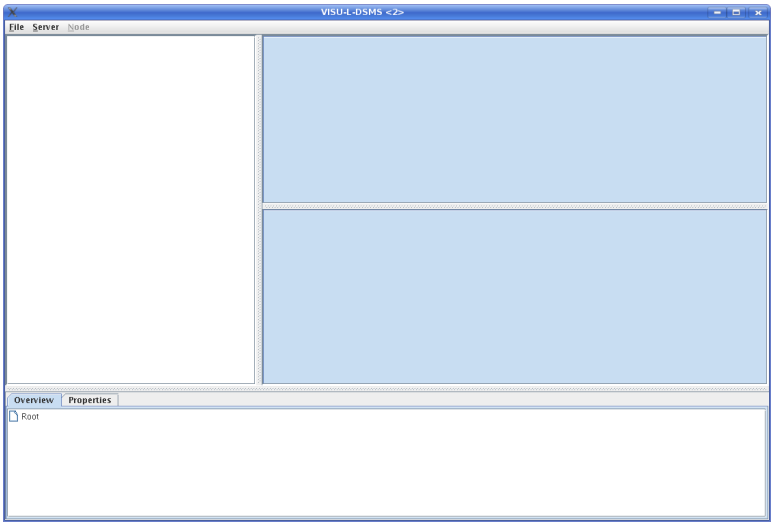


Figure 4.1: VISU-L-DSMS after start up

contains three areas and one menu panel. At the left side is the graphic representation of all components that are used and their relationship to each other, we will call it the **Network View**. Each component is represented as a node (colored symbol) and their relationships to each other by edges (black lines).

At the right side is the **Capturing View**. This area contains two subwindows. The upper one is used to show the incomming data of a component selected in the network area and the lower one is used to show the outgoing data of a component (the same or another one).

In the bottom area are two tabs that provide additional informations about the selected components of the network window (the **Overview Tab**) and their properties (the **Properties tab**).

These three areas are empty until VISU-L-DSMS is connected to a running L-DSMS instance. This will be explained in the following paragraph.

4.0.0.5 Connect VISU-L-DSMS to an instance of L-DSMS

To manage an instance of L-DSMS, a connection between VISU-L-DSMS and this instance has to be established first. This is done by selecting the `Connect` item from the `Server` menu (cf. figure 4.2). A connection dialog will be opened (cf. figure 4.3) where the necessary connection parameters for the L-DSMS instance have to be specified. The connection parameters are:

- hostname (optional, default = localhost)
- port (optional, default = 1099)
- name of the instance (not optional, no default value)

The L-DSMS instance name was configured as a parameter of the RMIPlugin. If you don't know this name, please refere to the configuration file of the instance of your interest or contact your local L-DSMS administrator.

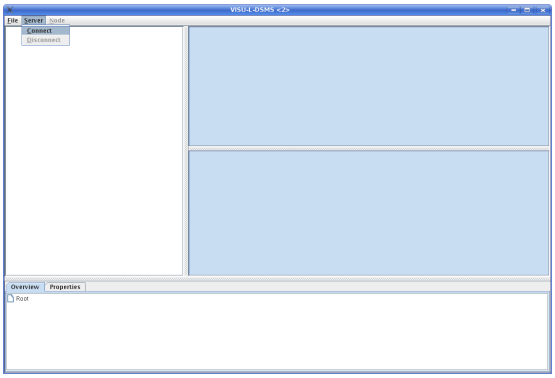


Figure 4.2: server -> connect

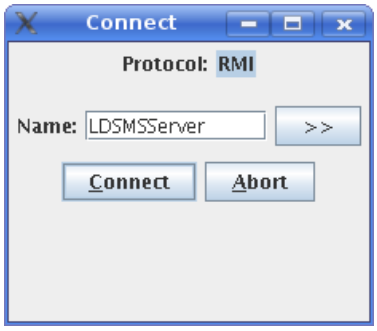


Figure 4.3: connection dialog

Figure 4.4: network area after connecting

4.0.0.6 The Overview tab

The overview tab (cf. figure 4.5) shows the names of all used components in a tree structure with sources being the parents and their drains being the children (recursive). Each component is listed only once, even if it has more than one source. In this case, it is only listed once as a child of the first of its sources. If a component is selected in the network area, its representation in the overview tab will be selected also. The same counts vice versa. So if a component is selected in the overview tab, it's representation in the network area will be selected also. This gives a compact overview about the network structure while the names of the components are listed in the overview tab. To select more than one component, hold down the `CTRL` button.

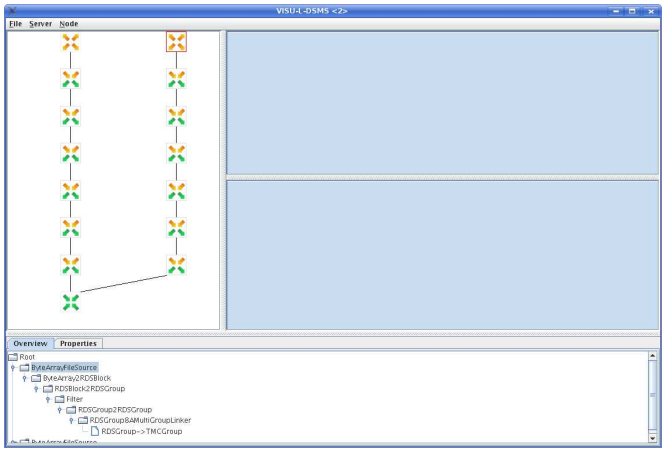


Figure 4.5: overview tab

4.0.0.7 The Properties tab

The properties tab (cf. figure 4.6) is used to display and edit the properties for the last component, that has been selected. If a component is selected in the network area, the properties tab will update itself automatically and display the properties for this component. Each property is presented as either

- a text field,
- a list or
- a check box.

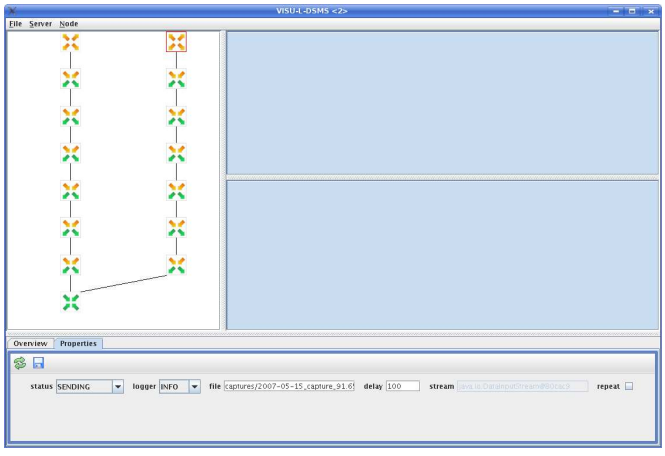


Figure 4.6: properties register

Not all properties are editable, therefore some values can be edited and others not. Which property is editable and wich not depends on the selected component but editing values doesn't

affect the L-DSMS instance, until the changes are saved (cf. figure 4.7). The reload button (cf. figure 4.8) can be used to revoke changes, before they have been saved.

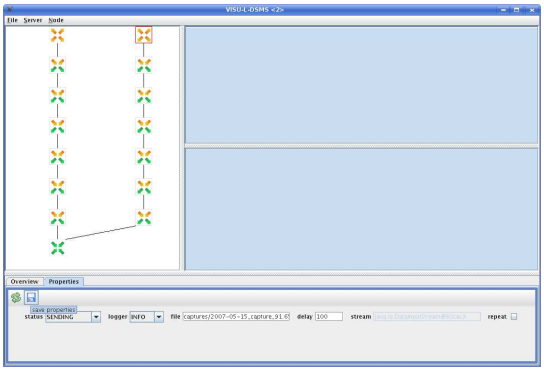


Figure 4.7: save property changes

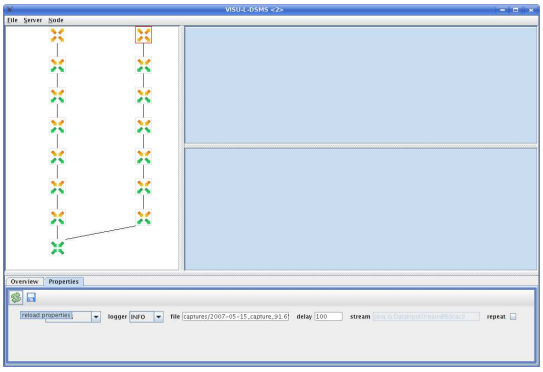


Figure 4.8: reload properties

4.0.0.8 The Capturing Area

The capturing area is used for monitoring the incomming and outgoing data of the components. To capture data, select one or more components, either in the network or the overview area, open the `node` menu and select `start listening` (cf. figure 4.9). This will open a tab for the incomming data in the upper subwindow and a tab for the outgoing data in the lower subwindow for each selected component.

Each tab has the same functionality to manage the capturing process. The buffer size field

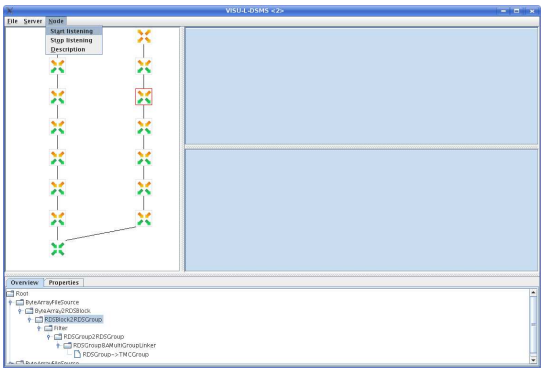


Figure 4.9: open capture window for selected component

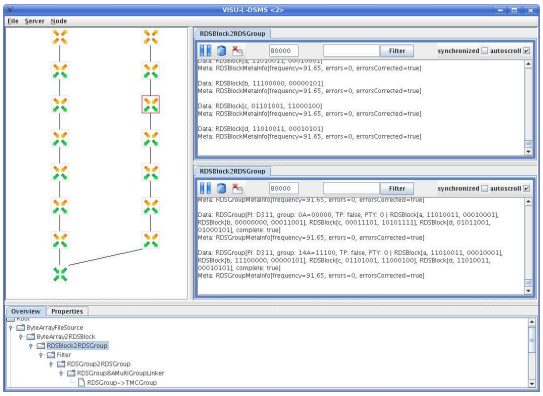


Figure 4.10: capture component input/output

(cf. figure 4.11) defines the number of characters that can be captured, until the oldest data has to be deleted to free space for new data. The clear button (cf. figure 4.12) can be used to delete all captured data from the textarea.

