ReportServer

Administrator's Guide 6.0

ReportServer

Administrator's Guide 6.0

InfoFabrik GmbH, 2026

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

Contents

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 described 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 "ClassicModelCars". 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 reports 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, then 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-RS{VERSION}}.zip
demobuilder-RS{VERSION}.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** Allows administrators to create, modify, and manage users, organizational units and user groups.
- **Datasources** This section is used to configure and manage database connections that can be used as data sources for reports.
- **Datasinks** Datasinks are used to send data to a defined target location and can be considered the counterpart to datasources. They are typically used for exporting report results, for example to file systems, FTP/SFTP servers, or other external systems.
- **Report management** Provides functionality to create, modify, and manage report objects.
- **File system** The file system is used by ReportServer in various contexts. It contains configuration files and ReportServer scripts, and can also be used to provide shared resources (e.g. images) for reports.
- **Dashboard library** Dadgets (short for dashboard gadgets) are used by users to assemble dashboards. Within the dashboard library, administrators can provide preconfigured dadgets that users can easily add to their dashboards.
- **Remote servers** Remote servers are used to configure connections to other ReportServer instances via the REST API. They enable the exchange of data (e.g. reports or users) between different ReportServer installations, such as copying content from a production system to a test system.
- **Transports** Transports are used to bundle reports, datasources, datasinks, and files into transport packages. They enable the controlled and consistent transfer of changes between different ReportServer environments such as development, test, and production systems. In this section, transport objects can be created, inspected, and applied (i.e. integrated into the system).

Transport management This section allows administrators to fetch transport objects from a remote ReportServer instance. Unlike the "Transports" section above, only transports that have not yet been applied are listed here. These transports can be reviewed and applied from this section.

TeamSpace TeamSpaces allow administrators to create and manage collaborative workspaces for teams. They provide team-specific views on shared report objects without affecting the global object hierarchy, while supporting collaboration and role-based permissions within each TeamSpace.

Permission management Defines access permissions for the different ReportServer modules and functions.

Import The import module allows administrators to import ReportServer objects that have previously been exported.

Global constants Global constants can be used within reports to externalize configuration parameters and manage them centrally.

Scheduler In contrast to the Scheduler main module, the administration view provides insight into all scheduler jobs created by users. This allows administrators to monitor and manage scheduled executions centrally.

In the following section, we provide a brief introduction to ReportServer. Key concepts and functions are explained using selected examples.

If you have not done so already, you may now load the demo data; however, the demo content is not required for the upcoming introduction. For further details, see Section 1 on page 4.

2.3 Creating a datasource

The basic requirement for running reports is the configuration of a datasource. Datasources 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 programs. 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 datasources section within the Administration module. Below the Root folder, the folder "internal datasources" should already be located. Beside the existing folder, create another one: Right click on "Datasource 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 datasource which enables to access the internal demo data. Below the newly created folder, create then the datasource 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 “ClassicModelCars”. 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 datasource (click on the magnifying glass) **demo data** that you have just created.

The configuration displayed varies with the datasource 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 “ClassicModelCars”. 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 “ClassicModelCars” 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** datasource that you created as default datasource. For the import it will be entered in the report as datasource. 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 subfolder 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 `https://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 submodule. 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` or clicking on the upper-right dropdown "Tools" -> "Open Terminal Session". 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 submodule 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 (https://groovy-lang.org/). For working with ReportServer, scripts are not necessarily required, however, you can use them to accomplish many special tasks (such as integrating complex datasources 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.groovy` by entering the command `createTextFile hello.groovy`. 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.groovy`. You can run the script with the command `exec hello .groovy`. You should see the line

```
Hello ReportServer Admin
reportserver$
```

on the terminal.

With this we would like to finish our quick ReportServer introduction. Up to this point we have only encountered a small 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 independently 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 does not need to 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. A 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

Generic rights control access to global components and functions of ReportServer. Examples include the permission to log in to the system or to access the Administration section.

In addition to generic rights, ***object-related rights*** can be assigned. These permissions apply to ReportServer objects that are organized in tree-based structures, such as datasources, reports, or users. Unlike generic rights, object-related permissions support inheritance, which can significantly reduce administrative effort when managing large or complex permission hierarchies.

In ReportServer, permissions are managed using ***Access Control Lists (ACLs)*** and ***Access Control Entries (ACEs)***. An ACL exists for every object whose access can be restricted, including reports, datasources, folders, and other tree-based objects, as well as for each generic permission. Each ACL consists of a set of ACEs, where each ACE defines the following information:

Folk Specifies the subject to which the permission is granted. This can be an individual user, a user group, or an organizational unit.

Access type Defines whether the permission entry extends the rights granted by other entries or further restricts them.

Basic rights Basic rights can be assigned to all ReportServer objects. Their specific meaning depends on the type of the respective object and is explained in the following sections.

Determining if a Permission is Granted

When evaluating whether a user has a specific permission, ReportServer processes the associated Access Control Entries (ACEs) sequentially in the order in which they are defined.

For each ACE, ReportServer checks whether it applies to the user by evaluating the ***folk***: if the folk is a group, the system checks whether the user is a member of that group; if the folk is an organizational unit, it is verified whether the user belongs to the corresponding subtree; if the folk is an individual user, the identities are compared directly.

If an ACE applies, its ***access type*** determines whether the permission is granted or denied. ***The first matching ACE is decisive. This means that the order of the ACEs is relevant.*** This means that if an earlier ACE denies a permission, the permission is denied even if a later ACE would grant the same permission. If no ACE applies to the user, the permission is not granted.

Consider the following example of an ACL consisting of three ACEs: the first denies read and write permissions to members of group A; the second grants read permission to members of group B; the third grants write permission 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, ReportServer would first test whether 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.

> Note that you can use the `haspermission` terminal command described in Section 20.26 haspermission for checking generic permissions of a user.

In the following we discuss the various generic permissions. Note that the `target` is necessary for checking generic permissions with the `haspermission` terminal command described in Section 20.26 haspermission.

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 submodules must be released individually. Rights other than the reading right are not queried.

Target: `net.datenwerke.gf.service.genrights.AdministrationViewSecurityTarget`

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.

Target: `net.datenwerke.rs.dashboard.service.dashboard.genrights.DashboardViewSecurityTarg`

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

Target: `net.datenwerke.rs.dashboard.service.dashboard.genrights.DashboardAdminSecurityTar`

Datasinks The datasinks generic right controls access to the datasinks tree in the Administration module. The read permission determines whether this section is visible in the Administration module. Access to individual datasinks is governed by object-related permissions.

Target: `net.datenwerke.rs.core.service.datasinkmanager.genrights.DatasinkManagerAdminView`

Datasources The datasources generic right controls access to the datasources tree in the Administration module. The read permission determines whether this section is visible in the Administration module. Access to individual datasources is governed by object-related permissions.

Target: `net.datenwerke.rs.core.service.datasourcemanager.genrights.DatasourceManagerAdmin`

Export The generic right Export determines whether a user is allowed to export objects from a ReportServer Admin module in the xml format. Executing right (x) is queried for this.

Target: `net.datenwerke.rs.eximport.service.genrights.ExportSecurityTarget`

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.

Target: `net.datenwerke.rs.fileserver.service.fileserver.genrights.FileServerManagerAdminV`

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.

Target: `net.datenwerke.security.service.genrights.security.GenericSecurityTargetAdminViewSe`

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.

Target: `net.datenwerke.rs.globalconstants.service.globalconstants.genrights.GlobalConstants`

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

Target: `net.datenwerke.rs.eximport.service.genrights.ImportSecurityTarget`

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.

Target: `net.datenwerke.rs.license.service.genrights.LicenseSecurityTarget`

Remote Servers. The remote servers generic right controls the access to the remote servers tree in the Admin module. Reading right is queried for this section to be visible in the Admin module. The actually granted access rights to single remote servers will be controlled by object rights.

Target: `net.datenwerke.rs.remoteserver.service.remoteservermanager.genrights.RemoteServerMa`

Report manager. 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.

Target: `net.datenwerke.rs.core.service.reportmanager.genrights.ReportManagerAdminViewSecuri`

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.

Target: `net.datenwerke.rs.core.service.genrights.access.AccessRsSecurityTarget`

SFTP. The generic right SFTP enables to access ReportServer via SFTP. Here, execute right enables to access ReportServer via SFTP. Rights other than the execute right are not queried.

Target: `net.datenwerke.rs.remoteaccess.service.sftp.genrights.SftpSecurityTarget`

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.

Target: `net.datenwerke.rs.adminutils.service.su.genrights.SuSecurityTarget`

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.

Target: `net.datenwerke.rs.scheduler.service.scheduler.genrights.SchedulingBasicSecurityTa`

Scheduler Administrations. The generic right Scheduler Admin View enables to access the Scheduler module in the Admin section. Here, reading right enables to view the module. The execute right enables to modify scheduling entries. It further allows to change the job executor.

Target: `net.datenwerke.rs.scheduler.service.scheduler.genrights.SchedulingAdminSecurityTa`

System Console. The generic permission System Console controls the access to the system console in the Administration module. If reading right is granted the section is visible. All system console subsections will be visible if the user has read permission on this generic permission. Note that the "System Console" is an Enterprise feature.

Target: `net.datenwerke.rs.adminutils.service.systemconsole.genrights.SystemConsoleSecurit`

TeamSpace. The generic right TeamSpaces controls the permissions 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.

Target: `net.datenwerke.rs.teamspace.service.teamspace.genrights.TeamSpaceSecurityTarget`

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

Target: `net.datenwerke.rs.terminal.service.terminal.genrights.TerminalSecurityTarget`

Transport management. The generic right "Transport Management" controls access to the "Transport Management" section in the Administration module. The read permission determines whether this section is visible to the user.

Target: `net.datenwerke.rs.transport.service.transport.genrights.TransportManagementAdminV`

Transports. The generic right "Transports" controls access to the "Transports" section in the Administration module. The read permission determines whether this section is visible to the user.

Target: `net.datenwerke.rs.transport.service.transport.genrights.TransportAdminViewSecurit`

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.

Target: `net.datenwerke.rs.uservariables.service.genrights.UserVariableAdminViewSecurityTa`

User and groups. The generic right User management controls the visibility of the Administration submodule 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.

Target: `net.datenwerke.security.service.genrights.usermanager.UserManagerAdminViewSecurityT`

Object Related Rights

In addition to generic rights, permissions can be assigned at a fine-grained level to individual objects, such as reports, datasources, or users. For this purpose, object-related permissions also use Access Control Lists (ACLs).

Object-related permissions are managed directly on the respective object. For example, to manage the permissions of a report, navigate to the Report Management module and open the report. The permission management view is accessible via the **Permission management** tab.

> Note that you can use the `haspermission` terminal command described in Section 20.26 haspermission for checking object rights of a user.

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***, ***Dashboard 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 conclusive decision can be made, ReportServer goes on to object 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

Datasources

In ReportServer, datasources provide the primary data foundation for reports; each report retrieves the data it displays from a defined datasource. Like most other objects, datasources are organized in a hierarchical tree structure. Datasource management is available in the Administration module under ***Datasources***. Within this tree, the following object types can be created:

Folder Used to organize and structure datasources.

Datasource Represents a concrete datasource. Multiple datasource types are available and are described in more detail in the following sections.

Datasources in ReportServer are configured in two stages. First, datasources are created and basic connection settings are defined in the Datasource Management. For relational databases, this typically includes parameters such as the connection URL, username, and password.

The usage-specific configuration is defined at the point where the datasource is used, most commonly within a report. For example, in the case of relational databases, the SQL query on which a report is based is specified there.

ReportServer supports the following datasource types.

4.1 Relational Databases

It is possible to access common relational databases via the datasource type "Relational databases". Use the option "database" to control the SQL dialect created by ReportServer. Currently, Report-Server supports the following datasources[1]:

- Athena `https://aws.amazon.com/athena/`
- Db2 for IBM i `https://www.ibm.com/products/db2`

[1] Many other datasources and dialects can be integrated via scripting. For further information, refer to the ReportServer Scripting Guide.

- Db2 for z/OS https://www.ibm.com/products/db2
- Derby https://db.apache.org/derby/
- CockroachDB https://www.cockroachlabs.com/
- CrateDB https://crate.io/
- Exasol https://www.exasol.com/
- Firebird https://firebirdsql.org/
- Google BigQuery https://cloud.google.com/bigquery
- H2 https://www.h2database.com/
- HSQL https://hsqldb.org/
- Incorta https://www.incorta.com/
- Informix https://www.ibm.com/products/informix
- MariaDB https://mariadb.org/
- MonetDB https://www.monetdb.org/
- MySQL https://www.mysql.com/
- Oracle https://www.oracle.com/en/database/
- PostgreSQL https://www.postgresql.org/
- Redshift https://aws.amazon.com/redshift
- SAP HANA https://www.sap.com/products/technology-platform/hana/what-is-sap-hana.html
- SQL Server https://www.microsoft.com/en-us/sql-server
- SQL Server (Old driver, jTDS) https://www.microsoft.com/en-us/sql-server
- SQLite https://www.sqlite.org/
- Sybase https://www.sap.com/
- Teradata https://www.teradata.com/
- Vertica https://www.vertica.com/
- YugabyteDB https://www.yugabyte.com/

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 datasource.

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 datasource, for instance, if you want to create a dynamic list on the basis of this datasource.

> Note that you can fetch any metadata information of your datasource supported by your JDBC driver with the `datasourceMetadata` terminal command. More defails on Section 20.13 datasourceMetadata.

4.2 Athena

Amazon AWS Athena `https://aws.amazon.com/athena/` is an interactive query service that makes it easy to analyze data directly in Amazon Simple Storage Service (Amazon S3) using standard SQL. With a few actions in the AWS Management Console, you can point Athena at your data stored in Amazon S3 and begin using standard SQL to run ad-hoc queries and get results in seconds.

ReportServer supports AWS Athena by its official Simba JDBC driver, downloadable from `https://docs.aws.amazon.com/athena/latest/ug/connect-with-jdbc.html`. Download the JDBC 4.2-compatible version and install it in your ReportServer lib directory. We strongly recommend using the external Configdir for this, as explained in the Configuration Guide. In such a way, the driver will be separated from the ReportServer libraries and will not get overwritten with your next ReportServer upgrade. At the moment of writing, the version of the JDBC driver was 2.0.35.1000 (AthenaJDBC42-2.0.35.1000.jar). Note that you should download the "without the AWS SDK" version: (`JDBC 4.2 compatible driver version 2.1 (without the AWS SDK)`).

As stated in the JDBC documentation, you have to obtain your JDBC URL from your AWS. An example URL of Athena is:

```
jdbc:awsathena://AwsRegion=eu-central-1;S3OutputLocation=s3://examplebucket/exampledirectory;
```

In the ReportServer datasource user and password fields, put your user and password (i.e. your access key ID (e.g. `AKIAIOSFODNN7EXAMPLE`) and your secret access key (e.g. (`wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEK` `https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html`) previously configured in AWS. More information can be found in the AWS Athena documentation.

The JDBC URL parameters available are documented here: https://docs.aws.amazon.com/athena/latest/ug/connect-with-jdbc.html in the JDBC documentation, e.g. https://s3.amazonaws.com/athena-downloads/drivers/JDBC/SimbaAthenaJDBC-2.0.35.1000/docs/Simba+Amazon+Athena+JDBC+Connector+Install+and+Configuration+Guide.pdf.

More information can be found here: https://docs.aws.amazon.com/athena/latest/ug/getting-started.html

4.3 Db2 for IBM i

Db2 https://www.ibm.com/products/db2 is IBM's cloud-native database built to power scalable real-time analytics.

