InfoFabrik gmbh

ReportServer

Administrator's Guide 3.0.4

InfoFabrik gmbh

ReportServer

Administrator's Guide 3.0.4

InfoFabrik GmbH, 2018

http://www.infofabrik.de/
http://www.reportserver.net/

Contents

Contents i

1 Preamble 3

2 First Steps 7
2.1 Configuration and installation . . . 7
2.2 Login . . . 7
2.3 Creating a data source . . . 9
2.4 Creating your First Report . . . 9
2.5 Importing a Graphical Report . . . 10
2.6 Creating users . . . 11
2.7 Terminal and FileServer . . . 12

3 User and Permission Management 15
3.1 The User Tree . . . 15
3.2 Permission Management . . . 16

4 Data Sources 23
4.1 Relational Databases . . . 23
4.2 Storage of Database Passwords . . . 24
4.3 The Internal Data Base . . . 24
4.4 Datasource Pool . . . 25
4.5 CSV Lists . . . 25
4.6 Script Data Sources . . . 27
4.7 BIRT Report Data Source . . . 27
4.8 Mondrian Datasource . . . 28
4.9 Data Source Bundle . . . 29
4.10 Configuration of a Standard Data Source . . . 30

5 File System 33
5.1 Configuration Files . . . 34
5.2 Filing of Scripts . . . 34
5.3 Accessing Resources by URL . . . 34

6 Report Management **37**
6.1 Fundamentals . . . 38
6.2 The Dynamic List . . . 40
6.3 Working with Parameters . . . 46
6.4 JasperReports . . . 61
6.5 Eclipse Birt . . . 62
6.6 SAP Crystal Reports . . . 63
6.7 Saiku / Mondrian Reports . . . 64
6.8 JXLS Reports . . . 64
6.9 Script Reports . . . 66
6.10 Grid Editor Reports . . . 69
6.11 Executing Reports via the URL . . . 84
6.12 Report Properties . . . 87
6.13 Report Metadata . . . 88
6.14 Drill Down Reports . . . 88

7 Global Constants **91**

8 User Variables **93**
8.1 Defining User Variables . . . 93
8.2 Allocating User Variables . . . 94
8.3 Using User Variables in Reports . . . 94

9 Import and Export **97**
9.1 Exporting . . . 97
9.2 Importing . . . 97

10 Scheduling of Reports **101**
10.1 Technical Backgrounds to Scheduler Jobs . . . 101
10.2 Filtering by the Status of a Job . . . 102
10.3 Notifications . . . 102
10.4 Conditional Scheduling . . . 102
10.5 Enhanced Scheduling . . . 103

11 Theming **105**

12 Terminal **111**
12.1 Using the Terminal . . . 111
12.2 The Virtual File System . . . 112
12.3 Assigning Aliases . . . 114
12.4 Scripts . . . 115

13 ReportServer Scripting **117**
13.1 A first Hello World . . . 118
13.2 How to Handle Errors . . . 118
13.3 Administrative Scripts . . . 121
13.4 Changing the Data Model . . . 123
13.5 Enhancing ReportServer with Scripts . . . 124

13.6 Scheduling of Scripts . . . 125
13.7 Accessing Scripts by URL . . . 125

14 Terminal Commands **127**
14.1 birt . . . 127
14.2 cd . . . 127
14.3 clearInternalDbCache . . . 127
14.4 config . . . 127
14.5 cp . . . 128
14.6 createTextFile . . . 128
14.7 desc . . . 128
14.8 echo . . . 128
14.9 editTextFile . . . 128
14.10eliza . . . 128
14.11exec . . . 129
14.12export all . . . 129
14.13hello . . . 129
14.14import all . . . 129
14.15kill . . . 129
14.16listpath . . . 129
14.17locate . . . 130
14.18ls . . . 130
14.19meminfo . . . 130
14.20mkdir . . . 130
14.21mv . . . 130
14.22pkg . . . 130
14.23ps . . . 130
14.24pwd . . . 131
14.25rcondition . . . 131
14.26reportmod . . . 131
14.27rev . . . 131
14.28rm . . . 132
14.29scheduleScript . . . 132
14.30scheduler . . . 132
14.31sql . . . 133
14.32teamspacemod . . . 133
14.33unzip . . . 133
14.34updateAlias . . . 133
14.35updatedb . . . 133
14.36usermod . . . 134
14.37xslt . . . 134
14.38zip . . . 134

15 Dashboards and Dadgets **137**
15.1 Static HTML Dadgets . . . 138
15.2 Embedding Dashboards . . . 140

16 SFTP Server **143**

17 Maintenance **145**
17.1 Testing User Specific Settings (su) 145
17.2 Logging 145
17.3 Recovering of Objects 146

A Expression Language **149**

B Demo Data **153**

Chapter 1

Preamble

Business Intelligence

Business Intelligence (BI) describes the ability to jointly analyze all of a company's data, distilling relevant information to be used to foster better business decisions. The foundation of any BI solution is the careful preprocessing of existing data, for example, in a data warehouse.

ReportServer acts as the gateway between end-users and the collected data, allowing users to efficiently access and analyze the available data. From camera-ready evaluations to fine-grained ad-hoc reporting; ReportServer provides you with the tools to support your daily work.

Target Audience

This document is designed for future administrators of ReportServer.

Separate manuals and instructions illustrate the various aspects of ReportServer.

ReportServer Configuration Guide: Describes the installation of ReportServer as well as the basic configuration options.

ReportServer User Guide: The user guide describes ReportServer from the point of view of the ultimate user. It includes an in-depth coverage of dynamic lists (ReportServer's adhoc reporting solution), execution of reports, scheduling of reports, and much more.

ReportServer Administrator Guide: The administrator guide describes ReportServer from the point of view of administrators that are tasked with maintaining the daily operation of the reporting platform including the development of reports, managing users and permissions, monitoring the system state, and much more.

ReportServer Scripting Guide: The ReportServer scripting guide covers the scripting capabilities of ReportServer which can be used for building complex reports as well as for extending the functionality of ReportServer or performing critical maintenance tasks. It extends the introduction to these topics given in the administrator guide.

System State

We proceed on the assumption that the basic configuration has been completed as de- scribed in the installation and configuration instructions. All examples given in this book are based on the available demo data.

ReportServer Demo Content

The demo data used in this book are part of the standard delivery scope of ReportServer and can be automatically loaded when starting the system. Additionally we provide a demo content package that preconfigures ReportServer for a fictitious company called "1 to 87". This includes setting up various users including permissions, TeamSpaces where these users have access and can collaborate as well as a number of demo reports.

Loading the demo warehouse. ReportServer comes with the option to install a demo database, which is the base for the demo report shipped with ReportServer. The demo data will be installed in an internal database (we will later see the significance of this internal database). To load the demo data on startup change the configuration file `/etc/datasources/internaldb.cf` to include the tag `<installdemodata>true</installdemodata>`. You will find this configuration file, if you log into ReportServer as root, go to the administration module.[1] For this choose Administration from the module bar at the top of the screen, and then choose File Server from the aspects on the left. Navigate to file `/etc/datasources/internaldb.cf` in the tree. To change the file choose the tab ***edit*** on the bottom. A sample configuration could, for example, look like:

```
<configuration>
  <internaldb>
    <location>dbtmp</location>
      <encryption>
        <disable>false</disable>
        <password>SecretPassphrase</password>
      </encryption>
      <installdemodata>true</installdemodata>
  </internaldb>
</configuration>
```

Note that this change will only take effect after a restart of ReportServer.

Loading the demo content. To install the demo content, log into ReportServer as root.After logging in, press CTRL+ALT+T to open the terminal. The terminal is a powerful tool to administer ReportServer. We will see various use cases for it throughout this manual. Type

```
pkg list
```

and confirm the command by pressing enter. You should see

```
baseconfig-RS2.1.1-5517-2013-11-06-18-02-40.zip
demobuilder-RS2.1.1-5517-2013-11-06-18-02-40.zip
reportserver$
```

[1] Look at the next section "First Steps" for an introduction to ReportServer and a guide to logging in, the user interface and the various interfaces.

The example content is supplied as a ReportServer package, which is basically a groovy script with some additional files. The `pkg list` terminal command lists all the available packages. In the example above, there were two packages. The ***baseconfig*** package is automatically installed whenever you conduct a fresh install of ReportServer. It installs the default configuration files in the `etc` directory of ReportServer's internal file server. The ***demobuilder*** package contains the example data we are going to install. The precise filename may differ from the one pictured here, depending on your ReportServer version.

Warning, the installation of the demo content, will remove any existing content. To install the example data use the following command:

```
pkg install -d <packagename>
```

where `packagename` denotes the name of the package as returned by the `pkg list` command. Note that you don't actually have to type the whole name, but can only enter the first few characters and press the tab-key to have ReportServer autocomplete the filename for you. After you issued the `pkg install` command ReportServer unzips and installs the example data. This might take a minute or two, but after a while you will be presented with the ***ok***-confirmation. If you reload your browser you can now use the example data.

Chapter 2

First Steps

This section will provide you with a first impression of how an administrator will work with ReportServer. On the basis of the demo data provided we will guide you step by step through the various sections of the Administration interface. We will use examples to explain the basic concepts that you will meet throughout ReportServer.

2.1 Configuration and installation

You will find a detailed description in the configuration and installation instructions in the freely available ReportServer configuration guide. As we will frequently refer to configuration files and options, we list below the most important locations where to configure ReportServer:

reportserver.properties To be found in the directory WEB-INF/classes; it includes basic configuration settings which have to be rarely adapted. This applies to login control settings, or passwords to encrypt sensitive data, etc.

persistence.xml To be found in the directory WEB-INF/classes/META-INF, it includes the configuration of the database connection.

All further settings will be made within ReportServer. ReportServer has an internal file system which you find in the administration module. There, in the spirit of UNIX systems, you will find the configuration files in the folder /etc. For further information on the configuration of ReportServer please refer to the configuration manual.

2.2 Login

Use your web browser and open the ReportServer home page in order to log into ReportServer. By default you will find it under the URL `http://SERVERNAME:PORT/reportserver/`. If you are on the server where the program is installed, the address is normally `http://127.0.0.1:8080/reportserver/`. In order to log into ReportServer for the first time, it must be preconfigured so that users can log-in with their user name and password (this is the default setup). A description of the various authentication procedures is given in the ReportServer configuration guide. Now, log in by entering the user name “root” and password “root”.

After login you get to the Dashboard module. The user interface is structured as follows: At the top screen margin you will find the module bar. Here you can switch between the different ReportServer modules. At the right hand side you will find the option to log off from the system as well as your user profile (click on your name) and global search. For further information on your profile and ReportServer's search functionality, please refer to the ReportServer user guide. As root you have access to the following modules:

Dashboard The dashboard enables the user to get important information at a glance. For further information, please refer to the ReportServer user guide as well as to the Dashboard section.

TeamSpace The TeamSpace module enables users or user groups to organize their individual work areas. The users may attend to nearly all of their settings themselves. For further information on TeamSpace, please refer to ReportServer user guide.

Scheduler The Scheduler module lists all scheduled reports. For further information on the Scheduler module, please refer to section Scheduling of reports as well as to the ReportServer user guide.

Administration The administration module provides access to a collection of various sub modules used by administrators. The administration module is split into the following sub modules.

User management Here users and user groups may be maintained.

Report management Enables to manage report objects.

Dadget library Dadgets (abbreviation for Dashboard Gadget) are used by the users to assemble their dashboard. In the Dadget library, administrators can provide pre-assembled dadgets which the users may simply select.

File system ReportServer uses the File system in many different ways.You will find here common configuration files or ReportServer scripts. But you also can provide resources (e.g. pictures) for reports.

Datasources This section enables to manage database connections which may be used as a basis for reports.

Global constants Global constants may be used in reports to swap out configuration parameters so that they can be managed centrally.

Import The import module enables to import ReportServer objects that had been exported before.

Permission management The permission management defines the access permissions for the varying ReportServer modules/functions.

Scheduler As compared to the Scheduler main module, the Administration module provides insight into any scheduler job made by the users. Now we are going to get acquainted with ReportServer in a quick resumé. We try to introduce important concepts and functions by presenting examples. If you haven't done so, you should now load the demo data. For the upcoming introduction the demo content is not necessary. See Section 1 on page 4.

2.3 Creating a data source

The basic requirement for running reports is the configuration of a data source. Data sources are organised hierarchically in a tree structure like many other objects. The ReportServer trees are organised in a similar way as you may know from files and folders of common file systems. The hierarchic structure enables you to retain the overview even when dealing with a large number of objects. In addition, the hierarchic structure enables to map even the most complicated access rules in a compact and comprehensible manner.

You will work with ReportServer trees just in the same way as you are used to from other programmes.You will create new objects by using the context menu (right-clicking an object), and with drag and drop you can move objects. Now, select the Data sources section within the Administration module. Below the Root folder, the folder "internal data sources" should already be located. Beside the existing folder, create another one: Right click on "Data source Root" -> Insert -> Folder. Now, select the newly created folder "unnamed". In the right part of the window you can edit the properties of the currently selected object. Rename the folder to "Demo data" and click on the ***Apply*** button.

Now, we will configure a data source which enables to access the internal demo data. Below the newly created folder, create then the data source type "relational database". Enter "Demo data" as the name for the new database connection. Select H2 as database type. User name and password are "demo". Enter the following JDBC connection URL: `jdbc:h2:dbtemp/rsdemodb`. The `dbtemp` path refers to the directory of the internal database as defined in configuration file etc/datasources/internaldb.cf in the internal file system of ReportServer. By default this is the dbtemp directory relative to the ReportServer directory. See the configuration guide on information on how to change this location.

Apply the chosen settings and run a test whether the connection can be established ("Test connection" button in the tool bar). Be aware to only test the connection after saving your changes.

2.4 Creating your First Report

In the following we will create a first basic report. To do so, switch to the section Report management in the Administration module. Here you see that reports are managed in a tree structure as well. As you did before, create a new folder with the name "demo reports". The provided demo data represents a small datamart for the fictitious model-making manufacturer "1 to 87". You will find the corresponding database scheme in the appendix. First, we want to create a customer list. For this purpose the reporting type "dynamic list" is suitable. Below the demo reports folder insert a report of type "dynamic list". Rename the report to **customer list** and select the data source (click on the magnifying glass) **demo data** that you have just created.

The configuration displayed varies with the data source selected. Relational databases require the setting of the appropriate SQL statement. We want to create a list showing any and all information about the customers of the company "1 to 87". In the demo data the `T_AGG_CUSTOMER` table has already been prepared for this. The following statement selects all data records from the table:

```
SELECT * FROM T_AGG_CUSTOMER
```

Apply the data and open the report by double clicking on the object in the tree. Now, the screen that opens should show the report execution area. This section is detailed in the ReportServer user manual. To get a quick overview of the "1 to 87" customers, select from ***Select columns*** all available columns and then on the left select ***Preview*** from the aspects. The first 50 data records should now display. If you wish to export the complete list, for instance, to an Excel file, select the button ***Excel export*** from the tool bar.

Then close the report by clicking on the button **x** on the right hand side above the tool bar to return to the Administration section. Alternatively, select the Administration module from the module bar.

2.5 Importing a Graphical Report

In the next step we want to import a graphical report from the ReportServer sample projects. Within your ReportServer installation you should find a folder called "pkg". Within there should be a file called `demobuilder-xx.zip` where xx denotes your ReportServer version. Copy that file to a temporary directory and there unzip the file.You should find a tmp directory containing various files of the type "export_...zip".

Now, select the section **Import** in the Administration module and then click on the **Start import** button. Select the file `export_birt_sample_reports.zip` and click on **Submit**. In the left part of the screen the sections Datasources, User management and Report management should display. This indicates that objects from these sections were exported to the export file. Click on Report management and select the folder **demo reports** as the import target. At the bottom click on the **objects** tab and select the **sales invoice** report.

Now switch to Datasources and choose the **demo data** data source that you created as default data source. For the import it will be entered in the report as data source. Then at the top, right click on **Finalize import** and answer the question whether you want to reset the configuration with **Yes**. Now select **Cancel job** from the tool bar to show ReportServer that no other objects are to be imported from this file.

Switch back to Report management and open the **Demo reports** folder. You might need to reload the tree to see the freshly imported report. For this click the reload button in the tree's toolbar. The imported report **sales invoice** should be located here. Open the report to ensure that the correct data connection has been set.

If you run the report you will first be directed to the parameters page of the report. Parameters enable the user to limit the data basis in particular for graphical reports (such as JasperReports or Eclipse-BIRT reports) according to its needs. However, parameters can be applied for all reports. For instance, enter 10167 as invoice number (order number) and click on **Preview**. The report will run for exactly this invoice number. When you return to the Administration section and select the **sales invoice** report, you can view the application of this parameter. Beside the general report settings you may switch to other aspects by clicking on the tabs at the bottom margin. Select **Parameter management**. The report shows the parameter **Order Number**. If you double click on the icon of the parameter, a dialogue box opens showing the parameter settings. Close the dialogue by clicking on the **Cancel** button.

2.6 Creating users

In the following we will create the **Report Management** user group as well as the **jondoe** user. So far you have been working under the user **root**. You should use this account only in an emergency case as the user **root** operates completely isolated from the management of permissions. It is reasonable to set up user accounts in such a way that the users will always be granted only those permissions they actually need for their work. In the Administration section now switch to User management. Like reports, users will be managed in ReportServer in a hierarchic structure. This enables to easily map, for instance, company hierarchies and to create users with similar or equal permissions in a common folder (or as we call them organisational unit). Beside hierarchic structuring you can additionally organise users by means of groups. We will get back to the various structuring options in more detail in later sections. First create the organisational unit **IT** and then the user **Jon Doe** in a sub-folder with **jondoe** as user name and password. Jon Doe shall be given the permissions to create and manage reports. Here it is recommendable to create a group which will be granted the respective permissions and to add jondoe to this group. To do this, create the organisational unit **roles** below the root directory as well as the group **Report management**. Add the user **Jon Doe** to the group either by using the corresponding button, or directly by Drag and Drop from the user tree (drag the user to the respective member list). Don't forget to save your changes by clicking on **Apply**.

jondoe needs the following permissions to manage reports: He must be able to access ReportServer and to use the Administration module, there, however, only the Report management. Within Report management we would like to allocate all rights to him. First, switch to module **Permission management** in the Administration section. Various subsections will now display. Now, select the subsection **Administration**. In ReportServer permissions are granted based on the ACL security model (for more information on ACLs, please refer to the section on user management or to http://de.wikipedia.org/wiki/Access_Control_List). Click on **Insert** to create a further permission entry. The entry will be added to the list. To edit the entry, double click on it. From the mask in column **Folk** (the receiver of rights) select the group **Report management** and allocate reading rights (r). To grant jondoe ReportServer access, go to the subsection **ReportServer Access** and create an ACE (an access control entry; a permission entry) that grants jondoe (or the group **Report management**) the execute (x) right. Now, when you log off and log on as jondoe, you will see that he will be allowed to see the Administration section, however, he will have no further access rights. Therefore, log in again with **root** and return to Permission management. Select subsection Report management and here as well add an access right by granting the group **Report management** reading access. Now Jon Doe is granted the rights to use the Administration module and there the Report management sub-module. However, he is lacking any rights with regard to single report objects. In ReportServer access rights can be granted fine-tuned down to the level of single objects. As jondoe is to account for the report management, we will assign him the rights so that he may access all report objects. To do this, go to Report management and select the root node. Switch to the permissions view of the root object by clicking on the tab below the expanded view. Add a further permission entry and double click on the newly created entry. The dialogue is identical to the already known permission dialogue box except for the option **inherit**. The option **inherit** controls the use of permission entries in trees. We wish to assign to jondoe any and all rights for all objects. To do so, under Folk we select again the group **Report management**. Keep the default settings for **access type** and **inherit**. In addition, we either set a tick at all rights individually, or we use the **Quick assign** and select **Full access**. If you now sign in as jondoe

you should be granted access to the complete Report section. When you are logged in with the user name **root** and you wish to quickly change the user you may also use the SU (switch user) command. You can do this by using the keyboard shortcut CTRL+L. In the dialogue box popping up select the user jondoe and click on **Submit**. The users who may execute the SU command may also be controlled in the Permission management.

2.7 Terminal and FileServer

Now log off and log in again as the user **root**. In the last part of the introduction we will get acquainted with two important Administration tools enabling to perform complex actions to ReportServer: the Terminal and ReportServer Scripts.

The Terminal follows the familiar Unix terminal and uses various commands to enable the set-up of a database connection, to move objects or retrieve system information. Open the Terminal by entering the keyboard shortcut CTRL-ALT-T. A window opens showing a prompt. Enter the command `ls` and confirm with ENTER. The `ls` command shows the objects in the current folder. All ReportServer trees (e.g. Report management or User management) are accessible from the Terminal, and are integrated as virtual file systems.

In the following we describe how to create a simple ReportServer script which goes into the internal **File System** (a sub-module of the administration module). The File System can be accessed via the Administration module or via the Terminal. Now enter the command `cd fileserver` to change the working directory to the root folder of the file system. The terminal supports the auto-completion function to enable quicker navigation within the Terminal. You can activate it by pressing the tab key. If you want to make sure which folder you are currently in, you can use the `pwd` command.

To verify whether the `bin` directory exists in FileServer, use the `ls` command. If it fails to exist create it by entering the `mkdir bin` command. With `cd bin` switch to the newly created directory. Now, we will create our first ReportServer script. In ReportServer, scripts have manifold tasks and are written in the programming language Groovy (http://groovy.codehaus.org/). For working with ReportServer, scripts are not necessarily required, however, you can use them to accomplish many special tasks (such as integrating complex data sources or importing of reports from third-party systems) and small enhancements. In this manual we will time and again meet with scripts. You will find a detailed introduction in the ReportServer scripting guide. Now create the script `hello.rs` by entering the command `createTextFile hello.rs`. A text editor opens. Enter the following line:

```
"Hello ReportServer Admin"
```

Be sure to enter the quotation marks. Click on Submit to save the script. To edit the file use the command `editTextFile hello.rs`. You can run the script with the command `exec hello.rs`. You should see the line

```
Hello ReportServer Admin
reportserver$
```

on the terminal.

With this we would like to finish our quick ReportServer resumé. Up to this point we have only en-

countered a fractional part of the ReportServer options. In the sections to come we will specifically outline individual ReportServer sections and discuss them in detail.

Chapter 3

User and Permission Management

3.1 The User Tree

Like other object types in ReportServer, users and groups are organized hierarchically in trees. The ***user tree*** can include the following object types.

Users Represent individual users of the system

Groups Arrange users independent of the structure given by the tree

Organisational Units Serve as folders to structure the user tree

New objects will be created in the user tree via the context menu of the folder node. When clicking on an object in the tree, the right part of the screen will display the properties of this object. The configurable properties vary with the object type.

The name and description fields are available for organisational units. Their name is displayed in the tree, the description field can include additional notes or comments. We will go into more detail explaining the tabs Permission management and User variables in the respective sections.

User objects will hold the personal data of the user. Beside the personal form of address, first and last name, they include the user name, e-mail address and, if applicable, the password. If a password has been manually entered in the Administration interface, it need not be changed by the user on the next login. The usual procedure to reset a password or set a password for the first time is to "activate" the user using the Activate button in the menu bar. One-time password is then sent by e-mail to the user's email address. The user will then be prompted to enter a new password when logging in for the first time.

The guidelines regulating the generation of new passwords, or to which the password chosen by the user has to comply, can by adapted via the configuration of the PasswordPolicy. For further information on PasswordPolicy as well as on the various authentication procedures supported by ReportServer we refer to the configuration guide.

The section Account blocking allows to inhibit individual users from accessing the system. The block can be set either manually or automatically by setting a certain expiration date. Accounts which have been blocked automatically, e.g. by exceeding the permissible number of login attempts, are unblocked and reactivated here.

User groups are mainly used in connection with the permission management, and therefore, they will be discussed in detail under this issue. Groups provide the possibility to summarize users who are not located in the same branch of the user tree, and to treat them equally when assigning rights. Beside the name and description fields, the user group configuration page has three fields to manage the group members. In the Direct Members field, individual users can be added to the group. The field Members (via groups) allows to add all members of another group to this group. The effect is identical, as if the members of the subgroup had been added manually and individually to the group. The field Members (via organisational units) eventually enables to add entire branches of the user tree to a group. Here as well it is to be interpreted that all users given in the branch are direct members of the group.

3.2 Permission Management

ReportServer principally distinguishes between two types of permissions. Generic rights are called those permissions which regulate the access to components or functions of ReportServer. Examples for generic rights are the basic right to login to the system, or to access the Admin section.

Beside the generic rights, object related rights can additionally be assigned. They always refer to ReportServer objects organized in tree based data structures such as data sources, reports, users, etc. Unlike generic rights, for the definition of object rights the inheritance of rights can be applied which can considerably reduce the administrative effort when dealing with large and complex user structures.

In ReportServer rights are assigned in form of Access Control Lists (ACL) and Access Control Entries (ACE). An ACL exists for each object for which the access is to be restricted that is an ACL exists for every report, data source, folder,etc., but also for every generic right. An ACL then consists of a list of ACEs that hold the following information:

Folk The user object which is granted the permission. This could be a user, a group, or an organisational unit.

Access type Determines whether the permission entry expands the rights attributed by other entries, or whether it further narrows them.

Basic rights Basic rights are assignable to all ReportServer objects. Their specific significance depends on the type of the relevant object and will be discussed in the following:

Determining if a Permission is Granted

To test if a user has been granted a certain right ReportServer checks for each ACE (in the order they are defined) if the ACE applies to the user, i.e., if the user is part of the folk (i.e., if the folk is a group it is checked if the user is a member in this group, if the folk is an organisational unit, it

is checked if the user is part of that sub-tree and if the folk is a user it is tested if the users are the same). If an ACE matches, the access type defines whether the right is granter or denied. This means, that even if an ACE later in the ACL grants a right, but an earlier one denies it, the right is not granted. If no ACE matches then the right is not granted. Consider the following example ACL consisting of three ACEs. The first denies the read and write right for members of group A. The second grants the read right to members of group B and the third grants the write right to members of group C.

```
group A  deny   read write
group B  allow  read
group C  allow  write
```

To test if a user has the write right, it would first test weather the user is in group A. If so the right is denied and no more checks are performed. In case the user is not in group A, ReportServer would move to check the second ACE to find that it doesn't say anything about the read right. It then goes to the third ACE and tests if the user is in group C. If so, the right is granted. If the user is not in group C, the right is denied as it has not been explicitly granted. This also means, that any user which is not in group B or C would not have any rights on that particular object.

Also let us stress that ACEs are inspected in the order they are defined. This means if we change the above ACL to

```
group A  deny   read write
group B  allow  read
group C  allow  write
group A  allow  read write
```

then still, any user that is in group A would not be granted read or write rights, since the check would stop after the first ACE.

Generic rights

Generic rights are configured in the Permission management module located in the Administration module. Here, the second column holds an entry for each generic permission, for example, there is an entry for every ReportServer section for which access can be controlled via generic rights. For each of these entries a single Access Control List is filed, the entries included in this list control the access to the respective function.

To manage the existing permissions, or to add new ones use the buttons Add/Remove. The configuration dialogue for this ACE will open by double clicking on an entry to edit its properties.

To edit generic rights (or object related rights, for that matter), beside the respective rights for the Generic Permission Management section, you require the permission "grant rights (g)" each for the object to which you want to assign permissions. In addition, you must own all the rights yourself that you want to assign.

In the following we discuss the various generic permissions.

Administration. The generic right Administration controls the access to the Administration module. If reading right (r) is granted the user concerned can open the Administration module. The access to Administration sub-modules must be released individually. Rights other than the reading right are not queried.

User variables. The user variables right controls the access to the User variables Administration section. Reading right (r) releases the section for reading access. Writing right (w) additionally en- ables to change the configuration.

User management. The generic right User management controls the visibility of the Administration sub-module for user management. Reading right is requested here, other rights will not be used. The visibility of sections in the user tree and in what way they are visible to or modifiable by a user will be controlled by the object rights in the user tree. Object rights will be discussed in detail in the following section.