The filter field can be used, to filter the data, that fulfils the given regular expression. Only the data matching the regular expression will be displayed and the rest will be hidden. No data

will be lost but the user gets a better overview during filtering. After deactivating the filter, all captured data will be visible again.

The pause button (cf. figure 4.15) can be used to pause the capturing. No more data will be captured while the capture window is in pause mode, but the captured data stays visible. In pause mode, the pause button changes it's appearance (cf. figure 4.16) and can be used to restart the capturing.

If a capture window isn't needed anymore, the close button (cf. figure 4.17) can be used to close it.

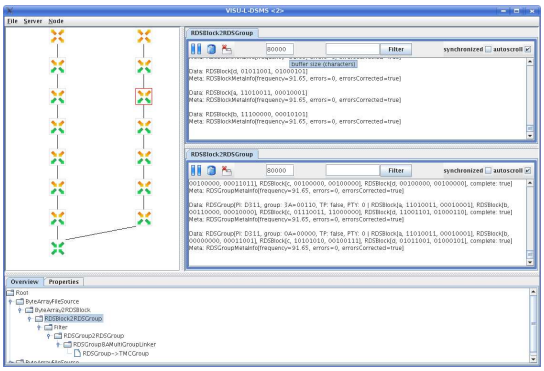


Figure 4.11: change capture window buffer size

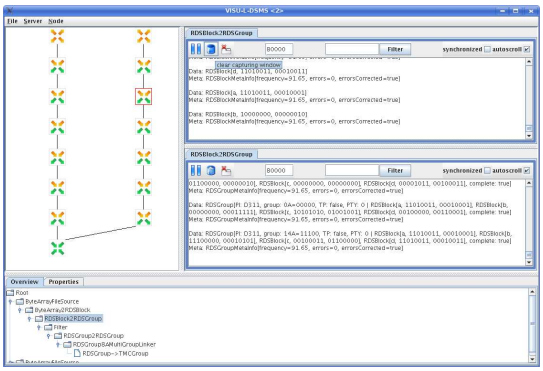


Figure 4.12: clear capture window

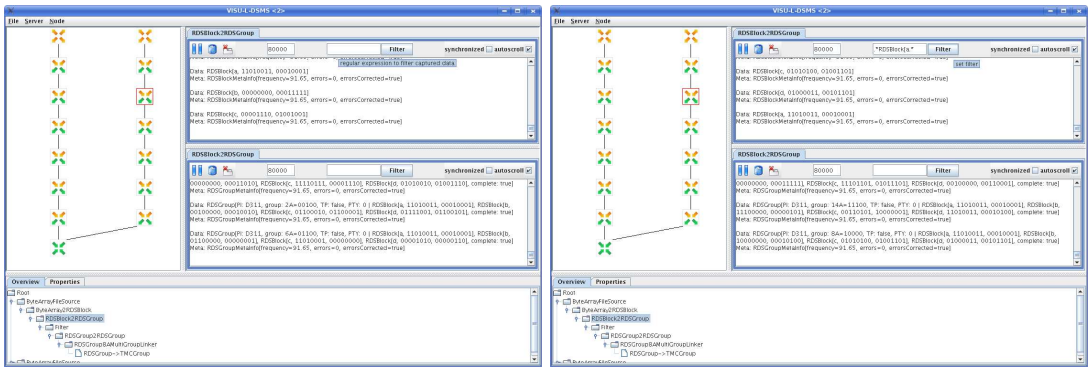


Figure 4.13: change filter for captured data

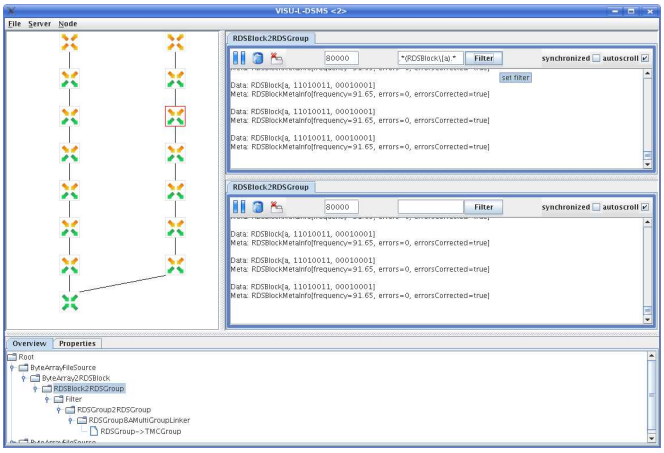


Figure 4.14: capture window with active filter

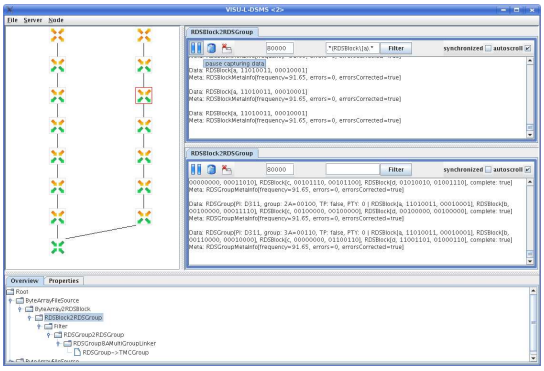


Figure 4.15: pause capturing

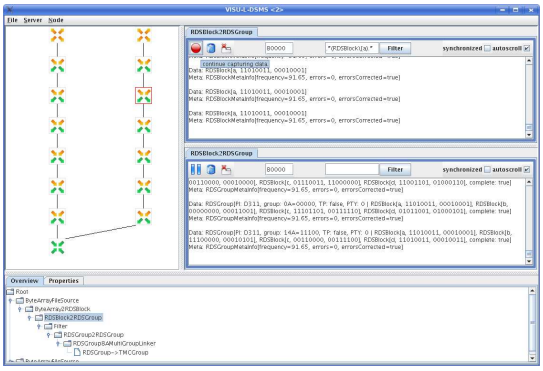


Figure 4.16: continue capturing

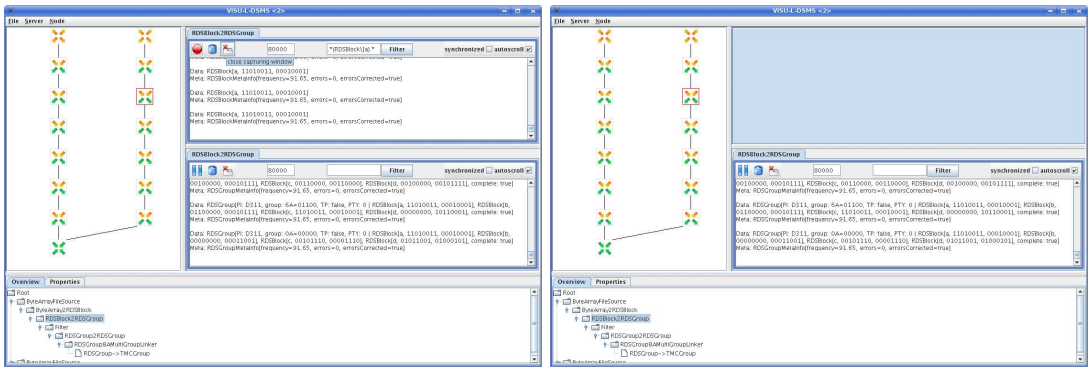


Figure 4.17: close capture window

Chapter 5

Components

5.1 Overview by packagename

This section gives you a brief overview for the components available in L-DSMS. This makes it possible to search for components by function. The detailed descriptions for each component are available in section 5.2. The excact subsection and page are listed along with each component.

Advice:
The current package name for L-DSMS core is 'de.lmu.ifi.pms.ldsms'. This may change in the future, therefore the package names are listed as <ldsms core package>.<*subpackage*>.

5.1.1 <ldsms core package>.generics

This package provides components, that were not designed for special fields, but to provide a core functionality for L-DSMS.

Name	Type	Section	Page
Buffer	Node	5.2.1	38
ByteArray2String	Node	5.2.4	43
ByteArrayFileDrain	Drain	5.2.5	44
ByteArrayFileSource	Source	5.2.6	45
ByteArraySocketDrain	Drain	5.2.7	46
ByteArraySocketSource	Source	5.2.8	48
Cast	Node	5.2.9	49
ConsoleDrain	Node	5.2.10	51
Filter	Node	5.2.14	57
ObjectSocketDrain	Drain	5.2.15	58
ObjectSocketSource	Source	5.2.16	60
String2ByteArray	Node	5.2.26	74
StringFileDrain	Drain	5.2.27	75
StringFileSource	Source	5.2.28	76
StringSocketDrain	Drain	5.2.30	79
StringSocketSource	Source	5.2.31	80

5.1.2 <ldsms core package>.rds

This package provides components, that were designed for processing RDS data.

Name	Type	Section	Page
ByteArray2RDSBlock	Node	5.2.2	39
ByteArray2RDSGroup	Node	5.2.3	41
RDSBlock2ByteArray	Node	5.2.18	62
RDSBlock2RDSGroup	Node	5.2.19	63
RDSGroup2ByteArray	Node	5.2.20	64
RDSGroup2RDSGroup	Node	5.2.21	66
RDSGroup2RDSGroup3A	Node	5.2.22	68
RDSGroup8AMultiGroupLinker	Node	5.2.24	71

5.1.3 <ldsms core package>.rds.easyway

This package provides components, that were designed for processing data produced by an EasyWay RDS reciever.

Name	Type	Section	Page
EasyWayFileSource	Source	5.2.11	52
EasyWayRDSChunk2RDSBlock	Node	5.2.12	54
EasyWaySource	Source	5.2.13	55

5.1.4 <ldsms core package>.string

This package provides components, that were designed for processing text data.

Name	Type	Section	Page
RegexCondition	Condition	5.3.8	90
RepeatedStringCondition	Condition	5.3.9	90
StringReplaceNode	Node	5.2.29	78
StringTokenizerNode	Node	5.2.32	81

5.1.5 <ldsms core package>.tmc

This package provides components, that were designed for processing TMC data.

Name	Type	Section	Page
OTNDrain	Drain	5.2.17	61
RDSGroup2TMCMessage	Node	5.2.23	69
TMCMessageManagement	Node	5.2.34	85

5.1.6 <ldsms core package>.xml

This package provides components, that were designed for processing XML data.