ReportServer supports Db2 for IBM i by its official JDBC driver `jt400.jar`, downloadable from https://jt400.sourceforge.net/ or https://mvnrepository.com/artifact/net.sf.jt400/jt400. Download the JDBC version and install it in your ReportServer lib directory. We strongly recommend using the external Configdir for this, as explained in the Configuration Guide. In such a way, the driver will be separated from the ReportServer libraries and will not get overwritten with your next ReportServer upgrade. At the moment of writing, the version of the JDBC driver was jt400-11.2 (jt400-11.2.jar).

As stated in the JDBC documentation, you have to enter your connection parameters required to connect into your JDBC URL. An example URL is:

```
jdbc:as400://my-host
```

In the ReportServer datasource user and password fields, enter your database user and password previously configured in Db2. More information can be found in the Db2 documentation.

The JDBC URL parameters available are documented here: https://www.ibm.com/docs/en/i/7.4?topic=ssw_ibm_i_74/rzahh/javadoc/com/ibm/as400/access/doc-files/JDBCProperties.htm.

If you get a "not found" error when creating a dynamic list, try prefixing your objects with your schema. E.g.: instead of entering this query: `select * from MYTABLE`, enter this query: `select * from MYSCHEMA.MYTABLE`, where `MYSCHEMA` is your schema.

4.4 Db2 for z/OS

Db2 https://www.ibm.com/products/db2 is IBM's cloud-native database built to power scalable real-time analytics.

ReportServer supports Db2 for z/OS by its official JDBC driver `db2jcc4.jar`, downloadable from https://www.ibm.com/support/pages/db2-jdbc-driver-versions-and-downloads or https://mvnrepository.com/artifact/com.ibm.db2.jcc/db2jcc/db2jcc4. Download the JDBC 4-compatible version and install it in your ReportServer lib directory. We strongly recommend using the external Configdir for this, as explained in the Configuration Guide. In such a way, the driver will be separated from the ReportServer libraries and will not get overwritten with your next ReportServer upgrade. At the moment of writing, the version of the JDBC driver was db2jcc4 4.32.28 (db2jcc4.jar).

As stated in the JDBC documentation, you have to enter your connection parameters required to connect into your JDBC URL. An example URL is:

```
jdbc:db2://6d9b95e7-dafb-4c5c-916c-0n79babe9ae0.bv7e8xuf0shslbo0krsg.databases.
    appdomain.cloud:30387/bludb:allowNextOnExhaustedResultSet=1;sslConnection=
    true;
```

It is very important to enter this parameter:

```
allowNextOnExhaustedResultSet=1
```

This is due to the fact that the Db2 driver automatically closes the cursor when all rows have been retrieved from a ResultSet, so this setting is necessary: `https://www.ibm.com/support/pages/invalid-operation-result-set-closed-error-data-server-driver-jdbc`.

In the ReportServer datasource user and password fields, enter your database user and password previously configured in Db2. More information can be found in the Db2 documentation.

The JDBC URL parameters available are documented here: `https://www.ibm.com/docs/en/db2/10.5?topic=information-properties-data-server-driver-jdbc-sqlj`.

If you get the following error when creating a dynamic list: `DB2 SQL Error: SQLCODE=-204, SQLSTATE=42704`, try prefixing your objects with your schema. E.g.: instead of entering this query: `select * from mytable`, enter this query: `select * from bludb.mytable`, where `bludb` is your schema.

4.5 Derby

Apache Derby `https://db.apache.org/derby/` is an open-source relational database management system (RDBMS) developed by the Apache Software Foundation that can be embedded in Java programs and used for online transaction processing.

ReportServer supports Apache Derby by its official JDBC driver `derbyclient.jar`, downloadable from `https://db.apache.org/derby/derby_downloads.html`. Download the JDBC 4-compatible version and install it in your ReportServer lib directory. Note that as ReportServer needs Java 21, you need to download the 10.17.1.0 version (For Java 21 and higher). You can find the `derbyclient.jar` inside the `lib` directory of the downloaded file. We strongly recommend using the external ReportServer Configdir for installing the JDBC driver, as explained in the Configuration Guide. In such a way, the driver will be separated from the ReportServer libraries and will not get overwritten with your next ReportServer upgrade. At the moment of writing, the version of the JDBC driver was 10.17.1.0 (For Java 21 and higher).

As stated in the JDBC documentation, you have to enter your connection parameters required to connect into your JDBC URL. An example URL is:

```
jdbc:derby://localhost:1527/mydb;create=false
```

In the ReportServer datasource user and password fields, enter your database user and password previously configured in Apache Derby. More information can be found in the Apache Derby documentation.

The JDBC URL parameters available are documented here: https://db.apache.org/derby/docs/10.15/ref/ and https://db.apache.org/derby/docs/10.15/ref/rrefattrib24612.html.

4.6 CockroachDB

CockroachDB https://www.cockroachlabs.com/ is a distributed SQL database built on a transactional and strongly-consistent key-value store. It scales horizontally; survives disk, machine, rack, and even datacenter failures with minimal latency disruption and no manual intervention; supports strongly-consistent ACID transactions; and provides a familiar SQL API for structuring, manipulating, and querying data.

ReportServer supports CockroachDB by the PostgreSQL driver shipped with ReportServer.

As stated in the JDBC documentation, you have to enter your connection parameters required to connect into your JDBC URL. An example URL is:

```
jdbc:postgresql://karmic-gopher-6507.7tc.cockroachlabs.cloud:26257/defaultdb?
    sslmode=verify-full
```

In the ReportServer datasource user and password fields, enter your database user and password previously configured in CockroachDB. More information can be found in the CockroachDB documentation.

Note that, if you haven't already, you must have a valid CA certificate located at /.postgresql/root.crt. See instructions under 'Download CA Cert (Required only once) in the CockroachDB documentation.

E.g. you can download the certificate like this:

```
curl --create-dirs -o $HOME/.postgresql/root.crt -O https://cockroachlabs.cloud/
    clusters/b446374f-9820-44e3-800a-77e0306e564d/cert
```

The JDBC URL parameters available are documented here: https://www.cockroachlabs.com/docs/stable/connection-parameters.html.

4.7 CrateDB

CrateDB https://crate.io/ is a distributed SQL database management system that integrates a fully searchable document-oriented data store.

ReportServer supports CrateDB by its official JDBC driver crate-jdbc.jar, downloadable from https://crate.io/docs/jdbc/en/latest/getting-started.html. Download the JDBC 4.2-compatible version and its dependencies (e.g. from its crateDB-jdbc maven dependency available here: https://mvnrepository.com/artifact/io.crate/crate-jdbc or https://crate.io/docs/jdbc/en/latest/getting-started.html and install it in your ReportServer lib directory. Add only the libraries not included already in ReportServer (i.e. add only the libraries not available here:

your/reportserver/installation/WEB-INF/lib) We strongly recommend using the external Configdir for this, as explained in the Configuration Guide. In such a way, the driver will be separated from the ReportServer libraries and will not get overwritten with your next ReportServer upgrade. At the moment of writing, the version of the JDBC driver was 2.6.0 (crate-jdbc-2.6.0.jar).

As stated in the JDBC documentation, you have to enter your connection parameters required to connect into your JDBC URL. An example URL is:

```
crate://maroon-luminara-unduli.aks1.eastus2.azure.cratedb.net:5432/
```

Note the / trailing slash after the URL. This is necessary, otherwise the driver will not accept your URL.

Also, note that the JDBC URL ***does not start*** with jdbc:, different from other JDBC URL types.

In the ReportServer datasource user and password fields, enter your database user and password previously configured in CrateDB. More information can be found in the CrateDB documentation.

The JDBC URL parameters available are documented here: https://crate.io/docs/jdbc/en/latest/connect.html.

4.8 Exasol

Exasol https://www.exasol.com/ is a parallelized relational database management system (RDBMS) which runs on a cluster of standard computer hardware servers. Following the SPMD model, on each node the identical code is executed simultaneously. The data is stored in a column-oriented way and proprietary in-memory compression methods are used.

ReportServer supports Exasol by its official JDBC driver exasol-jdbc.jar, downloadable from https://www.exasol.com/. Download the JDBC 4.2-compatible version (e.g. from its exasol-jdbc maven dependency available here: https://mvnrepository.com/artifact/com.exasol/exasol-jdbc/ and install it in your ReportServer lib directory. We strongly recommend using the external Configdir for this, as explained in the Configuration Guide. In such a way, the driver will be separated from the ReportServer libraries and will not get overwritten with your next ReportServer upgrade. At the moment of writing, the version of the JDBC driver was 7.1.17 (exasol-jdbc-7.1.17.jar).

As stated in the JDBC documentation, you have to enter your connection parameters required to connect into your JDBC URL. An example URL is:

```
jdbc:exa:mv374bgnpje2dktjlxqer22qoa.clusters.exasol.com:8563;schema=myschema
```

Note that the URL does not contain ://, different as other JDBC URLs.

In the ReportServer datasource user and password fields, enter your database user and password previously configured in Exasol. More information can be found in the Exasol documentation.

The JDBC URL parameters available are documented here: https://docs.exasol.com/db/latest/connect_exasol/drivers/jdbc.htm.

4.9 Google BigQuery

Google BigQuery https://cloud.google.com/bigquery/ is a serverless, highly-scalable, and cost-effective cloud data warehouse with an in-memory BI Engine and machine learning built in.

ReportServer supports Google BigQuery datasources by the official Simba JDBC driver. Download the JDBC 4.2-compatible version from the Simba web site (https://docs.cloud.google.com/bigquery/docs/reference/odbc-jdbc-drivers?hl=en) and add the all .jars in the downloaded zip to your lib directory.

At the moment of writing, the Simba JDBC driver had version 1.2.0.1000.

We strongly recommend using the external Configdir for this, as explained in the Configuration Guide. In such a way, the libraries will be separated from the ReportServer libraries and will not get overwritten with your next ReportServer upgrade.

The Simba JDBC driver download includes a configuration guide (Simba JDBC Driver for Google BigQuery Install and Configuration Guide.pdf). The "Configuring Authentication" section includes some authentication options; use e.g. the "Using a Google Service Account" option.

Your URL should look similar to this:

```
jdbc:bigquery://https://www.googleapis.com/bigquery/v2:443;ProjectId=MyBigQueryProject;
OAuthType=0;OAuthServiceAcctEmail=bqtest1@data-driver-testing.iam.gserviceaccount.
com;OAuthPvtKeyPath=/SecureFiles/ServiceKeyFile.json;
```

Your username and password fields should be empty in ReportServer, since they are not being used.

In some cases, you may also want to adapt the timeout parameter, you can change it by appending it into the URL:

```
jdbc:bigquery://https://www.googleapis.com/bigquery/v2:443;ProjectId=MyBigQueryProject;
OAuthType=0;OAuthServiceAcctEmail=bqtest1@data-driver-testing.iam.gserviceaccount.
com;OAuthPvtKeyPath=/SecureFiles/ServiceKeyFile.json;Timeout=3600
```

For more details refer to the Simba JDBC Driver Configuration Guide.

4.10 Incorta

Incorta https://www.incorta.com/ is an end-to-end data and analytics platform for acquiring, processing, analyzing and presenting business applications data.

ReportServer supports Incorta by the PostgreSQL driver shipped with ReportServer.

As stated in the JDBC documentation, you have to enter your connection parameters required to connect into your JDBC URL. An example URL is:

```
jdbc:postgresql://incorta-4439.sqli.cloud4.incorta.com:9603/default
```

In the ReportServer datasource user and password fields, enter your database user and password previously configured in Incorta. More information can be found in the Incorta documentation.

Note that you have to enable external BI tools connections if you are using the cloud as explained here: https://community.incorta.com/t5/data-schemas-knowledgebase/how-to-connect-to-incorta-from-ext ta-p/714.

4.11 MySQL

MySQL https://www.mysql.com/ is an open-source relational database management system that allows you to manage and organize your data using SQL. It is widely used for web applications, data warehousing, and various types of data storage. MySQL supports both SQL-based queries and various connectors, including JDBC for Java applications.

ReportServer supports MySQL via its official MySQL JDBC driver, shipped along with ReportServer.

To connect ReportServer to a MySQL database, you'll need to configure the JDBC URL in your data source configuration. Here is an example of a typical JDBC URL for MySQL:

```
jdbc:mysql://localhost:3306/mydatabase?serverTimezone=UTC
```

where `serverTimezone=UTC` ensures the connection uses UTC time zone, which can be adjusted to your server's time zone. Ensure that the `serverTimezone` is correctly set to match your MySQL server's configuration, especially if your server is in a different time zone than your application.

Depending on your configuration, you may need to setup SSL and ensure that the necessary certificates are properly configured. For testing purposes, you may use the following parameters:

```
jdbc:mysql://localhost:3306/mydatabase?useSSL=false&serverTimezone=UTC&
    allowPublicKeyRetrieval=true
```

where `useSSL=false` disables SSL if you're connecting over a trusted network (or in a testing environment) and `allowPublicKeyRetrieval=true` allows the retrieval of the public key for authentication, which is required if you are not using SSL.

In the ReportServer datasource configuration, provide your MySQL username and password in the respective fields.

MySQL uses the `TINYINT(1)` data type to represent boolean values (`true`/`false`) by default. In some cases, however, you may need to treat `TINYINT(1)` as a regular numeric type rather than a boolean type. In such cases, `TINYINT(1)` should not be automatically mapped to a `boolean` in ReportServer.

To prevent MySQL from treating `TINYINT(1)` as a `boolean`, you can use the `tinyInt1isBit=false` parameter in your JDBC connection string. This setting instructs the MySQL JDBC driver to treat `TINYINT(1)` columns as a numerical value (`TINYINT`) rather than converting them into a `boolean` data type.

Example JDBC URL:

```
jdbc:mysql://localhost:3306/mydatabase?useSSL=false&serverTimezone=UTC&
    tinyInt1isBit=false
```

For more information, refer to https://forum.reportserver.net/viewtopic.php?id=792.

For more detailed configuration options and parameter explanations, refer to the MySQL JDBC documentation: https://dev.mysql.com/doc/connector-j/en/connector-j-reference.html and https://dev.mysql.com/doc/connector-j/en/connector-j-reference-configuration-properties.html.

4.12 PostgreSQL

PostgreSQL is an advanced, open-source relational database system known for its strong compliance with SQL standards and extensive support for various data types and advanced features. More information on PostgreSQL can be found at https://www.postgresql.org/.

ReportServer supports PostgreSQL via its native JDBC driver, which is shipped together with ReportServer.

To connect ReportServer to a PostgreSQL database, you need to configure the appropriate JDBC URL. A typical JDBC URL for PostgreSQL might look like this:

```
jdbc:postgresql://hostname:port/dbname
```

In the ReportServer datasource configuration, enter your database username and password in the corresponding fields. These credentials should have the necessary permissions to access the database.

An example JDBC URL is shown below.

```
jdbc:postgresql://localhost:5432/reportserver
```

PostgreSQL JDBC drivers offer various parameters that can be appended to the JDBC URL for specific functionalities, such as connection timeouts, SSL encryption, and more. The full list of parameters and their usage is documented in the official PostgreSQL JDBC documentation at https://jdbc.postgresql.org/documentation/head/connect.html.

As of PostgreSQL 12, jit configuration is turned on by default. Note that this may have large performance drawbacks. You can easily turn this off by PostgreSQL configuration: https://www.postgresql.org/docs/13/runtime-config-query.html

Here you can find some information on this: https://dba.stackexchange.com/questions/257258/upgrade-from-postgres-11-to-postgres-12-made-some-queries-300x-slower-probably and https://dba.stackexchange.com/questions/257759/recursive-cte-based-on-function-values-significantly-slower-on-postgres-12-than/258493.

Background information can be found here: https://severalnines.com/blog/overview-just-time-compilation-jit-postgresql/.

When using the `stringtype=unspecified` setting in PostgreSQL, it allows the server to handle String parameters more flexibly. Normally, a `String` parameter is sent to the server as a `VARCHAR` type by default. However, with `stringtype=unspecified`, the parameters are sent as untyped values, meaning the server will try to infer the appropriate data type.