Report management. Report management access will be controlled by the generic right report management. Reading right (r) determines whether the respective section is visible in the Administration module. The release of rights in general or detail for individual objects in the report tree will be determined by means of the object rights described in the following section.

Dashboard. The generic dashboard right controls the access to the Dashboard module. If reading right is set, the module is visible and can be used in a read only fashion. This means that users can import predefined dashboards but they cannot make any changes or create custom dashboards. If additionally the write permission is granted, users can fully use the dashboard component.

Dashboard (admin). The generic right Dashboard Admin controls the access to the Dadget library in the Administration module. If reading right is granted the section is visible, the rights to the individual sub-trees of the Dadget library will be controlled by object rights.

File system. The access to the File system section in the Admin module will be controlled via the generic right File system. If reading right is granted the section is visible. The effectively assigned access rights are controlled by setting the respective object right.

Data sources. The Data sources generic right controls the access to the Data sources tree in the Admin module. Reading right is queried. The actually granted access rights to single data sources will be controlled by object rights.

Export. The generic right Export determines whether a user is allowed to export objects from a Re- portServer Admin module in the xml format (cf. Export/Import). Executing right (x) is queried.

Generic permission management. Access to Generic permission management is controlled via the right Generic permission management. Reading right here controls the visibility of the respective module in the Admin section. To forward rights to individual modules the Grant rights (g) right must have been additionally assigned for the respective section. Furthermore, the user itself must hold the right that it wants to forward.

Global constants. The Global constants generic right controls the access to the section Global constants in the Administration module. Here the reading right enables to read the defined constants. For editing the global constants, write permissions are additionally required.

Import. The generic right Import grants access to the Import module in the Administration section. The execute (x) right will be checked.

License management. The generic right Import grants access to the License Management module in the Administration section. To view the license information the read (r) permission is necessary. To adapt the license information the execute (x) permission is needed.

ReportServer access. The ReportServer Access right controls who has access to ReportServer, i.e., on login ReportServer checks if the user has the execute (x) right.

SU command. Whether the SU function can be used will be controlled via the generic right SU command. The execute right determines whether the user may use this function.

TeamSpaces. The generic right TeamSpaces controls the permissions awarded for TeamSpaces. Here, the reading right controls the general visibility of the section. The writing right additionally enables the user to create new TeamSpaces. The delete permission allows users to delete TeamSpaces if they either own the TeamSpace or have the administration role for that particular TeamSpace. The specific Administrator permission grants a user administrative rights to all TeamSpaces, which means the user can access all TeamSpaces and has full rights in every TeamSpace. For details on TeamSpaces please refer to the ReportServer user guide.

Terminal. The generic right Terminal controls the access to the terminal window. Here the execute (x) right will be checked.

Scheduler. The generic right Scheduler controls the access to the user components of the Scheduler module. The reading right enables users to open the Scheduler module in order to edit the orders they scheduled before. The execute right enables to schedule reports.

Scheduler Admin View. The generic right Scheduler Admin View enables to access the Scheduler module in the Admin section. Here, reading right enables to principally view the module. The execute right enables to modify scheduling entries.

Object Related Rights

Besides generic rights, rights can be assigned at a fine granular level down to specific objects (such as, reports, data sources, users, etc.). For such object related rights also ACLs are used. The management of object related rights can be found at the object in question, that is for a report, go to the report management module and access the report. The permission management view can be accessed via the tab **Permission management**.

Besides the ACE fields that are also available for generic rights, you can take advantage of the hierarchical structure of objects for object related rights. That is, you can decide if an ACE only applies to the object, whether it is inherited by all its descendants, or whether it applies to the current object and is inherited by all descendants.

Object rights can be assigned in the areas ***Reportmanager***, ***Dadget Library***, ***Datasources***, ***File System***, and ***User management***. In general, the right to read (r) allows users to see (resp. select) the object, the right to write (w) allows to make changes and the right to delete (d) allows users to remove the object. The execute (x) right controls whether a user can execute a report (resp. ReportServer Script). Note that it is not necessary to have read rights to execute. Finally, the grant rights (g) right gives a user the permission to grant rights to other users. Note that users can only grant rights that they themselves hold. Furthermore, they must have read rights for the folk (user, organisational unit or group) to whom they want to grant the right.

Verifying Object related Rights

To check if a user holds an object related right, ReportServer first checks all ACEs that are assigned at the object in question and which apply to that object. If a decision cannot be made, ReportServer goes on to the parent object and there checks all ACEs which are marked to be inherited by descendants. This process is continued until a decision can be made or the root node has been processed. By default rights ar not granted. We consider the following example:

```
Object A
+--- Object B
+--- Object C
```

In the example we consider three objects A, B, and C. Object B is a direct descendant (child) of object A, and object C is a child of B. To test a right on Object C, ReportServer first checks the ACEs defined at C (which also apply to C and in the order they are defined). If no con- clusive decision can be made, ReportServer goes on to obejct B and there checks all ACEs that are marked to be inherited by descendants. If again, no conclusive decision can be made, ReportServer goes on to A. If also here no decision is made, ReportServer will deny the acess.

Virtual Roots

If a user is given read access for a folder (or any other object) but does not have read access for one of the parents of that object, then ReportServer will display the object as a virtual root in the topmost level.

Chapter 4

Data Sources

In ReportServer data sources serve as principal data basis for report objects, i.e. a report draws the data to be displayed from a defined data source. As with most other objects, data sources are maintained in a hierarchical structure. The data source management module is to be found in the Administration module under Data sources. The following object types can be created in the tree:

Folder Serve to structure data sources.

Datasource Here various data sources are optionally available which we will discuss in more detail in the following.

Data sources will be configured in two steps. In Data source management, data sources will be created and the basic settings made. For relational databases, here, for instance, user name, password and access URL are stored. However, the specific configuration per use will be set at the point where the data source will be used (this is mostly with the respective report). Here, for instance, for a relational database the SQL query can be set on which the report is based.

ReportServer supports the following data sources.

4.1 Relational Databases

It is possible to access common relational databases via the data source type "Relational databases". Use the option "database" to control the SQL dialect created by ReportServer. At present, ReportServer supports the following SQL dialects[1]:

- DB2
- H2
- HSQL

[1] Many other dialects can be integrated via ReportServer scripts. For further information we refer to the ReportServe scripting guide.

- Informix
- MSSQL
- MySQL
- Postgre SQL
- Oracle

Make sure to integrate the respective JDBC database driver prior to use. (For more detailed information refer to the database driver description.) Information on how to set user name and password as well as the URL is given in the data- base manual. In the following example we will demonstrate how to configure a MySQL data source.

After the initial installation, MySQL can usually be started by entering user name "root" and password "root". A JDBC URL could be as follows:

```
jdbc:mysql://127.0.0.1:3306/ClassicModels
```

After having transferred the data, and you wish to test whether ReportServer can establish a database connection, apply the button **Test connection**. Be aware to always test the saved connection.

The query will actually be configured when selecting the data source, for instance, if you want to create a dynamic list on the basis of this data source.

4.2 Storage of Database Passwords

In the ReportServer development we particularly emphasized the safety of the system to the greatest possible extent. One of the main issues was to store sensitive data as securely as possible. Therefore, the data source settings are of special importance as they provide the potential to access your data warehouse. To store passwords as securely as possible, we follow a two-way strategy. Firstly, database passwords should never be transferred to the client (to your web browser), but only be used to establish database connections. This results in the data source password field always being empty upon reloading the form. However, you may safely change the data source, the password will only be reset when you add an entry to the password field. The second safety measure is that ReportServer database passwords will be encrypted when saving. For further information, please refer to the configuration instructions.

4.3 The Internal Data Base

Non-relational database data (such as CSV lists) will be cached in an internal H2 database to be able to quickly and comfortably access them. For this purpose, H2 stores some files on your local hard disk. By default this location points to the folder `dbtmp` in your ReportServer installation. For setting the location as well as other properties please refer to the configuration instructions. You may access the internal database in just the same way as you do with all other data sources. To

do so, create a new data source of type “relational database”, set the database type to H2 and configure as follows.

Username:	sa
Password:	Refer to the configuration file `reportserver.properties`. If you activated the encryption of the internal data- base, the password is "encryption password user password". If encryption is deactivated, the password is the user password set in reportserver.properties.
URL:	`jdbc:h2:VERZEICHNIS/rsdb;CIPHER=AES` (however, omit “;CIPHER=AES” if database encryption was deactivated).

Demo Data

If desired, you can load the demo data on the fictitious organization “1 to 87” as soon as you have started ReportServer. They will then become available in the internal H2 database. For further information see Section 1 on page 4 as well as the configuration guide.

You can easily configure a data source with access to the internal demo data. To do this, create a new H2 data source with the following settings:

Username:	demo
Password:	demo
URL:	`jdbc:h2:DIRECTORY/rsdemodb`

Be aware that the demo database will never be encrypted, therefore, there is no need to configure any encryption in the URL.

4.4 Datasource Pool

Data source connections can be provided in a pool in ReportServer. This can clearly increase the performance and enables to better control the individual connections. If pooling is activated, ReportServer keeps a pre-defined number of connections open per relational database integrated. ReportServer will recycle these connections by user to save the costs incurred for setting up the connection. In the configuration guide you will find detailed information on how to pool databases exactly and which settings to enter.

4.5 CSV Lists

Apart from relational databases, ReportServer can provide data in form of CSV files (Comma-Separated Values; further information on CSV you will find, for example, at `http://en.wikipedia.org/wiki/Comma-separated_values`) as data basis for reports. To work with a **CSV List**, create one. To set the format of your CSV file, use the fields

Quotes:	Delimiting characters for an individual data record.
Separator:	Separating character between data records

Use the Connector setting to define the location of the CSV data. The following connectors are presently supported:

Text-Connector:	Allows to directly enter data in a text field at the data source.
Argument-Connector:	Allows to directly enter data in a text field when selecting the data source. This enables, for instance, to easily simulate static lists for report parameters (refer to the section on datasource parameters).
URL-Connector:	Allows to load data by using an URL. Please observe that you have the option to load the data from the internal ReportServer file system (refer here to the Section File Server).

Database Cache

As described before, prior to their use CSV data will be loaded to an internal database. This may take some time if you load a larger data volume for the first time. Therefore, it is quite often reasonable not to continuously load the data. Use the **Cache database** setting to determine after how many minutes the data should be reloaded from the source to the internal database. If you set −1, the data will be loaded only one time. If you set 0, the data will be reloaded every time you use them.

If you wish to load the data manually (because they were changed) it is sufficient to simply save the data source again. After every saving process, ReportServer will initiate that the data will be removed from the internal cache and reloaded again.

Please observe that in case of the **Argument Connector**, ReportServer will ignore the cache setting, so the data will not be cached.

Configuration at the Object

Similar to relational databases, you can make additional settings for CSV data sources at the location where the data source will be selected (e.g. at the report or parameter). As already explained, CSV data will be buffered to an internal database. The query type to be used is

```
SELECT * FROM TMP_TABLENAME
```

where `TMP_TABLENAME` is a temporary table name assigned by ReportServer. Using the **Query Wrapper** setting, now you can extend the query created automatically. Here, use the syntax for parameters (see section on report parameters). The following replacements will be available to you:

TMP_TABLENAME:	The name of the table
query:	The basic query.

For instance, by using the following query you could limit the data volume to all those data records where the attribute REGION has value 3.

```
SELECT * FROM ($!{TMP_TABLENAME}) WHERE REGION = "3"
```

Please observe that any CSV data will generally be treated as if it were of type string. In addition, we want to point out that if you use replacements, you need to use $!{} instead of ${} as replacements need to be directly written into the query (for further information on the replacement syntax refer to Section 6.3 Working with Parameters on page 46).

4.6 Script Data Sources

If you wish to load data which are in a format that has not been supported so far, or if you wish to perform complex pre-processing of data, it is advisable to use script data sources. Script data sources provide data by running a ReportServer script and, therefore, they can be applied very flexibly. Similar to CSV lists, the result of a script data source will first be buffered in the internal database. Here as well, you will have the option to define in the **Cache database** setting how often the data will be reloaded.

Script data sources run a ReportServer script whenever the datasource is accessed. This script will be filed with the data source and, if selected, it can be parameterized with a report (i.e. parameters can be transferred to the script). The return value of the script must be an object of type RSTableModel (included in the package `net.datenwerke.rs.base.service.reportengines.table.output.object`). In the following we will give a simple example script which builds up a static table consisting of 3 columns.

```
import net.datenwerke.rs.base.service.reportengines.table.output.object.*;

def definition = new TableDefinition(['a','b','c'],
         [Integer.class,Integer.class,Integer. class]);

def model = new RSTableModel(definition);
model.addDataRow(1,2,3);
model.addDataRow(4,5,6);
model.addDataRow(7,8,9);

return model;
```

Configuration at the Object

You can further reduce the data volume by using **Query wrapper** in the same way as you proceeded with CSV data sources. In addition, you can pass arguments to the script.

4.7 BIRT Report Data Source

The BIRT report engine enables to define data records within BIRT Reports. For instance, they can be used for feeding parameters. By using BIRT Report data sources, you can access these data in ReportServer.

As the most frequent application case for BIRT Report data sources will surely be the reading out of parameters, the BIRT data source will directly be configured at the report. This means you only have to create a data source in the data source tree, any further configuration will be entered at the location where it is used. When using the data source, you eventually have to select the

respective BIRT report and enter the name of the data set. Please observe that BIRT provides the option to access so-called "data sets" as well as parameter data. Depending on the origin of the data, you have to set the respective type.

In the same way as you proceeded with CSV and script data sources, you can modify the query by using the **Query wrapper** configuration.

4.8 Mondrian Datasource

Mondrian is a java-based OLAP engine (Online Analytical Processing) developed by Pentaho (http://mondrian.pentaho.com) allowing you to perform multi-dimensional analysis on your data. For an introduction to OLAP and Mondrian we refer to the Mondrian documentation available online at http://mondrian.pentaho.com/documentation and we assume basic familiarity with Mondrian and OLAP for the following discussion.

Mondrian datasources are used to define so called Mondrian schemas which can then be used by the Mondrian backed Saiku reporting format within ReportServer (see Chapter 6.7 Saiku / Mondrian Reports). The main configuration options of Mondrian datasources are

Properties: The properties define the connection to the underlying relational database.

Schema: The schema describes the data warehouse semantics.

Note: in order to configure your Mondrian instance you can create a "mondrian.properties" file in your WEB-INF/classes directory or modify it if it already exists. In this file you can set the Mondrian properties needed, e.g. "mondrian.rolap.queryTimeout=3" (without the quotes). This property gives you an error if your query runs more than 3 seconds. Refer to https://mondrian.pentaho.com/documentation/configuration.php for all Mondrian configuration options.

When you create a new Mondrian datasource, ReportServer already specifies an example definition for the properties pointing at a MySQL database called ***foodmart***[2].

```
type=OLAP
name=Foodmart
driver=mondrian.olap4j.MondrianOlap4jDriver
location=jdbc:mondrian:Jdbc=jdbc:mysql://localhost/foodmart
jdbcDrivers=com.mysql.jdbc.Driver
jdbcUser=
jdbcPassword=
```

A Mondrian schema defines multi-dimensional data warehouses on top of relational databases that are usually assumed to be managed in a star schema like form. Following is a simple schema definition based on the foodmart demo data and taken from the Mondrian documentation[3]. The schema consists of a single cube called ***Sales*** which is made up of two dimensions (Gender and Time) and four measures.

```
<Schema>
```

[2] The foodmart is Mondrian's demo dataset that comes with various cubes demonstrating the OLAP capabilities of Mondrian.

[3] http://mondrian.pentaho.com/documentation/schema.php

```
<Cube name="Sales">
  <Table name="sales_fact_1997"/>
    <Dimension name="Gender" foreignKey="customer_id">
      <Hierarchy hasAll="true" allMemberName="All Genders" primaryKey="customer_id">
        <Table name="customer"/>
        <Level name="Gender" column="gender" uniqueMembers="true"/>
      </Hierarchy>
    </Dimension>
    <Dimension name="Time" foreignKey="time_id">
      <Hierarchy hasAll="false" primaryKey="time_id">
        <Table name="time_by_day"/>
        <Level name="Year" column="the_year" type="Numeric" uniqueMembers="true"/>
        <Level name="Quarter" column="quarter" uniqueMembers="false"/>
        <Level name="Month" column="month_of_year" type="Numeric" uniqueMembers="false"/>
      </Hierarchy>
    </Dimension>
    <Measure name="Unit Sales" column="unit_sales" aggregator="sum" formatString="#,###"/>
    <Measure name="Store Sales" column="store_sales" aggregator="sum" formatString="#,###.##"/>
    <Measure name="Store Cost" column="store_cost" aggregator="sum" formatString="#,###.00"/>
    <CalculatedMember name="Profit" dimension="Measures" formula="[Measures].[Store Sales] - [Measures ↙
        ↳ ].[Store Cost]">
      <CalculatedMemberProperty name="FORMAT_STRING" value="$#,##0.00"/>
    </CalculatedMember>
  </Cube>
</Schema>
```

Schema and **properties** are sufficient for the Mondrian server to query the defined ware house using queries written in the MDX language, an SQL like query language first introduced by Microsoft in 1997 (see, e.g., http://en.wikipedia.org/wiki/Multidimensional_Expressions). In order to use Mondrian within ReportServer you will need to create so called Saiku reports (see Chapter 6.7 Saiku / Mondrian Reports) which provide a beautiful user interface to access a data specified in a cube in a Pivot like fashion.

4.9 Data Source Bundle

The **datasource bundle** allows you to use the same report for different datasources an have the users select which database to use. To use this feature you first have to define sets or bundles of similar datasources from which a selection can be made. For this, in the datasource manager create a new datasource of type ***database bundle***. After you configured the bundle, instead of using a specific datasource, you use the bundle as the datasource for your report.

The database bundle needs two options to be configured: The **Key Provider** defines where the key used for the lookup of the actual datasource (the datasource that is selected for a single execution) is taken from. There are two key providers:

Login Key Provider	The login dialog contains a dropdown list that allows the user to select the key the bundle uses to lookup the assigned datasource. For this to work properly you have to configure the available values in the /etc/datasources/databasebundle.cf file. Please refer to the Configuration guide for additional information.
Report Parameter Provider	One of the report parameters is used to provide the key the bundle uses to lookup the assigned datasource. You also have to enter the parameter-key of the parameter that will be used.

The **Mapping provider** defines how a datasource gets selected from the key. There are three providers to choose from:

Static Mapping	The static mapping allows you to manualy specify a map of keys and associated datasources.
Auto: Ds-Node (by ID)	Instead of manually adding all the datasources for the bundle to your mapping this mapping provider automatically chooses the datasource that has an id matching the provided key. The mapping table is used to specify the search path. You can add single datasources or whole folders to your mapping. If you add a folder to your mapping, the datasources must be direct children of the given folder. The key column is ignored in this configuration.
Auto: Ds-Node (by Name)	Similar to the previous strategy this mapping provider automatically chooses from a set of datasource without explicitly defining a key for each datasource. Instead of using the id to find a matching datasource, the datasources name is used. If your bundle contains multiple datasources with the same name, the result is undefined.

Tip. You can also use a datasource bundle as the datasource for a database parameter. If you use the Report Parameter Key Provider you have to make the parameter that uses the bundle dependent on the paramter that is used as the key source.

4.10 Configuration of a Standard Data Source

For a quick configuration of reports, ReportServer allows to define a default data source. It can then be configured by a single click at the locations where data sources can be selected. The default data source can be set up by using the configuration file etc/datasources/data- sources.cf (in the internal file system, refer also to the configuration guide). In the following please find a sample configuration selecting the data source by name. It can optionally be selected by its ID.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <datasource>
    <defaultDatasourceName>Demodaten</defaultDatasourceName>
  </datasource>
</configuration>
```

Please observe to run the terminal command `config reload` when configuration files have been modified. For further information see the ReportServer configuration guide as well as Chapter 12 Terminal.

Chapter 5

File System

The ReportServer File system meets a variety of functions. First, it serves to file various files and resources so as to use them, for instance, as a basis for data sources (e.g. CSV files), or as resources for reports (e.g. images). In addition, a major part of ReportServer settings (e.g. the mail server configuration) can be performed with configuration files that you will find in the file system's sub-directory `etc`. Furthermore, so-called ReportServer scripts can be filed in the file system that cover a great number of varying cases of use. These scripts can serve as a basis for data sources (script data sources) or for reports (script reports). They can, however, also provide additional functionality, or be used to perform maintenance tasks.

The file system has a hierarchical structure, just like file systems of common operating systems. You will find the file system in the Administration module under **File system**. The following object types can be created in the File system tree:

Folder:	Serve to structure files
File:	Any file (e.g. image, text file, CSV, etc.)

Both objects share the usual attributes **name** and **description**. In addition, folders can be configured for web access. This means that objects in these folders (including subfolders) can be accessed by URL without prior authentication (i.e., permissions are not checked for accessing these objects).

After having selected a folder, in the right window above the **Apply** button you will find the area marked **Drop files here**. Here you can quickly copy files from your local file system to the ReportServer File system. Drag files, for instance, from your Windows Explorer or Mac OS Finder to the marked area. As soon as the files have been uploaded they will be dis- played in the file tree.

If you create files in the file tree by using the context menu (right click on a folder), or you select already existing files, you can enter a name and description as well as the Mime-type for the file. Additionally, you will be given the current file size as well as a download link. By means of a form, you can upload a new file to the server and update the currently stored file. Text files can directly be edited in ReportServer. In the tree, click on the text file and then select the tab **Edit file** (next to Properties).

5.1 Configuration Files

ReportServer configuration files are located in the `etc` directory, following the Unix operating system structure. Configuration files are in XML format and are marked with the file extension `.cf` by default. A configuration file could be as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <report>
    <id>41390</id>
  </report>
</configuration>
```

For further information on the individual configuration options refer to the ReportServer configuration guide.

5.2 Filing of Scripts

By default ReportServer scripts are filed in the `bin` directory (this location can be changed; for further information see the ReportServer configuration guide) with the file extension `.rs` or `.groovy`. ReportServer scripts are written in the language Groovy (`http://groovy.codehaus.org/`). They are a powerful tool to enhance ReportServer, to perform administrative tasks, or to generate complex reports. Please observe that a user requires the **Execute** right to run scripts. The explanation of ReportServer scripts in more detail would go beyond the scope of this manual. You will find a brief introduction in Chapter 13 ReportServer Scripting. For a detailed introduction to ReportServer scripts we refer to the separate scripting guide.

5.3 Accessing Resources by URL

You can access files in the ReportServer File system by URL. This facilitates the embedding of images in reports. The appurtenant URL is:

`http://SERVER/APPLICATIONFOLDER/reportserver/fileServerAccess?id=XX`

Here, `SERVER` gives the server address (e.g. `demo.raas.datenwerke.net`), and `APPLICATIONFOLDER` the installation path in your application server (e.g. Tomcat). The default setting is "reportserver". By using the property ID you can access certain files. In addition, you can use the following URL attributes:

id	ID of a file
path	Path leading to a file, e.g. `resources/images/img.jpg`
thumbnail	If the file is a MimeType image/png or image/jpeg, the image is automatically scaleable. The thumbnail option activates the scaling mode for ReportServer.
twidth	Gives the width of the resulting preview image.

Please observe that reading rights are required for a file by default (refer to Chapter 3 User and Permission Management) to be able to access it by URL. This indicates in particular that a user

must be logged in to the system. To fully share files you may place them in a folder where a tick is set at **Share folder for web access**.

Chapter 6

Report Management

Reporting is a highly dynamic field. Data need to be consolidated, prepared and evaluated at increasingly shorter intervals. Due to the communication overhead, these dynamics can, however, only be met to a limited extent with the typical approach in reporting where a report is first specified by the specialist department and then implemented by the IT department. ReportServer is designed to get the responsibility just as well as the opportunities to create new, relevant information to the greatest possible extent back to whom it concerns and where it is needed, but also where the knowledge about how to generate data is available: To the specialists respectively to the ultimate user.

ReportServer first distinguishes between (base) reports and variants. Base reports are comparable to a shell. They define the basic data structure and, depending on the format, the design how to present these data. In addition, they provide the users with the options that enable them to adapt this data basis to their requirements. Depending on the report format, various options are here available. However, variants represent reports which are fully configured by the user.They include the necessary settings to execute the report or the schedule it. Users can save their configurations—the variants—to assure prompt access to their data.

All report types supported by ReportServer have in common that they are configurable by setting the so-called parameters. For instance, a parameter may be a simple text field where the user can enter an invoice number, or a date range to limit a time span. The parameters applicable for a report are invariably specified by the report designer/administrator.The users can select from the parameter values and save them as a variant. Depending on the report type, the ultimate user is given additional options to control the output of the report. We will discuss them more closely in the following depiction of the single report types. Naturally, you will find a detailed description in the user manual. The ultimate goal is to empower the users to gather the data required for their task independently.

Of course, it must be ensured to document the reports in an audit-proof manner if they can be adjusted by a user after a possible acceptance in the data warehouse. ReportServer here supports you in two ways. First, ReportServer automatically creates detailed documentation for each report/variant outlining all user defined settings. Then, history objects will be created for any changes to objects. They enable to trace the type and time of a modification executed on an object.

6.1 Fundamentals

As with users and data sources, ReportServer manages reports in a hierarchical tree structure. You will find it in the Administration modules under **Report management**. Report management exclusively provides the administrative management of reports. By using TeamSpaces the users may create their own view to report objects released to them. For further information on TeamSpaces refer to the User manual.

As you are used to you may structure objects in folders. Apart from folders, there will be one object per report type as well as variant objects. Variant objects will not be created in the tree itself but automatically by ReportServer as soon as a user creates a new variant for a report. ReportServer provides you with the following report types:

Dynamic list

The dynamic list shifts the major part of report logics to the ultimate user who can compile the data relevant for it individually. Here users can draw from simple filters and complex aggregations up to computed fields. In this respect, the dynamic list can be regarded as ad-hoc reporting. The user may, of course, save all settings in a variant. The dynamic list is designed to be highly performant when dealing with large data volumes. The dynamic list outputs data primarily as a table which can be exported to Microsoft Excel just as well as to PDF, HTML or CSV. In addition, by using templates data can directly be uploaded to pre-defined Excel spreadsheets, or transferred to any XML dialects. We will look at the dynamic list from the administrator's viewpoint in section **The Dynamic List**. The description of the dynamic list from the user's viewpoint will be given in the User manual. The dynamic list is discussed in detail in Section 6.2.

Graphical Reports

The dynamic list supplies data in raw format, and therefore it is ideally suited for daily control. To generate graphically sophisticated analyses, ReportServer supports reports in the JasperReports and Eclipse BIRT report formats. These popular and open libraries enable to create reports with elaborate graphics. We often refer to these as **graphical reports**. In addition to the open formats BIRT and Jasper, ReportServer also provides support for the commercial SAP Crystal Reports engine which similarly allows you to create pixel perfect reporting.

JasperReports

JasperReports Library (`http://community.jaspersoft.com/`) designed by JasperSoft is a powerful report engine to generate graphical reports. Reports are defined in an XML dialect which will be translated to Java Source Code for report execution. Apart from the function to directly generate XML sources, JasperSoft provides the report designer IReports. It helps to create reports in the style of WYSIWYG applications (what you see is what you get). Jas- perReports are particularly suited to output reports in the PDF format (e.g. for pixel precise printing). However, Jasper reports can be provided in other formats such as HTML or RTF. JasperReports are discussed in detail in Section 6.4.

Eclipse BIRT

Eclipse BIRT (http://www.eclipse.org/birt) is the second open report engine next to JasperReports. Actuate (http://www.birt-exchange.com/be/home/) takes the lead in further developing Eclipse BIRT which offers functionalities comparable to JasperReports. BIRT is based on the Eclipse platform (http://www.eclipse.org/) and includes a comprehensive designer for visualizing reports. Just as well as with JasperReports, Eclipse BIRT defines reports in an own XML dialect which transfers to an executable Java Code at report runtime. The primary output format for BIRT reports is PDF as well. BIRT is discussed in detail in Section 6.5.