Name	Type	Section	Page
SpexNode	Node	5.2.25	73
TMCMessage2XML	Node	5.2.33	83
XML2TMCMessage	Node	5.2.35	86

5.2 Core Components

This section gives you an alphabetic overview with a detailed description for each component, deliverd in the L-DSMS core package. For Sources, the data/metadata input types will be listed. For Drains, the data/metadata output types will be listed. For Nodes, the data/metadata input and output types will be listed. If it is possible to parametrize the input or output types no types will be listed. By parametrizing a class, you can specify the input or output types in the configuration file as needed.

5.2.1 Buffer

Package: de.lmu.ifi.pms.ldsms.generics
Type: Node
Input data type: parametrizable by attribute 'data'
Input meta type: parametrizable by attribute 'meta'
Output data type: same as input data
Output meta type: same as input meta
Dependencies: no additional external library needed
Abstract: A Buffer caches informations until they can be delivered to underlying consumers (drains). It can be used, to cache data until one ore more drains are able to consume it. The recieved content is send according to the first in first out principle (FIFO). This means, if the data D_1 was recieved bevore data D_2, data D_1 will be send before data D_2.

Attributes:

Attribute-name	Type	Default	Optional	Description
data	String		No	Specifies the data input/output type.
meta	String		No	Specifies the meta input/output type.
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma separated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.1: Example

```
<?xml version="2.0" encoding="ISO-8859-1" ?>

<server>
```

```
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.generics.Buffer" level="DEBUG
        " />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
          file="examples/hello_world/input.txt" repeat="true" delay=
        "1000">
        <node class="de.lmu.ifi.pms.ldsms.generics.Buffer" data="
            String" meta="Object">
          <drain class="de.lmu.ifi.pms.ldsms.generics.
              StringSocketDrain" port="6543" />
        </node>
      </source>
    </network>
  </services>
</server>
```

This example creates an instance of Buffer, that caches every incomming data of type String and incomming metadata of type Object. New data will be appended to the end of the chache. If at least one consumer (drain) is connected to the StringSocketDrain (at port 6543), the cache will be emptied beginning with its head. To test the Buffer (and recieve the cached data), open a telnet connection to the StringSocketDrain (e.g. 'telnet localhost 6543').

5.2.2 ByteArray2RDSBlock

Package:	de.lmu.ifi.pms.ldsms.rds
Type:	Node
Input data type:	byte array (byte[])
Input meta type:	byte array (byte[])
Output data type:	de.lmu.ifi.pms.ldsms.rds.RDSBlock
Output meta type:	de.lmu.ifi.pms.ldsms.rds.RDSBlockMetaInfo
Dependencies:	no additional external library needed

Abstract:

Reads a RDSBlock and its Metainformation from two byte arrays (byte[]). The first array (with a length of three) represents the data and is interpreted as follows:

- data byte 0: Block id
- data byte 1: Block high byte
- data byte 2: Block low byte

The second array (with a length of six) represents the metadata and is interpreted as follows:

- meta byte 0-3: Frequency
- meta byte 4: number of Errors
- meta byte 5: bit 0 wether all errors are corrected

Attributes:

Attribute-name	Type	Default	Optional	Description
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.2: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.rds.ByteArray2RDSBlock" level
       ="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.
         ByteArrayFileSource" file="examples/rds/RDSBlock.bin">
        <node class="de.lmu.ifi.pms.ldsms.rds.ByteArray2RDSBlock">
          <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"/
             >
        </node>
```

```
      </source>
    </network>
  </services>
</server>
```

This example reads the binary data from 'examples/rds/RDSBlock.bin' and forwards it to an instance of ByteArray2RDSBlock, where it is converted from ByteArrays to RDSBlocks. The instance of ConsoleDrain is used to print the result to the console.

5.2.3 ByteArray2RDSGroup

Package:	de.lmu.ifi.pms.ldsms.rds
Type:	Node
Input data type:	byte array (byte[])
Input meta type:	byte array (byte[])
Output data type:	de.lmu.ifi.pms.ldsms.rds.RDSGroup
Output meta type:	de.lmu.ifi.pms.ldsms.rds.RDSGroupMetaInfo
Dependencies:	no additional external library needed
Abstract:	Reads a RDSGroup and its Metainformation from two byte arrays (byte[]). The first array (with a length of 13) represents the data and is interpreted as follows:

- data byte 0: Block A id
- data byte 1: Block A high byte
- data byte 2: Block A low byte
- data byte 3: Block B id
- data byte 4: Block B high byte
- data byte 5: Block B low byte
- data byte 6: Block C id
- data byte 7: Block C high byte
- data byte 8: Block C low byte
- data byte 9: Block D id
- data byte 10: Block D high byte
- data byte 11: Block D low byte

The second array (with a length of six) represents the metadata and is interpreted as follows:

- meta byte 0-3: Frequency
- meta byte 4: number of Errors
- meta byte 5: bit 0 wether all errors are corrected

Attributes:

Attribute-name	Type	Default	Optional	Description
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.3: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.rds.ByteArray2RDSGroup" level
        ="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.
          ByteArrayFileSource" file="examples/rds/RDSGroup.bin">
        <node class="de.lmu.ifi.pms.ldsms.rds.ByteArray2RDSGroup">
          <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"/
              >
        </node>
      </source>
    </network>
  </services>
</server>
```

This example reads the binary data from 'examples/rds/RDSGroup.bin' and forwards it to an instance of ByteArray2RDSGroup, where it is converted from ByteArrays to RDSGroups. The instance of ConsoleDrain is used to print the result to the console.

5.2.4 ByteArray2String

Package: de.lmu.ifi.pms.ldsms.generics
Type: Node
Input data type: byte array (byte[])
Input meta type: byte array (byte[])
Output data type: java.lang.String
Output meta type: java.lang.String
Dependencies: no additional external library needed
Abstract: A ByteArray2String node transforms incomming byte arrays into strings, using the specified encoding format. It can be used, to connect a source with a drain, where the source produces byte arrays but the drain is expecting strings.

Attributes:

Attribute-name	Type	Default	Optional	Description
encoding	String	ISO-8859-1	Yes	Specifies the encoding format used to transform a byte array into a string.
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma separated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.4: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.generics.ByteArray2String"
        level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.
          ByteArrayFileSource" file="examples/hello_world/input.bin">
        <node class="de.lmu.ifi.pms.ldsms.generics.ByteArray2String"
            encoding="ISO-8859-1">
          <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"/
              >
```

```
        </node>
      </source>
    </network>
  </services>
</server>
```

This example reads the binary data from 'examples/hello_world/input.bin' and forwards it to an instance of ByteArray2String, where it is converted from ByteArrays to Strings, using the encoding 'ISO-8859-1'. The instance of ConsoleDrain is used to print the result to the console.

5.2.5 ByteArrayFileDrain

Package: de.lmu.ifi.pms.ldsms.generics
Type: Drain
Input data type: byte array (byte[])
Input meta type: byte array (byte[])
Dependencies: no additional external library needed
Abstract: A ByteArrayFileDrain drain writes the incomming byte arrays into the specified file. The byte arrays are written the stream as follows:

- the first four bytes describe the length of the data array as an integer, let it be n.
- the next n bytes make up the data array.
- the next four bytes describe the length of the metainfo array as an integer, let it be m.
- the next m bytes make up the metainfo array.

Attributes:

Attribute-name	Type	Default	Optional	Description
file	String		No	Specifies the path of the destination file.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.5: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
```

```
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.generics. ByteArrayFileDrain"
        level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
          file="examples/hello_world/input.txt">
        <node class="de.lmu.ifi.pms.ldsms.generics.String2ByteArray"
            >
          <drain class="de.lmu.ifi.pms.ldsms.generics.
              ByteArrayFileDrain" file="examples/hello_world/output.
              bin" />
        </node>
      </source>
    </network>
  </services>
</server>
```

This example reads the data from 'examples/hello_world/input.txt' and forwards it to an instance of String2ByteArray, where it is converted from Strings to byte arrays. These byte arrays are forwarded to an instance of ByteArrayFileDrain, that writes the incomming data to 'examples/hello_world/output.bin'. For examining the result, open the output file.

5.2.6 ByteArrayFileSource

Package: de.lmu.ifi.pms.ldsms.generics
Type: Source
Output data type: byte array (byte[])
Output meta type: byte array (byte[])
Dependencies: no external library needed
Abstract: ByteArrayFileSource produces binary data by reading it from a filestream. The data is received in the form of byte arrays. The byte arrays are read from the stream as follows:

- the first four bytes describe the length of the data array as an integer, let it be n.
- the next n bytes make up the data array.
- the next four bytes describe the length of the metainfo array as an integer, let it be m.
- the next m bytes make up the metainfo array.

Attributes:

Attribute-name	Type	Default	Optional	Description
delay	Integer	100	Yes	Specifies a delay (in ms). After reading a part of the file, the thread will sleep for the specified time, before the next part will be read.
file	String		No	Specifies the path of the source file.
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
repeat	Boolean	false	Yes	Specifies, whether the reading should restart after the end of file was reached.

Example:

Listing 5.6: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.generics.ByteArrayFileSource"
        level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.
          ByteArrayFileSource" file="examples/hello_world/input.bin"
          delay="1000" repeat="true">
        <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain" />
      </source>
    </network>
  </services>
</server>
```

This example reads the binary data from 'examples/hello_world/input.bin' with a delay of 1 second. If, during reading, the end of file is reached, ByteArrayFileSource will repeat the reading at the begin of the file. The instance of ConsoleDrain is used to print the data to the console.

5.2.7 ByteArraySocketDrain

Package: de.lmu.ifi.pms.ldsms.generics
Type: Drain

Input data type: byte array (byte[])
Input meta type: byte array (byte[])
Dependencies: no additional external library needed
Abstract: A ByteArraySocketDrain drain writes the incomming byte arrays to every client, connected at the specified port. The byte arrays are written to the stream as follows:

- the first four bytes describe the length of the data array as an integer, let it be n.
- the next n bytes make up the data array.
- the next four bytes describe the length of the metainfo array as an integer, let it be m.
- the next m bytes make up the metainfo array.

Attributes:

Attribute-name	Type	Default	Optional	Description
period	Integer	1	Yes	Specifies the interval in seconds, the drain will repeat looking for locked up client connections.
port	Integer		No	Specifies the port at which the SocketDrain will accept client connections.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.7: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.generics.ByteArrayFileSource"
        level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.
          ByteArrayFileSource" file="examples/hello_world/input.bin">
```

```
        <drain class="de.lmu.ifi.pms.ldsms.generics.
            ByteArraySocketDrain" port="6543" />
      </source>
    </network>
  </services>
</server>
```

This example reads the binary data from 'examples/hello_world/input.bin' and forwards it to ByteArraySocketDrain. ByteArraySocketDrain sends the input to every client connected at port 6543. The example for ByteArraySocketSource (cf. subsection 5.2.8) can be used, to test this example.