In the context of JSON, this means that even if ReportServer sends a string value, PostgreSQL won't strictly expect a `VARCHAR` type. Instead, it will automatically convert the string into a JSON data type in the database if that's what the context requires. This is particularly useful when dealing with JSON data in ReportServer as it is sent as `VARCHAR` even though the actual data type in the database is JSON.

An example configuration is shown below:

```
jdbc:postgresql://localhost:5432/reportserver?stringtype=unspecified
```

4.13 Redshift

Amazon AWS Redshift https://aws.amazon.com/redshift/ is a fast, fully managed data warehouse that makes it simple and cost-effective to analyze all your data. It allows you to run complex analytic queries against petabytes of structured data.

ReportServer supports Amazon Redshift by its official JDBC driver, downloadable from https://docs.aws.amazon.com/en_us/redshift/latest/mgmt/jdbc20-download-driver.html. Download the JDBC 4.2-compatible version and install it in your ReportServer lib directory. We strongly recommend using the external Configdir for this, as explained in the Configuration Guide. In such a way, the driver will be separated from the ReportServer libraries and will not get overwritten with your next ReportServer upgrade. At the moment of writing, the version of the JDBC driver was 2.1.0.10 (redshift-jdbc42-2.1.0.10.jar). Note that you should download the "without the AWS SDK" version: (`JDBC 4.2 compatible driver version 2.1 (without the AWS SDK)`).

As stated in the JDBC documentation, you have to obtain your JDBC URL from your AWS. An example URL of Redshift is:

```
jdbc:redshift://redshift-cluster-1.cchhqpzb0zgh.eu-central-1.redshift.amazonaws.
    com:5439/sample_data_dev
```

An example URL of Redshift serverless is:

```
jdbc:redshift://my-redshift-workspace.196198473025.eu-central-1.redshift-
    serverless.amazonaws.com:5439/sample_data_dev
```

In the ReportServer datasource user and password fields, put your database user and password previously configured in AWS. More information can be found in the Amazon Redshift documentation.

The JDBC URL parameters available are documented here: https://docs.aws.amazon.com/en_us/redshift/latest/mgmt/jdbc20-install.html.

Note: If you get the following error: "Error setting/closing connection: Not Connected" try adding to your connection pool's configuration (pool.cf) the following property:

```
<idleConnectionTestPeriod>5</idleConnectionTestPeriod>
```

With this property set, c3p0 will test all idle, pooled but unchecked-out connections, every this number of seconds. Add this property to your Redshift datasource. You can find more information on the connection pool in the Configuration Guide.

Also, note that incoming traffic should be configured correctly in order to be able to access Redshift (if you need to access from outside VPC). For this, configure your Redshift incoming traffic in your VPC security groups (using your JDBC port, in the above example 5439). You may also need to set "Publicly Accessible" to yes if you need to access it from outside VPC. Refer to the Redshift documentation for details.

4.14 SAP HANA

SAP HANA https://www.sap.com/products/technology-platform/hana/what-is-sap-hana.html is a multi-model database that stores data in its memory instead of keeping it on a disk. The column-oriented in-memory database design allows you to run advanced analytics alongside high speed transactions in a single system. Why is this so important? Because it lets companies process massive amounts of data with near-zero latency, query data in an instant, and become truly data-driven. By storing data in column-based tables in main memory and bringing online analytical processing (OLAP) and online transactional processing (OLTP) together, SAP HANA is unique and significantly faster than other database management systems (DBMS) on the market today.

ReportServer supports SAP HANA by its official JDBC driver `ngdbc.jar`, downloadable from https://www.sap.com. Download the JDBC 4.2-compatible version (e.g. from its ngdbc maven dependency available here: https://mvnrepository.com/artifact/com.sap.cloud.db.jdbc/ngdbc/ or https://jar-download.com/?search_box=ngdbc) and install it in your ReportServer lib directory. We strongly recommend using the external Configdir for this, as explained in the Configuration Guide. In such a way, the driver will be separated from the ReportServer libraries and will not get overwritten with your next ReportServer upgrade. At the moment of writing, the version of the JDBC driver was 2.15.12 (ngdbc-2.15.12.jar).

As stated in the JDBC documentation, you have to enter your connection parameters required to connect into your JDBC URL. An example URL is:

```
jdbc:sap://e3bd540b-325a-4e89-8404-1e17a1a6cbb8.hna0.prod-eu10.hanacloud.ondemand.
    com:443
```

In the ReportServer datasource user and password fields, enter your database user and password previously configured in SAP HANA. More information can be found in the SAP HANA documentation.

The JDBC URL parameters available are documented here: https://help.sap.com/docs/SAP_HANA_PLATFORM/0eec0d68141541d1b07893a39944924e/109397c2206a4ab2a5386d494f4cf75e.html.

4.15 SQL Server

Microsoft SQL Server (MSSQL) databases can by accessed by its official driver, available for download here: https://learn.microsoft.com/en-us/sql/connect/jdbc/download-microsoft-jdbc-driver-for- or here: https://mvnrepository.com/artifact/com.microsoft.sqlserver/mssql-jdbc. Please check the Java and JDBC specification matrix for the exact version depending on your environment: https://learn.microsoft.com/en-us/sql/connect/jdbc/microsoft-jdbc-driver-for-sql-server-suppor view=sql-server-ver16#java-and-jdbc-specification-support.

At the moment of writing, the latest version was 13.2.1 (13.2.1.jre11.jar, which is compatible with Java 21). Note that as ReportServer does not support Java 8, the 13.2.1.jre8.jar can not be used.

As stated in the JDBC documentation, you have to enter your connection parameters required to connect into your JDBC URL. An example URL is:

```
jdbc:sqlserver://IP:PORT;database=DATABASE
```

You may have to enter `encrypt=false` as well if you do not use encryption. Example:

```
jdbc:sqlserver://IP:PORT;database=DATABASE;encrypt=false
```

In the ReportServer datasource user and password fields, enter your database user and password previously configured in your MSSQL installation. More information can be found in the MSSQL documentation.

The JDBC URL parameters available are documented here: https://learn.microsoft.com/en-us/sql/connect/jdbc/setting-the-connection-properties.

4.16 SQL Server (Old driver, jTDS)

You can access old Microsoft SQL Server (MSSQL) installations by the jTDS driver, available for download here: https://jtds.sourceforge.net/. At the moment of writing, the latest jtds-1.3.1.jar driver supports the following Microsoft SQL Server versions: 6.5, 7, 2000, 2005, 2008 and 2012 (as per jTDS documentation).

As stated in the JDBC documentation, you have to enter your connection parameters required to connect into your JDBC URL. An example URL is:

```
jdbc:jtds:sqlserver://IP:PORT/DATABASE
```

In the ReportServer datasource user and password fields, enter your database user and password previously configured in SQL Server. More information can be found in the jTDS documentation.

The JDBC URL parameters available are documented here: https://jtds.sourceforge.net/faq.html.

4.17 SQLite

SQLite https://www.sqlite.org/ is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, file-based SQL database engine.

ReportServer supports SQLite datasources as of version 4.2.0 and contains the necessary driver as well.

An example JDBC-URL pointing to the "db" database file is the following:

jdbc:sqlite:/path/to/your/sqlite/db

If you want to use your SQLite datasource for read-only purposes, you can add the following JDBC-property to your datasource definition:

```
open_mode=1
```

4.18 Teradata

Teradata https://www.teradata.com/ is a fully scalable relational database management system produced by Teradata Corp. It is widely used to manage large data warehousing operations.

ReportServer supports Teradata datasources by the official Teradata JDBC driver. Download the driver from https://downloads.teradata.com/download/connectivity/jdbc-driver and add it to your libs directory. We strongly recommend using the external Configdir for this, as explained in the Configuration Guide. In such a way, the libraries will be separated from the ReportServer libraries and will not get overwritten with your next ReportServer upgrade.

At the moment of writing, the Teradata JDBC driver had version 16.10.00.07.

Your URL should look similar to this: jdbc:teradata://IP/DATABASE=myDatabase

4.19 YugabyteDB

YugabyteDB https://www.yugabyte.com/ is a high-performance, cloud-native, distributed SQL database that aims to support all PostgreSQL features. It is best suited for cloud-native OLTP (i.e., real-time, business-critical) applications that need absolute data correctness and require at least one of the following: scalability, high tolerance to failures, or globally-distributed deployments.

ReportServer supports YugabyteDB by its official JDBC driver, downloadable from https://docs.yugabyte.com/preview/drivers-orms/java/yugabyte-jdbc/. Download the JDBC 4.2-compatible version (e.g. from its jdbc-yugabytedb maven dependency available here: https://mvnrepository.com/artifact/com.yugabyte/jdbc-yugabytedb) and install it in your ReportServer lib directory. We strongly recommend using the external Configdir for this, as explained in the Configuration Guide. In such a way, the driver will be separated from the ReportServer libraries and will not get overwritten with your next ReportServer upgrade. At the moment of writing, the version of the JDBC driver was 42.3.5-yb-2 (jdbc-yugabytedb-42.3.5-yb-2.jar).

As stated in the JDBC documentation, you have to enter your connection parameters required to connect into your JDBC URL. An example URL is:

```
jdbc:yugabytedb://us-west-2.52d9ad18-1759-4973-bc46-75eb219218f0.aws.ybdb.io:5433/
    yugabyte?load-balance=true&yb-servers-refresh-interval=240&topology-keys=
    cloud.region.zone1,cloud.region.zone2&connectTimeout=120&loginTimeout=120&
    socketTimeout=120
```

In the ReportServer datasource user and password fields, enter your database user and password previously configured in YugabyteDB. More information can be found in the YugabyteDB documentation.

Note that in some cases you might need to set the timeouts: `connectTimeout`, `loginTimeout` and `socketTimeout`.

The JDBC URL parameters available are documented here: https://github.com/yugabyte/jdbc-yugabytedb/blob/master/docs/documentation/94/connect.md.

4.20 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 datasource 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 datasource password field always being empty upon reloading the form. However, you may safely change the datasource, 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 hashed when saving. For further information, please refer to the configuration instructions.

4.21 Datasource Pool

Datasource 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.

4.22 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 `https://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 datasource.
Argument-Connector:	Allows to directly enter data in a text field when selecting the datasource. 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 (per URL) from the internal ReportServer file system (refer here to the Section File Server). In order for this to work, note you have to select the "Share folder for web access" checkbox of the folder containing your CSV file.

Database Cache

As described before, prior to its use, CSV data will be loaded to internal temporary tables. 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 temporary tables. 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 it.

If you need to reload the data manually (for example, because it has changed), it is sufficient to simply save the datasource again. Each save operation causes ReportServer to clear the corresponding entries from the internal cache and reload the data.

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 datasources at the location where the datasource will be selected (e.g. at the report or parameter). As already explained, CSV data will be buffered to internal, temporary tables. The query type to be used is

```
SELECT * FROM _RS_TMP_TABLENAME
```

where `_RS_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:

_RS_TMP_TABLENAME:	The name of the table
_RS_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 $!{_RS_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 7.3 Working with Parameters on page 82).

If you wish to use CSV datasources together with Jasper, BIRT, or JXLS reports, refer to the Scripting Documentation, Chapter "Script Datasources", as this can be done in an analogous way.

4.23 Script datasources

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 datasources. Script datasources provide data by running a ReportServer script and, therefore, they can be applied very flexibly. Similar to CSV lists, the result of a script datasource will first be buffered in the internal temporary tables. Here as well, you will have the option to define in the **Cache database** setting how often the data will be reloaded.

Script datasources run a ReportServer script whenever the datasource is accessed. This script will be filled with the datasource 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.*

TableDefinition definition = new TableDefinition(
    ['a',        'b',        'c'],
    [Integer,    Integer,    Integer]
)

RSTableModel 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 datasources. In addition, you can pass arguments to the script, which can be referred in the script with the `args` variable. E.g., refer to the following example:

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

TableDefinition definition = new TableDefinition(
   ['a',        'b',        'c'],
   [String,     String,     String]
)

def model = new RSTableModel(definition)
model.addDataRow(args[0],"2","3")
model.addDataRow("4","5","6")
model.addDataRow("7","8","9")

return model
```

The `args[0]` prints the 0th argument passed to the script. You can either pass a text, e.g."myValue", or the value of a given report parameter, e.g. `${myParam}` for a "myParam" parameter. Note that if the value contains blank spaces, quotation marks are needed.

Refer to the Script Guide, Chapter "Script Datasources" for more details. Further, if you wish to use script datasources together with Jasper, BIRT, or JXLS reports, also refer to the Scripting Documentation, Chapter "Script Datasources".

Further, useful script datasource examples can be found in our GitHub samples project: https://github.com/infofabrik/reportserver-samples.

4.24 BIRT Report datasource

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 datasources, you can access this data in ReportServer.

As the most frequent application case for BIRT Report datasources will surely be the reading out of parameters, the BIRT datasource will directly be configured at the report. This means you only have to create a datasource in the datasource tree, any further configuration will be entered at the location where it is used. When using the datasource, 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 datasources, you can modify the query by using the **Query wrapper** configuration.

4.25 Mondrian Datasource

Mondrian is a java-based OLAP engine (Online Analytical Processing) developed by Pentaho (`https://github.com/pentaho/mondrian`) 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 `https://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 7.7 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`. 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.

ReportServer provides an example configuration for Mondrian properties, which points to an H2 database named ***foodmart***[2]. Below is the relevant configuration:

```
type=OLAP
name=foodmart
driver=mondrian.olap4j.MondrianOlap4jDriver
jdbcDrivers=org.h2.Driver
```

The demo JDBC URL is a specialized H2 URL:

```
rs:mondrian:demodata
```

If you are using your own datasource, your JDBC URL should be structured similarly to the following example for Microsoft SQL Server:

```
jdbc:mondrian:Jdbc=jdbc:sqlserver://IP:PORT;jdbc.databaseName=DBNAME
```

Please take note of the following key points:

- The JDBC URL must begin with the `jdbc:mondrian:Jdbc=` prefix.
- The specific JDBC URL for your driver follows. For Microsoft SQL Server, this URL is: `jdbc:sqlserver://IP:PORT;jdbc.databaseName=DBNAME`. Refer to Section 4.15 for details.
- All parameters in your JDBC URL should be prefixed with `jdbc.`. For example, if your original JDBC URL is `jdbc:sqlserver://IP:PORT;databaseName=DBNAME`, you should modify it to `jdbc:sqlserver://IP:PORT;jdbc.databaseName=DBNAME`.

[2]The foodmart database is Mondrian's demo dataset, showcasing various cubes that demonstrate Mondrian's OLAP capabilities.

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>
  <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., https://en.wikipedia.org/wiki/Multidimensional_Expressions). In order to use Mondrian within ReportServer you will need to create so called Saiku reports (see Chapter 7.7 Saiku / Mondrian Reports) which provide a beautiful user interface to access a data specified in a cube in a Pivot like fashion.

4.26 Using XMLA with SQL Server Analysis Services

In Section 4.25 we discussed how to input your entire Mondrian cube definition into ReportServer to generate reports based on this. Alternatively, by using XMLA, you can manage your cube in SQL Server Analysis Services (SSAS) and use XMLA (supported by ReportServer) for communication.

In this case, you should configure HTTP access to Analysis Services on IIS as explained here: https://learn.microsoft.com/en-us/analysis-services/instances/configure-http-access-to-analysis-se Specifically, you can enable HTTP access by configuring MSMDPUMP.dll, an ISAPI extension that runs in Internet Information Services (IIS) and pumps data to and from client applications and an Analysis Services server.

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

After you have configured Analysis Services HTTP access, you can enter the following JDBC URL into ReportServer:

```
jdbc:xmla:Server=http://IP:80/OLAP/msmdpump.dll
```

where `/OLAP/msmdpump.dll` is the path to the `msmdpump.dll` that you configured earlier. Since port 80 is the default for HTTP, you can omit it from the URL.

You can enter the following properties:

```
type=OLAP
name=Reporting
driver=org.olap4j.driver.xmla.XmlaOlap4jDriver
```

Your "cube" section should be left empty, as your cube definition is managed in Analysis Services.

You can also enter additional properties needed by your specific needs, e.g. regarding cache:

```
type=OLAP
name=Reporting
driver=org.olap4j.driver.xmla.XmlaOlap4jDriver
Cache=org.olap4j.driver.xmla.cache.XmlaOlap4jNamedMemoryCache
Cache.Name=ConnectionCache
Cache.Mode=LFU
Cache.Timeout=600
Cache.Size=1000
```

4.27 Datasource Bundle

The **datasource bundle** allows you to use the same report for different datasources and 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 parameter that is used as the key source.

4.28 Configuration of a Standard Datasource

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <datasource>
    <defaultDatasourceKey>DEMO_DATA</defaultDatasourceKey>
  </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 18 Terminal.

Chapter 5

Datasinks

Datasinks are the counterpart of datasources in ReportServer. Reports can be sent to any datasink defined, e.g. to a FTP or SFTP datasink. Reports may be sent directly to a datasink (via the SendTo button) or scheduled to be sent to a specific datasink. Multiple datasinks of the same type are allowed, analogously as datasources. Further, as with most other objects in ReportServer, datasinks are maintained in a hierarchical structure. The datasink management module can be found in the Administration module under Datasinks. The following object types can be created in the tree:

Folder Serve to structure datasinks.

Datasink Here various datasinks are optionally available which we will discuss in more detail in the following.

Datasinks will be configured in two steps. In datasink management, datasinks will be created and the basic settings made. For FTP datasinks, here, for instance, user name, password, default folder and FTP URL are stored. However, the specific configuration per use will be set at the point where the datasink will be used (this is mostly with the respective report). Here, for instance, for a FTP datasink, the specific folder can be set where the report should be stored. The default folder can be used or a new folder can be defined for the specific use.

Datasinks may either completely disabled, or disabled for a specific type, or scheduling may be disabled for this type. Disabling datasinks may be done in `/fileserver/etc/datasinks/datasinks.cf`. Details on this can be found in the Configuration Guide.

ReportServer supports the following datasinks.

- Email - SMTP
- Table datasinks (available in Enterprise Edition)
- SFTP
- FTPS

- FTP
- Samba - SMB/CIFS (available in Enterprise Edition)
- Amazon S3 (available in Enterprise Edition)
- SCP (available in Enterprise Edition)
- Local filesystem (available in Enterprise Edition)
- Printer (available in Enterprise Edition)
- Script datasinks (available in Enterprise Edition)
- OAuth2-authenticated datasinks:
  - Dropbox (available in Enterprise Edition)
  - OneDrive - SharePoint (O365) (available in Enterprise Edition)
  - Google Drive (available in Enterprise Edition)
  - Box (available in Enterprise Edition)

5.1 Email - SMTP

Email SMTP datasinks enable the sending and scheduling of reports via an SMTP server. The configuration for an Email SMTP datasink can be managed within the Administration area, and its permissions are administered similarly to other objects in ReportServer.

To allow ReportServer to send emails, you need to specify the appropriate mail server settings. Configure your Email SMTP datasink as follows.

Setting up the SMTP server. Provide the necessary values for `host`, `port`, `username`, and `password` according to your SMTP server's configuration.

> ***Important***: If your SMTP server does not require a password, simply removing the password characters in your Email SMTP datasink will not be sufficient, as ReportServer will internally store an empty string. To completely remove the password, right-click on the password field and select "Remove password." This will set the password to `NULL`, ensuring compatibility with servers that do not require authentication.

```
Host: mail.yourmailserver.com
Port: 25
Username: rs@yourmailserver.com
Password: passwordsecret
SSL: false
TLS enable: false
TLS require: false
```

If you are using SSL or TLS please also specify these values. Next, configure the sender name, email address and forceSender options. If the forceSender option is set to true, the emails will be sent using the given (generic) sender details. If set to false, the specific user sending the email will determine the sender details.