Crystal Reports

Crystal Reports is a commercial reporting engine developed by SAP AG (http://www.crystalreports.com/) and hence not directly part of ReportServer. ReportServer, however, comes with everything you need to run reports generated with Crystal Reports given that you have a Crystal license allowing you to use SAP Crystal Reports for Java runtime components (or short, the Java Reporting Component JRC). Similarly to Jasper and Birt the primary output format of Crystal reports is PDF.

Versions of the Report Engines

With every new release of ReportServer, the libraries in use will also be updated so that normally the latest JasperReports and Eclipse BIRT versions will be integrated. The current versions are given in the license documentation which is part of the download package. As JasperReports are designed as a simple library, you can easily exchange it yourself by a current or an older version. To do this, copy the corresponding `.jar` files to the ReportServer lib directory. Further information on this you will also find in the ReportServer Configuration guide. Unfortunately, Eclipse BIRT is not as easy to handle in this respect because it requires multiple libraries in specific versions. Therefore, we advise you not to upgrade the BIRT engine on your own without any support.

Saiku Reports – Multi-dimensional Reporting

Saiku reports allow you to access Mondrian datasources (see Section 4.8). The user interface is provided by Saiku (http://meteorite.bi/saiku) who created beautiful OLAP UI that we adopted in ReportServer. Saiku reports are the preferred way if you want to access multi-dimensional data that is organized with Mondrian.

JXLS Reports – Excel Reporting

JXLS (http://jxls.sourceforge.net/) is a template engine for Microsoft Excel. ReportServer allows to use JXLS templates, for example, with dynamic lists such that users can directly insert their data in a predefined Excel sheet (for further information have a look at the ReportServer User guide). Besides being available as a template engine for ReportServer's dynamic list, JXLS is also available as a first class report object. In this form it provides further reporting capabilities, as you can directly use SQL queries within your Excel templates.

Script Reports

Beside the dynamic list there is another native ReportServer report type, the script report. Script reports are written in Groovy and offer full Java VM flexibility and functionality. They are primarily

used to generate dynamic analyses allowing user interaction. But they can also access the ReportServer object model, and therefore they are particularly suited for the reporting of warehouse metadata as well as of ReportServer itself. An example for a meta report is a documentation report which generates an up to date documentation for all kinds of reports.

In the following sections we will provide you with in-depth information on the four different report types. Even if you intend to primarily work with the graphical report engines (Jasper or BIRT) we recommend you to read the Dynamic list section as we will explain here some of the principle techniques such as working with parameters. All of the following sections show a similar structure. First, we describe the principle techniques and concepts on which the report engine is based. This is not necessarily required for the creation and configuration of reports, and may be skipped on first-time reading. In the next step, we will explain the single configuration options in detail. For Jasper and BIRT we will additionally elaborate on the interoperability between ReportServer and the report designers offered by the manufacturers.

Grid Editor Reports

The grid editor component allows users to change data within database tables. The grid editor is backed by ReportServer scripts to define what data is loaded and how changed data is to be stored. In that way it is a very flexible component when you want to provide a simple data editor for users.

6.2 The Dynamic List

The dynamic list is very easy to configure as compared to the graphical report engines BIRT and Jasper. Basically, the administrator defines a table (which might be very concise at times) with the basis data. From this table the users can then independently arrange their data of relevance. In many cases the definition of a dynamic list only requires a single SQL query (provided the data basis is stored in a relational database). By setting parameters you can provide the users with predefined filter and configuration options.

To obtain highest possible efficiency, ReportServer relocates all filtering steps to the reference database, if possible. If a data source other than a relational database has been selected for your report, the data will be buffered in an internal database as described in section "„Data Sources“". Basically, the execution of a dynamic list report is as follows. Proceeding on a basic SQL statement, ReportServer constructs a complex SQL query considering all filter options set by the user. This SQL statement will either be predefined by you if you have chosen a relational database as your data source, or will automatically be generated by ReportServer if the data are buffered in an internal database. Resulting from this, there will be a few consequences with regard to the formulation of your basis SQL statement that we want to discuss in the following.

If you have chosen the demo data source (see Section 4.3), your base query could be as follows:

```
SELECT * FROM T_AGG_CUSTOMER
```

It would provide all processed customer data of the model company “1 to 87” (the imaginary company behind our demo data) as a data basis for the dynamic list. The query assembled by ReportServer could then be as follows (depending on the configuration of the variant):

```
SELECT * FROM
```

```
    (SELECT
        xx__rs_col_0,
            xx__rs_col_1,
            xx__rs_col_2,
            xx__rs_col_3,
            xx__rs_col_4,
            xx__rs_col_5
    FROM
        (SELECT
        *
    FROM
        (SELECT
        CUS_CONTACTFIRSTNAME AS xx__rs_col_0,
            CUS_CONTACTLASTNAME AS xx__rs_col_1,
            CUS_CUSTOMERNUMBER AS xx__rs_col_2,
            CUS_PHONE AS xx__rs_col_3,
            CUS_CITY AS xx__rs_col_4,
            CUS_POSTALCODE AS xx__rs_col_5
    FROM
        (SELECT
        *
    FROM
        T_AGG_CUSTOMER) colQry) filterQry
    WHERE
        xx__rs_col_4 IN ('Barcelona' , 'Auckland', 'Bern')) aliasQry) limitQry
LIMIT 50 OFFSET 0
```

Please consider that ReportServer will adapt the generated SQL source text to the database used. In the given example the syntax `LIMIT 50 OFFSET 0` is not generally applicable for all databases.

In this example the user has chosen the columns

- CUS_CONTACTFIRSTNAME
- CUS_CONTACTLASTNAME
- CUS_CUSTOMERNUMBER
- CUS_PHONE
- CUS_CITY
- CUS_POSTALCODE

and defined a simple inclusion filter on the CUS_CITY column.

You see here how ReportServer builds the SQL statement around the basic query `SELECT * FROM T_AGG_CUSTOMER`. As a consequence for the formulation of the basic query, the syntax has to allow to be used as inner `SELECT` statement. Here only the line limiting instructions such

as LIMIT and OFFSET basically provide a problem. Please observe that depending on the database dialect used the formulation has to be adapted. Surely, some attention has also to be paid to more complex basic queries. For example, if you wish to specify an aggregation (this is usually not required as users can aggregate data by themselves), your query could be as follows:

```
SELECT SUM(CUS_CREDITLIMIT) FROM T_AGG_CUSTOMER GROUP BY OFF_CITY
```

Here the sum of the customers' credit limit per location is defined. When executing the report, ReportServer would translate the above query as follows (here an additional filter was added on values greater than 4.000.000).

```
SELECT * FROM
    (SELECT
        xx__rs_col_0
    FROM
        (SELECT
        *
    FROM
        (SELECT
        SUM(CUS_CREDITLIMIT) AS xx__rs_col_0
    FROM
        (SELECT
        SUM(CUS_CREDITLIMIT)
    FROM
        T_AGG_CUSTOMER
    GROUP BY OFF_CITY) colQry) filterQry
    WHERE
        xx__rs_col_0 >= 4000000) aliasQry) limitQry
LIMIT 50 OFFSET 0
```

If you manually run the statements on a database you will see that the original statement is valid whereas the one built by ReportServer will not be valid at this place. In line 4 ReportServer assigns to column SUM(CUS_CREDITLIMIT) a unique (internal) name. However, the database interprets this statement as a further summation. This means that an ***admissible*** attribute name should already have been assigned to the aggregation in the basic query. In this case the basic query should have been as follows:

```
SELECT SUM(CUS_CREDITLIMIT) AS SUMME FROM T_AGG_CUSTOMER GROUP BY OFF_CITY
```

ReportServer would then correctly translate the following query:

```
SELECT * FROM
    (SELECT
        xx__rs_col_0
    FROM
        (SELECT
        *
    FROM
        (SELECT
        SUMME AS xx__rs_col_0
    FROM
        (SELECT
        SUM(CUS_CREDITLIMIT) AS SUMME
```

```
    FROM
        T_AGG_CUSTOMER
    GROUP BY OFF_CITY) colQry) filterQry
    WHERE
        xx__rs_col_0 >= 4000000) aliasQry) limitQry
LIMIT 50 OFFSET 0
```

Variants of the Dynamic List

Using dynamic lists, users have far reaching options to design the report according to their needs. In a first step, the users select a sub-set from the available report attributes (columns). This sub-set forms the basis for their report. In the following steps, they can then specifically reduce the data basis to the number they actually need for their analysis by setting various filter options. A detailed description of all options would go beyond the scope of this instruction. Therefore, we make reference to the User manual which details all options how to adapt a report by the user.

Output Formats of the Dynamic List

The underlying output format of the dynamic list is tabular.Therefore, the data can be exported to Excel or CSV (comma separated value, http://en.wikipedia.org/wiki/Comma-separated_values). In addition, when using dynamic lists ReportServer provides the possibility to export to PDF and HTML.

Beside outputting the data in a simple tabular structure, users can upload them to predefined Excel spreadsheets by means of templates (JXLS templates), transform them in any text format (velocity templates), or issue them in any XML dialect (XLS templates). You will find a short introduction to the various template languages in the User manual.

Configuring the Dynamic List

All reports have in common that you may assign a name and a description to them. Note that these will be indexed for ReportServer's search engine. In addition, you can assign a key to reports. This key represents a unique (plain text) name for referencing the report, for instance, in URLs.

To define the data basis select a data source and configure it (configuration depends on the data source type, refer to chapter 4). Normally, you will use data from relational databases (you can use the demo data source as an example which refers to the internal H2 database). To complete the configuration, you have to enter an SQL query. In the following we will base our description on a relational database serving as a data source and only refer to other data source types if the configuration appears to be fundamentally different.

Based on the demo data source, a report could provide, for instance, all data of the table `T_AGG_CUSTOMER` (i.e., all processed data concerning the customers of the company "1 to 87") as described above. Here you use the query

```
SELECT * FROM T_AGG_CUSTOMER
```

Once the data source is specified, the dynamic list is executable. To start executing it, double click on the respective node in the tree. You will find a detailed explanation how to use the dynamic list from the viewpoint of a user in the User manual.

By actuating the **Execute** button in the tool bar of the data source configuration you can easily test the currently entered query.

The Metadata Data Source

The optional metadata data source serves to further define the report's base data fields. If there is no metadata data source given, the users will only see the technical database name each when selecting attributes/columns. By using the metadata data source it is possible to enter an additional plain text name and description per attribute. ReportServer expects the data source output to consist of three columns where the first one shows the technical column name, the second one gives the appurtenant plain text name and the third one the appurtenant description.

A metadata example will also be supplied with the demo data. The appurtenant query is:

```
SELECT column_name, default_alias, description FROM METADATA
```

The result of this call, for instance, will look as follows:

COLUMN_NAME	DEFAULT_ALIAS	DESCRIPTION
addressLine1		First line of the address
addressLine2		Second line of the address
amount		Amount of the payment effected
buyPrice		Purchase price
checkNumber		Payment reference number

Customizing Dynamic Lists via Report Properties

There are several report properties (see Section 6.12) that can influence how the dynamic list. By default, the preview view, as well as filter views count the number of rows in the results. When working with very large database tables, this can become a performance bottleneck and you can thus disable this behavior. Furthermore, you can control whether filters are, by default, in ***linked*** mode, or not. The ReportServer standard is to have filters in linked mode, as this is the more intuitive when using a dynamic list: filters will only show those results which are still valid results given all the other filters. As this again may be a performance bottleneck, you can disable this behavior via a report property.

For an overview over all report properties see Section 6.12.

Theming HTML and PDF Output

ReportServer Enterprise Edition allows to customize the PDF and HTML exports of dynamic lists via the configuration files:

`etc/dynamiclists/htmlexport.cf`	Configuration file customizing the HTML export
`etc/dynamiclists/pdfexport.cf`	Configuration file customizing the PDF export

Following is a sample configuration for the HTML export:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
   <htmlexport>
      <!--
        <title>Some title</title>
        <head>Some additional content for the header</head>
        <script>Some Javascript</script>
       -->
      <style><![CDATA[
         @page {
                size: A4 landscape;
                @top-left {
                    content: "${report.name}";
                    font-family: DejaVu Sans, Sans-Serif;
                    font-size: 8pt;
                 }
                @top-right {
                    content: "${now}";
                    font-family: DejaVu Sans, Sans-Serif;
                    font-size: 8pt;
                 }
                @bottom-right {
                    content: "${page} " counter(page) " ${of} " counter(pages);
                     font-family: DejaVu Sans, Sans-Serif;
                    font-size: 8pt;
                }
           }
           ]]>
      </style>
      <pre><![CDATA[
<div class="wrap">
<div class="header">
<span class="logo">
<img src="" alt="logo"></img>
</span>
<div class="reportdata">
<span class="name">${report.name}</span>
<span class="date">${now}</span>
</div>
<div class="clear"></div>
</div>]]>
      </pre>
      <post><![CDATA[</div>]]></post>
   </htmlexport>
</configuration>
```

The configuration consists of six high level elements

title	The page title
head	Additional content to go in the HTML head element
script	Script elements
style	Custom CSS
pre	Custom HTML going before the table export
post	Custom HTML going after the exported table

Using the Formula Language

For all templates (i.e., pre, post, etc.) you have access to a ReportServer expression language object (see Appendix A) with some predefined replacements:

columncount	The number of columns
report	The current report
user	The current user
now	The current date as a formatted string
page	A localized text for ***page***
of	A localized text for ***of***
parameterMap	A map containing parameters
parameterSet	The parameterSet object

One should be careful when working with parameters within the template, as not compiling templates may cause unexpected errors.

6.3 Working with Parameters

By setting parameters you can specify additional configuration options for the users. First, we want to discuss the basics of parameters based on a simple example. Then we will look more closely to the various parameter types.

In the following we will again use the supplied demo data and set up a report on the orders of the company "1 to 87". Here we want to enable the users of the report to restrict the data basis to certain customers. To do this, we will create a new dynamic list, select again the demo data source (refer to Chapter 4), and first apply the following basic query:

```
SELECT * FROM T_AGG_ORDER
```

Run the report in its actual state to get better acquainted with the demo data. In the following we would like to introduce a parameter which enables the users to filter the report by customer numbers. To do this, we return to **Report management**, select the report and switch to the parameter tab (at the bottom). From the tool bar we add a new **data source parameter**. The parameter properties will open by double clicking on the icon of the parameter. We will assign the following properties:

Name:	Customer
Description:	Selection of the customer to be displayed only.
Key	P_CUSTNUM

Now we switch to the **Specific properties** of the parameter. Here we enter/activate the settings valid for the parameter type (data source parameter). Data source parameters draw their data from a data source. Here we enter the demo data source again. ReportServer expects a result giving two columns: the value column (this value can later be used in the query) and the display column (this column will be shown to the user on selection). We place the following query.

```
SELECT DISTINCT
    OR_CUSTOMERNUMBER, CUS_CUSTOMERNAME
FROM
    T_AGG_ORDER
ORDER BY 2
```

You may keep the default values of the other settings. Apply the settings by clicking on **Apply**. If you now run the report (double click in the tree) you will discover that the aspect displayed first is the page **Report parameters**, and no longer **Lists configuration**. To set the parameter data, double click into the corresponding data grid.

Now we added a parameter, and it can be configured by the users, but so far this parameter had no effect on the underlying data volume. We still have to embed it in the base query of the report. To do this, in Report management we return to the report settings and modify the query as follows:

```
SELECT * FROM T_AGG_ORDER WHERE $X{IN, OR_CUSTOMERNUMBER, P_CUSTNUM}
```

ReportServer interprets the syntax `$X{IN, OR_CUSTOMERNUMBER, P_CUSTNUM}` as an IN-Clause where the field `OR_CUSTOMERNUMBER` must correspond to a value selected in the parameter `P_CUSTNUM` ↙ ↳ (key of the parameter). Translated to SQL, the query would be as follows:

```
SELECT
    *
FROM
    T_AGG_ORDER
WHERE OR_CUSTOMERNUMBER IN ('WERT_1','WERT_2','WERT_3')
```

Now, if you execute the report again, you will discover that when selecting the parameter it will have the desired effect on the data volume. Please observe that if no parameter is selected, this will by default translate in “All values are valid”.

If you are familiar with the JasperReports report engine, you will find out that you can use the parameter syntax applied there for ReportServer dynamic list as well. We will discuss the syntax more closely at a later point.

The Parameter Types

In the following we will first introduce the parameters supported by ReportServer individually and then look at them in detail. ReportServer distinguishes two types of parameters. True ***parameters***

are the ones that enable the user to make settings. So-called ***separators*** enable the administrator to design the aspect report parameter, i.e. to add descriptive texts, etc.

The following parameters are available:

Text entry parameter Enables the user to make an entry to a text field

Date parameter Enables to enter a date or a time field

Data source parameter Enables to select from a number of possible pre-set values

User variable Enables to readout so-called user variables. User variables are discussed in detail in Chapter 8.

File-selection parameter Allows users to upload files, which can then be used in the report creation process. This is especially powerful in combination with script reports or custom export targets.

Script parameter Allows you to specify a parameter in HTML and JavaScript. This gives you the flexibility to master almost any requirement.

The following separators are available:

Display text Enables to display text

Heading Enables to integrate a sub-heading

Separator Enables to set a division

In the following we will present a close look at the individual parameter type settings. Then we will explain how to use parameters in data sources.

General Usage of Parameters

You add further parameters or separators to a report via the respective buttons in the tool bar in tab **Parameters**. By clicking on the **Remove** button you can remove either the selected or all report parameters. In addition, by using copy & paste you can add parameters to your current report or to another one. To copy one or more parameters to the Clipboard, activate the respective parameters and press `CTRL+SHIFT+C` (you will get a short notice about the successful move to the Clipboard). To add parameters, first click on the parameters list (if there is no parameter available click on the header), then press `CTRL+SHIFT+V`.

To edit the parameter properties, double click on the parameter icon, or activate the parameter and select "Edit" from the tool bar. Parameter name and key can directly be edited in the list. To do this click on the respective cell.

There are properties specific to each parameter type, but all parameters have the following properties in common:

Name:	Plain text name. Visible to the user.
Description:	Description of the parameter. Visible to the user.
Key:	A technically unique name. It is used to access the parameter from the data source.
Hidden:	Indicates whether users can see the parameter in the parameter page or not.
Editable:	Indicates whether users are allowed to modify the parameter.
Mandatory	Whether or not the parameter is mandatory.
Display inline:	The option **Display inline** controls the layout of the parameter page. By default parameters are displayed in a block. This means that line feed ends each parameter which results in the parameters being displayed one below the other. If a parameter is displayed inline, no line feed will be inserted. This enables to position the parameters next to each other.
Label width:	Allows to set a width for the label (i.e., name and description). The label width is inherited by all following parameters unless it is overwritten there. To set the label width to ***auto*** (which is the default) set it to -1

Parameter Instances

When a user allocates parameters and saves their configuration in a report variant, the parameter settings made will be stored in parameter instances. If you modify a parameter subsequently, the instances of the variants will not automatically change. This can be best explained by giving an example:

We assume that we have created a multiple choice list parameter following the above example. The appurtenant basic query was as follows:

```
SELECT * FROM T_AGG_ORDER WHERE $X{IN, OR_CUSTOMERNUMBER, P_CUSTNUM}
```

In the meantime, some variants have already been created, and the users have edited the variants' parameter `P_CUSTNUM`. If you modify the parameter definition so as to select only one value, and adapt the query as follows:

```
SELECT * FROM T_AGG_ORDER WHERE OR_CUSTOMERNUMBER = $P{P_CUSTNUM}
```

This would cause problems with the existing variants as here a list of values is given to the query and not only a single value. In this case, by applying the button "Adapt instances", you could reset the instances for the selected parameter to the initial value. However, this is not necessary for all parameter modifications, and as a reset will lead to the loss of the user's selection, ReportServer will not automatically delete the instances with every parameter change. Let us assume you did not set the parameter from multiple choice to single choice, but you only rearranged the query to offer less selection options. Here it is not necessarily required to adapt the existing instances.

The Text Entry Parameter

With the text entry parameter you can provide users a simple text field to enter parameter values. By applying the special characteristics of the parameter you control the presentation of the text field (height and width) by defining the type, and which type of object is to be re- turned. Beside the

`java.lang.String` (for a text object) you can choose from various number formats. In addition, you can specify a default value for the parameter.

With the pattern field you can state a regular expression that will be checked when entered by the user. If the entry is not made in the required format, an error message will display.

The Date Parameter

With the date parameter you can enable the users to select from a date or time field. As a default value you can either use the current date and current time (**Now as default**), you can allocate a fixed date, or you can determine the default value by using a ReportServer formula expression (e.g. `${today.firstDay()}` for the first day of the current month). For further information on the ReportServer formula language refer to the User manual.

The Data Source Parameter

The data source parameter provides you with the option to enable the user to choose from a selection of pre-set values. The specified values will be provided by a data source, giving it its name. ReportServer expects a table with one or two columns to be returned from the data source. Here, if there is one column the values will be taken as display value and parameter value. If two columns are returned, by default the first column is regarded as parameter value and the second one as display value. If you configured a data source with the following contents:

1	A
2	B
3	C
4	D

the user will be displayed “A,B,C,D”. When selecting **A** the value 1 would be passed on as a parameter value.

Directly Entering the Value List

In some cases you may wish to enter the parameter values directly at the parameter with- out a detour via the data base. Here, the solution is a CSV data source with an argument connector (refer to "„Data Sources“"). This enables you to enter the values for the parameter directly at the parameter. When generating the data source please observe to deactivate the database cache. Please consider as well that the first line of the CSV data hold the descrip- tion of the data. To enter the above data as CSV table proceed as follows

```
VALUE;DISPLAY
1;A
2;B
3;C
4;D
```

Subsequent Processing of Values

By using post-processing you can make complex changes to parameter data. Please note that post-processing needs to be activated in the configuration (see ReportServer configuration guide).

Here you may give a Groovy script which will run for each data line. The current line will be given to the script as **data**. To change the place of VALUE and DISPLAY you can use the following simple script:

```
data.reverse()
```

ReportServer expects a list or an array to be returned. To exclude values you can return NULL. Instead of returning a single value you can also return multiple values by means of a nested list. If you return

```
[[1,'A'],[2,'B']]
```

this would, for instance, result in adding two data records.To better control the output, beside the replacement **data**, the replacement **cnt** (an integer value) will be provided. **isLast** here indicates whether the value currently processed is the last value. **cnt** counts the number of data records processed beginning with 0.

Please consider that ReportServer will for consistency reasons ensure that there will be no duplicates in the return. This means that when the data hold exactly the same value twice, only one will be presented to the user for selection.

Security

Allowing users to execute Groovy bears a security risk. Thus, you should be careful when enabiling this option, since this means that any user that is able to edit reports can potentially execute arbitrary code.

Selection Mode

You can determine via the selection mode whether the user can select multiple values or only one. Depending on the setting chosen, there are various views available to you. For multiple selection options you can choose from the usual selection dialogue (pop up) and check boxes. The height and width settings control the display of the selected values when selected from the dialogue. When opting for the check boxes you can control the ranging of the check boxes via the settings **layout**, **packing mode** and **number per**. Here the layout option controls the basic orientation, i.e. whether the values will be entered first from top to bottom or from left to right. The "packing mode" setting controls whether you rather want to opt for the number of columns or the number of packages. Then with the **number per** option you control this number. The best way to explain this is by giving you an example. Let us assume we can choose from 11 values. If we now select **Column** as packing mode and 2 as number, the values will be arranged in two columns where the first one will hold 6 values and the second one 5 values (with the layout setting top/bottom, left/right).

A G
B H
C I
D J
E K
F

However, if you choose **package** as the packing mode, ReportServer will arrange columns of size 2. Consequently, the objects would be arranged as follows:

A C E G I K
B D F H J

For the single selection setting you can draw from selection list (drop down), selection dialogue (pop up), or radio buttons. If you take the last option, the layout will be controlled in the same way as with multiple choice check boxes.

Typing

By typecasting you control the type of objects that ReportServer will attempt to return. Of course, this must be based on the data available. If a value is to be returned as a date, you can specify the available date format by the setting **format**. The syntax to be used here is explained under:

http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html

Default Values

By using the option default values you can determine the values selected by default. Please consider here that for a single selection from a selection list (drop down) the convention applies that if there is no determined default value, normally the first value will be selected.

File-Selection Parameter

The file selection parameter provides users with the means to select or upload one or more files that can then go into the report generation process. This is especially useful, when working with custom script reports or export targets that can make use of such additional information.

The main configuration of the file upload parameter is to specify where files are coming from. This can either be from a TeamSpace or the internal file server, or via uploads. Additionally, you can specify a minimum and maximum number of files, as well as allowed file extensions and a maximal file size. Via the option "**enable file download**" you can control whether or not users can download files that have been selected.

When working with script reports you can access files that were selected via the parameter instance. In the following example we assume that we have a selection parameter with key **selectionParame-**

terKey. The script shows the various ways to access the files. For further information on scripting have a look at Chapter 13 and the ReportServer Scripting Guide.

```
import net.datenwerke.rs.base.ext.service.parameters.fileselection.SelectedParameterFile
import net.datenwerke.rs.base.ext.service.parameters.fileselection.UploadedParameterFile
import net.datenwerke.rs.core.service.reportmanager.entities.reports.Report;
import net.datenwerke.rs.fileserver.service.fileserver.entities.FileServerFile
import net.datenwerke.rs.scheduleasfile.service.scheduleasfile.entities.ExecutedReportFileReference;
import net.datenwerke.rs.tsreportarea.service.tsreportarea.entities.AbstractTsDiskNode
import net.datenwerke.rs.tsreportarea.service.tsreportarea.entities.TsDiskFolder;
import net.datenwerke.rs.tsreportarea.service.tsreportarea.entities.TsDiskReportReference;
import net.datenwerke.rs.tsreportarea.service.tsreportarea.entities.TsDiskRoot;

/*
 * the parameter instance is if type net.datenwerke.rs.base.ext.service.parameters.fileselection. ↙
      ↳ FileSelectionParameterInstance
 * its value is a List of net.datenwerke.rs.base.ext.service.parameters.fileselection. ↙
      ↳ SelectedParameterFile
 */

List<SelectedParameterFile> files = parameterMap['selectionParameterKey'];

for(SelectedParameterFile file : files){

  /*
   * A SelectedParameterFile contains an object from one of these sources
   *
   * Upload
   * Teamspace
   * Fileserver
   *
   */

  /* there are some common properties */
  String filename = file.getName();
  byte[] fileContent = file.getContent(); // returns null for objects without content; variants, folders

  /* and some for which you need to access the wrapped object */
  Object fileObject = file.getSelectedFile();

  /* when accessing the wrapped object you need to differentiate different types of content*/
  if(fileObject instanceof UploadedParameterFile){
    /* uploaded file */
    UploadedParameterFile uploadedFile = fileObject;

    /* has nothing but getContent() */
    uploadedFile.getContent();

  }else if(fileObject instanceof AbstractTsDiskNode){
    /* selected from teamspace */
    AbstractTsDiskNode tsObject = fileObject;

    /* can be one of */
    if(tsObject instanceof TsDiskRoot){
      /* a teamspace root folder */
      TsDiskRoot tsRoot = tsObject;
      tsRoot.getName();

    }else if(tsObject instanceof TsDiskFolder){
      /* a teampsace folder */
      TsDiskFolder tsFolder = tsObject;
      tsFolder.getName();

    }else if(tsObject instanceof ExecutedReportFileReference){
```

```
     /* a reference to an executed report */
     ExecutedReportFileReference tsFileRef = tsObject;

     tsObject.getData()
     tsObject.getDataContentType()
     tsObject.getOutputFormat()
     tsObject.getSize()

   }else if(tsObject instanceof TsDiskReportReference){
     /* a reference to a report/variant */
     TsDiskReportReference tsReportRef = tsObject;
     Report report = tsReportRef.getReport();
   }

  }else if(fileObject instanceof FileServerFile){
    /* selected from file server */
  }
}
```

Script Parameter

The script parameter is a very flexible parameter that basically allows you to write custom HTML and javascript. In order to communicate with ReportServer, ReportServer will call a special JavaScript method that you need to implement and hand over the currently stored value, selected configuration options as well as a callback which allows you to update the value.

When you create a scripting parameter you need to specify a ReportServer script which needs to return an HTML page. You can either create a groovy script or directly write HTML and tell ReportServer that the script is HTML by adding

```
#html
```

in the very first line. A script could hence look as follows

```
#html
<html>
  <head>
  </head>
  <body>
    <div>This is a Script Parameter. Click Me</div>
  </body>
</html>
```

What is missing from this script is the javascript method expected by ReportServer. For this we add a custom method called `initParameter`. The `initParameter` method takes two arguments

params	An object containing various configuration choices as well as the currently selected value.
callback	A callback to update the selected value

The params object contains the following information

editable	Reflects the option whether the parameter is supposed to be editable.
isDefault	true, if the parameter was not yet set.
name	The parameter's name.
mandatory	Whether or not the parameter is mandatory.
key	The parameter's key.
value	The currently selected value.
defaultValue	The specified default value.

The following is a fully functional example. If you click on the text the parameter value is updated. If you then create a variant of the report, you should be prompted with the updated value. Also note the `validate` method which is called once the user wants to close the parameter view.