5.2.8 ByteArraySocketSource

Package: de.lmu.ifi.pms.ldsms.generics
Type: Source
Output data type: byte array (byte[])
Output meta type: byte array (byte[])
Dependencies: no additional external library needed
Abstract: ByteArraySocketSource produces binary data by reading it from a socket connection. The data is received in the form of byte arrays. The byte arrays are read from the stream as follows:

- the first four bytes describe the length of the data array as an integer, let it be n.
- the next n bytes make up the data array.
- the next four bytes describe the length of the metainfo array as an integer, let it be m.
- the next m bytes make up the metainfo array.

Attributes:

Attribute-name	Type	Default	Optional	Description
host	String		No	Specifies the hostname for the socket connection.
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
port	Integer		No	Specifies the portnumber for the socket connection.
retry	Integer	1000	Yes	Specifies the interval for reconnect attempts, if the connection is broken.

Example:

Listing 5.8: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.generics.
        ByteArraySocketSource" level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.
          ByteArraySocketSource" host="localhost" port="6543">
        <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"/>
      </source>
    </network>
  </services>
</server>
```

This example recieves the binary data from localhost (port 6543). The instance of ConsoleDrain is used to print the data to the console. You can use this configuration, to test the example for ByteArraySocketDrain (cf. subsection 5.2.7).

5.2.9 Cast

Package:	de.lmu.ifi.pms.ldsms.generics
Type:	Node
Input data type:	java.lang.Object
Input meta type:	java.lang.Object
Output data type:	parametrizable by attribute 'data'
Output meta type:	parametrizable by attribute 'meta'
Dependencies:	no additional external library needed
Abstract:	A Cast node casts the input to the specified type and filters out any incompatible data or meta objects. It can be used to connect a source with a drain, where the output types of the source are different but compatible to the input types of the drain. This works if for data and metadata one of the following cases (doesn't need to be the same) matches: 1. The input type can be casted to the output type. Therfore, it is a subtype of the output type. 2. The output type is either String, Boolean, Integer, Double or Float and the input can be transformed to this type.

Attributes:

Attribute-name	Type	Default	Optional	Description
data	String		No	Speciefies the data output type.
meta	String		No	Speciefies the metadata output type.
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.9: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.generics.Cast" level="DEBUG"
        />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
          file="examples/numbers/input.txt">
        <node class="de.lmu.ifi.pms.ldsms.generics.Cast" data="
            Integer" meta="Object">
          <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"/
              >
        </node>
      </source>
    </network>
  </services>
</server>
```

This example creates an instance of Cast, that forwards every incomming data and metadata after converting the data to Integer.

5.2.10 ConsoleDrain

Package: de.lmu.ifi.pms.ldsms.generics
Type: Drain
Input data type: java.lang.Object
Input meta type: java.lang.Object
Dependencies: no additional external library needed

Abstract: A ConsoleDrain prints every incomming data without any formating to the console.

Attributes:

Attribute-name	Type	Default	Optional	Description
printData	Boolean	true	Yes	Specifies, if the data should be printed to the console.
printMeta	Boolean	false	Yes	Specifies, if the metadata should be printed to the console.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.10: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain" level=
       "DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
           file="examples/hello_world/input.txt">
        <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain" />
      </source>
          </network>
    </services>
</server>
```

This example creates an instance of StringFileSource, that reads from examples/hello_world/input.txt. The strings read from the file, are printed to the console by the ConsoleDrain.

5.2.11 EasyWayFileSource

Package: de.lmu.ifi.pms.ldsms.rds.easyway
Type: Source
Output data type: de.lmu.ifi.pms.ldsms.rds.easyway.EasyWayRDSChunk
Output meta type: de.lmu.ifi.pms.ldsms.rds.easyway.
EasyWayRDSChunkMetaInfo

Dependencies: no additional external library needed
Abstract: Emulates an EasyWaySource by reading the RDS-TMC raw data from a source file.

Attributes:

Attribute-name	Type	Default	Optional	Description
file	String		No	Specifies the path of the source file.
frequency				
interval	Integer		No	Specifies a delay (in ms). After reading a part of the file, the thread will sleep for the specified time, before the next part will be read.
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
repeat	Boolean	false	Yes	Specifies, whether the reading should restart after the end of file was reached.
version	String		No	Specifies the version of the attached simulated Easyway light RDS reciever.

Example:

Listing 5.11: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.rds.easyway.EasyWayFileSource
        " level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.rds.easyway.
          EasyWayFileSource" file="examples/easyway/input.bin"
          interval="100" frequency="95.1" version="1.0">
        <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain" />
      </source>
          </network>
    </services>
</server>
```

This example simulates an EasyWay light RDS reciever of version '1.0', that is tuned to the frequency 95.1Mhz. The data will be read from 'examples/easyway/input.bin' every 100ms. The instance of ConsoleDrain is used to print the data to the console.

5.2.12 EasyWayRDSChunk2RDSBlock

Package: de.lmu.ifi.pms.ldsms.rds.easyway
Type: Source
Input data type: de.lmu.ifi.pms.ldsms.rds.easyway.EasyWayRDSChunk
Input meta type: de.lmu.ifi.pms.ldsms.rds.easyway.EasyWayRDSChunkMetaInfo
Output data type: de.lmu.ifi.pms.ldsms.rds.RDSBlock
Output meta type: de.lmu.ifi.pms.ldsms.rds.RDSBlockMetaInfo
Dependencies: no additional external library needed
Abstract: Converts incomming EasyWayRDSChunk into RDSBlock. This is done by creating a new RDSBlock using the EasyWayRDSChunk block id and the EasyWayRDSChunk current high and low byte. The EasyWayRDSChunk block id is transformed into a RDSBlock id as follows:

- 0x00 − > (byte)'a'
- 0x20 − > (byte)'b'
- 0x40 − > (byte)'c'
- 0x60 − > (byte)'d'
- 0x80 − > (byte)'C'
- 0xA0 − > (byte)'e'
- 0xC0 − > (byte)'E'
- 0xE0 − > (byte)'I'

The EasyWayRDSChunk current high and low byte are copied without any transformations. The metainformation is forwarded unchanged.

Attributes:

Attribute-name	Type	Default	Optional	Description
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.12: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.rds.easyway.EasyWayFileSource
        " level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.rds.easyway.
          EasyWayFileSource" file="examples/easyway/input.bin"
          interval="100" frequency="95.1" version="1.0">
        <node class="de.lmu.ifi.pms.ldsms.rds.easyway.
            EasyWayRDSChunk2RDSBlock">
          <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"
              />
        </node>
      </source>
          </network>
    </services>
</server>
```

This example simulates an EasyWay light RDS reciever of version '1.0', that is tuned to the frequency 95.1Mhz. The data will be read from 'examples/easyway/input.bin' every 100ms and transformed from EasyWayRDSChunk to RDSBlock. The instance of ConsoleDrain is used to print the data to the console.

5.2.13 EasyWaySource

Package:	de.lmu.ifi.pms.ldsms.rds.easyway
Type:	Source
Output data type:	de.lmu.ifi.pms.ldsms.rds.easyway.EasyWayRDSChunk
Output meta type:	de.lmu.ifi.pms.ldsms.rds.easyway. EasyWayRDSChunkMetaInfo
Dependencies:	RXTX (Version $\geq$ 2.1, `http://www.rxtx.org`)
Abstract:	Reads RDS data from an EasyWay Light RDS-TMC reciever and produces EasyWayRDSChunk. An EasyWayRDSChunk encapsulates a chunk (block) of raw RDS data, built of exactly seven bytes.

Byte 0: [BL2, BL1, BL0, SNYC, DOFL, RSTD, ELB1, ELB0]

Byte 1: [M15, M14, M13, M12, M11, M10, M09, M08] Current HIGH byte

Byte 2: [M07, M06, M05, M04, M03, M02, M01, M00] Current LOW byte

Byte 3: [P15, P14, P13, P12, P11, P10, P09, P08] Previous HIGH byte

Byte 4: [P07, P06, P05, P04, P03, P02, P01, P00] Previous LOW byte

Byte 5: [BEC5, BEC4, BEC3, BEC2, BEC1, BEC0, EPB1, EPB0]

Byte 6: [BP2, BP1, BP0, undefined, SQI3, SQI2, SQI1, SQI0]

Attributes:

Attribute-name	**Type**	**Default**	**Optional**	**Description**
device	String		No	Specifies the device name of the RDS-TMC reciever within the system (e.g. /dev/ttyS0).
dx	Boolean	false	Yes	Not used.
frequency	String		No	Specifies the frequency (in MHz), the RDS-TMC reciever will be atuned (e.g. 98.1).
mono	Boolean	false	Yes	Indicates, wheter the signal should be recieved in mono.
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
searchLevel	Integer	15	Yes	Specifies the sensibility for the signal strength of automatic searched frequencies.
search	String	down	Yes	Indicates, if the next available frequency should be automatically searched. Valid values are 'up' and 'down').

Example:

Listing 5.13: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.rds.easyway.EasyWaySource"
        level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.rds.easyway.EasyWaySource"
          device="/dev/ttyS0" frequency="95.1">
        <node class="de.lmu.ifi.pms.ldsms.rds.easyway.
            EasyWayRDSChunk2RDSBlock">
          <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"
              />
        </node>
      </source>
          </network>
    </services>
</server>
```

This example creates an instance of EasyWaySource, that reads the data from an EasyWay Light RDS-TMC reciever, which is registered under '/dev/ttyS0' and tuned to 95.1MHz. The instance of ConsoleDrain is used to print the data to the console.

5.2.14 Filter

Package:	de.lmu.ifi.pms.ldsms.generics
Type:	Node
Input data type:	depends on selected conditions
Input meta type:	depends on selected conditions
Output data type:	same as input data
Output meta type:	same as input meta
Dependencies:	no additional external library needed
Abstract:	A Filter tests whether the data and metainfo fullfills a certain condition and passes it to its drains only if the condition is fullfilled. Otherwise, the data and metainfo is omitted. Section 5.3 describes the filter, filter conditions and how to use them in detail.