```
Sender: rs@yourmailserver.com
Sender name: ReportServer
Force sender: false
Encryption policy: allow_mixed
```

The encryption policy option controls whether or not mails have to be encrypted or whether it is ok to send mails unencrypted if a user's public key is not specified. Choose between `strict` and `allow_mixed`. Note that if you choose `strict` then mails to users that do not have public key registered with ReportServer will not receive any messages.

Note that you can specify a default datasink per datasink type in the `/etc/datasinks/datasinks.cf` configuration file. Details can be found in the Configuration Guide.

5.2 Table datasinks

Table datasinks allow you to send and/or schedule dynamic list reports into to a given table in any datasource supported by ReportServer. This allows you to easily transfer data from MySQL to Oracle, for example. The destination table must exist and must be compatible with the dynamic list, i.e. the fields must exist, have the same name, and have compatible data types.

Table datasink configuration can be made in the Administration area. Its permissions can be administrated analogous as other objects in ReportServer.

In order to configure a table datasink, the following must be set:

Datasource The datasource to send the data to.

Destination table name The table name. This table must exist in the given datasource, be reachable, and compatible to the dynamic list transfering the data.

;-separated list of primary keys in destination table The destination table may of course contain one or more primary keys. These can be entered here, separating them by ";". For example, if your destination table has one primary key "ENTITY_ID", you can enter "ENTITY_ID". If your destination table has two primary keys "ENTITY_ID" and "ENTITY_ID2", you can enter "ENTITY_ID;ENTITY_ID2".

Copy primary keys If your destination primary keys are of autoincrement-type, their values should not be copied to the destination table, as your DB calculates the ids automatically. In this case, you should not set this checkbox. However, if you want to copy the primary key values, you should set this checkbox.

Batch size The table is copied in batches for performance reasons. You may adapt the batch size here.

When you schedule a dynamic list to a table datasink, you should select "Stream table" as an export type.

Note that you can specify a default datasink per datasink type in the /etc/datasinks/datasinks.cf configuration file. Details can be found in the Configuration Guide.

5.3 SFTP

SFTP datasinks allow to send and/or schedule reports to a SFTP server. SFTP datasink configuration can be made in the Administration area. Permissions to SFTP datasinks can be given analogously as to other objects in ReportServer.

The host can be entered using the following format:

sftp://sftpHost

Currently, ReportServer supports the following authentication methods:

- Username Password Authentication
- Public Key Authentication

All passwords and private keys are encrypted and saved into the ReportServer database. Note that the private key passphrase may be empty, but it is highly recommended to use a passphrase.

Further, note that for SFTP to work, you have to add your SFTP host to your /etc/ssh/known_hosts file (https://en.wikibooks.org/wiki/OpenSSH/Client_Configuration_Files#~/.ssh/known_hosts) in order to verify the identity of the remote host, thus protecting against impersonation or man-in-the-middle attacks. Its location can be configured in the /fileserver/etc/security/misc.cf file as described in the Configuration Guide. For manually adding a public key to the /etc/ssh/known_hosts file, check here: https://en.wikibooks.org/wiki/OpenSSH/Client_Configuration_Files#Manually_Adding_Public_Keys_to_~/.ssh/known_hosts.

E.g.: Use ssh-keyscan IP » known_hosts in order to add the given IP to your known_hosts file.

Note that if the (extension) folder/folders does/do not exist, it/they is/are created. If the report already exists in the same path, it is overwritten.

5.4 FTPS

FTPS datasinks allow to send and/or schedule reports to a FTPS server. FTPS datasink configuration can be made in the Administration area. Permissions to FTPS datasinks can be given analogously as to other objects in ReportServer.

The host can be entered using the following format:

ftps://sftpHost

Currently, ReportServer supports the following authentication method:

- Username Password Authentication

All passwords are encrypted and saved into the ReportServer database.

Note that if the (extension) folder/folders does/do not exist, it/they is/are created. If the report already exists in the same path, it is overwritten.

5.5 FTP

FTP datasinks allow to send and/or schedule reports to a FTP server. FTP datasink configuration can be made in the Administration area. Active and passive FTP modes are supported. Permissions to FTP datasinks can be given analogously as to other objects in ReportServer.

The host can be entered using the following format:

`ftp://ftpHost`

Note that if the (extension) folder/folders does/do not exist, it/they is/are created. If the report already exists in the same path, it is overwritten.

> Note that that FTP datasinks are supported as of ReportServer 3.4.0. For scheduled reports using the legacy FTP functionality, a dummy FTP datasink is created on ReportServer start (found in the Administration area, in the datasink root folder). This dummy FTP datasink must be adapted in order for these reports to further work.

5.6 Samba - SMB/CIFS

Samba - SMB/CIFS datasinks allow to send and/or schedule reports to a given Samba server using SMB/CIFS protocols. Samba - SMB/CIFS datasink configuration can be made in the Administration area. Permissions to local Samba - SMB/CIFS datasinks can be given analogously as to other objects in ReportServer.

The host can be entered using the following format:

`smb://sambaHost`

Note that the path must exist and must be writable by your Samba's user. If the report already exists in the same path, it is overwritten.

If you get the error "SmbException: The network name cannot be found.", you should probably add or correct your domain name or check if the folder is correct. Note that the folder must be complete, e.g. if your Samba share is "myshare", and you have a directory "temp" inside this share, your folder should be "myshare/temp".

5.7 Amazon S3

Amazon S3 datasinks allow to send and/or schedule reports to a given S3 bucket. The datasink configuration can be made in the Administration area. Permissions to local S3 datasinks can be given analogously as to other objects in ReportServer.

You can create an access and a secret key as described here: https://docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html#access-keys-and-secret-access-keys. Further, the unique bucket name and region must be entered into the datasink definition.

If the report already exists in the same path, it is overwritten.

More information can be found here: https://docs.aws.amazon.com/en_en/AmazonS3/latest/userguide/Welcome.html.

5.8 SCP

SCP datasinks allow to send and/or schedule reports to a given SCP (SSH) server. SCP datasink configuration can be made in the Administration area. Permissions to local SCP datasinks can be given analogously as to other objects in ReportServer.

The host can be entered using the following format:

`scp://scpHost`

Note that the path must exist and must be writable by your SCP user. An absolute destination folder is required, so relative paths like "./" do not work here. If the report already exists in the same path, it is overwritten.

Currently, ReportServer supports the following authentication methods:

- Username Password Authentication
- Public Key Authentication

All passwords and private keys are encrypted and saved into the ReportServer database. Note that the private key passphrase may be empty, but it is highly recommended to use a passphrase.

Further, note that for SCP datasinks to work, you have to add your SCP server host to your `/etc/ssh/known_hosts` file (https://en.wikibooks.org/wiki/OpenSSH/Client_Configuration_Files#~/.ssh/known_hosts) in order to verify the identity of the remote host, thus protecting against impersonation or man-in-the-middle attacks. Its location can be configured in the `/fileserver/etc/security/misc.cf` file as described in the Configuration Guide. For manually adding a public key to the `/etc/ssh/known_hosts` file, check here: https://en.wikibooks.org/wiki/OpenSSH/Client_Configuration_Files#Manually_Adding_Public_Keys_to_~/.ssh/known_hosts.

E.g.: Use `ssh-keyscan IP » known_hosts` in order to add the given IP to your `known_hosts` file.

5.9 Local Filesystem

Local filesystem datasinks allow to send and/or schedule reports to a given directory of your local filesystem, i.e. where your ReportServer is installed. Local filesystem datasink configuration can be made in the Administration area. Permissions to local filesystem datasinks can be given analogously as to other objects in ReportServer.

You can enter a path to a given folder in your filesystem as follows:

`/users/myuser/myreports`

in Linux/Mac, and

`C:/myuser/myreports`

in Windows.

The path in the datasink definition determines the base path of the datasink. It also contains a default folder/default folders, which can be overriden in specific uses of this datasink, analogously to other datasink types.

Note that the base path must exist and must be writable by your Tomcat's user. If the (extension) folder/folders does/do not exist, it/they are created. If the report already exists in the same path, it is overwritten.

5.10 Printer Datasinks

Printer Datasinks allow to send and/or schedule reports to a given printer. Printer datasink configuration can be made in the Administration area. Permissions to printer datasinks can be given analogously as to other objects in ReportServer.

Note that, while scheduling, you have to select the "PDF" export type in order to use printer datasinks.

> Be careful when sending large reports to the printer, this may stuck your printer depending on your configuration.

5.11 Script Datasinks

Script Datasinks allow you to send and/or schedule reports to virtually any custom location you need. They also allow you to add custom files/logic to your datasinks. Script datasink configuration can be made in the Administration area. Permissions to script datasinks can be given analogously as to other objects in ReportServer.

Details on script datasinks and an example datasink can be found in the Script Guide.

5.12 OAuth2-authenticated datasinks

OAuth2-authenticated datasinks use, as their name suggest, OAuth2 https://oauth.net/2/, the industry-standard protocol for authorization.

Before being able to use oauth2-datasinks, you have to create an app in the service needed and authorize file uploads in this app. You can then add your app key and app secret into the respective ReportServer fields and save the datasink.

When this is achieved, you can click the "Datasink OAuth2 Authentication Setup", which redirects you to the service authorization. Please note that you have to add the "redirect URI" into your app. This redirect URI is shown when clicking the before-mentioned button. The URI should be similar as this: http://localhost:8080/ReportServer/reportserver/oauth. When this is done, after app authorization and automatic redirection, you should be able to use your oauth2-datasinks. For testing your datasink, as with other ReportServer datasinks, you can use the "Test datasink" button.

5.13 Dropbox

Dropbox OAuth2-authorized datasinks allow you to send and/or schedule reports to a given directory in your Dropbox https://www.dropbox.com/ account. Note that Dropbox for Business is also supported: https://www.dropbox.com/business/.

For getting your "app key" and "secret key", you have to create an app here: https://www.dropbox.com/developers/apps/ and give it the appropriate permissions. Specifically, `files.content.write` and `files.metadata.write` is needed. More information on this can be found here: https://developers.dropbox.com/de-de/oauth-guide. Note that when permissions are changed, it is necessary to create a new "refresh token" with the "Datasink OAuth Authentication Setup" button.

The path in the datasink definition determines the base path of the datasink. It also contains a default folder/default folders, which can be overriden in specific uses of this datasink, analogously to other datasink types.

Note that the base path must exist. If the (extension) folder/folders does/do not exist, it/they is/are created. If the report already exists in the same path, it is overwritten. The path should be entered like this: `/my/path`. Note that, different as in other datasinks, `./my/path` does not work.

5.14 OneDrive - SharePoint (O365)

OneDrive - SharePoint (O365) OAuth2-authorized datasinks enable you to send and schedule reports directly to a specified directory in your OneDrive or SharePoint (O365) account (https://onedrive.com/). This functionality supports **OneDrive for Business (Office 365)** as well as **SharePoint O365**.

Please Note: Older versions of SharePoint are not supported.

Below, you will find a step-by-step guide for setting up a OneDrive - SharePoint (O365) datasink in ReportServer.

1. Configure the Datasink in ReportServer

1. Log in to ReportServer with the necessary permissions.
2. Navigate to **Administration → Datasinks**.
3. Right-click on **Datasink Root** (or on any datasink folder), select **Insert**, and choose **OneDrive - SharePoint (O365)**.
4. Fill in the required fields as follows:
   a) **Name**: Enter a meaningful name for the datasink.
   b) **Tenant ID**: For the initial setup, leave the default value `"common"`.
   c) **Base Root**: Leave this at the default setting (`/me/drive/items/root:`) for the initial setup. Details on custom drive configuration are provided below.
   d) **Folder**: Set this to `/` (default) or specify a subfolder path if needed.
   e) **App Key and Secret Key**: For the initial setup, enter `0` in both fields.
5. Click **Apply** to save the settings.

2. Obtain and Register App Credentials in Microsoft Azure

1. In ReportServer, click the **Datasink OAuth2 Authentication Setup** button, located at the top of the screen next to the "Test Datasink" button.
   a) A new window opens, displaying the Redirect URI. Ensure this URI uses the **"HTTPS"** protocol.
   b) Copy the URI and close the window using the **"(x)"** button.
2. A OneDrive Administrator must register the app in Microsoft Azure:
   a) Log in to the Microsoft Azure Portal at `https://portal.azure.com/`.
   b) Navigate to **"App Registrations → New Registration"**.
   c) Provide a **"Display Name"** for the app.
   d) Select **"Accounts in this organizational directory only"** under **"Supported Account Types"** (or as per organizational requirements).
   e) Select the **"Web"** option in the **platform** dropdown menu.
   f) Enter the Redirect URI from the ReportServer setup in the field next to the **"Web"** option.
   g) Click **"Register"** to complete the registration.
3. A OneDrive Administrator must assign the required API permissions to the app:
   a) Navigate to **"Manage → API Permissions → Add Permissions"**.
   b) Choose **"Microsoft Graph → Application Permissions"**.

c) Enter “Files” in the search field.
d) Select **“Files.ReadWrite.All”**.
e) Click **“Add Permissions”** to finalize.

4. A OneDrive Administrator must generate the secret key:

   a) Navigate to **“Manage → Certificates & Secrets → New Client Secret”**.
   b) Provide a description (e.g., `‘‘ReportServer Key’’`) and specify a validity period (e.g., `‘‘1 year’’`).
   c) Click on **“Add”**.
   d) **IMPORTANT**: After creating the key, **make sure to note down its value immediately**, as it will be required for the configuration and ***cannot be retrieved later***.

3. Finalize the Datasink Configuration in ReportServer

1. Return to **Administration → Datasinks** in ReportServer and open the created datasink.

2. Update the following fields:

   a) **App Key**: Enter the Application (Client) ID from the registered app in Azure. Your administrator can find this in the overview of the registered app in Azure, under *Application (client) ID*.
   b) **Secret Key**: Enter the value of the secret key generated in Azure in Step 2.
   c) **Tenant ID**: Enter the tenant ID. Your administrator can find this in the overview of the registered app in Azure, under *Directory (tenant) ID*.

3. Click **Apply** to save the settings.

4. Complete the authentication process:

   a) Select the datasink and click the **Datasink OAuth2 Authentication Setup** button.
   b) Click **Start Authentication Process** and follow the prompts to grant permissions.
   c) Return to ReportServer and click the **Test Datasink** button.

A test file will be created in the user’s OneDrive under **My Files**. You can check this here, e.g.,: `https://your-company-my.sharepoint.com/my`.

4. Configuring a Company-Specific Drive (SharePoint)

If a company-related SharePoint drive needs to be integrated:

1. The user (account) must have access to (must be a member of) the SharePoint site. This ensures the user can upload files to the Documents area.

2. The user must obtain the **Group ID** and **Drive ID** for the desired drive. This can be done through two methods: directly via ReportServer (recommended) or using Microsoft Graph Explorer. Both approaches are outlined below.

Using the ReportServer Terminal (RECOMMENDED):

1. Obtain the user's Access Token from Graph Explorer: `https://developer.microsoft.com/en-us/graph/graph-explorer` (Access token tab).
2. Execute the following commands in the ReportServer terminal:
   a) **onedrive group getmygroups [path datasink] [accessToken]**: Lists all groups accessible to the user. Copy the Group ID.
   b) **onedrive group getdrivesof [group-id] [path datasink] [accessToken]**: Lists all drives for the group. Copy the Drive ID.

Using Microsoft Graph Explorer:

1. Log in to Microsoft Graph Explorer: `https://developer.microsoft.com/en-us/graph/graph-explorer` with the user's account.
2. Retrieve the Group ID:
   a) **Endpoint**: `https://graph.microsoft.com/v1.0/me/memberOf`. Identify the desired group and copy its Group ID.
3. Retrieve the Drive ID:
   a) **Endpoint**: `https://graph.microsoft.com/v1.0/groups/GROUP-ID/drives`. Verify the drive (e.g., by checking its webUrl) and copy the Drive ID.

Update the Datasink Configuration in ReportServer:

Using the Drive ID for setting a **Base Root** in ReportServer:

Base Root Enter the Drive ID obtained from either the ReportServer terminal or Microsoft Graph Explorer into the `Base Root` field of the datasink configuration as follows: `/drives/DRIVE-ID/root:`

Folder Specify the folder path if necessary (e.g., `/Reports/`).

1. Save and test the configuration by clicking the **Test Datasink** button.

Additional Notes:

- An OneDrive Administrator must ensure all API permissions and keys are properly configured in Azure. For more information, refer to: **Microsoft Graph API Documentation**: `https://learn.microsoft.com/en-us/graph/api/resources/onedrive?view=graph-rest-1.0`
- To use ReportServer terminal commands, refer to Section 20.46 onedrive in the documentation.
- When permissions are changed, it is necessary to create a new `refresh token` with the **Datasink OAuth Authentication Setup** button.

- You can use the default `tenant id` if a custom `tenant id` is not needed or if you don't have a custom `tenant id`.
- Note that the base path must exist. The base path is the `Base Root` of your datasink together with the `Folder` path.
- If the report already exists in the same path, it is overwritten.
- The path should be entered like this: `/my/path`. Note that, unlike in other datasinks, `./my/path` does not work.

By following these detailed steps, both users and administrators can successfully configure the OneDrive - SharePoint (O365) datasink in ReportServer.

5.15 Google Drive

Google Drive OAuth2-authorized datasinks allow you to send and/or schedule reports to a given directory in your Google Drive https://drive.google.com/ account.

For getting your "app key" and "secret key", you have to create an app registration here: https://console.developers.google.com/ (Google API Console) and give it the appropriate API permissions.

The "base root" path in the datasink definition, together with the "folder" path, determines the base path of the datasink. The default folder/folders are also included, which can be overriden in specific uses of this datasink, analogously to other datasink types.

Note that the base path must exist. If the (extension) folder/folders does/do not exist, it/they is/are created. If the report already exists in the same path, it is overwritten. The path should be entered like this: `/my/path`. Note that, different as in other datasinks, `./my/path` does not work.

More information on this can be found here: https://developers.google.com/drive/api/v3/about-sdk.

5.16 Box

Box OAuth2-authorized datasinks allow you to send and/or schedule reports to a given directory in your Box Drive https://box.com/ account.

For getting your "app key" and "secret key", you have to create an app registration here: https://app.box.com/developers/console (Box Developer Console) and give it the appropriate API permissions.

The "base root" path in the datasink definition, together with the "folder" path, determines the base path of the datasink. The default folder/folders are also included, which can be overriden in specific uses of this datasink, analogously to other datasink types.

Note that the base path must exist. If the (extension) folder/folders does/do not exist, it/they is/are created. If the report already exists in the same path, it is overwritten. The path should be entered like this: `/my/path`. Note that, different as in other datasinks, `./my/path` does not work.

More information on this can be found here: https://developer.box.com/reference/.

Chapter 6

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 datasources (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 subdirectory `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 datasources (script datasources) 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).

6.1 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.

6.2 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 (`https://groovy-lang.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 21 ReportServer Scripting. For a detailed introduction to ReportServer scripts we refer to the separate scripting guide.

6.3 Rendering Graphviz DOT files

You can render Graphviz DOT files as SVG graphs directly in the RS file system if your DOT file contains the content type “text/vnd.graphviz”. Note that this is necessary for being able to convert the DOT to SVG.

> Note that if the file’s content type is not set to “text/vnd.graphviz”, you won’t see the preview tab or preview buttons.

You can find more information on Graphviz DOT in ReportServer in Section 14.

6.4 Rendering Markdown files

You can render Markdown (`.md`) files as HTML directly in the ReportServer file system if your Markdown contains the content type “text/markdown”. Note that this is necessary for being able to convert the markdown to HTML.

> Note that if the file’s content type is not set to “text/markdown”, you won’t see the preview tab or preview buttons.

You can find more information on Markdown in ReportServer in Section 13.

6.5 Accessing Resources by URL

Accessing files in the ReportServer File system via URL is possible, which greatly facilitates the embedding of images in reports. The corresponding URL format is:

`http://SERVER/APPLICATIONFOLDER/reportserver/fileServerAccess?id=123`

In this context, `SERVER` represents the server address (e.g., `demo.raas.datenwerke.net`), and `APPLICATIONFOLDER` corresponds to the installation path in your application server (e.g., Tomcat). The default setting is "reportserver". By utilizing the property ID, you can directly access a specific file.

You also have the option to access the file using its key:

`http://SERVER/APPLICATIONFOLDER/reportserver/fileServerAccess?key=MY_KEY`

where `MY_KEY` represents the key associated with your file.

Furthermore, accessing the file by its path is supported:

`http://SERVER/APPLICATIONFOLDER/reportserver/fileServerAccess?path=/path/to/your/file`

For example, the following URL retrieves the contents of your `alias.cf` configuration file:

`http://SERVER/APPLICATIONFOLDER/reportserver/fileServerAccess?path=/etc/sso/ldap.cf`

Additionally, you can utilize the following URL attributes:

`id`	Represents the ID of a file, for example: `id=123`.
`key`	Denotes the key associated with your file, such as: `key=MY_KEY`.
`path`	Specifies the path to a file, for instance: `path=/resources/images/img.jpg`.
`thumbnail`	If the file is of type image/png or image/jpeg, it can be automatically scaled. Activating the `thumbnail` option enables scaling mode in ReportServer. Example: `thumbnail=true`.
`twidth`	Defines the width of the resulting thumbnail preview image, e.g., `twidth=200`. Note that `thumbnail` must be set to `true` for this attribute to take effect.
`download`	Setting this attribute to `true` results in the file being downloaded instead of displayed in the browser. Example: `download=true`.
`folder`	Setting this attribute to `true` allows you to download an entire folder as a ZIP file. Note that `download` must be `true` for this attribute to work.
`render`	Setting this attribute to `true` enables rendering of a specified file instead of displaying the original file. Currently supported file types include DOT and Markdown. Ensure that `format` is set and the file's content type is specified. Refer to Section 14 and 13 for more details.
`format`	In conjunction with `render`, allows a file to be directly rendered. Supported formats include: `html` for Markdown, and `svg`, `png`, `ps`, `json`, `json0`, `xdot`, `plain`, and `plain_ext` for DOT. For more details on supported DOT formats, refer to `https://github.com/nidi3/graphviz-java`.

Please note that to access a file via URL, reading rights are required by default (see Chapter 3 User and Permission Management). This means that users must be logged in to the system. To enable full file sharing, consider placing them in a folder with the **Share folder for web access** option enabled.

Complete folders can also be accessed, but only via `id` and `path`, as they do not currently support keys. For folders, the following properties are available:

`id`	Represents the ID of a folder, for example: `id=123`
`path`	Indicates the path leading to a folder, e.g., `path=/resources/images`
`folder`	This attribute should be set to `true` for folders. It is a mandatory field for folders, e.g., `folder=true`
`download`	This attribute should also be set to `true` for folders. It is a mandatory field for folders, e.g., `download=true`

As an example, the following URL downloads a ZIP file containing your `/resources` folder:

`http://SERVER/APPLICATIONFOLDER/reportserver/fileServerAccess?path=/resources&folder=true&download=true`

Chapter 7

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.

7.1 Fundamentals

As with users and datasources, 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 7.2.

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 (`https://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 7.4.

Eclipse BIRT

Eclipse BIRT (https://eclipse-birt.github.io/birt-website/) is the second open report engine next to JasperReports. BIRT is based on the Eclipse platform (https://www.eclipse.org/) and includes a comprehensive designer for visualizing reports. Just 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 7.5.

Crystal Reports

Crystal Reports is a commercial reporting engine developed by SAP AG (https://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.

Saiku Reports – Multi-dimensional Reporting

Saiku reports allow you to access Mondrian datasources (see Section 4.25). The user interface is provided by Saiku (https://docs.meteorite.bi/) 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 (https://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. Note that JXLS2 is supported in ReportServer. The legacy JXLS1 is not supported.

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.

7.2 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 datasource other than a relational database has been selected for your report, the data will be buffered in an internal database as described in Section "Datasources". 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 datasource, 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 datasource (see Section **??**), 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 9 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
```

If you need to modify the query before it is being executed, take a look at the `net.datenwerke.rs.base.service.dbhelper.hooks.StatementModificationHook`.

Common Table Expressions (CTEs)

Because of the same reasons as explained above, you have to take special care if you wish to use Common Table Expressions (CTEs) together with dynamic lists. Refer to the following example in MSSQL syntax

```
WITH USERS_CTE (username, firstname, lastname)
AS  (
SELECT username, firstname, lastname from RS_USER
)
SELECT * from USERS_CTE
```

If you write this valid query into your dynamic list's query, you will get an error. As explained above, this error appears because of the query generated by ReportServer. In order to solve this problem, you can mark the CTE expression as follows.

```
/*<rs:cte>*/
WITH USERS_CTE (username, firstname, lastname)
AS  (
SELECT username, firstname, lastname from RS_USER
)
/*</rs:cte>*/
SELECT * from USERS_CTE
```

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 subset from the available report attributes (columns). This subset 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, https://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 datasource and configure it (configuration depends on the datasource type, refer to chapter 4). Normally, you will use data from relational databases (you can use the demo datasource 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 datasource and only refer to other datasource types if the configuration appears to be fundamentally different.

Based on the demo datasource, 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 datasource 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 datasource configuration you can easily test the currently entered query.

The Metadata Datasource

The optional metadata datasource serves to further define the report's base data fields. If there is no metadata datasource given, the users will only see the technical database name each when selecting attributes/columns. By using the metadata datasource it is possible to enter an additional plain text name and description per attribute. Further, the default column width in the dynamic list preview can be set here (per attribute). ReportServer expects the datasource output to consist of three or four columns, where the first one shows the technical column name, the second one gives the appurtenant plain text name, the third one the appurtenant description, and the fourth (optional) the appurtenant default column width in the dynamic list preview.

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

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

Note that this query has only three columns. As mentioned above, the fourth one is optional (default column width in the dynamic list preview).

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

If you want to set the default column width in the dynamic list preview, you can write a similar query analogous to:

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

The query would then result in e.g.:

COLUMN_NAME	DEFAULT_ALIAS	DESCRIPTION	DEFAULT_WIDTH
addressLine1		First line of the address	100
addressLine2		Second line of the address	200
amount		Amount of the payment effected	150
buyPrice		Purchase price	50
checkNumber		Payment reference number	100

Please note that the demo data does not include a default column width, as this is optional, so the query above will not work with the demo data. If you need a default column width in your datasource, create a similar query as the above in your metadata datasource definition.

The metadata datasource further allows you to link your dynamic list report to a URL. Details of this can be found in the next section.

Linking Dynamic Lists to URLs

ReportServer allows you to link your dynamic lists to a (possibly external) URL. This may be very practical if the link contains additional information of the record clicked on. Useful examples include opening a new report based on a given id (e.g. by httpauthexport, refer to Section 7.11 for more details), performing a Google search based on a given value, etc.

This is achieved in ReportServer by using the metadata datasource described in Section 7.2. Along with the four columns described in Section 7.2, the metadata datasource query may contain a fifth column for this purpose. Therefore, the fourth column (default column width in the dynamic list preview) is required in this case although it is normally optional. Its value may be of course NULL if no default width is required.

An example query containing the five columns could be:

```
SELECT column_name, default_alias, description, default_width,
'linkto|http://URL/reportserver/httpauthexport?&user=myuser&id=20616&format=PDF&
    p_id=$' + '{_value}' as link
FROM METADATA
```

Note that the string begins with a ***linkto|*** substring. In this case, ReportServer recognizes the rest of the string as a link. The example link points to a PDF report which expects an "id" parameter. You can use `${_value}` for passing the value of the cell where the user right-clicked on to your URL. The `${_value}` is being concatenated in the query in order to avoid ReportServer handling it as a parameter.