```
#html
<html>
  <head>
    <script type="text/javascript">
     var callback;
     function initParameter(param, cb){
       callback = cb;
       alert("The current value is: " + param.value);
     }

     function setValue(value){
            alert("Updating the value to: " + value);
             callback(value);
     }

     function validate(){
       return "The parameter is not valid!"
     }
    </script>
  </head>
  <body>
    <div onclick="val = prompt('set new value:', ''); setValue(val);">This is a ↙
     ↪ Script Parameter. Click Me</div>
  </body>
</html>
```

Setting Values

As described above, to set values from within the JavaScript code, you need to call the callback object with the value to be set. Note, that ReportServer expects this value to be of type String. Besides setting a basic string value, you can also return a JSON formatted string. For this, consider the following `setValue` function:

```
function setValue(){
        var json = JSON.stringify({
            valA : 'A',
            valInt: 15
```

```
    });
    callback(json);
}
```

Assume that the script parameter has key `scriptParam`. If the parameter returned a JSON string, then you can access each of the JSON properties via the replacement `ParameterKey_PropertyName` ↙ ↳ . That is, you can access the integer value in the above example via `${scriptParam_valInt}` and the property ***valA*** via `${scriptParam_valInt}`. The replacement `${scriptParam}` would contain the entire JSON formatted string.

Cascading Parameters

By using cascading parameters you can use parameters within parameters and thus nest parameters. Source parameters can here be any parameters. At present, only the data source parameter supports the configuration by means of depending parameters. To do this, on the general configuration page of the data source parameter set the parameters on which the data source parameter will depend. In the following you can use them in the data source configuration as describe below.

Using Parameters

Now, the specified parameters have to be used in the report definition so that they will take effect in the report. How to do this varies with the report type and data source. In the follow-through ing we describe the parameter use in the dynamic list. How to use the parameters in Jasper BIRT or script reports is given in each section dealing with the respective report type.

For the dynamic list, parameters are used in data sources. In the following we treat the individual data source types and introduce how parameters can influence data selection.

Parameters in Data Sources: Relational Database

For relational databases, parameters are used in the appurtenant query. Here you have basically two options to embed parameters in queries: either by query replacement or by direct replacement. In the first case, the query will first be processed and the replacements will only be integrated afterwards by verifying the data type. In the second case the replacement will be added prior to processing and, therefore, this changes the query to be processed. Let us have a look at the following sample query:

```
SELECT * FROM MY_TABLE WHERE ID = ?
```

In the example, an ID shall be given by a parameter. Now by replacing "?" in the query by a value it would be ensured that the transferred value (say 1234) is of the correct data type (in the example it is the same as the attribute ID), and only then the replacement will be made. In case of a direct replacement the query would be changed to

```
SELECT * FROM MY_TABLE WHERE ID = 1234
```

and will be processed only afterwards. The result is that such a query can maliciously be changed by skillfully selecting the parameter value (a so-called SQL injection attack http://en.wikipedia.org/wiki/Sql_injection). Now, if a user enters for instance the value `1 OR 1=1` instead of a valid ID, the query would be changed as follows:

```
SELECT * FROM MY_TABLE WHERE ID = 1 OR 1=1
```

By using these techniques it might be possible that a user can display values to which it should not have access. Please consider that SQL injection examples often show the follow- ing attack: The user selects string "`1; DROP TABLE MY_TABLE;`" which would transfer to the following statement (respectively to the following two statements)

```
SELECT * FROM MY_TABLE WHERE ID = 1; DROP TABLE MY_TABLE;
```

ReportServer will intercept this attack by allowing a query to only include exactly one statement. Still, we want to emphasize that direct replacement represents a potential security risk, and, therefore, should only be applied with particular caution.

In the following we present you the various possibilities how to integrate parameter values in a query.

`$P{key}`

A parameter referenced in this way will be converted according to the standard behaviour of the parameter type and will be added to the query (query replacement) by applying the internal JDBC parameter mechanism. Quotation marks and special escape characters will automatically be set at the right place. `$P{key}` parameters can only be used as part of a WHERE condition. Our sample query would therefore be as follows:

```
SELECT * FROM MY_TABLE WHERE ID = $P{key_ID}
```

where `key_ID` is the key of a parameter which returns a single value of the `ID` attribute type.

`$P!{key}`

Unlike the `$P{}` parameter reference, `$P!{}` parameter will directly be entered in the query by omitting the JDBC parameter mechanism (direct replacement). This enables to use this parameter type at positions in the query where no JDBC parameters are allowed. As demonstrated in the example, a parameter can, for instance, provide part of a source table name. Please consider that for this parameter type, quotation marks will not automatically be set or special characters escaped. In addition, by directly interfering in the query you run the risk that your report will be an open gate for SQL injection. Therefore, use this parameter type with caution. Example:

```
SELECT * FROM tblpfx_$P!{paramTblSuffix}
```

`$X{function, COLUMN, key1 [, key2]}`

The `$X{}` parameter reference facilitates the use of SQL clause functions and provides an ***injection safe*** way of use. Furthermore it properly handles NULL values and is able to handle lists (for example, for `IN` or `NOTIN` clauses).

Functions supported are:

`$X{IN, column, parameterkey}` : If all parameter values are other than `NULL`, an expression in the form `<column> IN (?, ?, .., ?)` will be generated. If the list of values includes `NULL` values as well as values other than `NULL` the generated expression is:

(<column> IS NULL OR <column> IN (?, ?, .., ?))
If all delivered values are NULL, the expression generated becomes <column> IS NULL.

$X{NOTIN, column, parameterkey} : If all parameter values are other than NULL an expression in the form <column> NOT IN (?, ?, .., ?) will be generated. If the list of values includes NULL values as well as values other than NULL, the generated expression is:

(<column> IS NOT NULL AND <column> NOT IN (?, ?, .., ?))
If all delivered values are NULL, the expression generated becomes <column> IS NOT NULL.

$X{EQUAL, column, parameterkey} : If the parameter value is other than NULL, an expression in the form <column> = ? will be generated. If the parameter value is NULL, the generated expression is <column> IS NULL.

$X{NOTEQUAL, column, parameterkey} : If the parameter value is other than NULL, an expression in the form of <column> <> ? will be generated. If the parameter value is NULL, the expression generated is <column> IS NOT NULL.

$X{LESS, column, parameterkey} : If the parameter value is other than NULL, an expression in the form of <column> < ? will be generated. If the parameter value is NULL, an expression always evaluating to true will be generated, e.g. 0=0.

$X{LESS], column, parameterkey} : If the parameter value is other than NULL, an expression in the form of <column> <= ? will be generated. If the parameter value is NULL, an expression always evaluating to true will be generated, e.g. 0=0.

$X{GREATER, column, parameterkey} : If the parameter value is other than NULL, an expression in the form of <column> > ? will be generated. If the parameter value is NULL, an expression always evaluating to true will be generated, e.g. 0=0.

$X{[GREATER, column, parameterkey} : If the parameter value is other than NULL, an expression in the form of column >= ? will be generated. If the parameter value is NULL, an expression always evaluating to true will be generated, e.g. 0=0.

$X{BETWEEN, column, lowerParameterkey, upperParameterkey} : If both parameter values are other than NULL, an expression in the form of (<column> >? AND column < ?) will be generated. If one of the two parameters is NULL, it will only be compared with the other parameter, the then generated expression will be <column> > ? or <column> < ?. If both parameter values are NULL, an expression always evaluating to true will be generated, e.g. 0=0.

$X{[BETWEEN, column, lowerParameterkey, upperParameterKey} : If both parameter values are other than NULL, an expression in the form of (<column> >= ? AND column <?) will be generated. If one of the two parameters is NULL, it will only be compared with the other parameter, the then generated expression will be <column> >= ? or <column> < ?. If both parameter values are NULL, an expression always evaluating to true will be generated, e.g. 0=0.

$X{BETWEEN], column, lowerParameterkey, upperParameterKey} : If both parameter values are other than NULL, an expression in the form of <column> > ? AND <column> <= ?) will be generated. If one of the two parameters is NULL, it will only be compared with the other

parameter, the then generated expression will be `<column> > ?` or `<column> <= ?`. If both parameter values are NULL, an expression always evaluating to true will be generated, e.g. `0=0`.

`$X{[BETWEEN], column, lowerParameterkey, upperParameterKey}` If both parameter values are other than `NULL`, an expression in the form of `(<column> >= ? AND column <=?)` will be generated. If one of the two parameters is `NULL`, it will only be compared with the other parameter, the then generated expression will be `<column> > =?` or `<column> <= ?`. If both parameter values are `NULL`, an expression always evaluating to true will be generated, e.g. `0=0`.

Parameters referenced by `$X{}` will also be added to the actual query via the standard JDBC parameter mechanism, they also include correct quoting and escaping just like `$P{}` parameter references, but they can also only be used in `WHERE` clauses.

The advantage of using `$X{}` parameter references over absolute `$P{}` parameters is the correct processing of parameters of the value `NULL`.

`${key}`, or `$!{key}`

Beside the `$P`, `$P!` and `$X` replacements that we have already discussed, you are provided with the `${}` and `$!{}` options to integrate parameters by using ReportServer `${}` formula expressions. The exclamation mark effects the avoidance of the parameter mechanism (direct replacement) as it is the case with the `$P!{}` parameter.

The ReportServer `${}` formula language builds on the Java-Unified-Expression-Language (http://juel.sourceforge.net/, https://www.jcp.org/en/jsr/detail?id=245 and http://www.oracle.com/technetwork/java/unifiedel-139263.html) and enables you to run simple operations as for instance date arithmetic, etc. based on the parameter values. Here, for each parameter the following replacements will be provided.

`key` Allows to access the parameter value in the form as it was inserted in the query by entering `$P{key}`.

`_key` Enables to access the Java object lying below the parameter instance (a sub-class of `net.datenwerke.rs.core.service.parameters.entities.ParameterInstance` which is designed to save the values selected by the user). Here, each parameter type can define differing functions to be executed on the parameter. The resulting values will then be entered to the query in the correct data type and, if applicable, in quotation marks.

`__key` Enables to access the parameter definition object (a sub-class of `net.datenwerke.rs.core.service.parameters.entities.ParameterDefinition` which is designed to save the parameter settings). This enables to access, for instance, the name or other metadata of the parameter.

For further information on the `${}` formula language refer to Appendix A as well as to the User manual.

Parameters in Data Sources: CSV List

For CSV lists you can use parameters in the Wrapper query, as described for relational databases.

Parameters in Data Sources: Script Data Sources

For script data sources you can use parameters in the Wrapper query, as described for relational databases. In addition, you can use parameters with the `${}` Syntax in the script arguments.

Special Parameters

In addition to the parameters defined per report, ReportServer adds a few special parameters to the parameter set of each report. These special parameters enable the report designer to access special properties.

Current User

The properties of the current user can be called up by using `${_RS_USER.xx}`, as a substitute `getFirstname()`, `getLastname()`, `getTitle()`, `getUsername()`, `getEmail()`, `getId()` can be used.

Additionally, the `$P`, or `$P!` syntax can be used. Here, the following replacements are available.

- `_RS_USER_FIRSTNAME`
- `_RS_USER_LASTNAME`
- `_RS_USER_TITLE`
- `_RS_USER_USERNAME`
- `_RS_USER_EMAIL`
- `_RS_USER_ID`

Current Report

The properties of the current report can be accessed by `${_RS_REPORT.xx}`, and as methods you can access `getName()`, `getDescription()`, `getKey()`, `getId()`, and `isVariant()`.

Additionally, the `$P`, or `$P!` syntax can be used. Here, the following replacements are available.

- `_RS_REPORT_NAME`
- `_RS_REPORT_DESCRIPTION`
- `_RS_REPORT_KEY`
- `_RS_REPORT_ID`

Locale

You can access the system's default locale as well as the locale of the current user via the replacements

- `_RS_LOCALE_SYS`
- `_RS_LOCALE_USER`

Global Constants

Values defined as global constants may be used in queries in the same way as parameters. For further information on this refer to Chapter 7 Global Constants.

Report Metadata

Metadata which were defined at the report can be used by means of the parameter syntax. For further information on this refer to Section 6.13 Report Metadata.

6.4 JasperReports

ReportServer supports the creation of graphical reports by using the OpenSource library JasperReports Library (`http://www.jaspersoft.com`). It is primarily designed to specify pixel perfect reports to directly print them, or to export them as a PDF at a later time. In Jasper re- ports, reports are defined in a specific XML dialect (JRXML). Reports can be directly written by using a text editor, however, in practice reports are mostly drawn up by using the open report designers IReports (`http://community.jaspersoft.com/project/ireport-designer`).

A JasperReport always consists of exactly one master report and one or several sub-reports. There is an XML file for the master report and for each sub-report with the extension `.jrxml`. These files are required to embed the report in ReportServer.

For further information on the creation of reports by using JasperReports refer to the relevant documentation of JasperSoft (`http://community.jaspersoft.com/documentation`).

To embed Jasper reports in ReportServer switch to the Report management and create a **Jasper report** in a folder. As with dynamic lists, in the following dialogue you can give a name and a description to the report as well as a unique key to call up the report by URL, for instance.

To complete the configuration you have to upload the master report (i.e. the corresponding `.jrxml` file) as well as all sub-reports.You can create sub-reports either by the **Add sub-report** button, or by dragging and dropping it to the respective window.

During report execution, sub-reports will be loaded exclusively via the subreport's name; possible path information will be ignored. Therefore, it doesn't matter how you locally organized the files belonging to a report, you need not adapt them in order to use them in ReportServer. However, you must ensure that no two sub-reports have the same name. Conversely, however, you note that ReportServer never loads sub-reports from the local server file system. Accordingly, all sub-reports

always have to be registered together with the master report. Furthermore, it is not possible to share sub-reports between several reports.

Finally, you have to assign a data source to the report. Here, you can exclusively draw from relational databases as the source of your data. In ReportServer there is no need to enter queries as you do with dynamic lists for they have already been included in the `.jrxml` files.

Just like dynamic lists, Jasper reports can be controlled by parameters. They will be created in the reports themselves (for detailed information refer to JasperReports/Ireport Ultimate Guides http://community.jaspersoft.com/documentation). In ReportServer you have to additionally configure all parameters entered in the report by the parameter management (tab at the bottom of the screen). Here, ReportServer will support you by automatically creating parameters from the master report. This function can be activated via the **Suggest parameter** button in the tool bar. We note that the automatic extraction of parameters is rudimentary and for may need manual adjustments for more complex parameter settings. The configuration of the parameters is otherwise identical to the configuration of dynamic list parameters.

Unformatted Excel Output

JasperReports attempts to present pixel perfect reports in all output formats.As a result, when outputting it to Excel they are usually nicely made up, however, numbers will not appear as such but as text, this complicates further data processing unnecessarily. Here, ReportServer provides the option to add an additional data source to a Jasper report. If this function is activated you will be offered the additional output format **Excel (unformatted)** when executing the report. Here the ResultSet of the data source execution will be presented as an Excel file.

6.5 Eclipse Birt

Eclipse BIRT is a reporting system to generate pixel perfect reports in diverse output formats. A BIRT report is defined in form of an XML document. For the creation and editing of reports usually the BIRT Report Designer is used which simplifies your draft report by a graphical presentation. Some of the sample reports realized in Eclipse BIRT are given in the demo data supplied with ReportServer as well as in the annex. For further information on Eclipse BIRT refer to http://www.eclipse.org/birt/.

Apart from the XML documents defining the actual report, BIRT reports can additionally include resource bundles and library files. These will be stored in a specifically identified folder in the file server separate from the actual report.

To create a new BIRT report in ReportServer, proceed as follows. Switch to the Administration module and then to the section Report management. From the context menu of a folder you create a new BIRT report. The configuration mask presents the same fields as all other report types do and allows to enter a name, description and key. In addition to these data, it is required to upload the Rpt-design-file of the report.

Possibly available report libraries or resource bundles need to be copied to a folder in ReportServer' internal file system. Which folder is controlled by the configuration file `etc/reportengines/`

`reportengines.cf`. For further information on this refer to the ReportServer configuration guide.

Finally, you may select the data source to be used. If a data source is provided in ReportServer, it will be used instead of a data source possibly defined in the report itself. But if you wish to use the data source given in the report, you may not select a data source in ReportServer.

Similarly to parameters in JasperReports, parameters are defined directly in Eclipse Birt, but have to additionally be specified within ReportServer. ReportServer supports you by trying to extract parameter settings from your report. This will, however, only capture basic parameters and for more sophisticated parameter settings (for example cascading parameters), manual adjustments will be necessary.

Birt allows to specify datasets directly within the report. Theses data sets are often used as data basis for parameters. ReportServer allows to use these data sets as ReportServer data sources. For further information see Chapter 4 (Data Sources).

6.6 SAP Crystal Reports

ReportServer comes with a basic support for SAP Crystal Reports[1] SAP Crystal Reports is the only proprietary reporting component supported by ReportServer and thus, we are not allowed to ship the necessary libraries with the ReportServer release. If you do have a proper Crystal Reports license you are, however, good to go and in the following we describe the necessary steps to work with Crystal from ReportServer.

Prepare ReportServer for Crystal

In order to use Crystal Reports with ReportServer you must first install the SAP Crystal Reports for Java runtime components (short, the Java Reporting Component (JRC)). You'll find these, for example, on the SAP download `http://scn.sap.com/docs/DOC-29757`. Here, look for Runtime Libraries and download the there located archive (around 45 MB). Unzip the archive and locate the lib directory. This includes all the additional libraries you have to install to get going with Crystal Reports. The following jars are needed:

- `com.azalea.ufl.barcode.1.0.jar`
- `CrystalCommon2.jar`
- `CrystalReportsRuntime.jar`
- `cvom.jar`
- `DatabaseConnectors.jar`
- `icu4j.jar`
- `jai_imageio.jar`
- `JDBInterface.jar`

[1] `http://www.crystalreports.com/`

- jrcerom.jar
- keycodeDecoder.jar
- logging.jar
- pfjgraphics.jar
- QueryBuilder.jar
- webreporting-jsf.jar
- webreporting.jar
- XMLConnector.jar
- xpp3.jar

That is, all jars except the commons-*.jar and log4j.jar. Note that these should not be included as ReportServer already ships with newer versions of these components. Copy the above mentioned jars to the ReportServer WEB-INF/lib directory and start ReportServer.

Use Crystal Reports

Reports for Crystal Reports come in the .rpt file format. All that is needed to configure a Crystal report in ReportServer is to upload the corresponding .rpt-file. Similarly to Birt, you can either configure a datasource via the interface or leave it open in which case Crystal will use the datasource configured within the report.

6.7 Saiku / Mondrian Reports

Saiku reports allow you to generate multi-dimensional OLAP style reports. The name Saiku stems from the beautiful open source user interface for Mondrian called Saiku (http://community.meteorite.bi/) that we use to display such OLAP reports. For an introduction to OLAP and Mondrian we refer to the Mondrian documentation available online at http://mondrian.pentaho.com/documentation.

Saiku reports are easily configurable once you have your Mondrian datasource (see Section 4.8). A Mondrian datasource defines one or more cubes. A Saiku report takes a Mondrian datasource as datasource and you additionally select the cube the report should use. This is already everything you need to use OLAP with ReportServer. For an introduction to the OLAP UI we refer to the ReportServer user guide.

6.8 JXLS Reports

JXLS (http://jxls.sourceforge.net/1.x/) is a templating engine for Microsoft Excel and can be used together with the Dynamic List in order for users to export data into a predefined Excel spreadsheet (see the ReportServer User Guide). JXLS can, however, also be used as a first class

report type which offers some advantage over the use as a Dynamic List templating engine: In particular you can define SQL queries directly from within the template and work with parameters.

At the moment ReportServer only supports version 1.0 of the jXLS library. As the current version 2.x is not backwards compatible and would break existing templates we cannot simply update. We are however working on supporting v2 templates alongside the legacy v1 templates.

In order to define a JXLS report, all that is needed in terms of configuration is an Excel file which serves as the template and a datasource which can be accessed from within the template. In addition as with any other report type you can specify parameters that can also be accessed from within the template.

The Basics

Templates are formed using two basic constructs: tags and replacements. A so called tag surrounds an area and defines how the cells within are to be interpreted:

```
<jx:forEach items="${data}" var="department">
  ${department.name} | ${department.chief}
</jx:forEach>
```

In the example each line of the example becomes one line in the Excel document and the pipe symbol (|) marks the end of cell, that is the second line consists of two cells. It is also important to note that commands should start at the very beginning of a cell and that no leading whitespace may exist.

In the example the tag `<jx:forEach ...>` marks the beginning of a block. The block is closed by the corresponding end tag `</jx:forEach>`. All cells within the block are repeated for every record in the data variable. To access the individual attributes within the "forEach-block" the variable defined in the "var" attribute can be used: in the example "department". To access an individual attribute use `${department.colname}` where "department" is the variable bound to the current data row and "colname" is the name of the attribute.

When used as a template for the dynamic list (see the chapter on the Dynamic List in the User guide), then `${data}` contains the data selected by the dynamic list. When used as a first-class report type you need to select the data from the specified datasource. For this, you can use the object `${sql}` which provides access to the underlying datasource. As an example consider the following template:

```
<jx:forEach items="${sql.exec('SELECT CUS_CUSTOMERNAME name, CUS_PHONE phone FROM T_AGG_CUSTOMER')} ↙
↳  " var="customer">
       ${customer.name} | ${customer.phone} |
</jx:forEach>
```

Variable `${rm}` can be used synonymously to `${sql}` in order to be consistent with the documentation of JXLS (http://jxls.sourceforge.net/samples/reportsample.html.

Here we select two fields from table `T_AGG_CUSTOMER`. In order to use parameters within queries,

you can use the standard syntax for parameters in the query. This is best understood with an example:

```
<jx:forEach items="${sql.exec('SELECT CUS_CUSTOMERNAME name, CUS_PHONE phone FROM T_AGG_CUSTOMER
 ↳  WHERE CUS_CUSTOMERNAME = ${parameterKey}')}" var="customer">
       ${customer.name} | ${customer.phone} |
 </jx:forEach>
```

Here we assume that there exists a parameter with key "parameterKey". Additionally you can access parameters via the ${parameters} object. Again assuming that there is a parameter with key "parameterKey" you could access the parameters value via

```
${parameters.parameterKey}
```

Following is a complete example, which works for the demo data that is shipped with ReportServer. It shows a very simple employee report showing some basic information of the employee and the customers served by the employee. The report uses a single parameter with key employee which is assumed to hold an employee number.

```
Employee Number: | ${parameters.employee}
<jx:forEach items="${rm.exec('SELECT EMP_FIRSTNAME as firstname, EMP_LASTNAME as lastname FROM
      ↳  T_AGG_EMPLOYEE WHERE EMP_EMPLOYEENUMBER=${employee}')}" var="employee">
First name: | ${employee.firstname}
Last name: | ${employee.lastname}
</jx:forEach>

List of Customers

Customer Name | Customer Number | Volume
<jx:outline detail="true">
<jx:forEach items="${rm.exec('SELECT CUS_CUSTOMERNAME as name, CUS_CUSTOMERNUMBER as num, Y_VOLUME as
      ↳  volume FROM T_AGG_CUSTOMER WHERE EMP_EMPLOYEENUMBER=${employee}')}" var="customer">
${customer.name}  | ${customer.num}  | ${customer.volume}
</jx:forEach>
</jx:outline>

  | TOTAL: | $[SUM(C11)]
```

A full documentation of the possibilities offered by JXLS is out of scope of this documentation, and we refer the interested reader to the official JXLS documentation available on http://jxls.sourceforge.net/1.x/index.html.

6.9 Script Reports

In this section we give you a short introduction to script reports. A detailed documentation on scripts and script reports will be given in the ReportServer scripting guide. This section builds upon the introduction of scripts given in Chapters 12 and 13. On a first read it might thus be advisable to skip this section.

Script reports allow to create complex and interactive reports. Moreover, they are suited for the generation of documentation reports that directly process metadata from ReportServer. So, for instance, the report documentation supplied with the demo data has been realized in form of a script report. To run a script report you only need a script. Create the following Hello World script in the directory bin/tmp (in the File system).

```
"Hello World"
```

Now, in the Administration module switch to Report management, create a new script re- port and assign the newly created script. If you run the report, you will see the output `"Hello World"`. Note that for the execution of a script report, the user does not only need to have the (execute) right to execute the report, but also needs execute rights on the underlying script. Script reports will be displayed in an IFrame by default, and therefore, the HTML output is appropriate. In the following we will accordingly give our `"Hello World"` script a makeover.

Basically, you could simply return HTML. Instead of displaying `"Hello World"`, for instance

```
"<html><body><h1>Hello World</h1></body></html>"
```

In the following we will use Groovy's Markup Builder to create HTML comfortably:

```
import groovy.xml.*
def writer = new StringWriter()

new MarkupBuilder(writer).html {
  head {
    title( "Hello World" )
  }
  body {
    h1("Hello World")
    p("This is a hello world script")
    p("The time is: " + new Date())
  }
}
writer.toString()
```

Arguments

While configuring the script report you can add arguments to it that you can process via the variable **args** just like command line arguments.

```
import groovy.xml.*
def writer = new StringWriter()

new MarkupBuilder(writer).html {
  head {
    title ( "Hello World" )
  }
  body {
    h1("Hello World")
    p("This is a hello world script")
    p("The time is: " + new Date())
    p("Arguments:" + args.join(", "))
  }
}
writer.toString()
```

Parameters

Parameters specified at the report will be passed to the script via the **parameterMap**. If there is, for instance, the parameter with the key **param**, you can access it by using the following statement.

```
parameterMap['param']
```

Output Formats

Beside arguments and parameters you can define multiple output formats with the report. They will be entered separated by commas, e.g. “HTML, PDF”. If you now execute the report, you will see that the output formats were changed to export options. In the script itself, you can query the output format by using the variable **outputFormat**. The output format for preview is “preview”:

```
import groovy.xml.*
def writer = new StringWriter()

new MarkupBuilder(writer).html {
  head {
    title ( "Hello World" )
  }
  body {
    h1("Hello World")
    p("This is a hello world script")
    p("The time is: " + new Date())
    p("Arguments:" + args.join(", "))
    p("Output format:" + outputFormat)
  }
}
writer.toString()
```

By using the open source library Flying-Saucer you can easily create PDF documents from HTML. ReportServer supports you here by providing a pre-set renderer for PDF creation. In the following example a PDF object will be returned provided you have selected **pdf** (please ensure to pass the format in lower case letters to the script).

```
import groovy.xml.*
def writer = new StringWriter()

new MarkupBuilder(writer).html {
  head {
    title ( "Hello World" )
  }
  body {
    h1("Hello World")
    p("This is a hello world script")
    p("The time is: " + new Date())
    p("Arguments:" + args.join(", "))
    p("Output format:" + outputFormat)
  }
}
if("pdf".equals(outputFormat.trim()))
  return renderer.get("pdf").render(writer.toString())
writer.toString()
```

Data Sources

Beside parameters, arguments and output formats you can also add a data source to a script. The data source object will be passed to the script via the "connection" variable. The following example uses the internal demo data (refer to Section 4.3) to present a customer list.