Attributes:

Attribute-name	Type	Default	Optional	Description
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.14: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.generics.Filter" level="DEBUG
        " />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
          file="examples/filter/input.txt">
        <node class="de.lmu.ifi.pms.ldsms.generics.Filter">
          <condition class="de.lmu.ifi.pms.ldsms.string.
              RegexCondition" data="Hello|world.*" />
          <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"/
              >
        </node>
      </source>
    </network>
  </services>
</server>
```

This example reads the data from 'examples/filter/input.txt' and forwards it to an instance of Filter. This Filter contains only one condition, the RegexCondition (cf. subsection 5.3.8) that drops each line, that doesn't contain either 'Hello' or 'world'. The instance of ConsoleDrain is used to print the result to the console.

5.2.15 ObjectSocketDrain

Package: de.lmu.ifi.pms.ldsms.generics
Type: Drain
Input data type: java.lang.Object
Input meta type: java.lang.Object
Dependencies: no additional external library needed

Abstract: A ObjectSocketDrain writes the incomming data and metadata into the output stream of a socket (in this order). The incomming objects must implement the java.io.Serializable interface. If no client is connected, the data will be droped.

Attributes:

Attribute-name	Type	Default	Optional	Description
period	Integer	1	Yes	Specifies the interval in seconds, the drain will repeat looking for locked up client connections.
port	Integer		No	Specifies the port at which the SocketDrain will accept client connections.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.15: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.generics. ObjectSocketDrain"
        level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics. StringFileSource
          " file="test/Hello_World.txt" interval="0" repeat="true"
          forwardEOL="true">
        <drain class="de.lmu.ifi.pms.ldsms.generics.
            ObjectSocketDrain" port="6543" />
      </source>
    </network>
  </services>
</server>
```

This example creates an instance of ObjectSocketDrain, that recieves its incomming data and metadata from a StringFileSource and forwards everything to each client connected at port 6543.

5.2.16 ObjectSocketSource

Package: de.lmu.ifi.pms.ldsms.generics
Type: Source
Output data type: parametrizable by attribute 'data'
Output meta type: parametrizable by attribute 'meta'
Dependencies: no additional external library needed
Abstract: ObjectSocketSource produces data by reading objects from a socket connection. The objects are read from the stream as follows:

- at first, the data object will be read and casted to the specified data output type
- at second the metadata object will be read and casted to the specified metadata output type

The ObjectSocketSource tries to cast the incomming data and metadata to their specified types. If either the incomming data or metadata could not be casted, both will be droped.

Attributes:

Attribute-name	Type	Default	Optional	Description
data	String		No	Specifies the data output type.
meta	String		No	Specifies the metadata output type.
host	String		No	Specifies the hostname for the socket connection.
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
port	Integer		No	Specifies the portnumber for the socket connection.
retry	Integer	1000	Yes	Specifies the interval for reconnect attempts, if the connection is broken.

Example:

Listing 5.16: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
```

```
    <logger name="de.lmu.ifi.pms.ldsms.generics. ObjectSocketSource"
        level="DEBUG" />
  </logging>
  <services>
  <network>
    <source class="de.lmu.ifi.pms.ldsms.generics. ObjectSocketSource
        " host="localhost" port="6543" data="String" meta="String">
          <drain class="de.lmu.ifi.pms.ldsms.generics. ConsoleDrain"
              />
    </source>
    </network>
  </services>
</server>
```

This example creates an instance of ObjectSocketSource, that builds up a connection to localhost at port 6543. Every data object recieved, will be casted to a String.

5.2.17 OTNDrain

Package: de.lmu.ifi.pms.ldsms.tmc
Type: Drain
Input data type: java.util.List<TMCMessage>
Input meta type: java.util.List
Dependencies: no additional external library needed
Abstract: Creates an OWL representation of the given data and writes it to 'Ontologie/transformation/TMCEvents.owl' in the specified base dir. The metainformation is not used.

Attributes:

Attribute-name	Type	Default	Optional	Description
dir	String		No	The base directory. This directory is used to find the ontologie file 'Ontologie/transformation/TMCEvents.owl'
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.17: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
```

```
    <logger name="de.lmu.ifi.pms.ldsms.rds.OTNDrain" level="DEBUG" />
  </logging>
  <services>
    <network>
    </network>
  </services>
</server>
```

5.2.18 RDSBlock2ByteArray

Package: de.lmu.ifi.pms.ldsms.rds
Type: Node
Input data type: de.lmu.ifi.pms.ldsms.rds.RDSBlock
Input meta type: de.lmu.ifi.pms.ldsms.rds.RDSBlockMetaInfo
Output data type: byte array (byte[])
Output meta type: byte array (byte[])
Dependencies: no additional external library needed
Abstract: Transforms a RDSBlock and its Metainformation into two byte arrays (byte[]). The first array (with a length of three) represents the data and its structure is as follows:

- data byte 0: Block id
- data byte 1: Block high byte
- data byte 2: Block low byte

The second array (with a length of six) represents the metadata and its structure is as follows:

- meta byte 0-3: Frequency
- meta byte 4: number of Errors
- meta byte 5: bit 0 wether all errors are corrected

Attributes:

Attribute-name	Type	Default	Optional	Description
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.18: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.rds.RDSBlock2ByteArray" level
        ="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.rds.easyway.EasyWaySource"
          device="/dev/ttyS0" frequency="95.1">
        <node class="de.lmu.ifi.pms.ldsms.rds.easyway.
            EasyWayRDSChunk2RDSBlock">
          <node class="de.lmu.ifi.pms.ldsms.rds.RDSBlock2ByteArray">
            <drain class="de.lmu.ifi.pms.ldsms.generics.
                ByteArrayFileDrain" file="examples/rds/RDSBlock.bin"/
                >
          </node>
        </node>
      </source>
    </network>
  </services>
</server>
```

This example creates an instance of EasyWaySource, that reads the data from an EasyWay Light RDS-TMC reciever, which is registered under '/dev/ttyS0' and tuned to 95.1MHz. The data is first grouped to RDSBlocks. These RDSBlocks are transformed to byte arrays, to be stored to 'examples/rds/RDSBlock.bin' as binary data.

5.2.19 RDSBlock2RDSGroup

Package:	de.lmu.ifi.pms.ldsms.rds
Type:	Node
Input data type:	de.lmu.ifi.pms.ldsms.rds.RDSBlock
Input meta type:	de.lmu.ifi.pms.ldsms.rds.RDSBlockMetaInfo
Output data type:	de.lmu.ifi.pms.ldsms.rds.RDSGroup
Output meta type:	de.lmu.ifi.pms.ldsms.rds.RDSGroupMetaInfo
Dependencies:	no additional external library needed
Abstract:	Collects single RDSBlock and RDSBlockMetaInfo elements, groups them to a RDSGroup and RDSGroupMetaInfo and passes both together to the children.

Attributes:

Attribute-name	Type	Default	Optional	Description
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.19: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.rds.RDSBlock2RDSGroup" level=
        "DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.rds.easyway.EasyWaySource"
          device="/dev/ttyS0" frequency="95.1">
        <node class="de.lmu.ifi.pms.ldsms.rds.easyway.
            EasyWayRDSChunk2RDSBlock">
          <node class="de.lmu.ifi.pms.ldsms.rds.RDSBlock2RDSGroup">
            <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain
                "/>
          </node>
        </node>
      </source>
    </network>
  </services>
</server>
```

This example creates an instance of EasyWaySource, that reads the data from an EasyWay Light RDS-TMC reciever, which is registered under '/dev/ttyS0' and tuned to 95.1MHz. The data is first grouped to RDSBlocks and then to RDSGroups. The instance of ConsoleDrain is used to print the result to the console.

5.2.20 RDSGroup2ByteArray

Package: de.lmu.ifi.pms.ldsms.rds
Type: Node
Input data type: de.lmu.ifi.pms.ldsms.rds.RDSGroup
Input meta type: de.lmu.ifi.pms.ldsms.rds.RDSGroupMetaInfo
Output data type: byte array (byte[])

Output meta type: byte array (byte[])
Dependencies: no additional external library needed
Abstract: Transforms a RDSGroup and its meta information into two byte arrays (byte[]). The first array (with a length of 13) represents the data and its structure is as follows:

- data byte 0: Block A id
- data byte 1: Block A high byte
- data byte 2: Block A low byte
- data byte 3: Block B id
- data byte 4: Block B high byte
- data byte 5: Block B low byte
- data byte 6: Block C id
- data byte 7: Block C high byte
- data byte 8: Block C low byte
- data byte 9: Block D id
- data byte 10: Block D high byte
- data byte 11: Block D low byte

The second array (with a length of six) represents the metadata and is interpreted as follows:

- meta byte 0-3: Frequency
- meta byte 4: number of Errors
- meta byte 5: bit 0 wether all errors are corrected

Attributes:

Attribute-name	Type	Default	Optional	Description
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.20: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.rds.easyway.EasyWaySource"
        level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.rds.easyway.EasyWaySource"
          device="/dev/ttyS0" frequency="95.1">
        <node class="de.lmu.ifi.pms.ldsms.rds.easyway.
            EasyWayRDSChunk2RDSBlock">
          <node class="de.lmu.ifi.pms.ldsms.rds.RDSBlock2RDSGroup">
            <node class="de.lmu.ifi.pms.ldsms.rds.RDSGroup2ByteArray
                ">
              <drain class="de.lmu.ifi.pms.ldsms.generics.
                  ByteArrayFileDrain" file="examples/rds/RDSGroup.bin
                  "/>
            </node>
          </node>
        </node>
      </source>
    </network>
  </services>
</server>
```

This example creates an instance of EasyWaySource, that reads the data from an EasyWay Light RDS-TMC reciever, which is registered under '/dev/ttyS0' and tuned to 95.1MHz. The data is first grouped to RDSBlocks and then to RDSGroups. These RDSGroups are transformed to byte arrays, to be stored to 'examples/rds/RDSGroup.bin' as binary data.

5.2.21 RDSGroup2RDSGroup

Package:	de.lmu.ifi.pms.ldsms.rds
Type:	Node
Input data type:	de.lmu.ifi.pms.ldsms.rds.RDSGroup
Input meta type:	de.lmu.ifi.pms.ldsms.rds.RDSGroupMetaInfo
Output data type:	de.lmu.ifi.pms.ldsms.rds.RDSGroup
Output meta type:	de.lmu.ifi.pms.ldsms.rds.RDSGroupMetaInfo
Dependencies:	no additional external library needed

Abstract: Tries to convert the given RDSGroup into one of the following RDS groups:

- RDSGroup0A
- RDSGroup2A
- RDSGroup3A
- RDSGroup4A
- RDSGroup8A

Data, that can't be transformed into one of the mentioned groups, will be dropped.

Attributes:

Attribute-name	Type	Default	Optional	Description
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.21: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.rds.easyway.EasyWaySource"
        level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.rds.easyway.EasyWaySource"
          device="/dev/ttyS0" frequency="95.1">
        <node class="de.lmu.ifi.pms.ldsms.rds.easyway.
            EasyWayRDSChunk2RDSBlock">
          <node class="de.lmu.ifi.pms.ldsms.rds.RDSBlock2RDSGroup">
            <node class="de.lmu.ifi.pms.ldsms.rds.RDSGroup2RDSGroup"
                >
              <drain class="de.lmu.ifi.pms.ldsms.generics.
                  ConsoleDrain"/>
```

```
            </node>
          </node>
        </node>
      </source>
    </network>
  </services>
</server>
```

This example creates an instance of EasyWaySource, that reads the data from an EasyWay Light RDS-TMC reciever, which is registered under '/dev/ttyS0' and tuned to 95.1MHz. The data is first grouped to RDSBlocks and then to RDSGroups. The instance of ConsoleDrain is used to print the result to the console.