Also note that in order for the example to work, the user has to right-click a column in the dynamic list preview containing the id to be passed as a parameter to the URL. The context menu will contain a "Link to..." entry which opens a new window containing the URL for the given entry.

If you need to pass the value of ***another*** (visible) column to your URL, you can use the analogous syntax, passing the column name or its alias. Thus, the string passed can be either the technical name of the column or its given alias. In the example below, `${id}` is being used to pass the value of the "id" column. The value passed is independent of the cell right-clicked by the user.

```
SELECT column_name, default_alias, description, default_width,
'linkto|http://URL/reportserver/httpauthexport?&user=myuser&id=20616&format=PDF&
    p_id=$' + '{id}' as link
FROM METADATA
```

Of course, you can also combine values passed. An example is given below.

```
SELECT column_name, default_alias, description, default_width,
'linkto|http://URL/reportserver/httpauthexport?&user=myuser&id=20616&format=PDF&
    p_id=$' + '{id}' + '&p_name=$' + '{name}' as link
FROM METADATA
```

If you need further customization, you can always implement the hook `net.datenwerke.rs.base.client.reportengines.table.hooks.TableReportPreviewCellEnhancerHook`. Refer to the Scripting Manual for more information on implementing hooks.

Linking Geolocation Data in Dynamic Lists to Google Maps

Similarly to linking dynamic lists to URLs as described in Section 7.2, you can use the metadata datasource to link geolocation data in a dynamic list to the corresponding Google Maps location.

For this purpose, the fifth column of the metadata query can again be used. An example is shown below.

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

If the "column_geolocation" value is the string "geolocation", the corresponding column will link to Google Maps when right-clicking the value. The entry "Display on Map" will appear, which jumps to the corresponding Google Maps location. E.g., if the value right-clicked on is "48.860294,2.338629", the linked location will be `http://maps.google.de/?q=48.860294,2.338629`, which corresponds to the Louvre Museum in Paris.

If you need further customization, you can always implement the hook `net.datenwerke.rs.base.client.reportengines.table.hooks.TableReportPreviewCellEnhancerHook`. Refer to the Scripting Manual for more information on implementing hooks.

Customizing Dynamic Lists via Report Properties

There are several report properties (see Section 7.12) 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.

Further, you can control if the parameters and their respective values should be exported along with the dynamic list export. Even the complete configuration may be exported together with the dynamic list export. This may be a huge help in order to being able to analyze thoroughly the values exported. For this, you can use the properties `output_parameters`, `output_complete_configuration`, and `output_include_hidden_parameters`. For more information, refer to Section 7.2.

For an overview over all report properties see Section 7.12.

Exporting Dynamic List Configuration

For exporting the parameter and filter configuration along with the dynamic list, you can set the following report properties:

- `output_parameters`
- `output_filters`
- `output_complete_configuration`
- `output_include_hidden_parameters`

Please note that these report properties are only available in Enterprise Edition.

If only `output_parameters` is set to `true`, then only the "real" parameters are being printed out, without separators, text, headers, etc. If `output_complete_configuration` is set, ***everything*** will be printed out. This includes user information, report information, filters, pre-filters, report metadata, global variables, separators, headers, parameters, etc. With other words, everything that you can possibly include in the query.

Both `output_parameters` and `output_complete_configuration` take `output_include_hidden_parameters` into account. `output_include_hidden_parameters` controls whether hidden parameters are being exported (defaults to `false`).

For exporting filters and pre-filters, you can use the `output_filters` variable. `output_complete_configuration` includes filters and pre-filters as well.

Note that only the Excel export prints this information without further settings. If you want to export this information into the HTML and/or PDF export as well, you have to also configure your `etc/dynamiclists/htmlexport.cf` and/or `etc/dynamiclists/pdfexport.cf` files.

For example, the following prints the filters and parameters:

```
<div class="parameters">
${parameterMapSimple}
```

```
</div>
<div class="filters">
${filterMapSimple}
</div>
```

More information on configuring these files on Section 7.2.

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="data:image/png;base64,SOMELOGOINBASE64=" alt="logo"></img>
```

```
</span>
<div class="reportdata">
<span class="name">${report.name}</span>
<span class="date">${now}</span>
</div>
<div class="clear"></div>
</div>

<!-- output parameters / filters before report data -->
<!-- Activate per report with output_parameters, output_filters, or ↙
     ↪ output_complete_configuration report property.
  The property has to be set to true in order to activate. -->
<!-- <div class="parameters">
You can export parameters with the parameterMapSimple variable.
</div>
<div class="filters">
You can export filters with the filterMapSimple variable.
</div> -->
<!-- end output parameters / filters before report data -->

]]>
      </pre>
      <post><![CDATA[

<!-- output parameters / filters after report data -->
<!-- Activate per report with output_parameters, output_filters, or ↙
     ↪ output_complete_configuration report property.
  The property has to be set to true in order to activate. -->
<!-- <div class="parameters">
You can export parameters with the parameterMapSimple variable.
</div>
<div class="filters">
You can export filters with the filterMapSimple variable.
</div> -->
<!-- end output parameters / filters after report data -->

</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
parameterMapSimple	A user-friendly text representation of the parameter map
parameterSet	The parameterSet object

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

7.3 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 "ClassicModelCars". 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 datasource (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 **datasource 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 (datasource parameter). datasource parameters draw their data from a datasource. Here we enter the demo datasource 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 enter 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

Datasource 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 9.

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 subheading

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 datasources.

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 right-click. Then select "copy". To add the copied parameters, first right-click on the parameters list (if there is no parameter available, right-click on the empty grid), then select "paste".

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 datasource.
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. In this case, it is not necessary 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 changing the specific properties 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 returned. 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. If you want a NULL to be returned when the parameter is left empty instead of an empty string, select the "return NULL if empty" option.

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 be shown.

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 Datasource Parameter

The datasource 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 datasource, giving it its name. ReportServer expects a table with one or two columns to be returned from the datasource. 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 datasource 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 without a detour via the database. Here, the solution is a CSV datasource with an argument connector (refer to "datasources"). This enables you to enter the values for the parameter directly at the parameter. When generating the datasource please observe to deactivate the database cache. Please consider as well that the first line of the CSV data hold the description 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:

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/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.

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 create a groovy script that returns HTML. A script could hence look as follows

```
return '''
<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.

```
return '''
<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>
'''
```

As a further example, you can use a script parameter for installing a color picker which can be then used in the SQL query as any other parameter. For this purpose, download the latest jscolor javascript file from here: http://jscolor.com/. At the time of writing, the latest version was 2.0.5. Copy the downloaded jscolor.js file into this location: Fileserver Root/lib/jscolor/jscolor.js. Check in the "Properties" tab the URL of this file in your installation. It should be something similar to: http://localhost:8080/reportserver/reportserver/fileServerAccess?id=2189950.

Now you can create the script which will be the basis of your script parameter. We name this script PalettePicker.groovy:

```
return '''
<html>
  <head>
    <script type="text/javascript" src="http://localhost:8080/reportserver/
       reportserver/fileServerAccess?id=2189950"></script>
    <script type="text/javascript">
      var callback;

      function initParameter(param, cb){
        callback = cb;
        //alert("The current value is: " + param.defaultValue);
        document.getElementById("mycolor").value = param.defaultValue;
        document.getElementById("mycolor").style.backgroundColor = '#' + param.
             defaultValue;
        setValue(param.defaultValue);
      }

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

    </script>
  </head>
  <body>
    <input id="mycolor" class="jscolor" onchange="setValue(this.value);">
  </body>
</html>
```

```
'''
```

As you can see in the script, you install a color picker and use the default value for setting the initial value and background color of the input field. The URL mentioned previously must be adapted to your configuration:

```
<script type="text/javascript" src="http://localhost:8080/reportserver/
    reportserver/fileServerAccess?id=2189950"></script>
```

Finally, you can create the script parameter in your report and set the script field to point to your PalettePicker.groovy script in your parameter's "Specific Properties" dialog. Set your parameter's default value to e.g.: "FF0000" (without apostrophes). You can now use the color picker's value in your query as:

```
SELECT * FROM myTable where myField = ${mycolor}
```

Note that your parameter's name must be set to "mycolor" for using this query.

This approach works for virtually all javascript libraries. Script arguments are a powerful tool for extending ReportServer parameters to user-defined functionality.

Many thanks to Karolina Boboli for sending us this and allowing us to publish the script.

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 datasource parameter supports the configuration by means of depending parameters. To do this, on the general configuration page of the datasource parameter set the parameters on which the datasource parameter will depend. In the following you can use them in the datasource 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 datasource. In the following we describe the parameter use in the dynamic list. How to use the parameters in Jasper BIRT or script reports is described in each section dealing with the respective report type.

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

Parameters in Datasources: 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:

```sql
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

```sql
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 https://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:

```sql
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 following attack: The user selects string "`1; DROP TABLE MY_TABLE;`" which would transfer to the following statement (respectively to the following two statements)

```sql
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 (https://juel.sourceforge.net/, https://www.jcp.org/en/jsr/detail?id=245 and https://docs.oracle.com/cd/E19226-01/820-7627/gjddd/) 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 subclass 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 subclass 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 Datasources: Mondrian Schema

For Saiku Reports you can use parameters in the SQL tag of the Mondrian schema query as described for relational databases.

Parameters in Datasources: CSV List

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

Parameters in Datasources: Script Datasources

For script datasources 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`

The following allows you to access the complete locale which is used, e.g. in Jasper reports, for localization.

- `_RS_LOCALE`

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 8 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 7.13 Report Metadata.

7.4 JasperReports

ReportServer supports the creation of graphical reports by using the OpenSource library JasperReports Library (`https://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 reports, 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 report designer JasperSoft Studio (`https://community.jaspersoft.com/download-jaspersoft/download-jaspersoft/`).

A JasperReport always consists of exactly one master report and one or several subreports. There is an XML file for the master report and for each subreport 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 (`https://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 subreports. You can create subreports either by the **Add subreport** button, or by dragging and dropping it to the respective window.

During report execution, subreports 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 subreports have the same name. Conversely, however, please note that ReportServer never loads subreports from the local server file system. Accordingly, all subreports always have to be registered together with the master report. Furthermore, it is not possible to share subreports between several reports.

Finally, you have to assign a datasource 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 https://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 toolbar. 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.

Note that when you include images, if you plan to export these to HTML, you need to set `isLazy` to `true` as shown in the following example:

```
<image scaleImage="Clip" vAlign="Bottom" isLazy="true">
   <reportElement x="0" y="0" width="555" height="250"/>
   <imageExpression class="java.lang.String">"/path/to/your/image.gif"</
      imageExpression>
</image>
```

More information can be found here: https://jasperreports.sourceforge.net/api/net/sf/jasperreports/engine/JRImage.html.

Unformatted Excel Output: Plain Excel

Jasper Reports attempts to present pixel-perfect reports across all output formats. As a result, when exporting to Excel, the reports are usually well-formatted visually. However, numbers often appear as text rather than numeric values, which complicates further data processing.

Please note that the Excel export of Jasper reports essentially "draws" the report into Excel. This is handled by the Jasper engine and can vary in quality.

To address this issue, ReportServer provides an option “Plain Excel” to the Jasper report, which allows you to add an additional datasource to a Jasper report. When this function is activated, you will be offered an additional output format: “Excel (plain)”. This option presents the result of the query execution as an Excel file, allowing for a cleaner, more straightforward export that does not attempt to render the pixel-perfect layout of the report. Please consider using this “Plain Excel” option for better data processing.

Exporting Jasper reports containing images to HTML

When exporting a Jasper report that includes images to HTML, you may encounter an issue where the images are not initially visible in the export. This behavior is explained in detail here: https://jasperreports.sourceforge.net/api/net/sf/jasperreports/engine/JRImage.html.

To ensure images appear correctly in your HTML export of a Jasper report, consider setting the `isLazy` property to `true` as shown in the following example:

```
<image scaleImage="Clip" vAlign="Bottom" isLazy="true">
   <reportElement x="0" y="0" width="555" height="250"/>
   <imageExpression class="java.lang.String">"/path/to/your/image.gif"</
      imageExpression>
</image>
```

7.5 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 `https://eclipse-birt.github.io/birt-website/`.

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 datasource to be used. If a datasource is provided in ReportServer, it will be used instead of a datasource possibly defined in the report itself. But if you wish to use the datasource given in the report, you may not select a datasource 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 datasources. For further information see Chapter 4 (Datasources).

7.6 SAP Crystal Reports

ReportServer comes with a basic support for SAP Crystal Reports (`https://www.sap.com/products/data-cloud/crystal-reports.html`). 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 in the SAP website. You may have to register here: https://www.sap.com/cmp/td/sap-crystal-reports-eclipse-trial.html and then click on the "Runtime libraries" link. Download the there located archive (around 67 MB). At the time of writing the version available is CR4ERL27_0-80004572. 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`
- `commons-configuration-1.2.jar`
- `commons-lang-2.1.jar`
- `CrystalCommon2.jar`
- `CrystalReportsRuntime.jar`
- `cvom.jar`
- `DatabaseConnectors.jar`
- `icu4j.jar`
- `jai_imageio.jar`
- `JDBInterface.jar`
- `jrcerom.jar`
- `keycodeDecoder.jar`
- `logging.jar`
- `pfjgraphics.jar`
- `QueryBuilder.jar`
- `webreporting-jsf.jar`
- `webreporting.jar`
- `XMLConnector.jar`
- `xpp3.jar`

That is, all jars except some commons-*.jar and the 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.

Since ReportServer contains log4j libraries as of 4.3.0, you need the log4j adapter (log4j-1.2-api), available here: https://logging.apache.org/log4j/2.x/log4j-1.2-api/index.html. More information can be found here: https://logging.apache.org/log4j/log4j-2.14.1/manual/compatibility.html

Regarding this critical security issue: CVE-2021-44228. Crystal, on its current version CR4ERL27_0-80004572, is affected. For avoiding using the affected Crystal log4j libraries, use the log4j adapter (log4j-1.2-api) as described above. Please refer to this for more information: https://forum.reportserver.net/viewtopic.php?id=2926

Install Microsoft's TrueType core fonts on a Linux-Based ReportServer Installation

If your ReportServer is installed in a Linux environment, you have to install ***Microsoft's TrueType core fonts*** in order to run Crystal reports: https://corefonts.sourceforge.net/.

If your Linux distribution is based on Debian, you can run the following commands for installing the MS TrueType core fonts:

```
# search for the ttf-mscorefonts-installer package
apt search ttf-mscorefonts-installer
# if package not found:
echo "deb http://deb.debian.org/debian buster contrib" >> /etc/apt/sources.list
# install the package
apt-get update && apt-get install ttf-mscorefonts-installer
# create a "fonts" syslink in your java distribution pointing to the fonts
ln -s /usr/share/fonts/truetype/msttcorefonts/ /usr/lib/jvm/java-11-openjdk-amd64/
    lib/fonts
# restart tomcat
systemctl restart tomcat9
```

Here is an example installation for an Ubuntu distribution:

```
# search for the ttf-mscorefonts-installer package
apt search ttf-mscorefonts-installer
# if package not found:
echo "deb http://de.archive.ubuntu.com/ubuntu jammy main restricted" >> /etc/apt/
    sources.list
# install the package
apt-get update && apt-get install ttf-mscorefonts-installer
# create a "fonts" syslink in your java distribution pointing to the fonts
ln -s /usr/share/fonts/truetype/msttcorefonts/ /usr/lib/jvm/java-11-openjdk-amd64/
    lib/fonts
# restart tomcat
systemctl restart tomcat9
```

In case you use our docker image (https://hub.docker.com/repository/docker/infofabrik/reportserverenterprise/general) please run the following commands:

```
# search for the ttf-mscorefonts-installer package
apt search ttf-mscorefonts-installer
# if package not found:
```

```
echo "deb http://de.archive.ubuntu.com/ubuntu jammy main restricted" >> /etc/apt/ ↙
     ↳ sources.list
# install the package
apt-get update && apt-get install ttf-mscorefonts-installer
# create a "fonts" syslink in your java distribution pointing to the fonts
ln -s /usr/share/fonts/truetype/msttcorefonts/ /usr/lib/jvm/java-11-openjdk-amd64/ ↙
     ↳ lib/fonts
# restart reportserver container
```

The above should be run by a user with sudo/root permissions.

Note that the above applies for manual installations. In case you use our docker image (https://hub.docker.com/repository/docker/infofabrik/reportserverenterprise/general) the installation directory might be different. In this case, you'll need to create the symlink using a similar command as the following example:

```
# search for the ttf-mscorefonts-installer package
apt search ttf-mscorefonts-installer
# if package not found:
echo "deb http://de.archive.ubuntu.com/ubuntu jammy main restricted" >> /etc/apt/ ↙
     ↳ sources.list
# install the package
apt-get update && apt-get install ttf-mscorefonts-installer
# create a "fonts" syslink in your java distribution pointing to the fonts
ln -s /usr/share/fonts/truetype/msttcorefonts/ /usr/lib/jvm/java-11-openjdk-amd64/ ↙
     ↳ lib/fonts
# restart tomcat
systemctl restart tomcat9
```

```
ln -s /usr/share/fonts/truetype/msttcorefonts /opt/reportserverenterprise- ↙
     ↳ 4.0.0.6055-0/java/lib/fonts
```

Of course, if a different Java installation directory was set, you need to adapt the command to your specific installation.

Depending on your Linux distribution, you may also have to check the following directory:

```
/usr/lib/jvm/default-jvm/jre/lib/fonts
```

When you have successfully installed the ttf-mscorefonts package, you will be able to run Crystal reports on a Linux-based environment.

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.

7.7 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 (https://github.com/OSBI/saiku) that we use to display such OLAP reports. For an introduction to OLAP and Mondrian we refer to the Mondrian documentation available online at https://mondrian.pentaho.com/documentation/.

Saiku reports are easily configurable once you have your Mondrian datasource (see Section 4.25). 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.

7.8 JXLS Reports

JXLS (https://jxls.sourceforge.net/) 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. 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.

JXLS2

In JXLS2 (https://jxls.sourceforge.net/), you define the JXLS2 commands via Excel comments. The JXLS2 engine parses these comments and transforms the template accordingly. We will first explain some JXLS2 concepts and then we will show some examples to make it work together with ReportServer. For a complete JXLS2 documentation refer to https://jxls.sourceforge.net/.

You can find JXLS examples in ReportServer here: https://github.com/infofabrik/reportserver-samples/tree/main/src/net/datenwerke/rs/samples/templates/jxls.

> Note that you can export dynamic lists into JXLS templates as of ReportServer 3.4.0. This functionality may help you with manual JXLS template creation, since manual creation may be cumbersome in some cases. Many thanks to Karolina Boboli for sending us this script and allowing us to use it.

XLS Area

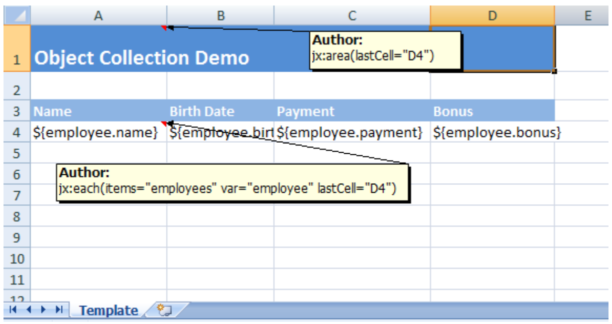
A XLS area represents a rectangular area in an Excel file which needs to be transformed. This basically defines the boundaries of your JXLS2 template in your Excel file. Each XLS Area may have a list of transformation commands associated with it and a set of nested child areas.

Constructing a XLS Area

You can use a special markup in your Excel template to construct a XLS area. The markup should be placed into an Excel comment in the first cell of the area. Its syntax is:

```
jx:area(lastCell = "<AREA_LAST_CELL>")
```

where <AREA_LAST_CELL> is the last cell of the defined area.



This markup defines a top-level area starting in the cell containing the markup comment and ending in the <AREA_LAST_CELL> cell. Refer to the following example (https://jxls.sourceforge.net/samples/object_collection.html):

The XLS area is defined in a comment in the A1 cell as

```
jx:area(lastCell="D4")
```

In the example we have an area covering the A1:D4 cell range.

Command Markup

A command represents a transformation action on a single or on multiple XlsAreas. It should be defined inside the boundaries of an XlsArea:

```
jx:<command_name>(attr1='val1' attr2='val2' ... attrN='valN' lastCell=<last_cell> areas=["<command_area1>", "<command_area2>", ... "<command_areaN>"])
```

Example commands include the following:

- each
- if
- image

attr1, attr2,..., attrN are the command specific attributes.

<last_cell> defines the bottom-right cell of the command body area, analogously to the XLS area.

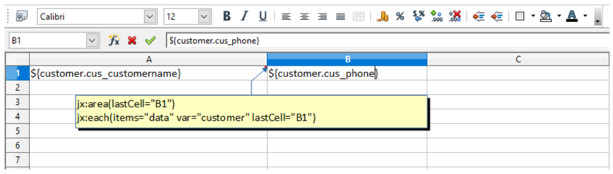
<command_area1>, <command_area2>, ... <command_areaN> - define XLS areas to be passed to the command as parameter.

Note that in a single cell comment you can define multiple commands. For example, in a single cell comment, you can have the following:

```
jx:each(items="department.staff", var="employee", lastCell="F8")
jx:if(condition="employee.payment <= 2000", lastCell="F8", areas=["A8:F8","A13:F13"])
```

Consider the following example.

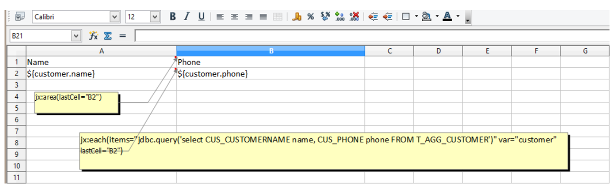
```
jx:area(lastCell="B1")
jx:each(items="data" var="customer" lastCell="B1")

${customer.cus_customername} | ${customer.cus_phone}
```



The cell A1 contains an Excel comment with the text `jx:area(lastCell="B1")`. It defines the boundaries of our template to be A1:B1. It also contains a Jxls Each-Command with the following text: `jx:each(items="data" var="customer" lastCell="B1")`. The Each-Command will iterate the collection of objects in the "data" collection and print the corresponding information. The body area of the Each-Command is A1:B1 (defined by the `lastCell` attribute), which means that the cells will be cloned and processed with each new `Customer` object in the context.
To access an individual attribute you can use `customer.colname`, where "customer" is the variable bound to the current data row and "colname" is the name of the attribute.

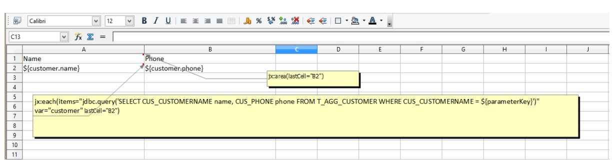
When used as a template for the dynamic list (refer to the chapter on the Dynamic List in the User Guide), the "data" variable contains the data selected by the dynamic list.

Note that you have to enter the fields in lower case so that these are correctly mapped by the JXLS engine.



When used as a first-class report type you need to select the data from a sql datasource. For this purpose, you can use the object `jdbc.query` inside the items attribute which provides access to the underlying datasource. Here, we select two fields from the table `T_AGG_CUSTOMER`.

```
jx:area(lastCell="B2")
jx:each(items="jdbc.query('select CUS_CUSTOMERNAME name, CUS_PHONE phone FROM T_AGG_CUSTOMER')" var="customer" lastCell="B2")

${customer.name} | ${customer.phone}
```

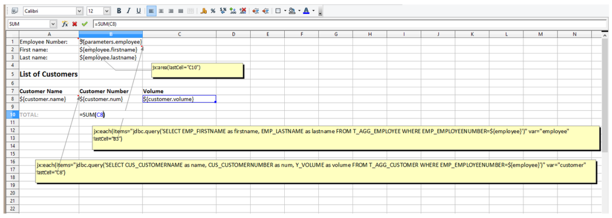


In order to use parameters within queries, you can use the standard syntax for parameters in the query described in Section 7.3. This is best understood with an example: Here, we assume that you previously created a parameter with key "parameterKey". You can also access parameters via the `parameters` object. Again, assuming that there is a parameter with key 'parameterKey", you could access the parameters value via

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

```
jx:area(lastCell="B2")
jx:each(items="jdbc.query('SELECT CUS_CUSTOMERNAME name, CUS_PHONE phone FROM T_AGG_CUSTOMER WHERE CUS_CUSTOMERNAME = ${parameterKey}')" var="customer" lastCell="B2")

Name | Phone
```

```
${customer.name} | ${customer.phone}
```



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.

```
jx:area(lastCell="C10")
jx:each(items="jdbc.query('SELECT EMP_FIRSTNAME as firstname, EMP_LASTNAME as lastname FROM
    T_AGG_EMPLOYEE WHERE EMP_EMPLOYEENUMBER=${employee}')" var="employee" lastCell="B3")
jx:each(items="jdbc.query('SELECT CUS_CUSTOMERNAME as name, CUS_CUSTOMERNUMBER as num, Y_VOLUME as
    volume FROM T_AGG_CUSTOMER WHERE EMP_EMPLOYEENUMBER=${employee}')" var="customer" lastCell="
    C8")

Employee Number: | ${parameters.employee}
First name: | ${employee.firstname}
Last name: | ${employee.lastname}

List of Customers

Customer Name | Customer Number | Volume
${customer.name}  | ${customer.num}  | ${customer.volume}
TOTAL: | =SUM(C8)
```

A full documentation of the possibilities offered by JXLS2 is out of scope of this documentation, and we refer the interested reader to the official JXLS documentation available on https://jxls.sourceforge.net. Also, the examples available here: https://github.com/infofabrik/reportserver-samples/tree/main/src/net/datenwerke/rs/samples/templates/jxls can be used directly in ReportServer.

7.9 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 18 and 21. 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.*
StringWriter 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.*
StringWriter 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 can be entered separated by commas, e.g. “HTML,PDF,DOCX”. 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.*
StringWriter 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 use the format in lower case letters inside the script).

The script report below supports PDF, HTML and DOCX outputs.

```
import groovy.xml.*
StringWriter 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())
if("html".equals(outputFormat.trim()))
  return renderer.get("html").render(writer.toString())
if("docx".equals(outputFormat.trim()))
  return renderer.get("docx").render(writer.toString())
```

```
writer.toString()
```

Datasources

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

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

StringWriter 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())
if("html".equals(outputFormat.trim()))
  return renderer.get("html").render(writer.toString())
if("docx".equals(outputFormat.trim()))
  return renderer.get("docx").render(writer.toString())
writer.toString()
```

7.10 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[1] and a ReportServer script. ReportServer scripts are covered in greater detail in Chapter 21 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.

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 provides 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.*

DbGridEditorDefinition definition = GLOBALS.getInstance(DbGridEditorDefinition)

DbGridEditorDefinitionAdapter adapter = definition.adapter
adapter.tableName = '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.

If you need to filter the data on your table, you can use set the `whereClause` property as shown in the following example:

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

DbGridEditorDefinition definition = GLOBALS.getInstance(DbGridEditorDefinition)

DbGridEditorDefinitionAdapter adapter = definition.adapter
adapter.tableName = 'T_AGG_CUSTOMER'
adapter.whereClause = 'CUS_CUSTOMERNUMBER > 100'

return definition
```

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

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

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 is unnecessarily complex, as the WHERE clause includes columns that are not part of the primary key. You should therefore always specify the table's primary key in ReportServer. Additionally, it may be desirable to display only a subset of the table's columns rather than the entire dataset as shown below.

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

DbGridEditorDefinition definition = GLOBALS.getInstance(DbGridEditorDefinition)

DbGridEditorDefinitionAdapter adapter = definition.adapter
adapter.tableName = 'T_AGG_CUSTOMER'
adapter.primaryKey = '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()`.

Auto-Increment Primary Keys

If you are updating a table containing an auto-increment primary key, you should add it to the list of columns and set its column configuration to be non-editable (setEditable(false)). You can further hide it if desired (setHidden(true)). As an example, here you can see a table "TABLE_NAME" having an "id" auto-increment primary key and a text column "your_column". The example uses the fluid API (see Section A Fluid API for more details).

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

DbGridEditorDefinitionAdapter adapter = gridHelper.initDbGridEditor()

adapter.configure(report, 'TABLE_NAME')
    .setPk('id')
    .columns()
        .add('id')
     .setEditable(false)
     .setHidden(true)
        .add('your_column')
    .done()

return adapter
```

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.*

DbGridEditorDefinition definition = GLOBALS.getInstance(DbGridEditorDefinition)

DbGridEditorDefinitionAdapter adapter = definition.adapter
adapter.tableName = 'T_AGG_CUSTOMER'
adapter.addColumns('CUS_CUSTOMERNUMBER', 'CUS_CUSTOMERNAME',  'CUS_CREDITLIMIT')

return definition
```

Now if a user updates the table ReportServer it may generate 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

In addition to the ***standard API*** for configuring the grid editor, a more compact and fluent API is available. This API simplifies the process of obtaining an adapter object. Within the scope of the script, an object named `gridHelper` is provided, which offers the method `initDbGridEditor`. To initialize the fluent API, the `configure` method must then be invoked on the returned adapter object. The example shown above can therefore be rewritten using the fluent API as follows:

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

DbGridEditorDefinitionAdapter 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()`	Enables sorting when set to `true`. (Default: `true`)
`setFilterable()`	Enables filtering when set to `true`. (Default: `true`)
`setCanAddRecords()`	Allows records to be added when set to `true`. (Default: `true`)
`setCanDuplicateRecords()`	Allows records to be duplicated when set to `true`. (Default: `true`)
`setCanRemoveRecords()`	Allows records to be removed when set to `true`. (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,

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

DbGridEditorDefinitionAdapter 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.*

DbGridEditorDefinition definition = GLOBALS.getInstance(DbGridEditorDefinition)

DbGridEditorDefinitionAdapter adapter = definition.adapter

adapter.tableName = 'T_AGG_CUSTOMER'
adapter.primaryKey = 'CUS_CUSTOMERNUMBER'
adapter.addColumns('CUS_CUSTOMERNUMBER', 'CUS_CUSTOMERNAME',  'CUS_CREDITLIMIT')

GridEditorColumnConfig nameConfig = new GridEditorColumnConfig()
nameConfig.displayName = 'NAME'
adapter.setColumnConfig('CUS_CUSTOMERNAME', nameConfig)

return definition
```

Or more compactly via the fluid API

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

```
DbGridEditorDefinitionAdapter 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','displayName')`.

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()` and `setOrderDesc()`.
`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` methods 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 given 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 https://docs.oracle.com/en/java/javase/21/docs/api/java.base/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)
addMaxValidator(max, message
```

Field Editors

By default, the Grid Editor creates form fields that correspond to the data type of each column. Text columns are rendered as text fields, date columns as date pickers, and boolean columns as checkboxes. For certain columns, this default behavior can be overridden by specifying a custom editor. To assign a custom editor to a column, use the `setEditor` method on a `GridEditorColumnConfig` instance. All available custom editors are located in the 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`.

```
GridEditorColumnConfig textbConf = new GridEditorColumnConfig()
textbConf.editor = new TextBooleanEditor()
adapter.setColumnConfig('textbool', textbConf)

GridEditorColumnConfig intbConf = new GridEditorColumnConfig()
intbConf.editor = new IntBooleanEditor()
adapter.setColumnConfig('intbool', intbConf)
```

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

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

DbGridEditorDefinitionAdapter 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 `https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/text/`

SimpleDateFormat.html for an introduction). The following is an example for how to use the TextDateEditor.

```
GridEditorColumnConfig dateConf = new GridEditorColumnConfig()
dateConf.editor = 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.