```
import groovy.xml.*
import groovy.sql.Sql

def writer = new StringWriter()

new MarkupBuilder(writer).html {
  head {
    title ( "Hello World" )
  }
  body {
    h1("Hello World")
    p("This is a hello world script")
    p("The current time is: " + new Date())
    p("Arguments:" + args.join(","))
    p("Output format:" + outputFormat) h1("Customers: ")
    ul {
      new Sql(connection).eachRow("SELECT DISTINCT CUS_CONTACTLASTNAME
                          FROM T_AGG_CUSTOMER ORDER BY 1 ASC" ){
        li(it.CUS_CONTACTLASTNAME)
      }
    }
  }
}
if("pdf".equals(outputFormat.trim()))
  return renderer.get("pdf").render(writer.toString())
writer.toString()
```

6.10 Grid Editor Reports

The ***Grid Editor component*** is not a report type in the classical sense. It is rather a very flexible spreadsheet like database editor that can be used in situations where you want to enable a user to do some basic data administration. Once defined the grid editor can be used as any report, that is, it can be used by users in their TeamSpace and users can even export the underlying data to Excel and schedule the report.

The grid editor component is configured by providing a datasource[2] and a ReportServer script. ReportServer scripts are covered in greater detail in Chapter 13 and in the separate ReportServer scripting guide and it might be helpful to skip the following details on a first read and come back to grid editors once you have a basic understanding of ReportServer scripts.

[2] Currently relational database datasources are directly supported but in theory you could also implement grid editors based on other types of datasources.

Remark. The grid editor is an experimental feature at the moment and its API might change in future versions. We are also very interested to hear your feedback in order to make the editor better fit your needs.

A Basic Grid Editor

The simplest use case for a grid editor is when you have a database table and you want to give a user the possibility to edit the data in that table. At the very basis you need to generate a GridEditorDefinition which handles the interaction with the user. For relational databases ReportServer comes with a helper class called DbGridEditorDefinition (located in `net.datenwerke.rs.grideditor.service.grideditor.definition.db`) which tries to handle as much of the interaction (loading data, storing values, etc.) as possible. Consider the following example script:

```
import net.datenwerke.rs.grideditor.service.grideditor.definition.db.*

def definition = GLOBALS.getRsService(DbGridEditorDefinition.class)

def adapter = definition.getAdapter()
adapter.setTableName("T_AGG_CUSTOMER")

return definition;
```

Here we obtain a new instance of a DbGridEditorDefinition and load an adapter object which is used for most of the configuration. At the very least you need to specify which database table you want to work on. With the above configuration ReportServer will attempt to load the data as

```
SELECT * FROM T_AGG_CUSTOMER
```

and display the data paged with a page size of 100 rows. To change the number of rows per page you can call `setPageSize()` on the adapter and set the number of rows. By default the editor will allow the user to edit every cell, to delete entire rows and to insert new rows. The entire editing process is cached on the client and only if the client calls **save** will the data be stored.

Consider the following data table (a small extract of the demo data):

CUS_CUSTOMERNUMBER	CUS_CUSTOMERNAME	CUS_CREDITLIMIT
386	Lordine Souveniers	121400
412	Extreme Desk Decorations, Ltd	86800
456	Microscale Inc.	39800

Now suppose the user changed the credit limit of Microsale to 50000. In this case ReportServer builds the following update statement

```
UPDATE T_AGG_CUSTOMER
    SET
        CUS_CUSTOMERNUMBER = 456 ,
        CUS_CUSTOMERNAME = 'Microsale Inc.' ,
        CUS_CREDITLIMIT = 50000
    WHERE
        CUS_CUSTOMERNUMBER = 456 AND
        CUS_CUSTOMERNAME = 'Microsale Inc.' AND
```

```
        CUS_CREDITLIMIT = 39800
```

Assuming that CUS_CUSTOMERNUMBER is the table's sole primary key, this statement would be a bit of an overkill as the WHERE clause lists fields which are not part of the primary key. You should thus always tell ReportServer the primary key of a table. In addition we might want to only display a fraction of the table:

```
import net.datenwerke.rs.grideditor.service.grideditor.definition.db.*

def definition = GLOBALS.getRsService(DbGridEditorDefinition.class)

def adapter = definition.getAdapter()

adapter.setTableName("T_AGG_CUSTOMER")
adapter.setPrimaryKey('CUS_CUSTOMERNUMBER')
adapter.addColumns('CUS_CUSTOMERNUMBER', 'CUS_CUSTOMERNAME',  'CUS_CREDITLIMIT')

return definition;
```

Now if a user updates the table the following statement will be generated behind the scenes:

```
UPDATE T_AGG_CUSTOMER
    SET
        CUS_CUSTOMERNUMBER = 456 ,
        CUS_CUSTOMERNAME = 'Microsale Inc.' ,
        CUS_CREDITLIMIT = 50000
    WHERE
        CUS_CUSTOMERNUMBER = 456
```

Tip. Instead of providing a list of the primary key columns you can also use the method `addPrimaryKeyColumn()`.

On the importance of primary keys

You should avoid using the Grid Editor on tables that do not have a primary key (or have the primary key not displayed). Assume the following configuration:

```
import net.datenwerke.rs.grideditor.service.grideditor.definition.db.*

def definition = GLOBALS.getRsService(DbGridEditorDefinition.class)

def adapter = definition.getAdapter()

adapter.setTableName("T_AGG_CUSTOMER")
adapter.addColumns('CUS_CUSTOMERNUMBER', 'CUS_CUSTOMERNAME',  'CUS_CREDITLIMIT')

return definition;
```

Now if a user updates the table ReportServer generates the following update query

```
UPDATE T_AGG_CUSTOMER
    SET
```

```
        CUS_CUSTOMERNAME = 'Microsale Inc.' ,
        CUS_CREDITLIMIT = 50000
    WHERE
        CUS_CUSTOMERNAME = 'Microsale Inc.' AND
        CUS_CREDITLIMIT = 39800
```

This update statement might not uniquely identify the data row and thus trigger an update on multiple rows and thus might not have the intended effect.

> Note that if you are displaying floating point numbers you always need to work with primary keys.

A Fluid API

Besides the ***standard API*** to configure the grid editor there exists a compact, fluid API. First we can make it easier to obtain an adapter object. For this there in the scope of the script you have access to an object called `gridHelper` which provides the method `initDbGridEditor`. Then, to initiate the API one needs to call the `configure` method on the adapter object. The above example can be rewritten in the fluid API as

```
def adapter = gridHelper.initDbGridEditor()

adapter.configure(report,"T_AGG_CUSTOMER")
   .columns()
      .add('CUS_CUSTOMERNUMBER')
      .add('CUS_CUSTOMERNAME')
      .add('CUS_CREDITLIMIT')
   .done()

return adapter
```

The `configure` method takes as parameter a report object (the corresponding report object is present in the script's scope) and the table name. From there you can access various configuration, amongst others, column configuration. By calling the `columns` method you start the column configuration which you end again by calling `done`.

For the remainder of the description of the grid editor we present features first with the "classical" API and then how to do the same with the fluid API.

Global Editor Configuration

A Grid Editor's `adapter` object provides several additional configuration options that we discuss next.

Paging

Per default the Grid Editor displays the data in a paged fashion showing 100 records on each page. In order to increase the number of records on each page you can call the `adapter` object on `setPageSize(pagesize)` specifying the size a page should have. To disable paging you can call `setPaging(false)`.

Sorting and Filtering

By default, users can filter the table by specifying a search string for every column. Furthermore users can sort the Grid Editor by every column. If you would like to globally disable sorting or filtering you can use the following methods of the adapter object:

`setSortable()`	If true then sorting is enabled. (Default: true)
`setFilterable()`	If true filtering is enabled. (Default: true)

Note that filtering and sorting can also be specified on a per column basis.

All the above configuration can also be done via the fluid API by calling, for example,

```
def adapter = gridHelper.initDbGridEditor()

adapter.configure(report,"T_AGG_CUSTOMER")
   .setPaging(false)
   .columns()
      .add('CUS_CUSTOMERNUMBER')
      .add('CUS_CUSTOMERNAME')
      .add('CUS_CREDITLIMIT')
   .done()

return adapter
```

Column Configs

In order to further configure how columns are presented to the user, you can specify so called column config objects which provide column specific configurations. Basic column configs are specified via instances of class `GridEditorColumnConfig` located in `net.datenwerke.rs.grideditor.service.grideditor.definition`:

```
import net.datenwerke.rs.grideditor.service.grideditor.definition.db.*
import net.datenwerke.rs.grideditor.service.grideditor.definition.*

def definition = GLOBALS.getRsService(DbGridEditorDefinition.class)

def adapter = definition.getAdapter()

adapter.setTableName("T_AGG_CUSTOMER")
adapter.setPrimaryKey('CUS_CUSTOMERNUMBER')
adapter.addColumns('CUS_CUSTOMERNUMBER', 'CUS_CUSTOMERNAME',  'CUS_CREDITLIMIT')

def nameConfig = new GridEditorColumnConfig();
nameConfig.setDisplayName('NAME')
adapter.setColumnConfig('CUS_CUSTOMERNAME', nameConfig)

return definition;
```

Or more compactly via the fluid API

```
def adapter = gridHelper.initDbGridEditor()
```

```
adapter.configure(report,"T_AGG_CUSTOMER")
    .setPk('CUS_CUSTOMERNUMBER')
    .columns()
        .add('CUS_CUSTOMERNUMBER')
        .add('CUS_CUSTOMERNAME')
            .setDisplay('Name')
        .add('CUS_CREDITLIMIT')
    .done()

return adapter;
```

For setting the display name you may also use the shortcut `.add('columnName','distplayName ↙ ↳ ')`.

Besides changing the name of columns you can also specify a number of display options

`setWidth()`	Defines the display width. (Default: 200)
`setEditable()`	If true then the column is editable. (Default: true)
`setHidden()`	If true then the column is not displayed. (Default: false)
`setSortable()`	If true then the column is sortable. (Default: true)
`setOrder()`	Allows to specify the order by supplying "asc" or "desc". For the fluid API there is the shortcut of calling `setOrderAsc()`.
`setFilterable()`	If true then the column can be filtered. (Default: true)
`setEnforceCaseSensitivity()`	If true then filtering on that column is always case sensitive. (Default: false)

Default values for new entries

When a user adds a record to a table, all values are by default set to NULL. You can specify a default value for a column by using the `setDefaultValue` method of a column configuration object.

Data Validation

Without further configuration ReportServer will only enforce that entered data is of the correct type. For example, if a field is of type INTEGER, then a user can only type in digits into the text field. In order to further restrict what users can enter you can add one or more Validators to each column. Validators are located in package `net.datenwerke.rs.grideditor.service.grideditor.definition.validator`. The following validators are available

`MaxIntegerValidator`	Allows to specify an upper bound for columns of type INTEGER
`MinIntegerValidator`	Allows to specify a lower bound for columns of type INTEGER
`MaxBigDecimalValidator`	Allows to specify an upper bound for columns of type DECIMAL
`MinBigDecimalValidator`	Allows to specify a lower bound for columns of type DECIMAL
`MaxLongValidator`	Allows to specify an upper bound for columns of type LONG

`MinLongValidator`	Allows to specify a lower bound for columns of type LONG
`MaxDoubleValidator`	Allows to specify an upper bound for columns of type DOUBLE
`MinDoubleValidator`	Allows to specify a lower bound for columns of type DOUBLE
`MaxFloatValidator`	Allows to specify an upper bound for columns of type FLOAT
`MinFloatValidator`	Allows to specify a lower bound for columns of type FLOAT
`MaxDateValidator`	Allows to specify an upper bound for columns of type DATE
`MinDateValidator`	Allows to specify a lower bound for columns of type DATE
`MaxLengthValidator`	Allows to specify a maximum length for character based columns
`MinLengthValidator`	Allows to specify a minimum length for character based columns
`RegExValidator`	Allows to restrict text based fields to match a pattern

In order to configure a validator you instantiate the corresponding object and provide the necessary configuration in the constructor. All Min/Max validators take as configuration the bound as well as an error message that is displayed in case a user enters a value that violates the bound. For example

```
new MaxIntegerValidator(15, "Values must be less than 15");
```

The `RegExValidator` takes as configuration a regular expression (see http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html for an introduction) and an error message. To, for example, restrict data to conform to a date format of type "yyyy-mm-dd" you could use

```
new RegExValidator("^\d{4}-\d{2}-\d{2}$", "Value should be of format yyyy-mm-dd");
```

> Note that the above pattern allows 9999-99-99.

With the fluid API there are also shortcuts to validators. You can call `addValidator(new ..)` when configuring a column. Additionally there are the following shortcuts:

```
addEmptyValidator(message)
addFixedLengthValidator(length, message)
addRegExValidator(regex, message
addMinLengthValidator(min, message
addMaxLengthValidator(max, message)
addMinValidator(min, message)
addMinValidator(max, message
```

Field Editors

By default the Grid Editor constructs form fields matching the type of a column. That is, for a text columns a text field is created, for date columns a date picker, for booleans a checkbox. For certain fields you can change the default behavior and specify a custom editor. In order to tell a column that it should use a custom editor use the `setEditor` method of a `GridEditorColumnConfig` option. All custom editors are located in package `net.datenwerke.rs.grideditor.service.grideditor.definition.editor`.

Quasi-Booleans

Sometimes boolean values are not stored as booleans in a database but as text or int values. For example, you might have a text column with the values "true" and "false" or an integer column with values 1 and 0. In this case you can use a `TextBooleanEditor` or an `IntBooleanEditor` in order to still present a user with a simple checkbox, rather than a textfield or a number field. As configuration you can tell the editor which value is representing `TRUE` and which value is representing `FALSE`. Per default `TextBooleanEditor` uses the strings `true` and `false` and `IntBooleanEditor` uses integers 1 and 0. To change the default use methods `setTrueText` (resp. `setFalseText`) and `setTrueInt` (resp. `setFalseInt`).

The following is an example assuming the column names `textbool` and `intbool`.

```
def textbConf = new GridEditorColumnConfig();
textbConf.setEditor(new TextBooleanEditor());
adapter.setColumnConfig('textbool', textbConf);

def intbConf = new GridEditorColumnConfig();
intbConf.setEditor(new IntBooleanEditor());
adapter.setColumnConfig('intbool', intbConf);
```

Using the fluid API, we can define quasi-boolean editors for columns by calling `withIntBooleanEditor` ↙ ↳ , or `withTextBooleanEditor`.

```
def adapter = gridHelper.initDbGridEditor()

adapter.configure(report, 'TABLE')
    .setPk('...')
    .columns()
        .add('column')
            .withIntBooleanEditor()
    .done()

return adapter;
```

Text-Dates

Sometimes dates are stored in text form, for example, as strings of the form `yyyy-mm-dd`. In these cases you can tell the Grid Editor not to use a basic text field but a date picker. For this use the editor `TextDateEditor`. For configuration you should provide the corresponding date pattern (see `http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html` for an introduction). The following is an example for how to use the TextDateEditor.

```
def dateConf = new GridEditorColumnConfig();
dateConf.setEditor(new TextDateEditor("yyyy-mm-dd"));
adapter.setColumnConfig('textdate', dateConf);
```

Using the fluid API, we can define quasi-boolean editors for columns by calling `withTextDateEditor` ↙ ↳ , or `withTextDateEditor(format)` and specifying the format.

Selection Lists

Often you might want to allow users to choose from a list of values. For text based columns and integer columns you can define selection lists via

TextSelectionListEditor	A drop down editor for text based columns.
IntSelectionListEditor	A drop down editor for text based integers
LongSelectionListEditor	A drop down editor for text based longs
DateSelectionListEditor	A drop down editor for text based dates
DecimalSelectionListEditor	A drop down editor for text based BigDecimals
FloatSelectionListEditor	A drop down editor for text based float
DoubleSelectionListEditor	A drop down editor for text based double

A selection list can be configured in two ways. Either you can specify a simple list of values or you can specify a map of label-value pairs. Each entry of a selection list consists of a label (the string that is shown to the user) and a value (the actual value that is stored in the database). In case you provide a simple list, each entry serves both as label and as value. Following is an example of a simple selection list for a text column.

```
def ddTextConf = new GridEditorColumnConfig();
def textddEditor = new TextSelectionListEditor();
textddEditor.setValues(['a','b','c','d']);
ddTextConf.setEditor(textddEditor);
adapter.setColumnConfig('textdd', ddTextConf);
```

It configures a selection list with the entries `a`, `b`, `c`, and `d`. Alternatively, you can specify each value individually:

```
def ddTextConf = new GridEditorColumnConfig();
def textddEditor = new TextSelectionListEditor();
textddEditor.addValue('a');
textddEditor.addValue('b');
textddEditor.addValue('c');
textddEditor.addValue('d');
ddTextConf.setEditor(textddEditor);
adapter.setColumnConfig('textdd', ddTextConf);
```

If you want to distinguish between labels and values you can either specify the map directly by calling `setValueMap()`. Or you can add each entry individually as follows:

```
def ddTextConf = new GridEditorColumnConfig();
def textddEditor = new TextSelectionListEditor();
```

```
textddEditor.addEntry('a','b');
textddEditor.addEntry('c','d');
textddEditor.addEntry('e','f');
ddTextConf.setEditor(textddEditor);
adapter.setColumnConfig('textdd', ddTextConf);
```

For integer columns the configuration works identical with the only difference that you assign a `IntSelectionListEditor` instead of a `TextSelectionListEditor` and that values are of type integer. Following is an example using a simple list to define values:

```
def ddIntConf = new GridEditorColumnConfig();
def intEditor = new IntSelectionListEditor();
intEditor.setValues([2,3,5,7,11])
ddIntConf.setEditor(intEditor);
adapter.setColumnConfig('intdd', ddIntConf);
```

And an example with custom labels.

```
def ddIntConf = new GridEditorColumnConfig();
def intEditor = new IntSelectionListEditor();
intEditor.addEntry('foo',2)
intEditor.addEntry('bar',7)
ddIntConf.setEditor(intEditor);
adapter.setColumnConfig('intdd', ddIntConf);
```

Fluid API for Selection Lists

With the fluid API we can also compactly generate selection editors. By calling `withSelectionEditor ↙ ↳ ()` you start the configuration of the editor which you end by calling `done()`. This allows you to configure an editor for example as:

```
def adapter = gridHelper.initDbGridEditor()

adapter.configure(report, 'TABLE)
    .setPk('...')
    .columns()
        .add('column')
            .withSelectionEditor()
               .addValue('A')
               .addValue('B')
            .done()
    .done()

return adapter;
```

Within the edit mode for the editor you have the very same methods `addValue` and `addEntry ↙ ↳ .` In addition you can add multiple values via the method `from` which either takes a list of values (corresponding to `addValue`) or a map (corresponding to `addEntry`).

```
def adapter = gridHelper.initDbGridEditor()

adapter.configure(report, 'TABLE)
    .setPk('...')
    .columns()
```

```
        .add('column')
            .withSelectionEditor()
               .from([1,2,3,4])
            .done()
    .done()

return adapter;
```

> Note that we did not specify the type of selection editor with the fluid API. The type is, instead, recognized by the provided values. This means, that in case you want to add, for example, "long" values you need to typecast.

Finally, a frequent objective is to construct the values for the selection list from a database query. To this end, you can use the `fromSql` which takes either a SQL statement or a connection object and a SQL statement. In case you provide no connection, the same connection as to the grid editor is used. In case you provide a connection, be sure to close the connection after usage. The SQL statement needs to return either two columns (key,value) or a single column.

```
def adapter = gridHelper.initDbGridEditor()

adapter.configure(report, 'TABLE')
    .setPk('...')
    .columns()
        .add('column')
            .withSelectionEditor()
               .fromSql('SELECT key, value FROM X')
            .done()
    .done()

return adapter;
```

Predefined Variables

Within your script you can access a couple of predefined variables that allow you to access the report object, as well as the current user and the parameters. The following variables are available:

`report`	The corresponding GridEditorReport object.
`user`	The current user.
`parameterSet`	The ParameterSet object with the current parameters.
`parameterMap`	A map allowing to easily access parameters.
`gridHelper`	Used to easily construct an adapter object

Obtaining a Database Connection

You can easily obtain a database connection for the datasource that is configured at the report by using the `getConnection()` method from the `DbGridEditorDefinition` object.

```
import net.datenwerke.rs.grideditor.service.grideditor.definition.db.*
```

```
import net.datenwerke.rs.grideditor.service.grideditor.definition.*

def definition = GLOBALS.getRsService(DbGridEditorDefinition.class)
def connection = definition.getConnection(report)

connection.close()
```

Be sure to close the connection when you are done using it.

Foreign Key Relationships

In the following we explain how you can use foreign key relationships to make editing easier. Consider a database table `Products` that has the following structure

Products	
`productNumber`	The primary key (INT)
`productName`	(VARCHAR)
`productCategory`	A foreign key pointing to Table Categories (INT).
`productSupplier`	A foreign key pointing to Table Suppliers (INT).

Here, we have to foreign key relationships, one pointing from products to a specific product category and a second one pointing to a supplier for the product. What we are modeling here is a ***many-to-one*** relationship. That is, a product is in exactly one category, but of course, a category can contain multiple products. Now, assume that we have category and supplier tables that look like:

Categories	
`categoryNumber`	The primary key (INT)
`categoryName`	(VARCHAR)
`categoryDescription`	(VARCHAR)

Suppliers	
`supplierNumber`	The primary key (INT)
`supplierFirstName`	(VARCHAR)
`supplierLastName`	(VARCHAR)

Now, consider that you use a basic grid editor instance to manage the product table:

```
import net.datenwerke.rs.grideditor.service.grideditor.definition.db.*

def definition = GLOBALS.getRsService(DbGridEditorDefinition.class)

def adapter = definition.getAdapter()
adapter.setTableName("Products")
adapter.setPrimaryKey('productNumber')
```

```
adapter.addColumns('productNumber', 'productName', 'productCategory',  ' ↙
    ↳ productSupplier')

return definition;
```

In this case, if you wanted to switch the category of a product, you would need to know the category number. In such cases, it might be helpful, to instead be able to choose a category using the category name rather, but this is stored separately from the Products table. For such cases you can specify ***foreign key columns*** which allow you to display different information in place of the information that is within the table. On update and insert this information is then replaced again by the correct value.

In order to specify a foreign key column you `addForeignKeyColumn()` methods provided by the adapter. They take four or five parameters:

`column`	Denotes the column in your table that has the foreign key relationship. In the example this would be ***categoryName***.
`fkTableName`	Denotes the table corresponding to the foreign key. In the example this would be ***Categories***.
`fkColumn`	Denotes the column within the foreign key table. In the example this would be ***categoryNumber***.
`displayExpression`	A SQL expression to select a unique value from the foreign key table that is then displayed. Usually this is a single column, but it could also hold a more complex expression. In the example we could simply set it to ***categoryName***
`displayName/config`	The final parameter takes either a String or a `GridEditorColumnConfig` object. If a string is specified this is used as the display name for the column. If a config object is specified this will be used as configuration for the column. If neither is specified then the column name is set to ***column***.

We could thus specify the relationship as follows in groovy code:

```
import net.datenwerke.rs.grideditor.service.grideditor.definition.db.*

def definition = GLOBALS.getRsService(DbGridEditorDefinition.class)

def adapter = definition.getAdapter()
adapter.setTableName("Products")
adapter.setPrimaryKey('productNumber')
adapter.addColumns('productNumber', 'productName')
           .addForeignKeyColumn('productCategory','Categories','categoryNumber',' ↙
    ↳ categoryName','Category')
           .addColumns('productSupplier')

return definition;
```

Similarly, you could additionally define the foreign key relationship also for the productSupplier column.

Usually, you would additionally define an editor that allows users to select a category. For this,

you can specify a config object that is used as fifth argument. For example something along the following lines (note also the additional imports).

```
import net.datenwerke.rs.grideditor.service.grideditor.definition.db.*
import net.datenwerke.rs.grideditor.service.grideditor.definition.*
import net.datenwerke.rs.grideditor.service.grideditor.definition.editor.*

import groovy.sql.Sql

def definition = GLOBALS.getRsService(DbGridEditorDefinition.class)

def adapter = definition.getAdapter()
adapter.setTableName("Products")
adapter.setPrimaryKey('productNumber')

// define config for categories
def categoryConfig = new GridEditorColumnConfig(displayName: 'Category')
def categoryEditor = new TextSelectionListEditor()
def connection = definition.getConnection(report)
try{
    new Sql(connection).eachRow('SELECT categoryName AS name FROM Categories ORDER BY 1' ){
        categoryEditor.addValue(it.name)
    }
} finally{
    connection.close();
}
categoryConfig.setEditor(categoryEditor);

adapter.addColumns('productNumber', 'productName')
           .addForeignKeyColumn('productCategory','Categories','categoryNumber','categoryName', ↙
      ↳  categoryConfig)
           .addColumns('productSupplier')

return definition;
```

Complex Display Expressions

In the above example we used the attribute categoryName to display category names instead of category numbers. Assume, that we wanted to display the suppliers as LASTNAME, FIRSTNAME. For this we can use SQL expressions in place of the display expression. For example, in MySQL to select LASTNAME, FIRSTNAME we can use the following query

```
SELECT CONCAT(supplierLastName, ", ", supplierFirstName) FROM Suppliers
```

Our example would, thus, change as follows (note the changes in the second to last line)

```
import net.datenwerke.rs.grideditor.service.grideditor.definition.db.*
import net.datenwerke.rs.grideditor.service.grideditor.definition.*
import net.datenwerke.rs.grideditor.service.grideditor.definition.editor.*

import groovy.sql.Sql

def definition = GLOBALS.getRsService(DbGridEditorDefinition.class)

def adapter = definition.getAdapter()
adapter.setTableName("Products")
adapter.setPrimaryKey('productCode')

// define config for categories
def categoryConfig = new GridEditorColumnConfig(displayName: 'Category')
def categoryEditor = new TextSelectionListEditor()
def connection = definition.getConnection(report)
```

```
try{
    new Sql(connection).eachRow('SELECT categoryName AS name FROM Categories ORDER BY 1' ){
        categoryEditor.addValue(it.name)
    }
} finally{
    connection.close();
}
categoryConfig.setEditor(categoryEditor);

adapter.addColumns('productCode', 'productName')
           .addForeignKeyColumn('productCategory','Categories','categoryNumber','categoryName', ↙
      ↳  categoryConfig)
           .addForeignKeyColumn('productSupplier','Suppliers', 'supplierNumber', 'CONCAT({{table}}. ↙
      ↳  supplierLastName, ", ", {{table}}.supplierFirstName)','Supplier')

return definition;
```

The change should be somewhat unexpected as we additionally added the string `{{table}}` twice. This becomes necessary since behind the scenes ReportServer needs to create a complex SELECT statement that joins together the foreign key tables. The replacement `{{table}}` is then used to plug in the correct temporary table name. Of course, we could also for the supplier add an editor. Here we would not need `{{table}}` replacement. Instead, the code for the editor is straight forward:

```
def supplierEditor = new TextSelectionListEditor()
def connection = definition.getConnection(report)
try{
    new Sql(connection).eachRow('SELECT CONCAT(supplierLastName, ", ", supplierFirstName) AS name FROM  ↙
      ↳  Suppliers ORDER BY 1' ){
        supplierEditor.addValue(it.name)
    }
} finally{
    connection.close();
}
```

Foreign Keys and the Fluid API

Of course, you can also define foreign key columns via the fluid API. For this, use the `fk` ↙ ↳ method. Additionally, to add the "default" selection list editor for a foreign key, i.e., the selection list that displays all possible choices you can use the `withFkEditor` method. To further fine tune the selection you may us the `withFkEditorWhere(...)` method which takes as input a where clause that is added to the underlying SQL query.