5.2.22 RDSGroup2RDSGroup3A

Package:	de.lmu.ifi.pms.ldsms.rds
Type:	Node
Input data type:	de.lmu.ifi.pms.ldsms.rds.RDSGroup
Input meta type:	de.lmu.ifi.pms.ldsms.rds.RDSGroupMetaInfo
Output data type:	de.lmu.ifi.pms.ldsms.rds.RDSGroup3A
Output meta type:	de.lmu.ifi.pms.ldsms.rds.RDSGroupMetaInfo
Dependencies:	no additional external library needed
Abstract:	If the given RDSGroup is a RDSGroup3A or can be converted into a RDSGroup3A and the application group type is 16 (0x10), the input will be send to all connected consumers as a new RDSGroup3A_8A, otherwise it will be dropped.

Attributes:

Attribute-name	Type	Default	Optional	Description
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.22: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
```

```
    <logger name="de.lmu.ifi.pms.ldsms.rds.RDSGroup2RDSGroup3A"
        level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.
          ByteArrayFileSource" file="examples/rds/RDSGroup.bin">
        <node class="de.lmu.ifi.pms.ldsms.rds.ByteArray2RDSGroup">
          <node class="de.lmu.ifi.pms.ldsms.rds.RDSGroup2RDSGroup3A"
              >
            <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain
                "/>
          </node>
        </node>
      </source>
    </network>
  </services>
</server>
```

This example does exactly the same as described in abstract. The instance of ConsoleDrain is used to print the result to the console.

5.2.23 RDSGroup2TMCMessage

Package:	de.lmu.ifi.pms.ldsms.rds
Type:	Node
Input data type:	de.lmu.ifi.pms.ldsms.rds.RDSGroup
Input meta type:	de.lmu.ifi.pms.ldsms.rds.RDSGroupMetaInfo
Output data type:	de.lmu.ifi.pms.ldsms.tmc.TMCMessage
Output meta type:	de.lmu.ifi.pms.ldsms.rds.TMCMessageMetaInfo
Dependencies:	no additional external library needed

Abstract:

This class maps the content of a RDSGroup8A into a convenient data structure that also reflects the "multiple information blocks" substructure of a RDSGroup8A Multigroup.
The Data structure reflects the semantic separation of single elements in the additional content and the scopes of the single optional message content elements.
Basically, the content is split into scopes by separator elements, but there is an additional scoping rule for destinations, events and quantifiers.
The following rules for scoping apply:

- The elements that may occur only once in a RDSMultiGroup are considered as having a global scope for the whole group.
- An event is considered as having a scope that reaches from the event to the next separator or event, whichever comes first. This scope determines the association of quantifiers to events.
- The destination element has a scope that reaches to the next destination element or separator, whichever comes first. If a diversion element precedes the destination, the scope of the destination ends after the diversion element.

Attributes:

Attribute-name	**Type**	**Default**	**Optional**	**Description**
eventDB	String		No	Name (and path) of the file, that stores the event list as a CSV table. This table is used, to transform the event codes into a user readable representation.
locales	String	de	Yes	A comma seperated list of locales (e.g. 'de' for german, 'en' for english) that represent the languages, the event code should be translated.
locationDB	String		No	Name (and path) of the file, that stores the location list as a CSV table. This table is used, to transform the location codes into a user readable representation.

Attribute-name	Type	Default	Optional	Description
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.23: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.rds.RDSGroup2TMCMessage"
        level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.
          ByteArrayFileSource" file="examples/rds/RDSGroup.bin">
        <node class="de.lmu.ifi.pms.ldsms.rds.ByteArray2RDSGroup">
          <node class="de.lmu.ifi.pms.ldsms.rds.RDSGroup2TMCMessage"
              >
            <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain
                "/>
          </node>
        </node>
      </source>
    </network>
  </services>
</server>
```

This example does exactly the same as described in abstract. The instance of ConsoleDrain is used to print the result to the console.

5.2.24 RDSGroup8AMultiGroupLinker

Package: de.lmu.ifi.pms.ldsms.rds
Type: Node
Input data type: de.lmu.ifi.pms.ldsms.rds.RDSGroup
Input meta type: de.lmu.ifi.pms.ldsms.rds.RDSGroupMetaInfo
Output data type: de.lmu.ifi.pms.ldsms.rds.RDSGroup
Output meta type: de.lmu.ifi.pms.ldsms.rds.RDSGroupMetaInfo
Dependencies: no additional external library needed

Abstract: RDSGroup8AMultiGroupLinker provides methods for linking RDS-TMC multi group messages together. Data, that is not of type RDSGroup8A, will be forwarded without any modifications. Data, that is of type RDSGroup8A, will be linked to the current first RDSGroup8A. If the given RDSGroup8A is a first group itself, the old one will be send and the given one will be abrogated.

Attributes:

Attribute-name	Type	Default	Optional	Description
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.24: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.rds.RDSGroup2RDSGroup3A"
        level="DEBUG" />
  </logging>
  <services>
  <network>
    <source class="de.lmu.ifi.pms.ldsms.generics.ByteArrayFileSource
        " file="examples/rds/RDSGroup.bin">
      <node class="de.lmu.ifi.pms.ldsms.rds.ByteArray2RDSGroup">
        <node class="de.lmu.ifi.pms.ldsms.rds.RDSGroup2RDSGroup">
          <node class="de.lmu.ifi.pms.ldsms.rds.
              RDSGroup8AMultiGroupLinker">
            <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain
                "/>
        </node>
      </node>
    </source>
    </network>
  </services>
</server>
```

This example reads the data from examples/rds/RDSGroup.bin, creates RDSGroups and links multi group messages together. The instance of ConsoleDrain is used to print the result to the console.

5.2.25 SpexNode

Package: de.lmu.ifi.pms.ldsms.xml
Type: Node
Input data type: java.lang.String
Input meta type: java.lang.Object
Output data type: java.lang.String
Output meta type: java.lang.Void
Dependencies:

name: **SPEX:** XPath Evaluation against XML Streams

version: Spex 1.0 (or higher)

url: `http://spex.sourceforge.net/`

Abstract: Filters data from a XML stream, using the de.lmu.ifi.pms.spex.main.SpexProcessor and a XPath expression.

Attributes:

Attribute-name	Type	Default	Optional	Description
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.
xpath	String		No	The XPath expression for querying the XML stream.

Example:

Listing 5.25: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.xml.SpexNode" level="DEBUG" /
        >
  </logging>
  <services>
```

```
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
          file="examples/xml/TMCMessage.xml" repeat="true">
        <node class="xml.SpexNode" xpath="/descendant::TMCMessage[
            descendant::eventName/child::text()='Unfall']">
          <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"
              />
        </node>
      </source>
    </network>
  </services>
</server>
```

This example reads xml data from examples/xml/TMCMessage.xml and filters all the events of type 'Unfall' using SpexNode. The instance of ConsoleDrain is used to print the result to the console.

5.2.26 String2ByteArray

Package: de.lmu.ifi.pms.ldsms.generics
Type: Node
Input data type: java.lang.String
Input meta type: java.lang.String
Output data type: byte array (byte[])
Output meta type: byte array (byte[])
Dependencies: no additional external library needed
Abstract: String2ByteArray takes as input Strings and forwards them as byte arrays. The parameter 'encoding' can be used to specify the encoding format (e.g. US-ASCII, UTF-8, UTF-16BE, UTF-16LE, UTF-16). More informations about valid encoding formats can be found in the specification of `java.nio.charset.Charset` at `http://java.sun.com/j2se/1.5.0/docs/api/java/nio/charset/Charset.html`.

Attributes:

Attribute-name	Type	Default	Optional	Description
encoding	String	ISO-8859-1	Yes	Specifies which encoding format should be used to transform the incomming Strings into byte arrays.
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.26: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.generics.String2ByteArray"
        level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
          file="examples/hello_world/input.txt" interval="0" repeat=
        "true" forwardEOL="true">
        <node class="de.lmu.ifi.pms.ldsms.generics.String2ByteArray"
            >
          <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"
              />
        </node>
      </source>
    </network>
  </services>
</server>
```

This example creates an instance of String2ByteArray, that recieves its incomming data and metadata from a StringFileSource. After encoding the incomming strings using ISO-8859-1, everything is forwarded to the ConsoleDrain. The ConsoleDrain prints every incomming data to the console.

5.2.27 StringFileDrain

Package:	de.lmu.ifi.pms.ldsms.generics
Type:	Drain
Input data type:	java.lang.String
Input meta type:	java.lang.Object
Dependencies:	no additional external library needed
Abstract:	A StringFileDrain writes the incomming data strings into the specified file using UTF-8 encoding. The metadata is not written into the specified file. The metadata is only used, to indicate that the corresponding data is a GoodbyeMessage. This data won't be written immediately but at last to the specified file, when the StringFileDrain stops.

Attributes:

Attribute-name	Type	Default	Optional	Description
file	String		No	Specifies the path of the destination file.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.27: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.generics.StringFileDrain"
        level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
          file="examples/hello_world/input.txt" interval="0" repeat=
        "true" forwardEOL="true">
        <drain class="de.lmu.ifi.pms.ldsms.generics.StringFileDrain"
            file="examples/hello_world/output.txt" />
      </source>
    </network>
  </services>
</server>
```

This example reads the data from 'examples/hello_world/input.txt' and copies it to 'examples/hello_world/output.txt' using StringFileDrain.

5.2.28 StringFileSource

Package: de.lmu.ifi.pms.ldsms.generics
Type: Source
Output data type: java.lang.String
Output meta type: java.lang.Void
Dependencies: no external library needed
Abstract: StringFileSource reads a file as a list of strings. It does not send any metainfo information, but merely the content of the file line by line. It also sends the '\n' newline delimiter.

Attributes:

Attribute-name	Type	Default	Optional	Description
delay	Integer	100	Yes	Specifies a delay (in ms). After reading a part of the file, the thread will sleep for the specified time, before the next part will be read.
file	String		No	Specifies the path of the source file.
forwardEOL	Boolean	false	Yes	Specifies, if line terminators will be eleminated (forwardEOL=false) or forwarded to the drains. forwardEOL is only used, when ignoreEOL is set to false.
ignoreEOL	Boolean	false	Yes	Specifies, if each line is send seperatly (ignoreEOL=false), or as much characters as possible should be send.
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
repeat	Boolean	false	Yes	Specifies, whether the reading should restart after the end of file was reached.

Example:

Listing 5.28: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
        level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
          file="examples/hello_world/input.txt" interval="0" repeat=
          "true">
        <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain" />
      </source>
    </network>
  </services>
</server>
```

This example creates an instance of StringFileSource, that reads from examples/hello_world/input.txt (relative to the L-DSMS startup location) with a delay of 1 second. The strings read from the file, are printed to the console by the ConsoleDrain. If the end of file is reached during reading, the StringFileSource will repeat the reading at the beginning of the file.

5.2.29 StringReplaceNode

Package: de.lmu.ifi.pms.ldsms.string
Type: Source
Output data type: java.lang.String
Output meta type: java.lang.String
Dependencies: no external library needed
Abstract: The StringReplaceNode can be used to do basic text transformations on an input stream. It is configured using a regular expression in the format '<match ex.> / <replace ex.>'.
The match expression specifies, which parts should be replaced.
The replace expression specifies, how the matched parts should be replaced.

Attributes:

Attribute-name	Type	Default	Optional	Description
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
regex	String		No	Specifies the match and the replace expression in the format '<match ex.> / <replace ex.>'.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.29: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.string.StringReplaceNode"
        level="DEBUG" />
  </logging>
  <services>
    <network>
```

```
      <source class="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
          file="examples/regex/input.txt" ignoreEOL="true">
        <node class="de.lmu.ifi.pms.ldsms.string.StringReplaceNode"
            regex="(\w*)(e|o)(\w*)(o|e)(\w*)/$1$4$3$2$5">
          <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"
              linebreak="false" />
        </node>
        <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"
            linebreak="false" />
      </source>
    </network>
  </services>
</server>
```

This example reads the data from 'examples/regex/input.txt' using StringFileDrain and changes the order of 'e' and 'o' within each word. The instances of ConsoleDrain are used to show the difference between the unchanged and the changed data and produce an output similar to the following one:

```
75 [main] DEBUG de.lmu.ifi.pms.ldsms.string.StringReplaceNode -
   Match expression set to (\w*)(e|o)(\w*)(o|e)(\w*).
75 [main] DEBUG de.lmu.ifi.pms.ldsms.string.StringReplaceNode -
   Replace expression set to $1$4$3$2$5.
102 [main] INFO de.lmu.ifi.pms.ldsms.network.Server - Initializing
   Server...
103 [main] INFO de.lmu.ifi.pms.ldsms.network.Server - Starting
   Server...
Holle Kitty is levod by peeplo all evor the world.
Hello Kitty is loved by people all over the world.
```

5.2.30 StringSocketDrain

Package:	de.lmu.ifi.pms.ldsms.generics
Type:	Drain
Input data type:	java.lang.String
Input meta type:	java.lang.Object
Dependencies:	no additional external library needed
Abstract:	A StringSocketDrain drain writes the incomming byte strings to every client, connected at the specified port.

Attributes:

Attribute-name	Type	Default	Optional	Description
period	Integer	1	Yes	Specifies the interval in seconds, the drain will repeat looking for locked up client connections.
port	Integer		No	Specifies the port at which the SocketDrain will accept client connections.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.30: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.generics.StringSocketDrain"
        level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
          file="test/Hello_World.txt" interval="0" repeat="true"
          forwardEOL="true">
        <drain class="de.lmu.ifi.pms.ldsms.generics.
            StringSocketDrain" port="6543" />
      </source>
    </network>
  </services>
</server>
```

This example creates an instance of StringSocketDrain, that recieves its incomming data and metadata from a StringFileSource. The incomming strings are send to every client connected at port 6543.

5.2.31 StringSocketSource

Package: de.lmu.ifi.pms.ldsms.generics
Type: Source
Output data type: java.lang.String
Output meta type: java.lang.Object
Dependencies: no additional external library needed

Abstract: StringSocketSource recieves the data from a socket connection as Strings.

Attributes:

Attribute-name	Type	Default	Optional	Description
host	String		No	Specifies the hostname for the socket connection.
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
port	Integer		No	Specifies the portnumber for the socket connection.
retry	Integer	1000	Yes	Specifies the interval for reconnect attempts, if the connection is broken.

Example:

Listing 5.31: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.generics.StringSocketSource"
        level="DEBUG" />
  </logging>
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.
          StringSocketSource" host="alice" port="6543" retry="5000">
        <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"/>
      </source>
    </network>
  </services>
</server>
```

This example creates an instance of StringSocketSource, that builds up a socket connection to host alice at port 6543. If the connection fails, reconnect attempts are repeated every 5 seconds.

5.2.32 StringTokenizerNode

Package: de.lmu.ifi.pms.ldsms.string
Type: Source
Output data type: java.lang.String

Output meta type: java.lang.String
Dependencies: no external library needed
Abstract: This node can be used, to break a large string (e.g. produced by an instance of StringFileSource) into smaller pieces (tokens). Every part of the input, that matches the given regular expression, will be interpreted as a delimiter. Each token will be forwarded seperatl, but the delimiters will be eliminated.
There are three different policies to handle the metadata:

- drop the metadata
  → metaPolicy="drop"
- repeat it with the first token
  → metaPolicy="repeatOnce"
- repeat it with every token
  → metaPolicy="repeat"

Attributes:

Attribute-name	Type	Default	Optional	Description
metaPolicy	String	drop	Yes	Specifies, which forward policy should be used. Valid values are 'drop', 'repeatOnce' and 'repeat'.
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
regex	String		No	Specifies the expression, that matches the delimiters.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.32: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.string.StringTokenizerNode"
        level="DEBUG" />
  </logging>
```

```
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
          file="src/regex/input.txt" ignoreEOL="true">
        <node class="de.lmu.ifi.pms.ldsms.string.StringTokenizerNode
            " regex="\s">
          <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"
              linebreak="true" />
        </node>
        <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"
            linebreak="true" />
      </source>
    </network>
  </services>
</server>
```

This example reads the data from 'examples/regex/input.txt' using StringFileDrain and breaks it into smaller pieces at every whitespace. The instances of ConsoleDrain are used to show the difference between the unsplitted and the splitted data and produce an output similar to the following one:

```
0 [main] INFO de.lmu.ifi.pms.ldsms.network.Server  - Configuring
    Server...
76 [main] DEBUG de.lmu.ifi.pms.ldsms.string.StringTokenizerNode  -
    Regex set to \s.
80 [main] DEBUG de.lmu.ifi.pms.ldsms.string.StringTokenizerNode  -
    Metadata will be dropped.
106 [main] INFO de.lmu.ifi.pms.ldsms.network.Server  - Initializing
    Server...
107 [main] INFO de.lmu.ifi.pms.ldsms.network.Server  - Starting
    Server...
Hello
Kitty
is
loved
by
people
all
over
the
world.
Hello Kitty is loved by people all over the world.
```

5.2.33 TMCMessage2Xml

Package: de.lmu.ifi.pms.ldsms.xml
Type: Node
Input data type: de.lmu.ifi.pms.ldsms.tmc.TMCMessage

Input meta type:	de.lmu.ifi.pms.ldsms.tmc.TMCMessageMetaInfo
Output data type:	java.lang.String
Output meta type:	de.lmu.ifi.pms.ldsms.network.SalutationMetaInfo
Dependencies:	no additional external library needed
Abstract:	Creates XML elements to represent the data and metadata in a plattform independend data structure.

Attributes:

Attribute-name	Type	Default	Optional	Description
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
prettyPrint	boolean	false	Yes	If pretty print is enabled, tabs and newlines are used to make the logical structure visible to humans. Each XML element will start at a new line and will be inteded relative to its parent element.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.
xpath	String		No	The XPath expression for querying the XML stream.

Example:

Listing 5.33: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO" />
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.rds.easyway.
          EasyWayFileSource" file="examples/easyway/input.bin"
          interval="100" frequency="95.1" version="1.0">
        <node class="de.lmu.ifi.pms.ldsms.rds.easyway.
            EasyWayRDSChunk2RDSBlock">
          <node class="de.lmu.ifi.pms.ldsms.rds.RDSBlock2RDSGroup">
            <node class="generics.Filter">
              <and>
                <condition class="rds.RDSGroupCorrectnessCondition"/
                    >
```

```
                <condition class="rds.RDSGroup8ARepetitionCondition"
                    />
              </and>
              <node class="rds.RDSGroup2RDSGroup">
                <node class="rds.RDSGroup8AMultiGroupLinker">
                  <node class="tmc.RDSGroup2TMCMessage" locationDB="
                      tmcdatabases/LocationList_de.csv" eventDB="
                      tmcdatabases/EventList_en_de.csv" locales="
                      de_DE">
                    <node class="xml.TMCMessage2Xml" prettyPrint="
                        true">
                      <drain class="generics.StringFileDrain" file="
                          examples/xml/TMCMessage.xml" />
                    </node>
                  </node>
                </node>
              </node>
            </node>
          </node>
        </source>
      </network>
    </services>
</server>
```

This example simulates an EasyWay light RDS reciever of version '1.0', that is tuned to the frequency 95.1Mhz. The data will be read from 'examples/easyway/input.bin' every 100ms. It is processed step-by-step to a TMCMessage. The TMCMessages are transformed to a XML format, using TMCMessage2XML. The instance of StringFileDrain is used to save the output to 'examples/xml/TMCMessage.xml'.

5.2.34 TMCMessageManagement

Package:	de.lmu.ifi.pms.ldsms.xml
Type:	Node
Input data type:	de.lmu.ifi.pms.ldsms.tmc.TMCMessage
Input meta type:	de.lmu.ifi.pms.ldsms.tmc.TMCMessageMetaInfo
Output data type:	java.lang.String
Output meta type:	de.lmu.ifi.pms.ldsms.network.SalutationMetaInfo
Dependencies:	no additional external library needed
Abstract:	Manages the TMCMessages and keeps an actual state of the TMC channel, i.e. caches all currently valid TMCMessages. This state is send to subsequent nodes in a certain delay, which is specified with the delay attribute.