```
def adapter = gridHelper.initDbGridEditor()

adapter.configure(report, 'Products')
    .setPk('productCode')
    .columns()
        .add('productCode')
        .add('productName')
        .add('productCategory')
           .fk('Categories','categoryNumber','categoryName')
           .withFkEditor()
        .add('productSupplier', 'Supplier')
           .fk('Suppliers', 'supplierNumber', 'CONCAT({{table}}.supplierLastName, ", ", {{table}}. ↙
      ↳  supplierFirstName)')
    .done()

return adapter;
```

6.11 Executing Reports via the URL

If you wish to integrate reports in external applications, you can use a specific URL to directly link the report export.

http://SERVER:PORT/reportserverbasedir/reportserver/reportexport

Here the following parameters control the export:

`id`	Specifies the report (also refer to key).
`p_`	Can be used to specify parameters. After the underscore character, the parameter will be prompted by its key. `p_myparameter=abc|def`, for instance, can be used to set a list parameter to abc and def.
`key`	As an alternative to id, key can be used to select reports.
`format`	Defines the output format. Possible values are: html,pdf,csv,excel,png as well as `TABLE_TEMPLATE`.
`page`	Allows to export a single page.

Depending on the format, additional properties are available.

`TABLE_TEMPLATE`	The template to be used needs to be specified via its ID as `tabletemplate_id` or via its key as texttttabletemplate_key.
`csv`	The delimiter is controlled via `csv_sep`, the quote character can be specified via `csv_q`. Additionally, you can control whether or not to print a header line via the property `csv_ph`.

Specialties of the Dynamic List

In addition to the control options stated above, you can set further properties for dynamic lists:

`pagesize`	Defines the pagesize to be used when exporting single pages. For example `&page=2&pagesize=10` will select records 11 to 20.
`c_1`	For dynamic lists this option specifies the columns to output. Separated by the pipe symbol, an alias can be entered. The figure following the underscore specifies the sequence. `c_2=ID|fooID` specifies the second column to be the ID column with the alias fooID.
`allcolumns`	Can be specified instead of `c_` to select all columns.
`ac_1`	Like `c_1`, however, this option selects a computed column.
`agg_i`	Sets an aggregation for column i. Admissible values are: `AVG, COUNT, MAX, MIN, SUM, VARIANCE, COUNT_DISTINCT`
`h_i`	Hides the i-th column
`or_i`	Controls i-th column sorting. Admissible values are: ASC (ascending), DESC (descending).

`fi_i`	Allows to define inclusion filters for the i column. Multiple filter values can be separated by the pipe (|) symbol. Here you will find an example for a configuration: `fi_1=FILTER_A|FILTER_B|FILTER_C`
`fri_i`	Allows to define inclusion filter sections for the i-th column. Multiple sections will be separated by the pipe (|) symbol. To separate the section use space-dash-space ("-"), as it is known from the filter dialogue. Additionally, to define open intervals, start the filter with "-␣" or end it with "␣-" (where ␣ denotes a space).
`fe_i`	Like `fi_` however, it defines an exclusion filter.

Here you will find an example for a possible configuration (spaces in URLs will be coded as %20, further information on URL encoding you wilL find, for instance under http://www.w3schools.com/tags/ref_urlencode.asp):

http://127.0.0.1:8888/reportserver/reportserver/reportexport?id=4&c_1=ENTITY_ID|FOO_ID&fri_1=2%20-%205|-%207&c_2=action&c_3=key_field&h_1&or_1=DESC&format=pdf

Configuring Reports in ReportServer by URL

In addition to exporting reports by URL, you can directly open pre-configured reports in ReportServer by URL. Here the URL is

http://SERVER:PORT/reportserverbasedir/ReportServer.html#reportexec/

Here you can use the above parameters to pre-configure the report. Please keep in mind to separate parameters from the appurtenant value by setting a colon (:) (instead of using the equal sign "=").

The above parameters are supplemented by "v:preview" to directly jump to the preview. Here we give you an example for a possible configuration:

http://SERVER:PORT/reportserverbasedir/ReportServer.html#reportexec/key:customer&c_1:CUS_CUSTOMERNUMBER&c_2:CUS_CUSTOMERNAME&c_3:CUS_ADDRESSLINE1&c_4:CUS_ADDRESSLINE2&c_5:CUS_CITY&fi_1:187&v:preview

Embedding Reports Without Login

The functionality was redesigned in ReportServer 3.0 and configuration changes are necessary when upgrading from older versions.

In some cases it is helpful to execute reports without having to login first. Here ReportServer's solution is an easy-to-use servlet. The **httpauthexport** servlet allows to execute reports without the user being logged in. The URL to be used is:

http://SERVER:PORT/reportserverbasedir/reportserver/httpauthexport

Apart from the usual parameters, you have to enter

user	User name
apikey	An apikey that is defined as a user property

Define the API Key and Define Appropriate Permissions

To define an API key for a user go to the user management view and select the user in question. Then select the tab **Use Properties** and add a new property called **apikey**. As value, you can use any string (preferably a random string that is on the longish side).

The user does not require a password, or any unusual permissions. The only permissions required are the ***execute*** permission on those reports that you plan to embed. Suppose, that we have specified the apikey ***79PKXGScP8r8*** on a fresh user ***exportuser*** which has no permissions except the permission to execute report 5000. Then, when everything goes right, then

`http://SERVER:PORT/reportserverbasedir/reportserver/httpauthexport?id=5000&user=exportuser&apikey=79PKXGScP8r8&format=HTML&download=false`

should execute the report with id 5000 and user ***exportuser***.

Embedding the Report Execution View

In the previous section we have seen how to execute reports directly via the URL. It is also possible to detach the report execution view (i.e., including parameter configuration or the complete configuration for dynamic lists) to, for example, embed it into a portal. The syntax is analogously to the **reportexec** functionality described above. The base URL for embedding the report view is

`http://SERVER:PORT/reportserverbasedir/ReportServer.html#inlinereport/`

Thus, to display report with id 29 you would use the URL

`http://SERVER:PORT/reportserverbasedir/ReportServer.html#inlinereport/id:29`

If you only want to display the preview view, then you can add the "type:preview" parameter, that is

`http://SERVER:PORT/reportserverbasedir/ReportServer.html#inlinereport/id:29&type:preview`

You can even specify exactly which views to display. Assuming that report 29 is a dynamic list, then the following would select the list config as well as the preview

`http://rstest.datenwerke.net/ReportServer.html#inlinereport/id:29&views:listconfig|preview`

The following views are available

parameter	The parameter view.
computedcolumns	The computed columns view of dynamic lists.

prefilter	The pre filter view of dynamic lists.
listconfig	The list configuration view of dynamic lists.
preview	The preview view

Note that you can still completely configure the report via the URL as seen in the following example where we configure a dynamic list:

http://SERVER:PORT/reportserverbasedir/ReportServer.html#inlinereport/key:customer&c_1:CUS_CUSTOMERNUMBER&c_2:CUS_CUSTOMERNAME&c_3:CUS_ADDRESSLINE1&c_4:CUS_ADDRESSLINE2&c_5:CUS_CITY&fi_1:187&views:preview

6.12 Report Properties

Report properties provide a means to further customize how report server treats reports. You can access report properties via the report management perspective in the administration module by selecting a report and then selecting **Report Properties**. You are then presented with a grid that allows to view and change the current properties for that particular report.

Tip. Report properties on a base report are inherited by all of its variants.

Report properties are simple key value pairs. You can add a new property, by clicking on **add** and remove existing ones by selecting them and clicking **remove**. Note that all changes are only committed once you hit the **Save** button in the toolbar.

Available Report Properties per Type

All Report Formats

output_format_auth	A comma separated list of available output formats for that particular report

The Dynamic List

ui:preview:count:default	Controls whether the report preview view directly starts counting the number of rows in the result. (Defaults to auto)
ui:filter:count:default	Controls whether filter views attempt to count the number of valid results. (Defaults to auto)
ui:filter:consistency:show	Controls whether the link/unlink button in the filter view is visible. (Defaults to true)
ui:filter:consistency:default	Controls whether link mode is per default enabled for filters. (Defaults to enable)

6.13 Report Metadata

It is often very useful to save additional data with the report object. Depending on the process, it might be helpful to save when and by whom a report was checked and accepted, or, maybe you wish to file the link leading to the documentation of a report.The type of metadata required, therefore, depends to a large extent on the processes applied in an enterprise.

ReportServer supports you to file report specific metadata by allowing to maintain a key value list for each report. In Report management you can recall it in the relevant report by clicking on the **Metadata** tab. With the **Add** and **Remove** buttons new key pairs can be added or removed. By clicking on the arrow next to the **Add** button you may choose from a list of key words already used.You can directly and easily edit the values in the list.

Using Metadata as Parameters

Available metadata can be used in reports as parameters. To do this, the following replacements are available.

`_RS_METADATA_NAME`	Includes the value for the key NAME.
`_RS_METADATA_SUPER_NAME`	The hierarchic structure of reports and variants allows to overwrite metadata of a report in the variant. To access the value that was overwritten, use the following replacement.

With the `${}` formula language, you can additionally use the replacement `_RS_METADATA`. It enables to access a HashMap including the existing metadata and relevant values.

6.14 Drill Down Reports

ReportServer provides support in creating drill down and drill across reports with Jasper- Reports and BIRT. For Drill Down and Drill Across operations, reports will be linked with each other to show more detailed information on specified data. ReportServer provides the special parameters `_RS_BACKLINK_ID` und `_RS_BACKLINK_URL` to each report execution . So when you are in a Jasper-Report and you want to link it to the report with the key **myKey** enter the link as follows:

```
"http://localhost:8888/reportserver/reportserver/reportexport?key=myKey&"
+ "format=html&bid=" + $P{_RS_BACKLINK_ID}
```

Of course, you have to replace the first part of the link by the server address. By entering the parameter **bid** you can easily set a **backlink**. You will get the required URL via the parameter `_RS_BACKLINK_URL`. If there is no backlink, the parameter will be empty.

Chapter 7

Global Constants

Global constants can be regarded as static text parameters. To create a global constant, go to the Administration module and there to the sub module **Global Constants**. Click on **Add constant** to create a new constant. Then enter the name and value of the constant by mouse click in the respective cell. To delete one or more constants, select the relevant constants and actuate the **Remove** button from the tool bar. To remove all constants, click on the arrow beside the **Remove** button and select **Remove all**.

In reports constants can be used as parameters. Once you have created a constant with the name `KONST_MY_CONSTANT`, you can work with the replacements `${KONST_MY_CONSTANT}` and `$ ↙ ↳ !{KONST_MY_CONSTANT}` in dynamic lists. As it applies with parameters, please have in mind to distinguish between `$` and `$!` where the replacement `$` will be used in a query, and S! directly. For further information refer to the paragraph Parameters in the Dynamic List section. When used in other report engines (e.g. BIRT or Jasper), the syntax applicable there must be considered. For Jasper it is for instance `$P{KONST_MY_CONSTANT}`. For further information refer to the relevant sections.

Chapter 8

User Variables

User variables are designed to adapt a report to the needs of the user executing it. By applying user variables, the same report holds a different base data entirety for different users.

We would like to explain user variables based on an example. The employees of an enterprise are distributed to two locations. They shall be given access to the sales figures of the enterprise, however, only to the data relevant for their respective location. You can model this requirement by establishing two separate reports (which are almost identical), and by granting the employees of location A access to the one report and the employees of location B to the other report. But you can also design a single report, and by applying user variables restrict the base data entirety per each executing user. Let us assume the user variable OFFICE was created, and the employees of location A have set it to "A", the employees of location B to "B". Now, you can create a parameter of type **User variable** for the report and select the variable `OFFICE` that you have defined before. (In the following example we assume to proceed on the basis of a dynamic list and a relational database.) Now, the query for the report can be designed as follows:

```
SELECT * FROM SALES WHERE SALES_OFFICE = ${OFFICE}
```

8.1 Defining User Variables

Before you can apply user variables, you have to create them.To do this, go to User management and select the root node (User root). Use the tab **User variable management** to add new user variables. You can choose from the following types:

Text variable	Allows to set plain text
List	Allows to set multiple values

When defining user variables you have to assign a name, and you can add a description at your discretion. To do this, click on the respective cell. You can define further properties for the variable by activating it and clicking on **Edit** in the tool bar.

8.2 Allocating User Variables

User variables are allocated to users directly from the User management. However, you can also assign user variables to organisational units. Here the user variable will be passed on to all users within the organisational unit. So it is possible to specify at the top organisational unit (User root) default values for variables which will overwrite the lower levels.

To specify user variables, select a user or an organisational unit and switch to the **User variables** tab. In the bottom table you see user variables which apply with this user/organisational unit by inheritance, and in the upper table you see the user variables specified here. To set a variable, click on **Add** and then select the desired variable. To specify the value of the variable, double click in the respective line of the table.

For text variables, enter the value in the entry field. For list parameters, you can enter the single values by separating them with | (pipe symbol).

8.3 Using User Variables in Reports

To use user variables in reports, create a parameter of type user variable and allocate the respective user variable to it. Now you can work with the parameter in the usual way. The replacement either hides an object of Type String (for a text user variable) or an object of Type Set <string> (for a list variable).

Chapter 9

Import and Export

You can export various objects of a ReportServer instance in order to save them or to import them to another instance. You will find the export options in the Administration section under the respective object (e. g. a report). You can not only export single objects but also entire sub-trees. With this you can move the complete report inventory of an installation to another one.

When exporting a ReportServer object, all reference objects required for using the object will be exported, too. So a report will, for example, be accompanied by the respective data source.

9.1 Exporting

To export an object or a sub-tree, open the respective section in the Administration module (e.g. Report management). Click on the object or the superordinate folder of the sub-tree to export. In the toolbar you will find the **Quick export** button. When exporting a report, you can decide whether or not to include variants in the export. As soon as the export has been finalized (consider that this may take some time for more comprehensive sub-trees) ReportServer will ask whether to immediately display, or to download the export. Direct download as a zipped archive is recommended for comprehensive exports.

Export Format

ReportServer exports objects in an XML dialect. This allows to edit export files with common XML tools. In addition, it ensures to easily adapt export files to later versions of ReportServer.

9.2 Importing

To upload exported objects to ReportServer, in the Administration module go to the **Import** section, and then click on **Start import** in the toolbar. In the window that opens you can either upload an export file (as a `.zip` file) or directly enter an exported XML file.

After the upload of the exported data you will see the object types included in the export in the left part of the window. There are differing import options available depending on the section.

After having successfully completed the configuration, in the toolbar click on **Finalize import** to execute the import. By clicking on **Cancel process** you can interrupt the current import process. To reset the configuration, select **Reset**. Please bear in mind that the import configuration will not directly be discarded after a successful import. Therefore, you can comfortably import multiple objects from an export file.

Importing Reports

Select the target folder for the import. From the **Objects tab**, select the reports to import. To avoid double key fields, remove the key field from all reports to import. To do this, tick the respective option. Additionally, you may add imported variants directly to a TeamSpace. From the tool bar select the **Import options** button and configure the respective TeamSpace.

Remember that reports include further objects such as data sources. They will only be im- ported if you set the relevant configuration options (refer to Importing of data sources). If you fail to set these options, the relevant fields will remain empty after the import.

Importing of Data Sources

Data sources are imported in the same way as reports. In the settings you define a target folder and from the **Objects tab** you select the objects to import. You have the option to specify a **Default data source** which will be entered at all locations where you want to avoid to import a data source that was included in the export. For instance, if you run a test and a production system and you wish to move reports from the test system to the production system, the underlying data source is usually the one of the production system. Select the production data source via the option **Default data source**; it will then be entered in all relevant reports and parameters while importing.

Importing Users and Files

Users or files are imported in the same way as reports. Here there are no further options available.

Chapter 10

Scheduling of Reports

ReportServer supports the timed execution of reports. In this section we will discuss how to view existing jobs with the administration interface as well as how to configure conditional schedule jobs. For a description of how to configure the e-mail server as well as the notifications about completed (or failed) executions refer to the ReportServer configuration guide.

The Scheduling module enables the user to browse their own jobs, to change or archive them. In addition, the module offers a variety of information on each job such as given times for next executions, or information on possible errors in previous executions. A detailed description on how to work with the module is given in the ReportServer user manual in section Scheduling.

The Administrator can call up an extended form of the Scheduling module via the Administration module. Here you see the jobs of all users, and, if required, may change or remove them. Please consider that after having changed a scheduler job, your user account will be entered in **Scheduled by**, and the user who originally created the job cannot edit it afterwards any more.

10.1 Technical Backgrounds to Scheduler Jobs

In the following we want to give a few technical details on scheduler job (or entries). A scheduler entry consists of a so-called job and one or more actions each. Here the job describes how to execute the report (further jobs could be the time controlled execution of scripts; refer to section "„Scheduling of reports“"), whereas actions describe what to do with the result (the completed report). At present, there are two options: Sending by e-mail or storing the result in a TeamSpace for later retrieval. Often instead of speaking of a scheduler entry (i.e., job plus actions) we simply speak of a scheduler job.

Errors may occur here on all levels. Running a job just as well as running individual actions might fail. If a report is not executable (for example, because the underlying data model has changed), the job cannot be executed. An example for a possible failure on the action level is that the e-mail server cannot be reached which would interrupt the sending of report results.

To display detailed information on successful or failed scheduler entries, from the list (on the left) select the respective entry and then double click in the details pane on a specific execution. In the pop-up window that opens, you will be given detailed information on the execution split up in

job and actions. If there is a failure here you will be given a stack trace of the execution to draw conclusions as to what might be the failure cause.

If, for instance, the connection to the data source is not available at the moment of execution, ReportServer will produce an error message similar to the following one:

```
net.datenwerke.rs.core.service.reportmanager.exceptions.ReportExecutorException: The
report could not be executed: Could not open connection to: jdbc:mysql://demo.db.raas.
datenwerke.net:3306/ClassicModels with user: demo. java.sql.SQLException: Timed out
waiting for a free available connection.
```

10.2 Filtering by the Status of a Job

From the filters in the tool bar you can search for a specific job by its failure status. Report- Server jobs are always in one of 4 statuses.

Inactive	The job is currently inactive If another execution at a later time is pending, the job will be selected at that time by the disposition module and prepared for execution.
Waiting to be executed	If a job has been selected by the scheduler for execution, it will change to the status “Waiting to be executed”. It will remain in this status until a free worker thread will start the execution.
Executing	The job is being executed.
Critical failure	This status will be set if an unforeseeable error occurred that requires manual action. The job will be exempt from further execution until the status is manually restored. To reset the status to "Inactive", click on the respective job and in Details double click on the field Execution status.

10.3 Notifications

After a scheduler job was executed ReportServer sends a notification by e-mail. If the report was scheduled into a TeamSpace, you can add a link to the completed report. Configure notification texts in the configuration file etc/scheduler/scheduler.cf. For further information on the configuration refer to the ReportServer configuration guide.

10.4 Conditional Scheduling

Beside regular scheduling of reports, ReportServer also supports conditional scheduling of reports. This allows users to specify requirements that will be checked before execution and only if they hold the job will be executed (refer to the ReportServer user manual section Scheduling). Possible ReportServer conditions will be configured with dynamic lists that have a single result line. Here is such a dynamic list that could be used as a scheduler condition:

A	B	C	D
5	17	23	42

The list consists of four attributes `A, B, C, D`. If the list has been configured as a scheduler condition, users now can define conditions based on this list. `${}` formula expressions define the conditions (here the user does not have to delimit the expression by "`${`" and "`}`". A feasible condition based on the list available would for instance be:

```
${A > 10 && D != 1}
```

Now, prior to execution, the expression would be compared to the current report values. Only if the condition evaluates to TRUE, the job will be executed.

To configure dynamic lists as a basis for conditional scheduling, use the terminal command **rcondition** (see Chapter 14 Terminal Commands).

10.5 Enhanced Scheduling

The Terminal enables to manually terminate or restart the Scheduler. In addition, it provides the option to schedule timer controlled scripts. For further information refer to Chapter Terminal commands (command **scheduler**) as well as to Chapter 13 ReportServer Scripting.

Chapter 11

Theming

ReportServer Enterprise Edition comes with a simple theming mechanism that allows you to adapt the look and feel of ReportServer and easily integrate adapt it to fit your corporate identity. The theming is controlled via the configuration file `/fileserver/etc/ui/theme.cf`. This file could look like

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <theme type="default">
    <header>
     <height>40</height>
    </header>
    <logo>
      <login>
       <html><![CDATA[<i class="icon-rs-logo rs-login-logo"></i><span class="rs- ↙
      ↪  login-bg"><i class="icon-rs-logo-square"></i></span>]]></html>
       <width>200px</width>
      </login>
      <header>
          <html><![CDATA[<span class="rs-header-logo"><i class="icon-rs-Report"></ ↙
     ↪  i><i class="icon-rs-Server"></i></span>]]></html>
       <width>185px</width>
      </header>
      <!--<url>Some URI pointing to a Logo</url> -->
    </logo>

    <colors>
      <color name="white" color="#FFFFFF"/>
      <color name="black" color="#000000"/>
      <color name="black-almost" color="#132834"/>

      <color name="purple-dark" color="#3E4059"/>
      <color name="purple-light" color="#DFE0EB"/>

      <color name="gray-light" color="#EEEEEE"/>
      <color name="gray-dark" color="#B8BDC0"/>
      <color name="gray-very-dark" color="#6D708B"/>
```

```
      <color name="terminal-green" color="#00B000"/>
    </colors>

    <colorMapping>
      <map useFor="bg" colorRef="gray-dark"/>
      <map useFor="bg.text" colorRef="black"/>

      <map useFor="bg.light" colorRef="white"/>
      <map useFor="light.text" colorRef="black"/>

      <map useFor="bg.shaded" colorRef="gray-light"/>
      <map useFor="shaded.text" color="#666666"/>

      <map useFor="hl.dark.bg" colorRef="purple-dark"/>
      <map useFor="hl.dark.text" colorRef="white"/>

    <map useFor="hl.light.bg" colorRef="purple-light"/>
      <map useFor="hl.light.text" colorRef="black"/>

      <map useFor="header.bg" colorRef="black-almost"/>
      <map useFor="header.text.active" colorRef="white"/>
      <map useFor="header.text.inactive" color="#BBBBBB"/>
    <map useFor="header.text.right" color="#BBBBBB"/>

      <map useFor="terminal.bg" colorRef="black"/>
      <map useFor="terminal.text" colorRef="terminal-green"/>
      <map useFor="terminal.hl.bg" colorRef="gray-very-dark"/>
    <map useFor="terminal.link" colorRef="white"/>

      <map useFor="border.light" colorRef="gray-dark"/>

      <map useFor="tbar.btn.bg" colorRef="gray-dark"/>

      <map useFor="icon.light" color="#999999"/>

    </colorMapping>

    <css>
      .icon-rs-Report {
        color: #FFF !important;
      }
    </css>
  </theme>
</configuration>
```

The configuration file consists of four parts. In the first part, you can replace the logos for the login screen, the main ReportServer screen and the documentation report. The logo used within the report documentation is either the ReportServer logo, or an image that is specified via `${logo ↙ ↳ .url}` in the config.

The second part of the configuration allows to provide names for colors which can then later be used to change the color scheme of ReportServer. In part three, (colorMapping) you can tell ReportServer what colors to use for certain elements. Colors can either be set via reference to a previously named color:

```
<map useFor="header.text" colorRef="white"/>
```

This sets the ***header.text*** element to the color specified as ***white***. They can be set directly

```
<map useFor="lighter.bg" color="#FFFFFF"/>
```

or they can be set by pointing to a different element. For example,

```
<map useFor="lighter.text" sameAs="light.text"/>
```

ensures that any element that uses the color ***lighter.text*** uses the same color as an element using ***light.text***. The following elements are currently available to be styled:

bg	The background.
text	Text when on background (bg).
bg.light	Light variant of background. For example used as background of panels.
light.text	Text on light background.
bg.shaded	A shaded variant of the background. Used, for example, as the background for toolbars.
shaded.text	Text on shaded background.
bg.dark	A darker variant of the background color.
border.light	A color used for (thin) borders on light background.
header.bg	The background of the top module bar (the header).
header.text.active	Text color of active modules and logo.
header.text.inactive	Text color of inactive modules.
header.text.right	Text color of user name and profile.
h3> hl.dark.bg	A dark highlight color.
hl.dark.text	Text on the dark highlight.
hl.light.bg	A lighter highlight color.
hl.light.text	Text on the lighter highlight color.
tbar.btn.bg	Background color of buttons in toolbars.
terminal.bg	Background of the terminal.
terminal.hl.bg	highlighted background of the terminal.
terminal.link	Links on the terminal.
terminal.text	Standard text on the terminal.

With the next versions we are planning on further fine-tuning the color classes and would be happy for any feedback you might have.

Finally, in the last part of the config, you can set individual CSS rules. As these rules are inserted after any other ReportServer CSS rule, you can overwrite any ReportServer specific CSS.

Chapter 12

Terminal

The ReportServer Terminal is a mighty tool for administrators. You open it with the keyboard shortcut CTRL+ALT+T (Please consider that you need the respective rights; refer to Chapter 3 User and Permission Management).

The ReportServer Terminal follows the shell of a Unix system. It includes a command prompt as well as a history of commands and the respective outputs. Beside general commands, the Terminal provides access to almost all ReportServer objects via the Virtual file system. In addition, the Terminal can be used to enter and execute scripts.

12.1 Using the Terminal

Enter commands at the command prompt and confirm with ENTER. The following command:

```
reportserver$ hello
```

will return the reply

```
> hello user
```

There are two different types of commands. Simple commands will provide their output directly in the Terminal. This applies for the majority of commands. Beside the simple ones there are also commands requiring further user interaction. After calling up this type of command, the command prompt will change and display a single opening angle bracket. The eliza program represents an example for this command type (originally by Joseph Weizenbaum, and here in an adaptation by Jesper Juul). You close complex commands with CTRL+C, or according to eliza also by entering the leave-taking expression: bye

```
reportserver$ eliza
*************************
ELIZA
CREATIVE COMPUTING
MORRISTOWN, NEW JERSEY
ADAPTED FOR IBM PC BY
PATRICIA DANIELSON AND PAUL HASHFIELD
Java version February 24th, 1999
By Jesper Juul - jj@pobox.com.
```

```
************************
> Hi, my name is Bob.
HI! I"M ELIZA. WHAT"S YOUR PROBLEM?
I DON"T CARE ABOUT NAMES --PLEASE GO ON.
> I do not know. Isn"t that why I am here?
HOW LONG HAVE YOU BEEN HERE?
> About one minute or so.
WHAT DOES THAT SUGGEST TO YOU?
> I guess you cannot help me after all.
OH, I CANNOT HELP YOU AFTER ALL.
> bye
Good Bye
reportserver$
```

The `-h` flag (or `-?` flag) offers help to a command listing the individual options. When pressing the TAB key, ReportServer attempts to complete the command presently entered. If it fails to do so because there are several options, these will be shown to you by pressing the TAB key again. With the arrow keys UP/DOWN you browse through the history of your commands. To clear the Terminal window, use the `clear` command.

You can leave the Terminal by entering the `exit` command.

You will find a command overview including a short description of all available commands in Chapter 14.

Tip: To copy data from the Terminal window, press the CTRL key when selecting the data. Otherwise, the cursor in the command prompt will be activated when clicking on the window.

12.2 The Virtual File System

ReportServer integrates various objects (reports, data sources, files, etc.) as virtual file systems in the Terminal. Each object tree (e. g. the report section) has here an own root node. When you open the Terminal you first arrive at the top level. Here you find the root nodes of the various virtual file systems. By entering the command `ls` you can display them.

```
reportserver$ ls
datasources     Data source management
reportmanager    Report management
dadgetlib      Dadget library
fileserver      File system
tsreport       TeamSpaces
usermanager     User management
```

Here `ls` shows the objects in the current directory. By entering the command `pwd` the current directory will display.

```
reportserver$ pwd
/
```

Use the command `cd directory` to change the directory. Multiple folders will be separated by / (slash). So, by entering

```
cd "reportmanager/Dynamic Lists/"
```

you can switch to the folder **Dynamic lists** in the report management tree. Please observe to set quotation marks when there are objects that include spaces in their name. When you enter `mkdir`, you can create a new sub-folder.

```
reportserver$ mkdir newFolder reportserver$ ls -l
12 T_AGG_ORDER - Basis TableReport   TableReport
17 T_AGG_PAYMENT - Basis        TableReport
22 T_AGG_CUSTOMER - Basis        TableReport
26 T_AGG_PRODUCT - Basis        TableReport
33 T_AGG_EMPLOYEE - Basis        TableReport
39 T_AGG_ORDER - Parametrized      TableReport
49 newFolder               ReportFolder
```

This is the output of `ls -l` which returns name and object as well as ID and type. Now, to move all reports to the new folder, use

```
reportserver$ mv T_AGG_* newFolder/
reportserver$ ls -l newFolder
17 T_AGG_PAYMENT - Basis       TableReport
33 T_AGG_EMPLOYEE - Basis      TableReport
12 T_AGG_ORDER - Basis        TableReport
22 T_AGG_CUSTOMER - Basis      TableReport
39 T_AGG_ORDER - Parametrized    TableReport
26 T_AGG_PRODUCT - Basis       TableReport
```

Be aware that the mv command at present expects the second parameter always to be a directory in contrast to the Unix equivalent. This is why it cannot be used to rename objects.