Attributes:

Attribute-name	Type	Default	Optional	Description
delay	Integer	15000	Yes	Specifies the time interval (in milliseconds) the current state is send repeatly to subsequent nodes.
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.34: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO">
    <logger name="de.lmu.ifi.pms.ldsms.xml. TMCMessageManagement"
        level="DEBUG" />
  </logging>
  <services>
    <network>
    </network>
  </services>
</server>
```

5.2.35 Xml2TMCMessage

Package: de.lmu.ifi.pms.ldsms.xml
Type: Node
Input data type: java.lang.String
Input meta type: java.lang.Object
Output data type: de.lmu.ifi.pms.ldsms.tmc.TMCMessage
Output meta type: de.lmu.ifi.pms.ldsms.tmc.TMCMessageMetaInfo

Dependencies:

name: Apache **Commons Digester**

version: Digester 1.6 (or higher)

url: `http://commons.apache.org/digester/`

name: Apache **Crimson**

version: Crimson 1.1 (or higher)

url: `http://xml.apache.org/crimson/`

Abstract: XML2TMCMessage provides methods for transforming the XML representation of a RDS-TMC message into a TMCMessage object.

Attributes:

Attribute-name	Type	Default	Optional	Description
name	String		Yes	Specifies a unique identifier within the current L-DSMS instance, for this component.
sourcerefs	String		Yes	A comma seperated list of identifiers from additional sources, that aren't direct parents.

Example:

Listing 5.35: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO" />
  <services>
    <network>
      <source class="de.lmu.ifi.pms.ldsms.generics.StringFileSource"
          file="examples/xml/TMCMessage.xml">
        <node class="de.lmu.ifi.pms.ldsms.xml.Xml2TMCMessage">
          <drain class="de.lmu.ifi.pms.ldsms.generics.ConsoleDrain"
              />
        </node>
      </source>
    </network>
  </services>
</server>
```

This example reads the TMC messages from a XML document and uses Xml2TMCMessage to transform it back to TMCMessage. The instance of ConsoleDrain is used to print the result to the console.

5.3 FilterConditions

FilterConditions are used by Filter nodes to verify, that the incomming data fulfils the required conditions. The Filter nodes ask the FilterConditions, if they would discard or accept a given input. If the input is forwarded or not, is based on these informations.

5.3.1 AndCondition

Package:	de.lmu.ifi.pms.ldsms.generics
Output data type:	depends on the types of the children
Output meta type:	depends on the types of the children
Dependencies:	no additional external library needed
Abstract:	AndCondition provides a method for combining several conditions by a boolean and. The request is forwarded to each of these conditions. The AndCondition accepts the input only, if all subconditions accept the input.

5.3.2 FalseCondition

Package:	de.lmu.ifi.pms.ldsms.generics
Output data type:	java.lang.Object
Output meta type:	java.lang.Object
Dependencies:	no additional external library needed
Abstract:	A false condition will always return false.

5.3.3 NotCondition

Package:	de.lmu.ifi.pms.ldsms.generics
Output data type:	same as the child condition
Output meta type:	same as the child condition
Dependencies:	no additional external library needed
Abstract:	A NotCondition negates the result of it's child condition. So the real checking is done by the child condition and NotCondition only forwards the request.

5.3.4 OrCondition

Package:	de.lmu.ifi.pms.ldsms.generics
Output data type:	depends on the types of the children
Output meta type:	depends on the types of the children

Dependencies: no additional external library needed
Abstract: OrCondition provides a method for combining several conditions by a boolean or. The request is forwarded to each of these conditions. The OrCondition accepts the input, if at least one subcondition accepts the input.

5.3.5 RDSGroup8ARepititionCondition

Package: de.lmu.ifi.pms.ldsms.rds
Output data type: de.lmu.ifi.pms.ldsms.rds.RDSGroup
Output meta type: de.lmu.ifi.pms.ldsms.rds.RDSGroupMetInfo
Dependencies: no additional external library needed
Abstract: RDSGroup8ARepititionCondition caches the last two recieved rds groups and returns true for an rds group if it is contained in its internal cache and if the group wasn't the last recieved. This condition implements the validation through repetition that the RDS specification requires (a group is only sent if it was immediately repeated at least once).

5.3.6 RDSGroupCorrectnessCondition

Package: de.lmu.ifi.pms.ldsms.rds
Output data type: de.lmu.ifi.pms.ldsms.rds.RDSGroup
Output meta type: de.lmu.ifi.pms.ldsms.rds.RDSGroupMetInfo
Dependencies: no additional external library needed
Abstract: RDSGroup8ACorrectnessCondition filters out RDS groups containing incorrect blocks. A RDS group contains incorrect blocks, if the RDSGroupMetaInfo indicates, that there are uncorrected errors.

5.3.7 RDSGroupFilterCondition

Package: de.lmu.ifi.pms.ldsms.generics
Output data type: de.lmu.ifi.pms.ldsms.rds.RDSGroup
Output meta type: de.lmu.ifi.pms.ldsms.rds.RDSGroupMetInfo
Dependencies: no additional external library needed
Abstract: RDSGroupFilterCondition provides methods for the filtering of RDS-TMC messages, based on their group identifier.

Attributes:

Attribute-name	Type	Default	Optional	Description
group	String		No	Specifies the group identifier of groups, that should be accepted.

5.3.8 RegexCondition

Package: de.lmu.ifi.pms.ldsms.string
Output data type: java.lang.String
Output meta type: java.lang.String
Dependencies: no additional external library needed
Abstract: RegexCondition provides a method for filtering strings, using regular expressions. RegexCondition only accepts the input, if the input data matches the regular expression for data and the input metadata matches the regular expression for metadata.

Attributes:

Attribute-name	Type	Default	Optional	Description
data	String		Yes	Specifies the regular expression, the incomming data has to match to be accepted. If no regular expression is specified for data, every incomming data will be accepted.
meta	String		Yes	Specifies the regular expression, the incomming metadata has to match to be accepted. If no regular expression is specified for metadata, every incomming metadata will be accepted.

5.3.9 RepeatedStringCondition

Package: de.lmu.ifi.pms.ldsms.string
Output data type: java.lang.String
Output meta type: java.lang.String
Dependencies: no additional external library needed
Abstract:

5.3.10 TrueCondition

Package: de.lmu.ifi.pms.ldsms.generics

Output data type:	java.lang.Object
Output meta type:	java.lang.Object
Dependencies:	no additional external library needed
Abstract:	A TrueCondition will always return true.

5.4 Plugins

The L-DSMS core functionality is to manage and process data streams by a chain of processing nodes. Other functions can be added as plugins, using the L-DSMS plugin mechanism. The L-DSMS plugin mechanism is configured, like everything else, with the configuration file (cf. section A.2). Therefore, adding and removing plugins is as easy as configuring the components and L-DSMS can be excecuted with a minimal set of functions, preserving resources.

RMIPlugin is the plugin, that is shiped with the L-DSMS core package. Using this plugin, enables L-DSMS to be managed by VISU-L-DSMS.

Example:

Listing 5.36: Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<server>
  <logging level="INFO" />
  <plugin-list>
    <plugin-prefix>de.lmu.ifi.pms.ldsms.plugin</plugin-prefix>
    <plugin-prefix>de.lmu.ifi.pms.ldsms.network</plugin-prefix>
    <plugin class="de.lmu.ifi.pms.ldsms.plugin.RMIPlugin" registry-
        name="LDSMSServer" security-policy="server.policy" stub-class
        ="de.lmu.ifi.pms.ldsms.remote.
        ConfigurationFacadeRemoteAdapter" rmi-codebase="file:/<
        LDSMS_HOME>/ldsms.jar file:/<LDSMS_HOME>/lib/avalon-framework
        -4.2.0.jar"/>
  </plugin-list>
  <services>
    <network>
    ...
    </network>
  </services>
</server>
```

Appendix A

Appendix

A.1 Dependencies

This section lists all additional Java libraries with their download locations, that L-DSMS needs to work properly. Libraries, that are not necessary for the core functionality, but needed by optional components, are listed within the component documentation.

Name	Version	Vendor	Download Location
Log4j	$\geq$ 1.2	Apache Software Foundation	`http://logging.apache.org/log4j/1.2/index.html`
Commons Configuration	$\geq$ 1.0.4	Apache Software Foundation	`http://commons.apache.org/configuration/`

Table A.1: External Libraries

A.2 Configuration file

All configuration informations for L-DSMS are in an external configuration file, that has to be a valid XML document. L-DSMS will use that configuration file during start-up to configure itself. You can specify:

- the plugins, L-DSMS should use
- the logging level for L-DSMS in general and for each specific component
- the network structure (the used components and their relationship to each other)

The following Document Type Definition (DTD) illustrates the structure, the configuration file has to follow.

```
<!ELEMENT server (logging, plugin-list?, services)>
<!ELEMENT logging (logger*)>
```

```
<!ATTLIST logging level (OFF|FATAL|ERROR|WARN|INFO|DEBUG|ALL) #
    REQUIRED>
<!ELEMENT logger EMPTY>
<!ATTLIST logger name CDATA #REQUIRED>
<!ATTLIST logger level (OFF|FATAL|ERROR|WARN|INFO|DEBUG|ALL) #
    REQUIRED>
<!ELEMENT plugin-list (plugin-prefix*, plugin*)>
<!ELEMENT plugin-prefix (#PCDATA)>
<!ELEMENT plugin ANY>
<!ATTLIST plugin class CDATA #REQUIRED>
<!ELEMENT services (network)>
<!ELEMENT network (source+,node*>
<!ELEMENT source (node|drain)*>
<!ATTLIST source class CDATA #REQUIRED>
<!ATTLIST source name ID #IMPLIED>
<!-- other attributes depend on the value of 'class' -->
<!ELEMENT node ((and?|or?),node*)>
<!ATTLIST node class CDATA #REQUIRED>
<!ATTLIST node name ID #IMPLIED>
<!ATTLIST node sourcerefs IDREFS #IMPLIED>
<!-- other attributes depend on the value of 'class' -->
<!ELEMENT drain EMPTY>
<!ATTLIST drain class CDATA #REQUIRED>
<!ATTLIST drain sourcerefs IDREFS #IMPLIED>
<!-- other attributes depend on the value of 'class' -->
<!ELEMENT and (or?,condition+)>
<!ELEMENT or (and?,condition+)>
<!ELEMENT condition EMPTY>
<!ATTLIST condition class CDATA #REQUIRED>
<!-- other attributes depend on the value of 'class' -->
```

For an example, take a look at chapter 3 (Examples).