Now we switch to the FileServer and create a temporary directory:

```
reportserver$ cd /fileserver/
reportserver$ mkdir tmp
reportserver$ cd tmp
reportserver$ pwd
/fileserver/tmp
```

Unlike typical file systems, ReportServer allows two objects having the same name in one folder. For example, two reports with identical names will be in the same report section. We can simulate this by creating two folders with the same name:

```
reportserver$ mkdir test
reportserver$ mkdir test
reportserver$ ls -l
124   test  FileServerFolder
125   test   FileServerFolder
```

Now, what will happen if we change to the test folder by entering `cd test`?

```
reportserver$ cd test
reportserver$ pwd
/fileserver/tmp/test
```

We have switched to the test folder. But to which one did we switch? Internally, ReportServer saves paths not via their names, but via the ID of objects which are unique in the individual virtual file systems. By entering `pwd -i` they will display.

```
reportserver$ pwd -i
/fileserver/id:123/id:124
```

As you see, we changed to the first test folder with the ID 124. To get to the second one with the ID 125, use the internal path:

```
reportserver$ cd ../id:125
reportserver$ pwd
/fileserver/tmp/test
reportserver$ pwd -i
/fileserver/id:123/id:125
```

In FileServer (i.e. in the virtual file system which represents ReportServer FileServer) we can use the commands `createTextFile` and `editTextFile` to create or edit text files.

```
reportserver$ createTextFile text.txt
file created
```

By entering `cp` now we can copy this text to other folders.

```
reportserver$ cp text.txt ..
reportserver$ cd ..
reportserver$ ls -l
124   test     FileServerFolder
125   test    FileServerFolder
127   text.txt   FileServerFile
```

To delete objects, use the `rm` command. Here wildcards can be used.

```
reportserver$ rm tes*
/fileserver/tmp/test has children
```

Here, ReportServer stated that the folder to delete still holds objects. To include them in the deletion, use `rm -r`.

```
reportserver$ rm -r tes*
reportserver$ ls -l
127   text.txt   FileServerFile
```

12.3 Assigning Aliases

For some recurring commands it might be useful to define shortcuts (aliases). To do this, use the configuration file `etc/terminal/alias.cf` (You will find it in the ReportServer File system, from the Terminal you will access it by entering `/fileserver/etc/terminal/`). If the file does not exist, create the respective directories by entering `mkdir` and the file with `createTextFile`.

The file is structured as follows

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

```
    <entry>
      <alias>ll</alias>
      <command>ls -l</command>
    </entry>
    </cmdaliases>
</configuration>
```

Here, alias **ll** was assigned to the command `ls -l`. To reload the aliases, use:

```
reportserver$ config reload
configuration reloaded
reportserver$ updateAlias
alias table updated
```

First, the cache which stores all configuration files will be cleared by entering `config reload`. Then you can re-enter the alias configuration with the command updateAlias. After a Report- Server restart it will be loaded automatically.

12.4 Scripts

A main task of the Terminal is to manage and execute scripts. In ReportServer scripts can be used for various purposes. Firstly, they can be used to create dynamic or complex reports. As they have access to the Java runtime environment and with this to the available metadata, scripts are perfectly suited for reports analyzing the system status. An example for a script report is the documentation report (for further information on script reports refer to Section 6.9). In addition, scripts can also be used for administrative tasks, or even to expand the functionality of ReportServer. Within the scope of this Administrator manual, we will give you some insight into the world of ReportServer scripts in the following section. You will find a detailed treatment including examples and concepts in our script guide.

Warning: The right to write scripts allows a user to execute arbitrary code. This right should thus only be granted to trusted system administrator's.

Chapter 13

ReportServer Scripting

ReportServer Enterprise Edition comes with scripting support. Scripts can be used in a variety of ways and can, for example, serve as a basis for reports and data sources, perform administrative tasks, or even contribute new functionalities. In this section we want to give you an introduction to the comprehensive script subject, and make you familiar with the various options provided by ReportServer scripts. Scripts and their fields of use will be treated in the ReportServer script manual in detail.

ReportServer scripts are written in Groovy (http://groovy-lang.org/) and run in the same VM where ReportServer is located. The decisive advantage here is that the scripts have access to the complete set of services provided by ReportServer. However, it also means that scripts represent a potential security and stability risk. Persons who are authorized to write or change scripts have full access to the system. The permissions to write scripts should thus be granted with care.

The following explanations address persons with basic knowledge in programming. Java and/or Groovy experience are not necessarily required but might be helpful. Under http://groovy.codehaus.org you will find many excellent tutorials on programming in Groovy.

Scripts are located in the internal file manager, by default beneath folder `bin`. Several configuration options can be configured in configuration file `/etc/scripting/scripting.cf` (also see the ReportServer configuration guide).

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <scripting>
    <enable>true</enable>
    <restrict>
      <location>bin</location>
    </restrict>
    <startup>
      <login>fileserver/bin/onlogin.rs</login>
      <rs>fileserver/bin/onstartup.rs</rs>
    </startup>
  </scripting>
</configuration>
```

The first option allows the global deactivation of scripts. Please consider that if you apply it, you will not be able to use script reports anymore, and that the documentation report available in the demo data will not work any longer. By the second option (restrict.location) you will define a root folder in which scripts have to be filed. This allows to give access rights for the file system to individual users without enabling them to create or change scripts. The last two options allow to configure two special scripts. As soon as a user has signed on, the `onlogin` script will run with the rights of this user. Here, for instance, interface enhancements can be loaded. To do this, ensure to grant users the execute right for the script. The start-up script, on the other hand, will run when starting the system. Note that then the script is run without any logged in user. Instead of specifying scripts, you can also specify folders. In this case, all scripts within the folder are executed.

13.1 A first Hello World

In the following we want to present a simple sample script. Open the terminal (press `CTRL+ALT+T`) and switch to the directory `fileserver/bin` (if you failed to create this directory so far, create it by using `mkdir`).

```
reportserver$ cd fileserver/bin
```

Now, create a `tmp` directory and switch to it.

```
reportserver$ mkdir tmp
reportserver$ cd tmp/
```

We can now create your a first ReportServer script by issuing `createTextFile hello1.rs`. The extension `.rs` has no relevance here, although, it has established as a standard for scripts (beside `.rs`, `.groovy` is frequently used):

```
reportserver$ createTextFile hello1.rs
file created
```

A pop-up window opens to edit the newly created file. For our first simple Hello World script we only want to induce the script to return Hello World. By default, scripts return the result of the last statement. So we can write the Hello World script simply by entering:

```
"Hello World"
```

Close the dialogue by clicking on the Submit button. With the "exec" command you can run the script.

```
reportserver$ exec hello1.rs
Hello World
```

13.2 How to Handle Errors

Before we continue, we will have a brief look at an error case. If the execution of a script fails, ReportServer will print an error message which tries to pin point the error. Let us consider the following simple script. Here we forgot to place the closing quotation mark:

```
return "Hello World
```

Here the error message would be as follows:

```
reportserver$ exec helloFail.groovy
Script execution failed.
error message: startup failed:
Script2.groovy: 1: unexpected char: 0xFFFF @ line 1, column 20.
return "Hello world
^

1 error
(java.util.concurrent.ExecutionException)
script arguments:
file: helloFail.groovy (id: 10074, line 1)
line number: 1
col. number: 20
```

Here the error message points to the problem: there is an unexpected character in line 1. In some cases, however, the error message might be insufficient to pinpoint the problem. In this case, you can tell ReportServer to print a detailed stack trace of the execution by running the script with the `-t` flag. In this case the output would be similar to the following

```
reportserver$ exec -t helloFail.groovy
net.datenwerke.rs.scripting.service.scripting.exceptions.ScriptEngineException: javax.script. ↙
      ↳  ScriptException: org.codehaus.groovy.control.MultipleCompilationErrorsException: startup failed ↙
      ↳  :
Script17.groovy: 1: unexpected char: 0xFFFF @ line 1, column 20.
return "Hello world
^

1 error

------- SCRIPT ERROR INFO -------
Script execution failed.
error message: startup failed:
Script17.groovy: 1: unexpected char: 0xFFFF @ line 1, column 20.
return "Hello world
^

1 error
(java.util.concurrent.ExecutionException)
script arguments:
file: helloFail.groovy (id: 10074, line 1)
line number: 1
col. number: 20

at net.datenwerke.rs.scripting.service.scripting.engines.GroovyEngine.eval(GroovyEngine.java:107)
at net.datenwerke.rs.scripting.service.scripting.ScriptingServiceImpl.executeScript(ScriptingServiceImpl ↙
      ↳  .java:217)
at net.datenwerke.rs.scripting.service.scripting.ScriptingServiceImpl.executeScript(ScriptingServiceImpl ↙
      ↳  .java:263)
at net.datenwerke.rsenterprise.license.service.EnterpriseCheckInterceptor.invoke( ↙
      ↳  EnterpriseCheckInterceptor.java:35)
at net.datenwerke.rs.scripting.service.scripting.ScriptingServiceImpl.executeScript(ScriptingServiceImpl ↙
      ↳  .java:317)
at net.datenwerke.rsenterprise.license.service.EnterpriseCheckInterceptor.invoke( ↙
      ↳  EnterpriseCheckInterceptor.java:35)
at net.datenwerke.rs.scripting.service.scripting.ScriptingServiceImpl.executeScript(ScriptingServiceImpl ↙
      ↳  .java:288)
at net.datenwerke.rsenterprise.license.service.EnterpriseCheckInterceptor.invoke( ↙
      ↳  EnterpriseCheckInterceptor.java:35)
at net.datenwerke.rs.scripting.service.scripting.terminal.commands.ExecScriptCommand.doRollbackExecute( ↙
      ↳  ExecScriptCommand.java:335)
at com.google.inject.persist.jpa.JpaLocalTxnInterceptor.invoke(JpaLocalTxnInterceptor.java:66)
at net.datenwerke.rs.scripting.service.scripting.terminal.commands.ExecScriptCommand$1$1.doFilter( ↙
      ↳  ExecScriptCommand.java:272)
```

```
at com.google.inject.servlet.FilterChainInvocation.doFilter(FilterChainInvocation.java:66)
at com.google.inject.servlet.FilterDefinition.doFilter(FilterDefinition.java:168)
at com.google.inject.servlet.FilterChainInvocation.doFilter(FilterChainInvocation.java:58)
at com.google.inject.servlet.FilterDefinition.doFilter(FilterDefinition.java:168)
at com.google.inject.servlet.FilterChainInvocation.doFilter(FilterChainInvocation.java:58)
at com.google.inject.servlet.FilterDefinition.doFilter(FilterDefinition.java:168)
at com.google.inject.servlet.FilterChainInvocation.doFilter(FilterChainInvocation.java:58)
at com.google.inject.servlet.ManagedFilterPipeline.dispatch(ManagedFilterPipeline.java:118)
at com.google.inject.servlet.GuiceFilter.doFilter(GuiceFilter.java:113)
at net.datenwerke.rs.scripting.service.scripting.terminal.commands.ExecScriptCommand$1.call( ↙
      ↳  ExecScriptCommand.java:263)
at net.datenwerke.rs.scripting.service.scripting.terminal.commands.ExecScriptCommand$1.call( ↙
      ↳  ExecScriptCommand.java:1)
at java.util.concurrent.FutureTask.run(FutureTask.java:266)
at java.lang.Thread.run(Thread.java:745)
Caused by: javax.script.ScriptException: org.codehaus.groovy.control.MultipleCompilationErrorsException: ↙
      ↳   startup failed:
Script17.groovy: 1: unexpected char: 0xFFFF @ line 1, column 20.
return "Hello world
^

1 error

at org.codehaus.groovy.jsr223.GroovyScriptEngineImpl.compile(GroovyScriptEngineImpl.java:181)
at net.datenwerke.rs.scripting.service.scripting.engines.GroovyScriptCache$1.load(GroovyScriptCache.java ↙
      ↳  :57)
at net.datenwerke.rs.scripting.service.scripting.engines.GroovyScriptCache$1.load(GroovyScriptCache.java ↙
      ↳  :1)
at com.google.common.cache.LocalCache$LoadingValueReference.loadFuture(LocalCache.java:3522)
at com.google.common.cache.LocalCache$Segment.loadSync(LocalCache.java:2315)
at com.google.common.cache.LocalCache$Segment.lockedGetOrLoad(LocalCache.java:2278)
at com.google.common.cache.LocalCache$Segment.get(LocalCache.java:2193)
at com.google.common.cache.LocalCache.get(LocalCache.java:3932)
at com.google.common.cache.LocalCache.getOrLoad(LocalCache.java:3936)
at com.google.common.cache.LocalCache$LocalLoadingCache.get(LocalCache.java:4806)
at net.datenwerke.rs.scripting.service.scripting.engines.GroovyScriptCache.get(GroovyScriptCache.java ↙
      ↳  :79)
at net.datenwerke.rs.scripting.service.scripting.engines.GroovyEngine.eval(GroovyEngine.java:73)
... 23 more
Caused by: org.codehaus.groovy.control.MultipleCompilationErrorsException: startup failed:
Script17.groovy: 1: unexpected char: 0xFFFF @ line 1, column 20.
return "Hello world
^

1 error

at org.codehaus.groovy.control.ErrorCollector.failIfErrors(ErrorCollector.java:309)
at org.codehaus.groovy.control.ErrorCollector.addFatalError(ErrorCollector.java:149)
at org.codehaus.groovy.control.ErrorCollector.addError(ErrorCollector.java:119)
at org.codehaus.groovy.control.ErrorCollector.addError(ErrorCollector.java:131)
at org.codehaus.groovy.control.SourceUnit.addError(SourceUnit.java:359)
at org.codehaus.groovy.antlr.AntlrParserPlugin.transformCSTIntoAST(AntlrParserPlugin.java:137)
at org.codehaus.groovy.antlr.AntlrParserPlugin.parseCST(AntlrParserPlugin.java:108)
at org.codehaus.groovy.control.SourceUnit.parse(SourceUnit.java:236)
at org.codehaus.groovy.control.CompilationUnit$1.call(CompilationUnit.java:164)
at org.codehaus.groovy.control.CompilationUnit.applyToSourceUnits(CompilationUnit.java:928)
at org.codehaus.groovy.control.CompilationUnit.doPhaseOperation(CompilationUnit.java:590)
at org.codehaus.groovy.control.CompilationUnit.processPhaseOperations(CompilationUnit.java:566)
at org.codehaus.groovy.control.CompilationUnit.compile(CompilationUnit.java:543)
at groovy.lang.GroovyClassLoader.doParseClass(GroovyClassLoader.java:297)
at groovy.lang.GroovyClassLoader.parseClass(GroovyClassLoader.java:267)
at groovy.lang.GroovyClassLoader.parseClass(GroovyClassLoader.java:253)
at groovy.lang.GroovyClassLoader.parseClass(GroovyClassLoader.java:211)
at org.codehaus.groovy.jsr223.GroovyScriptEngineImpl.getScriptClass(GroovyScriptEngineImpl.java:366)
```

```
at org.codehaus.groovy.jsr223.GroovyScriptEngineImpl.compile(GroovyScriptEngineImpl.java:173)
... 34 more
reportserver$
```

Tip: With the `-w` flag (e.g. `exec -w hello1.rs`) you can redirect the script output (or the error output) to a separate window.

13.3 Administrative Scripts

Now, we want to develop a more comprehensive script which returns all reports that access a defined table in the query. Here we will get acquainted with some sample services provided for scripts. You will find a detailed description of all services in the ReportServer script manual. In addition, the Java Doc API description by ReportServer provides you with initial information.

Now, we create our second script:

```
reportserver$ createTextFile searchReportByQuery.rs
file created
```

The entry to be made shall include a scrap text and output all dynamic lists for which the data connection is defined as a relational database, and the named scrap text is to be found in their query. Scripts can access arguments via the variable (array) **args**. So the following script would simply output the single argument again:

```
if(args.size() == 0)
  return "No arguments"
args[0]
```

If we execute this script we get the following result:

```
reportserver$ exec searchReportByQuery.rs
No argument stated
reportserver$ exec searchReportByQuery.rs A B C
A
```

Please note that we leave the script in line 2 by entering a `return` if no argument has been stated. In the following we want to browse through all dynamic lists. Via the GLOBALS object (an object that ReportServer adds to the scope of every script) you have access to the various services and auxiliary methods. For instance, the method `getEntitiesByType` allows to simply access all objects of a defined type. Dynamic lists are internally managed as a TableReport type (in the packet `net.datenwerke.rs.base.service.reportengines.table.entities`).

Tip: If you browse for a specific object in JavaDoc API (data source, report, etc.) you will be able to find many important objects because they are marked with the annotation `@Entity`. This annotation is element of all objects that are physically represented in the database.

In order to be able to use objects, you have to import them. Subsequently, we can pass the class related to dynamic lists (TableReport.class) to the method `getEntitiesByType`. By entering the statement .each we can then run a piece of code (in Groovy language a “Closure”) for each object found.

```
/* imports */
import net.datenwerke.rs.base.service.reportengines.table.entities.TableReport

/* argument handling */
if(args.size() == 0)
  return "No arguments"
def searchString = args[0]
GLOBALS.getEntitiesByType(TableReport.class).each {
  tout.println(it.getName())
}
""
```

By the object **tout** you can generate outputs on the console (the object is of type `java.io.PrintWriter`). Within the closure you have access to the loop object, here the current report, via the dynamically generated variable **it**. With this we output the name of all dynamic reports on the console. Please also have a look at the last line of the script `""`. It returns an empty string, as otherwise the return of `GLOBALS.getEntitiesByType(TableReport.class).each` will be output on the console.

When running the script (please ensure to pass an argument to the script) you will find out that not only basic reports have been processed but also the related variants. Reoprt variants inherit from their respective base classes and will therefore also be returned by

```
GLOBALS.getEntitiesByType(TableReport.class)
```

To exclude it we will test whether the currently processed object (within the closure) is of type ReportVariant (in the packet `net.datenwerke.rs.core.service.reportmanager.interfaces`). Now, the adapted script will only return the name of base reports.

```
/* imports */
import net.datenwerke.rs.base.service.reportengines.table.entities.TableReport
import net.datenwerke.rs.core.service.reportmanager.interfaces.ReportVariant

/* argument handling */
if(args.size() == 0)
  return "No arguments"
def searchString = args[0]
GLOBALS.getEntitiesByType(TableReport.class).each {
  if(it instanceof ReportVariant)
    return;
  tout.println(it.getName())
}
""
```

Now we have nearly reached our goal. The data source can be addressed via the field **datasourceContainer.datasource**, and the relational data sources are of type Database- Datasource (in the package `net.datenwerke.rs.base.service.datasources.definitions`). The corresponding configuration is also to be found in the data source container (`datasourceContainer.datasourceConfig`) and of type **DatabaseDatasourceConfig**.

```
/* argument handling */
if(args.size() == 0)
```

```
  return "No arguments"
def searchString = args[0]
GLOBALS.getEntitiesByType(TableReport.class).each {
  if(it instanceof ReportVariant)
    return;

  if(it.datasourceContainer?.datasource instanceof DatabaseDatasource){
    def query = it.datasourceContainer?.datasourceConfig?.query
    if( null != query && query =~ searchString)
      tout.println(it.getName() + ": " + query)
  }
}
""
```

Running on system with installed demo reports, the following output could result:

```
reportserver$ exec searchReportByQuery.rs T_AGG_ORDER
T_AGG_ORDER - Basis: SELECT * FROM T_AGG_ORDER
T_AGG_ORDER - Parametrized: SELECT * FROM T_AGG_ORDER WHERE $X{IN,
    OR_CUSTOMERNUMBER, P_CUSTNUM} AND OR_ORDERDATE &gt; ${P_DATE_FROM} AND
    OR_ORDERDATE &lt; ${P_DATE_TO}
```

13.4 Changing the Data Model

By using scripts, you can of course also change or create objects automatically. A slightly changed version of the above script resets the key of the reports found.

```
/* imports */
import net.datenwerke.rs.base.service.reportengines.table.entities.TableReport
import net.datenwerke.rs.core.service.reportmanager.interfaces.ReportVariant

def key = 1; GLOBALS.getEntitiesByType(TableReport.class).each {
  if(it instanceof ReportVariant)
    return;
  tout.println("set key for report " + it.getId())
  it.setKey("myKey" + key++)
}
"done"
```

If you execute this script you will find out that it runs smoothly, but the changes have not been adopted.

```
reportserver$ exec resetReportKeys.rs
done
set key for report 12
set key for report 17
set key for report 22
set key for report 26
set key for report 33
set key for report 39
```

By default, ReportServer performs a rollback on the database once the script is executed. However, in order to commit the changes, use the `-c` flag.

```
reportserver$ exec -c resetReportKeys.rs
```

13.5 Enhancing ReportServer with Scripts

Apart from the administrative tasks, scripts can be used to enhance ReportServer. Enhancements can be hooked up on the server side just as well as integrated on various points in the interface to, for example, display additional information, or to provide enhanced functionality. In the following we want to present an enhancement on the server side by giving a simple example.

Imagine, we run our business properly and want to ensure that our employees will only be able to retrieve reports during working time. Here, ReportServer provides the option to directly hook up in the report execution and, if required, to interrupt it. In the ReportServer jargon, enhancement interfaces are called **hooks**. They are provided at various locations. The easiest way to get an overview of the enhancement interfaces is by browsing through the JavaDoc API for interfaces which implement the interface **Hook**. For further information on hooks refer to the ReportServer script guide.

To delimit the working time we implement the hook **ReportExecutionNotificationHook**. It will be called up before and after report execution and allows to prevent it. In the following, a code is given which basically implements the interface and checks the current time in the method `doVetoReportExecution`, and if it lies outside the range of 9 a.m to 5 p.m it throws an exception. The callback will be "hooked in" added to the last line.

```
import net.datenwerke.rs.core.service.reportmanager.exceptions.*
import net.datenwerke.rs.core.service.reportmanager.hooks.*

def HOOK_NAME = "PROHIBIT_EXECUTION"
def callback = [
  notifyOfReportExecution : { report, parameterSet, user, outputFormat, configs ->  },
  notifyOfReportsSuccessfulExecution : { compiledReport, report, parameterSet, user,
outputFormat, configs -> },
  notifyOfReportsUnsuccessfulExecution : { e, report, parameterSet, user, outputFormat,
configs -> },
  doVetoReportExecution: { report, parameterSet, user, outputFormat, configs ->
    def cal = Calendar.instance
    def hour = cal.get(Calendar.HOUR_OF_DAY)
    if(hour > 17 || hour < 9)
      throw new ReportExecutorException("Please come back during office hours");
  }
] as ReportExecutionNotificationHook

GLOBALS.services.callbackRegistry.attachHook(HOOK_NAME, ReportExecutionNotificationHook.class,
  callback)
```

Now, if you try to run a report after 6 p.m. you will be welcomed with the message "Please come back during office hours.".

Please ensure to give the hook a name. By doing so, you prevent to apply the hook repeatedly when running the script repeatedly. Use the following script to remove the hook:

```
def HOOK_NAME = "PROHIBIT_EXECUTION"
GLOBALS.services.callbackRegistry.detachHook(HOOK_NAME)
```

Tip: Use the onStartup or onLogin script to hook up enhancements automatically.

13.6 Scheduling of Scripts

Scripts can be planned by a timer controlled schedule. To do this, use the `scheduleScript` command. For further information refer to Chapter 14.

13.7 Accessing Scripts by URL

Similar to accessing files, you can also directly access scripts by URL:

`http://SERVER/APPLICATIONFOLDER/reportserver/scriptAccess?id=XX`

The following URL attributes can be used

id	ID of a file.
path	Path leading to a file, e.g. `bin/script.rs`
args	Arguments passed on to the script.
exception	`true` to receive an error message in case of a failure

If you want to pass more than one argument to the script, you can achieve this separating the arguments through whitespaces, here an example: `http://SERVER/APPLICATIONFOLDER/reportserver/scriptAccess?id=XX&args=firstArg%20secondArg`

The return value of the script will be passed on to the browser as a text message. In addition, you have the option to directly impact the output with the substitutions **httpRequest** and **httpResponse**. This objects hide the Java objects HttpServletRequest (`http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html`) and **HttpServletResponse** (`http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html`). If the script has no value returned (zero), it is assumed that it files its output independently in the OutputStream. For further information refer to the Scripting guide.

```
httpResponse.getWriter().write("Hello world")
return null
```

Remark. If a script is in a folder that is marked as ***web accessible*** (see Chapter 5) then the script can be accessed also by users that are not logged in. This can, for example be used to create a custom login page where the script is used to authenticate the user.

For further information on ReportServer scripts please refer to the ReportServer scripting guide.

Chapter 14

Terminal Commands

In the following we will describe all Terminal commands presently available

14.1 birt

Allows to control the BIRT engine. For the execution of reports, BIRT requires a running Eclipse environment. It will start when the BIRT report will be executed for the first time and then continues to run. By entering the "birt" command, the runtime environment for BIRT reports will stop.

```
Use: birt shutdown
```

14.2 cd

Allows to change the current directory.

```
Use: cd directory
```

14.3 clearInternalDbCache

Clears the cache of the internal database. This is used to optimise the performance for CSV and script data sources (refer to „Data Sources").

```
Use: clearInternalDbCache
```

14.4 config

Configuration files will be cached to optimise the performance. When a configuration file was changed, the cache must be emptied.

```
Use: config reload
```

14.5 cp

Enables to copy one or more files to a new folder. The "-r" flag marks the process as being recursively. In this case sub-folders will also be copied.

```
Use: cp sourcefiles targetfolder
```

14.6 createTextFile

Creates a new text file in the file server File system.

```
Use: createTextFile file
```

14.7 desc

Allows the output of object definitions as they are internally used by ReportServer. For instance, by using `desc Report` you will get a list of all fields which will be saved for the report object of ReportServer. The desc function is primarily designed for developers who want to enhance ReportServer via scripts. For further information refer to the Script/Developer manual. In addition, you can display the object data saved. You wish to display the fields saved in the database for a Jasper Report, then enter the command `desc JasperReport myReport`. Here myReport stands for the name of your report. By setting the `-w` flag you control the output either directly in the console or in a new window.

Tip: The command `ls -l` displays the entity name of an object.

```
Use: desc [-w] EntityName [Entity]
```

14.8 echo

Outputs the given text on the console.

```
Use: echo hello world
```

14.9 editTextFile

Opens a text file in the FileServer File System for editing.

```
Use: editTextFile file
```

14.10 eliza

You want to communicate with Eliza? Enter `eliza` and then `hello`. You will terminate the communication by entering `CTRL+C` or `bye`.

```
Use: eliza
```

14.11 exec

Enables to execute scripts from the FileSystem. By entering the `-c` flag (for commit) you control whether changes to the script persist in the database. By entering the `-n` flag you prevent the script to run in an own monitored thread. For further information on scripts refer to the Script/Developer manual.

```
Use: exec [-c] [-g] Script
```

14.12 export all

Allows to export all metadata to a file. This file can be reloaded by entering "import all". Usually you should directly zip and file the export data in the File System to save memory, as they will be filed in an XML dialect. To do so, use `export all | zip > myExportFile.zip`.

```
Use: export all > myFile.zip
```

14.13 hello

Say hello

```
Use: hello
```

14.14 import all

Allows to import an export file which was created with "export all".

```
Use: import all export.xml
```

14.15 kill

Allows to terminate ongoing script executions. Refer also to `ps`. By entering the `-f` flag you can terminate the script execution thread. Keep in mind that this will be effected by Thread. stop which might provoke errors in ReportServer. For a discussion of the use of `Thread.stop()` please refer to http://docs.oracle.com/javase/6/docs/technotes/guides/concurrency/threadPrimitiveDeprecation.html.

```
Use: kill [-f] id
```

14.16 listpath

Enables the display of the object path. So "`listpath .`" returns the current path. By entering the `-i` flag you control the output either by displaying the path using the object name, or by IDs. ReportServer internally saves paths via object IDs, as they are unique in contrast to object names.

```
Use: listpath [-i] Object
```

14.17 locate

The locate command browses for objects by adding expressions in a similar way as with global search.

```
Use: locate expression
```

14.18 ls

Displaying files in the respective folder By entering `ls -l` additional information per file will be displayed.

```
Use: ls [-l] path
```

14.19 meminfo

Displays the storage utilization of ReportServer.

```
Use: meminfo
```

14.20 mkdir

Enables to create a new folder.

```
Use: mkdir folder name
```

14.21 mv

Enables to move files to a target folder. By using wildcards (e. g. prefix*) you can move sev- eral objects.

```
Use: mv source file target folder
```

14.22 pkg

Command to install ReportServer packages.

list	Lists all available packages in the local filesystem.
install	Install a package. Use flag `-d` to install package from local file system.

14.23 ps

Displays scripts which are currently running (and monitored). By entering the kill command you can interrupt executions.

```
Use: ps
```

14.24 pwd

Displays the current path. By entering the `-i` flag, the path will be given in ID presentation.

`Use: pwd [-i]`

14.25 rcondition

`rcondition` determines dynamic lists as a basis for conditional scheduling (refer also Chapter 10 Scheduling of Reports). The following sub-commands are available:

create	Creates a new condition.

The syntax for creating a new condition is

```
rcondition create objectReference key name description
```

Here reference can be made to the object by entering the path leading to the report. Key is unique to the condition. Name and description identify the same. Please observe to set enclosing quotation marks if spaces are included in names and descriptions, like “This is a description”.

list	Displays a list of the reports marked as conditions.
remove	Allows to remove a condition.

`Use: rcondition remove id`

14.26 reportmod

Enables to set and readout report properties (ReportProperty). They can, for instance, be used in connection with scripts and enhancements to save data with the report. In addition, it enables to set the unique report UUID.

14.27 rev

By entering the rev command you will have access to saved object versions.The list sub-command lists all existing versions of an object. The restore command enables to restore a former version of an object.

list	Lists all revisions of an object.	
restore	Restores an old version of an object	
Example:	rev list id:TableReport:123	Lists all revisions of an the dynamic list with id 123.
	rev restore id:TableReport:123 456 /reportmanager/test	Restores the revision 456 from the dynamic list 123 into /reportmanager/test

14.28 rm

Enables to delete files/objects. To recursively delete folders (which are not empty), the `-r` has to be added.

```
Use: rm [-r] object
```

14.29 scheduleScript

Enables to execute timer controlled scripts. "scheduleScript list" delivers a list showing the currently scheduled scripts. `scheduleScript execute` allows to enter further dispositions. `scheduler` removes dispositions made.

To schedule scripts use the following syntax:

```
scheduleScript execute script scriptArguments expression
```

Here, script is the object reference of a script. Expression determines the scheduling sequence. Please find here some examples:

```
scheduleScript execute myScript.rs " " today at 15:23
scheduleScript execute myScript.rs " " every day at 15:23
scheduleScript execute myScript.rs " " at 23.08.2012 15:23
scheduleScript execute myScript.rs " " every workday at 15:23 starting on 15.03.2011
   for 10 times
scheduleScript execute myScript.rs " " every hour at 23 for 10 times
scheduleScript execute myScript.rs " " today between 16:00 and 23:00 every 10 minutes
scheduleScript execute myScript.rs " " every week on monday and wednesday at 23:12
   starting on 27.09.2011 until 28.11.2012
scheduleScript execute myScript.rs " " every month on day 2 at 12:12 starting on
   27.09.2011 11:25 for 2 times
```

14.30 scheduler

The superordinate command scheduler includes the following commands to control the scheduler.

daemon — Enables to start and stop the scheduler. disable will stop the scheduler and prevent it to restart in case of a ReportServer restart. Commands prefixed by `wd` refer to Watchdog which is integrated in the Scheduler. For further information on this refer to the Developer manual.

```
scheduler daemon [start, stop, restart, enable, disable, status, wdstatus, wdshutdown,
  wdstart, wdrestart]
```

list — Lists jobid , type and nextFireTime.

```
scheduler list
```

listFireTimes — Lists the upcoming fire times for a given `jobid` for the next `numberofFireTimes`. If `numberofFireTimes` is not specified default is 10.

```
scheduler listFireTimes jobid numberofFireTimes
```

remove — Deletes a job with given `jobid` from the dispositions.

```
scheduler remove jobid
```

unschedule Cancels a job with given `jobid` from the dispositions.

```
scheduler unschedule jobid
```

14.31 sql

The SQL command enables to directly access a relational database to run normal SQL commands with the user filed in the object. By calling up `bye` you leave the console. A query always displays 100 result lines each. With `ENTER` you can browse through the results.

```
Use: sql ObjectReference
```

Example

```
reportserver$ cd "/datasources/internal datasources/"
reportserver$ sql "ReportServer Data Source
> SELECT COUNT(*) FROM RS_AUDIT_LOG_ENTRY
COUNT(*)
27783
> bye
Good Bye
```

14.32 teamspacemod

Enables to change TeamSpaces

14.33 unzip

Enables to unpack files presented in ZIP format.

```
Use: unzip file
```

14.34 updateAlias

Reloads the alias configuration. Refer also to section „Terminal".

```
Use: updateAlias
```

14.35 updatedb

Updates the search index. This may take some minutes depending on the data volume.

```
Use: updatedb
```

14.36 usermod

Enables to set UserProperties. They can be used for enhancements. For further information refer to the Script/Developer manual.

```
Use: usermod setproperty theProperty theValue theUser
```

14.37 xslt

Enables to perform an XSL transformation. Here, stylesheet input and output are FileServer files.

```
Use: xslt stylesheet input output
```

14.38 zip

Enables to pack files to a zip archive.

```
Use: zip outputFile.zip input files
```

Chapter 15

Dashboards and Dadgets

The Dashboard module is the first module users will see after signing in. Here, users can file simply prepared data for quick access. For further information on the use of the Dashboard refer to the ReportServer User manual.

Dashboards are privately owned. This means that users can configure their Dashboards themselves, and other users cannot change (or view) them. As the administrator you can provide pre-defined Dadgets and even complete Dashboards that users can easily import. To pre-configure a Dashboard or Dadget, switch to the Administration section and then to **Dadget library**.

Like many other objects in ReportServer, Dashboards and Dadgets are managed hierarchically. This enables to clearly manage even large object volumes and comfortably assign rights. Users must have read access to Dadgets and Dashboards they want to import. The following objects can be created in the Dashboard tree:

Folder	Serves to structure Dashboards and Dadgets.
Dadget	A pre-configured Dadget which users can import to proper Dashboards.
Dashboard	A pre-configured Dashboard that users can import.

The settings of a Dadget are identical to its configuration when a user imports it to the Dashboard. In a first step you have to configure the Dadget type. After having transferred the data, you can set the specific properties of the Dadget. Please observe that Dadgets which have been integrated by users cannot be changed, and the changes you made in the Administration section will be immediately visible to the user.

For details about the individual Dadget types refer to the User manual. As the most frequent case for Dadgets pre-defined by the administrator is certainly the configuration of complex HTML Dadgets, we have dedicated this subject a separate paragraph at the end of this section.

To pre-configure a complete Dashboard, create a new object of type Dashboard. Define its layout and then configure it in the usual way. Please observe here as well that users cannot change Dashboards that have been integrated by them. Therefore, the changes you made in the Administration section will also directly be applied to users.

15.1 Static HTML Dadgets

As an example we next show you how to add a simple chart to your dadget that combines several techniques.

ReportServer's dynamic list offers many export formats some of which, such as JSON, make it easy to use the returned data directly from within javascript. We are going to use the jplot library, a charting extension to the popular jquery library, to create just a very simple pie chart.

The RerportServer's demo data comes with a dynamic list on top of the customer data aggregate: `T_AGG_CUSTOMER`. This table contains all data relevant to specific customers. We are going to visualize the number of customers per office using a simple pie chart.

The first step is to get the data. ReportServer allows to export reports directly via the URL

`http://SERVER:PORT/reportserverbasedir/reportserver/reportexport`

You can access a particular report via its id or via its key (in this case ensure that it is unique). Suppose you have a report with the key **myreport** then you can export the report to say PDF by calling

`http://SERVER:PORT/reportserverbasedir/reportserver/reportexport&key=myreport&format=pdf`

There are two options to proceed, either create a variant first, that accesses the data we need for the chart, or use the base report and access the data via the URL directly. We are going for the second option here. Assuming the customer report has the key **customer** we can access the data via the following URL

`http://SERVER:PORT/reportserverbasedir/reportserver/reportexport&key=customer&c_1=OFF_CITY|country&c_2=CUS_CUSTOMERNAME|count&agg_2=COUNT&format=json`

You can test this on our demo system using the following URL (when prompted, login as `demoadmin/demoadmin`)

`http://demo.raas.datenwerke.net/reportserver/reportexport?id=22&c_1=OFF_CITY|country&c_2=CUS_CUSTOMERNAME|count&agg_2=COUNT&format=json`

Next we are going to create our pie chart. For this, log into your ReportServer and go to the dashboard. Create a new dadget of type static html. Following is the code needed to create a simple pie chart using the above data:

```
<html>
  <head>
    <script language="javascript" type="text/javascript"
      src="http://www2.datenwerke.net/files/blog/js/jqplot/jquery.min.js">
      </script>
    <script language="javascript" type="text/javascript"
      src="http://www2.datenwerke.net/files/blog/js/jqplot/jquery.jqplot.min.js">
      </script>
    <script class="include" type="text/javascript"
      src="http://www2.datenwerke.net/files/blog/js/jqplot/plugins/jqplot.pieRenderer.min.js">
      </script>
```

```
  <link rel="stylesheet" type="text/css"
    href="http://www2.datenwerke.net/files/blog/js/jqplot/jquery.jqplot.min.css" />
</head>
<body style="background-color:#fff">

  <div id="chart1" style="height:500px;width:500px; "></div>
  <script type="text/javascript">
  $.getJSON( 'http://rstest.datenwerke.net/reportserver/reportexport?id=22&c_1=OFF_
    CITY|country&c_2=CUS_CUSTOMERNAME|count&agg_2=COUNT&format=json', function(json) {
    var data = [];
    $.each( json, function( key, val ) {
      data.push( [val.country, Number(val.count)] );
    });
    var plot1 = jQuery.jqplot ('chart1', [data], {
      title: 'Customers per office',
      seriesDefaults: {
        renderer: jQuery.jqplot.PieRenderer,
        rendererOptions: {
          showDataLabels: true,
                      dataLabels: 'value'
        }
      }, grid: {
        background: "#fff",
        borderWidth: 0,
        shadow: false
      }, legend: {
        show: true
      }
    });
  });
  </script>
</body>
</html>
```

There are three basic parts to this script. The first is the head section of the HTML, where we load the required javascript libraries. Next is

```
$.getJSON( 'http://rstest.datenwerke.net/reportserver/reportexport?id=22&c_1=OFF_
  CITY|country&c_2=CUS_CUSTOMERNAME|count&agg_2=COUNT&format=json', function(json) {
  var data = [];
  $.each( json, function( key, val ) {
    data.push( [val.country, Number(val.count)] );
        });
```

which uses jquery to load the json data and to create a data array that is needed for jqplot. Finally we have the actual plotting of the data.

```
  var plot1 = jQuery.jqplot ('chart1', [data], {
    title: 'Customers per office',
    seriesDefaults: {
      renderer: jQuery.jqplot.PieRenderer,
      rendererOptions: {
        showDataLabels: true,
             dataLabels: 'value'
      }
    }, grid: {
      background: "#fff",
      borderWidth: 0,
      shadow: false
    }, legend: {
      show: true
    }
  });
```

15.2 Embedding Dashboards

Similarly to reports (see Section 6.11) you can also embed the ReportServer dashboard view or individual dashboards without the ReportServer corpus. This may be interesting, for example, to embed a dashboard in a portal like application. The base URL to embed dashboards is

`http://SERVER:PORT/reportserverbasedir/ReportServer.html#inlinedashboard/`

which then takes key value pairs where key and values are separated by colons (:) and the next key is separated by an ampersand (&). To display the dashboard of the currently logged in user add the type **user** that is:

`http://SERVER:PORT/reportserverbasedir/ReportServer.html#inlinedashboard/type:user`

This displays the complete dashboard view, that is, all dashboards of the currently logged on user. You can also access a specific dashboard, say the first one by adding the type:single and nr:1 parameters as follows

`http://SERVER:PORT/reportserverbasedir/ReportServer.html#inlinedashboard/type:single&nr:1`

Finally, you can also display a dashboard from the dadget library via its id. In this case, also select type:single but instead of nr use id. For example, to display the dashboard with id 25 use

`http://SERVER:PORT/reportserverbasedir/ReportServer.html#inlinedashboard/type:single&id:25`

Chapter 16

SFTP Server

As already described in previous sections, ReportServer primarily saves objects in tree structures. The SFTP server integrated in ReportServer provides a very comfortable approach to access these file system like structures. To connect to the integrated SFTP server, it needs to be configured beforehand. For further information on this refer to the Installation and Configuration instructions.

The standard configuration provides port 8022 for connections to the SFTP server. Connections use the SFTP (SSH File Transfer) protocol. The registration information (user name and password) usually used in ReportServer shall apply also here. Principally, all users who are entitled to log in, can also connect via SFTP. However, the assigned rights will be checked as it is the case for the access via the web interface. This ensures that also by logging in via SFTP it is not possible to access unreleased objects.

Once the connection is established you are in the root directory. Here, analogue to the presentation in Terminal, the main ReportServer modules are presented as a folder each.

In the module directories, the respective object trees are structured in folders and files as well. Directories whose name starts with `#v-` provide specific functions such as XML export of the object, or direct access to the report definition of graphical reports.

Chapter 17

Maintenance

In this chapter we consider general administrative maintenance tasks.

17.1 Testing User Specific Settings (su)

Especially when handling support requests it may be helpful to see ReportServer as a specific user sees it. To allow administrators to log in as a specific user without knowing their passwords ReportServer has a `su` function (substitute user identity).

To invoke the `su` function press `CTRL+Shift+L` which opens a dialogue that allows to select a specific user and to login as that user. As this is a highly sensitive function, only users with the necessary access rights can invoke it. Who can access the functionality is controlled via the generic right `su`. In addition a user needs the execute right on the user that s/he wants to log in (cf. Chapter 3 User and Permission Management).

> Note that granting the `su` functionality is a potential security risk, as it is difficult to control exactly what rights a user can obtain via logging in as another user.

17.2 Logging

All actions, such as changes to reports or execution of reports are logged in the ReportServer's audit log.

The log is split over two database tables. The table `RS_AUDIT_LOG_ENTRY` contains all logged actions while the table `RS_AUDIT_LOG_PROPERTY` contains additional information for each action.

Note that the tables are not truncated automatically and that on a system with heavy load the tables can thus become very big. It is hence recommended to set up an automatic archiving task. This can be achieved using a ReportServer script, or externally using the native scheduler of your RDBMS.

17.3 Recovering of Objects

Besides logging all relevant actions in the audit-log, ReportServer also documents object changes (for example, how a report was changed). This allows to compare objects to older versions or even to recover older versions if necessary. To access the versioning capabilities use the Terminal command `rev`. `rev list` displays a list with all versions of an object. `rev restore` allows to restore an older version.

Appendix A

Expression Language

In many cases ReportServer allows to insert formulas which are interpreted at runtime instead of static values. Such expressions are always initiated by a dollar sign and an opening curly bracket and closed with a closing curly bracket. The actual expression is given within the curly brackets: `${formula/expression}`. ReportServer uses the unified expression language (UE) standardized in JSR-245 (https://www.jcp.org/en/jsr/detail?id=245 and http://www.oracle.com/technetwork/java/unifiedel-139263.html).

An expression can be a simple calculation or string function such as `${3 + 5}` which would compute the number 8. Depending on the context different objects/replacements (such as the today object in filters, see the User Guide for further information) are available.

Besides the basic arithmetic operators you can use the mathematical functions defined in Table A.1.

To work with strings the following functions can be used in addition to the methods provided by the java string object:

`sutils:left(String, int)`	Returns the first n characters of the string.
`sutils:right(String, int)`	Returns the last n characters of the string.

The ternary operator can be used to define conditional expressions:

```
Condition ? Expression if condition evaluates to true : Expression
   if condition evaluates to false.
```

Thus the expression `${math:random() < 0.5 ? true : false}` returns boolean value which is `TRUE` if the random number is less than 0.5 and `FALSE` otherwise. Thus this expression returns `TRUE` with probability 50%. Depending on the context various objects can be accessed and methods can be called on these objects. In filters, for example, the object "today" can be used to specify dates. To call a method on an object write `${object.methodname()}`. The today object returns the current date. To return the first of the current month you can use the method `firstDay` and write: `${today.firstDay()}`.

Table A.1: List of mathematical functions

`math:random()`	Returns a random number between 0 and 1
`math:sin(Double)`	Computes the sine function.
`math:cos(Double)`	Computes the cosine function.
`math:tan(Double)`	Computes the tangent function.
`math:abs(Double)`	Returns the absolute value.
`math:ceil(Double)`	Returns the smallest double value that is greater or equal to the argument and which is equal to a mathematical integer.
`math:floor(Double)`	Returns the largest double value that is less or equal to the argument and which is equal to a mathematical integer.
`math:round(Double)`	Returns the rounded number (as an integer).
`math:max(Double, Double)`	Returns the greater of the two arguments.
`math:min(Double, Double)`	Returns the smaller of the two arguments.
`math:pow(Double, Double)`	Returns the first value raised to the power of the second.
`math:log(Double)`	Computes the natural logarithm.
`math:exp(Double)`	Computes the value e raised to the power of the argument.
`math:sqrt(Double)`	Computes the square root of the argument.
`math:signum(Double)`	Computes the signum function.

Appendix B

Demo Data

Table	Description
Customers	Master data of all customers
Employees	Information on all employees
Offices	Data on all offices
OrderDetails	Detailed data for every order
Orders	Base data on orders
Payments	Information on payments
ProductLines	Information on product lines
Products	Products
T_AGG_CUSTOMER	Aggregated data per customer
T_AGG_EMPLOYEE	Aggregated data per employee
T_AGG_ORDER	Aggregated data per order
T_AGG_PAYMENT	Aggregated data per payment
T_AGG_PRODUCT	Aggregated data per product

Customers

Column	Type
customerNumber	int(11)
customerName	varchar(50)
contactLastName	varchar(50)
contactFirstName	varchar(50)
phone	varchar(50)
addressLine1	varchar(50)
addressLine2	varchar(50)
city	varchar(50)
state	varchar(50)
postalCode	varchar(15)
country	varchar(50)
salesRepEmployeeNumber	int(11)
creditLimit	decimal(12,2)

Employees

Column	Type
employeeNumber	int(11)
lastName	varchar(50)
firstName	varchar(50)
extension	varchar(10)
email	varchar(100)
officeCode	varchar(10)
reportsTo	int(11)
jobTitle	varchar(50)

Offices

Column	Type
officeCode	varchar(10)
city	varchar(50)
phone	varchar(50)
addressLine1	varchar(50)
addressLine2	varchar(50)
state	varchar(50)
country	varchar(50)
postalCode	varchar(15)
territory	varchar(10)

OrderDetails

Column	Type
orderNumber	int(11)
productCode	varchar(15)
quantityOrdered	int(11)
priceEach	decimal(12,2)
orderLineNumber	smallint(6)

Orders

Column	Type
orderNumber	int(11)
orderDate	datetime
requiredDate	datetime
shippedDate	datetime
status	varchar(15)
comments	text
customerNumber	int(11)

Payments

Column	Type
customerNumber	int(11)
checkNumber	varchar(50)
paymentDate	datetime
amount	decimal(12,2)

PrductLines

Column	Type
productLine	varchar(50)
textDescription	varchar(4000)
htmlDescription	mediumtext
image	mediumblob

Products

Column	Type
productCode	varchar(15)
productName	varchar(70)
productLine	varchar(50)
productScale	varchar(10)
productVendor	varchar(50)
productDescription	text
quantityInStock	smallint(6)
buyPrice	decimal(12,2)
MSRP	decimal(12,2)

T_AGG_CUSTOMER

Column	Type
CUS_ADDRESSLINE1	varchar(50)
CUS_ADDRESSLINE2	varchar(50)
CUS_CITY	varchar(50)
CUS_CONTACTFIRSTNAME	varchar(50)
CUS_CONTACTLASTNAME	varchar(50)
CUS_COUNTRY	varchar(50)
CUS_CREDITLIMIT	decimal(12,2)
CUS_CUSTOMERNAME	varchar(50)
CUS_CUSTOMERNUMBER	int(11)
CUS_PHONE	varchar(50)
CUS_POSTALCODE	varchar(15)
CUS_SALESREPEMPLOYEENUMBER	int(11)
CUS_STATE	varchar(50)
CUS_LATITUDE	decimal(9,6)
CUS_LONGITUDE	decimal(9,6)

EMP_EMAIL	varchar(100)
EMP_EMPLOYEENUMBER	int(11)
EMP_EXTENSION	varchar(10)
EMP_FIRSTNAME	varchar(50)
EMP_JOBTITLE	varchar(50)
EMP_LASTNAME	varchar(50)
EMP_OFFICECODE	varchar(10)
EMP_REPORTSTO	int(11)
OFF_ADDRESSLINE1	varchar(50)
OFF_ADDRESSLINE2	varchar(50)
OFF_CITY	varchar(50)
OFF_COUNTRY	varchar(50)
OFF_OFFICECODE	varchar(10)
OFF_PHONE	varchar(50)
OFF_POSTALCODE	varchar(15)
OFF_STATE	varchar(50)
OFF_TERRITORY	varchar(10)
Y_VOLUME	decimal(12,2)
Y_ACC_BALANCE	decimal(12,2)

T_AGG_EMPLOYEE

Column	Type
EMP_EMAIL	varchar(100)
EMP_EMPLOYEENUMBER	int(11)
EMP_EXTENSION	varchar(10)
EMP_FIRSTNAME	varchar(50)
EMP_JOBTITLE	varchar(50)
EMP_LASTNAME	varchar(50)
EMP_OFFICECODE	varchar(10)
EMP_REPORTSTO	int(11)
OFF_ADDRESSLINE1	varchar(50)
OFF_ADDRESSLINE2	varchar(50)
OFF_CITY	varchar(50)
OFF_COUNTRY	varchar(50)
OFF_OFFICECODE	varchar(10)
OFF_PHONE	varchar(50)
OFF_POSTALCODE	varchar(15)
OFF_STATE	varchar(50)
OFF_TERRITORY	varchar(10)
Y_NUM_CUSTOMERS	bigint(21)
Y_SALES_AMOUNT	decimal(12,2)

T_AGG_ORDER

Column	Type
OD_ORDERLINENUMBER	smallint(6)

OD_ORDERNUMBER	int(11)
OD_PRICEEACH	decimal(12,2)
OD_PRODUCTCODE	varchar(15)
OD_QUANTITYORDERED	int(11)
OR_COMMENTS	text
OR_CUSTOMERNUMBER	int(11)
OR_ORDERDATE	datetime
OR_ORDERNUMBER	int(11)
OR_REQUIREDDATE	datetime
OR_SHIPPEDDATE	datetime
OR_STATUS	varchar(15)
PL_HTMLDESCRIPTION	mediumtext
PL_IMAGE	mediumblob
PL_PRODUCTLINE	varchar(50)
PL_TEXTDESCRIPTION	varchar(4000)
PRO_BUYPRICE	decimal(12,2)
PRO_MSRP	decimal(12,2)
PRO_PRODUCTCODE	varchar(15)
PRO_PRODUCTDESCRIPTION	text
PRO_PRODUCTLINE	varchar(50)
PRO_PRODUCTNAME	varchar(70)
PRO_PRODUCTSCALE	varchar(10)
PRO_PRODUCTVENDOR	varchar(50)
PRO_QUANTITYINSTOCK	smallint(6)
CUS_ADDRESSLINE1	varchar(50)
CUS_ADDRESSLINE2	varchar(50)
CUS_CITY	varchar(50)
CUS_CONTACTFIRSTNAME	varchar(50)
CUS_CONTACTLASTNAME	varchar(50)
CUS_COUNTRY	varchar(50)
CUS_CREDITLIMIT	decimal(12,2)
CUS_CUSTOMERNAME	varchar(50)
CUS_CUSTOMERNUMBER	int(11)
CUS_PHONE	varchar(50)
CUS_POSTALCODE	varchar(15)
CUS_SALESREPEMPLOYEENUMBER	int(11)
CUS_STATE	varchar(50)
CUS_LATITUDE	decimal(9,6)
CUS_LONGITUDE	decimal(9,6)
EMP_EMAIL	varchar(100)
EMP_EMPLOYEENUMBER	int(11)
EMP_EXTENSION	varchar(10)
EMP_FIRSTNAME	varchar(50)
EMP_JOBTITLE	varchar(50)
EMP_LASTNAME	varchar(50)
EMP_OFFICECODE	varchar(10)
EMP_REPORTSTO	int(11)
OFF_ADDRESSLINE1	varchar(50)

OFF_ADDRESSLINE2	varchar(50)
OFF_CITY	varchar(50)
OFF_COUNTRY	varchar(50)
OFF_OFFICECODE	varchar(10)
OFF_PHONE	varchar(50)
OFF_POSTALCODE	varchar(15)
OFF_STATE	varchar(50)
OFF_TERRITORY	varchar(10)

T_AGG_PAYMENT

Column	Type
CUS_ADDRESSLINE1	varchar(50)
CUS_ADDRESSLINE2	varchar(50)
CUS_CITY	varchar(50)
CUS_CONTACTFIRSTNAME	varchar(50)
CUS_CONTACTLASTNAME	varchar(50)
CUS_COUNTRY	varchar(50)
CUS_CREDITLIMIT	decimal(12,2)
CUS_CUSTOMERNAME	varchar(50)
CUS_CUSTOMERNUMBER	int(11)
CUS_PHONE	varchar(50)
CUS_POSTALCODE	varchar(15)
CUS_SALESREPEMPLOYEENUMBER	int(11)
CUS_STATE	varchar(50)
EMP_EMAIL	varchar(100)
EMP_EMPLOYEENUMBER	int(11)
EMP_EXTENSION	varchar(10)
EMP_FIRSTNAME	varchar(50)
EMP_JOBTITLE	varchar(50)
EMP_LASTNAME	varchar(50)
EMP_OFFICECODE	varchar(10)
EMP_REPORTSTO	int(11)
OFF_ADDRESSLINE1	varchar(50)
OFF_ADDRESSLINE2	varchar(50)
OFF_CITY	varchar(50)
OFF_COUNTRY	varchar(50)
OFF_STATE	varchar(50)
OFF_TERRITORY	varchar(10)
OFF_OFFICECODE	varchar(10)
OFF_PHONE	varchar(50)
OFF_POSTALCODE	varchar(15)
PAY_AMOUNT	decimal(12,2)
PAY_CHECKNUMBER	varchar(50)
PAY_CUSTOMERNUMBER	int(11)
PAY_PAYMENTDATE	datetime

T_AGG_PRODUCT

Column	Type
PL_HTMLDESCRIPTION	mediumtext
PL_IMAGE	mediumblob
PL_PRODUCTLINE	varchar(50)
PL_TEXTDESCRIPTION	varchar(4000)
PRO_BUYPRICE	decimal(12,2)
PRO_MSRP	decimal(12,2)
PRO_PRODUCTCODE	varchar(15)
PRO_PRODUCTDESCRIPTION	text
PRO_PRODUCTLINE	varchar(50)
PRO_PRODUCTNAME	varchar(70)
PRO_PRODUCTSCALE	varchar(10)
PRO_PRODUCTVENDOR	varchar(50)
PRO_QUANTITYINSTOCK	smallint(6)
Y_NUM_SOLD	decimal(32,0)
Y_AVG_PRICE	decimal(12,2)