```
GridEditorColumnConfig ddTextConf = new GridEditorColumnConfig()
TextSelectionListEditor textddEditor = new TextSelectionListEditor()
textddEditor.setValues(['a','b','c','d'])
ddTextConf.editor = 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:

```
GridEditorColumnConfig ddTextConf = new GridEditorColumnConfig()
TextSelectionListEditor  textddEditor = new TextSelectionListEditor()
textddEditor.addValue('a')
textddEditor.addValue('b')
textddEditor.addValue('c')
textddEditor.addValue('d')
ddTextConf.editor = 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:

```
GridEditorColumnConfig ddTextConf = new GridEditorColumnConfig()
TextSelectionListEditor textddEditor = new TextSelectionListEditor()
textddEditor.addEntry('a','b')
textddEditor.addEntry('c','d')
textddEditor.addEntry('e','f')
ddTextConf.editor = 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:

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

Following an example with custom labels.

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

Fluid API for Selection Lists

Using the fluent API, selection editors can also be created in a compact and expressive manner. Calling `withSelectionEditor()` initiates the editor configuration, which is finalized by invoking `done()`. This approach allows you to configure a selection editor, for example, as follows:

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

DbGridEditorDefinitionAdapter adapter = gridHelper.initDbGridEditor()

adapter.configure(report, 'T_AGG_CUSTOMER')
    .setPk('CUS_CUSTOMERNUMBER')
    .columns()
   .add('CUS_CUSTOMERNUMBER')
        .add('CUS_CUSTOMERNAME')
            .withSelectionEditor()
               .addValue('FunGiftIdeas.com')
               .addValue('Marseille Mini Autos')
            .done()
    .done()

return adapter
```

Within the editor's edit mode, the same methods `addValue()` and `addEntry()` are available. In addition, the fluent API provides the `from()` method, which allows multiple values to be added at once. This method accepts either a list of values (corresponding to `addValue()`) or a map (corresponding to `addEntry()`).

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

DbGridEditorDefinitionAdapter adapter = gridHelper.initDbGridEditor()

adapter.configure(report, 'T_AGG_CUSTOMER')
    .setPk('CUS_CUSTOMERNUMBER')
    .columns()
   .add('CUS_CUSTOMERNUMBER')
        .add('CUS_CUSTOMERNAME')
            .withSelectionEditor()
         .from(['FunGiftIdeas.com', 'Marseille Mini Autos'])
            .done()
    .done()

return adapter
```

Note that when using the fluent API, the type of the selection editor is not specified explicitly. Instead, it is inferred from the provided values. Consequently, if you intend to add values of type "long", an explicit type cast is required.

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.

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

DbGridEditorDefinitionAdapter adapter = gridHelper.initDbGridEditor()

adapter.configure(report, 'T_AGG_CUSTOMER')
    .setPk('CUS_CUSTOMERNUMBER')
    .columns()
    .add('CUS_CUSTOMERNUMBER')
        .add('CUS_CUSTOMERNAME')
            .withSelectionEditor()
               .fromSql('SELECT CUS_CUSTOMERNAME FROM T_AGG_CUSTOMER')
            .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

Foreign Key Relationships

In the following section, we explain how foreign key relationships can be used to simplify data editing. Consider a database table `Products` with 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).

In this example, two foreign key relationships exist: one linking a product to a product category and another linking it to a supplier. This models a ***many-to-one*** relationship. Each product belongs to exactly one category and one supplier, while a category or supplier may be associated with multiple products.

Assume the corresponding category and supplier tables have the following structure:

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

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

A MySQL example of the DDLs is shown below

```
CREATE TABLE Categories (
  categoryNumber INT AUTO_INCREMENT PRIMARY KEY,
  categoryName VARCHAR(255) NOT NULL,
  categoryDescription VARCHAR(1000)
) ENGINE=InnoDB;

CREATE TABLE Suppliers (
```

```
  supplierNumber INT AUTO_INCREMENT PRIMARY KEY,
  supplierFirstName VARCHAR(255) NOT NULL,
  supplierLastName VARCHAR(255) NOT NULL
) ENGINE=InnoDB;

CREATE TABLE Products (
  productNumber INT AUTO_INCREMENT PRIMARY KEY,
  productName VARCHAR(255) NOT NULL,
  productCategory INT NOT NULL,
  productSupplier INT NOT NULL,
  FOREIGN KEY (productCategory) REFERENCES Categories(categoryNumber),
  FOREIGN KEY (productSupplier) REFERENCES Suppliers(supplierNumber)
) ENGINE=InnoDB;
```

Now consider a basic grid editor configuration used to manage the `Products` table:

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

DbGridEditorDefinition definition = GLOBALS.getInstance(DbGridEditorDefinition)

DbGridEditorDefinitionAdapter adapter = definition.adapter
adapter.tableName = 'Products'
adapter.primaryKey = 'productNumber'
adapter.addColumns(
  'productNumber',
  'productName',
  'productCategory',
  'productSupplier'
  )

return definition
```

With this setup, changing a product's category requires knowing the numeric `categoryNumber`. In many cases, however, it is more convenient to select a category by its name. Since the category name is stored in a separate table, this is not directly possible with the basic configuration.

To address this, the grid editor allows you to define ***foreign key columns***. Foreign key columns enable you to display alternative values—such as descriptive names—instead of the raw foreign key values stored in the table. During insert or update operations, the displayed values are automatically translated back into the appropriate foreign key values.

Foreign key columns can be defined using the adapter's `addForeignKeyColumn()` method. This method accepts four or five parameters:

`column`	The column in the current table that holds the foreign key. In this example, this would be ***productCategory***.
`fkTableName`	The name of the referenced table. In this example, ***Categories***.
`fkColumn`	The primary key column of the referenced table. In this example, ***categoryNumber***.
`displayExpression`	A SQL expression that selects the value to be displayed from the foreign key table. This is typically a single column but may also be a more complex expression. In this example, ***categoryName***.
`displayName/config`	An optional parameter that accepts either a string or a `GridEditorColumnConfig` object. If a string is provided, it is used as the column's display name. If a configuration object is provided, it defines the column's behavior. If omitted, the column name defaults to the value of ***column***.

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

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

DbGridEditorDefinition definition = GLOBALS.getInstance(DbGridEditorDefinition)

DbGridEditorDefinitionAdapter adapter = definition.adapter
adapter.tableName = 'Products'
adapter.primaryKey = 'productNumber'
adapter.addColumns('productNumber', 'productName')
           .addForeignKeyColumn(
      'productCategory',
      'Categories',
      'categoryNumber',
      'categoryName',
      'Category'
      )
           .addColumns('productSupplier')

return definition
```

Using the same approach, you can also define a foreign key column for `productSupplier`.

In most cases, you will additionally configure an editor that allows users to select a category from a predefined list rather than entering values manually. To do so, you can pass a configuration object as the fifth parameter to `addForeignKeyColumn()`. An example configuration is shown below (note the additional imports):

```
import net.datenwerke.dbpool.DbPoolService
import net.datenwerke.rs.base.service.datasources.definitions.DatabaseDatasource
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 net.datenwerke.rs.terminal.service.terminal.TerminalSession
import net.datenwerke.rs.terminal.service.terminal.objresolver.ObjectResolverDeamon

import groovy.sql.Sql
import java.sql.Connection

// enter the path to your datasource here
```

```
String datasourcePath = '/datasources/RS_GRIDEDITOR_TEST'

DbGridEditorDefinition definition = GLOBALS.getInstance(DbGridEditorDefinition)
TerminalSession session = GLOBALS.getInstance(TerminalSession)
ObjectResolverDeamon objectResolver = session.objectResolver
DbPoolService dbPoolService = GLOBALS.getInstance(DbPoolService)

DatabaseDatasource datasource = (DatabaseDatasource) objectResolver.getObjects(datasourcePath)[0]

DbGridEditorDefinitionAdapter adapter = definition.adapter
adapter.tableName = 'Products'
adapter.primaryKey = 'productNumber'

// define config for categories
GridEditorColumnConfig categoryConfig = new GridEditorColumnConfig(displayName: 'Category')
TextSelectionListEditor categoryEditor = new TextSelectionListEditor()

dbPoolService.getConnection(datasource.connectionConfig).get().withCloseable { Connection connection ->
     new Sql(connection).eachRow('SELECT categoryName AS name FROM Categories ORDER BY 1' ){
        categoryEditor.addValue(it.name)
   }
}

categoryConfig.editor= categoryEditor

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

return definition
```

7.11 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. Valid output formats are: `EXCEL`, `CSV`, `PDF`, `XML`, `WORD`, `XLS`, `RTF`, `PNG`, `JSON`, `JSONC` (for compact JSON export), `SAIKU_CHART_HTML`, `TABLE_TEMPLATE`, and `RS_SIMPLE_BEAN`.

`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.

Particularities 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 (true/false)

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):

https://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 https://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:

https://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

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:

https://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 **User 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).

> Note that the apikey can be found as plain text in the URL, so make sure you use HTTPS. Treat the apikey like any password.

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 analogous 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`

7.12 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 the particular report. By setting this property, you can explicitly enable and disable export formats for the given report. If this property is not set, all export formats for the given report are available by default. Note that not all export formats are supported in all kind of reports. If you don't set `output_format_auth`, all available formats ***for the specific report*** will be shown. You can then use `output_format_auth` for allowing a ***subset*** from this set of formats. Valid output formats are: `EXCEL`, `CSV`, `PDF`, `XML`, `WORD`, `XLS`, `RTF`, `PNG`, `JSON`, `JSONC` (for compact JSON export), `SAIKU_CHART_HTML`, `TABLE_TEMPLATE`, `TABLE_EXCEL` and `RS_SIMPLE_BEAN`. `SAIKU_CHART_HTML` is needed for Pivot/Mondrian chart export, while the last two are necessary for dynamic list template export, e.g. JXLS.
`output_format_default`	The default export format of the particular report. The valid values are the same as in the `output_format_auth property`, with the difference that only one value may be set here.

The Dynamic List

export:veto_large_report	By default, when exporting a report to formats like Excel, ReportServer verifies whether the result set size exceeds certain limits. If it does, the export process is halted. However, if this property is configured with a value other than “auto”. this verification step is bypassed. It’s important to note that this approach could potentially cause the system to crash if the report size is exceptionally large.
output_parameters	Defines if the parameters and their values should be exported together with the report (defaults to false. Only available in Enterprise Edition). Note that the parameterMapSimple variable must be set in the pdfexport.cf or the htmlexport.cf configuration file for this to work in the PDF / HTML exports, respectively. In contrast, the Excel export does not need any additional configuration. Excel outputs the parameters into a “Configuration”-named sheet.
output_filters	Defines if the filters and pre-filters should be exported together with the report (defaults to false. Only available in Enterprise Edition). Note that the filterMapSimple variable must be set in the pdfexport.cf or the htmlexport.cf configuration file for this to work in the PDF / HTML exports, respectively. In contrast, the Excel export does not need any additional configuration. Excel outputs the filters into a “Configuration”-named sheet.
output_complete_configuration	Allows to export the complete dynamic list configuration variables and their respective values (defaults to false. Only available in Enterprise Edition). This includes the parameters, filters, prefilters, global variables, report metadata, etc. Everything you can include in the dynamic list query is listed here. This greatly helps the user to analyze the exported output. As parameters are included here, this report property overrides the output_parameters property. The same holds for output_filters. Analogously to output_parameters, the parameterMapSimple (or filterMapSimple for filters) is needed for the HTML / PDF export. The Excel export outputs this configuration into a “Complete configuration”-named sheet.
output_include_hidden_parameters	Controls if the parameter output should include hidden parameters (defaults to false. Only available in Enterprise Edition). Note that this property influences both output_parameters and output_include_hidden_parameters properties.
suppress_automatic_preview	Allows to disable the automatic preview of the report (defaults to false). Setting this option to “true” prevents the report from automatically executing upon opening from within the Administration panel or a TeamSpace. This capability is particularly beneficial in situations where reports generated by Birt, JXLS, Jasper, or Crystal have extended execution times, and users simply wish to export the report without initiating its direct execution. It’s important to note that when automatic preview is disabled, users can still manually trigger previews by explicitly clicking on "Preview" or the "Refresh preview" button. Furthermore, it’s noteworthy that dynamic lists cannot have their preview functionality disabled, as users are always required to explicitly click on "Preview".
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).

7.13 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.

7.14 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 JasperReport 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 8

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 9

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}
```

9.1 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.

9.2 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).

9.3 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 10

Import and Export

You can export various objects from a ReportServer instance to save or import them to another instance. The export options can be found in the Administration section under the respective object (e.g., a report). You can export individual objects or entire subtrees. This allows you to move the complete report inventory of one installation to another.

When exporting a ReportServer object, all the reference objects required to use the object will also be exported. For example, a report will be accompanied by the respective data source.

10.1 Exporting

To export an object or a subtree, open the respective section in the Administration module (e.g., Reports). Click on the object or the superordinate folder of the subt-ree to export. In the toolbar, click the **Export** button. When exporting a report, you can choose to include or exclude variants. Once the export is finalized, ReportServer will prompt you to either display or download the export. Please note that this process may take longer for more comprehensive subtrees. Direct download as a zipped archive is recommended for comprehensive exports.

Export Format

ReportServer exports objects in an XML dialect. This allows export files to be edited with common XML tools. Additionally, it makes it easy to adapt export files to later versions of ReportServer.

10.2 Importing

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

After uploading the exported data, the object types included in the export will appear in the left part of the window. Different import options are available depending on the section.

Once you have successfully completed the configuration, click on the **Complete import** in the toolbar to execute the import. Click on **Cancel import** to interrupt the current import process. To reset the configuration, select **Reset**. Note that after a successful import, you will be asked if you want to reset the import configuration. Therefore, you can comfortably import multiple objects from an export file.

Importing Reports

Select the target folder for the import in the **Options** tab and click on **Create random key if key exists:** to avoid double keys. From the **Objects** tab, select the reports to import. 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 datasources. They will only be imported if you set the relevant configuration options (refer to Importing Datasources). If you fail to set these options, the relevant fields will remain empty after the import.

Importing Datasources

Datasources are imported in the same way as reports. In the **Options** tab you define a target folder and from the **Objects** tab you select the objects to import. You have the option to specify a **Default datasource** which will be entered at all locations where you want to avoid to import a datasource 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 datasource is usually the one of the production system. Select the production datasource via the option **Default Datasource**; it will then be entered in all relevant reports and parameters while importing.

Importing Users and Files

Users and files are imported in the same way as reports. There are no additional options available here.

Merge Options

Under **Import Options Merge Options**, you can check for **Use Merge-Importer** (selected per-default) and the merge routine is applied. This involves taking one object and transferring its values to another object in the system, with the exception of the key and ID. No entity is deleted after merging. In general, only entities of the same type can be merged.

Chapter 11

Transports

11.1 General Introduction

ReportServer offers a comprehensive transport mechanism that securely moves objects such as reports, datasources, datasinks, and files between different system environments, such as development, testing, and production. This mechanism ensures that changes are carried out consistently and in a controlled manner, thereby guaranteeing the integrity and stability of the entire system landscape.

Overview of the Use Cases

The transport mechanism in ReportServer provides various use cases that allow users to efficiently manage their work and safely transfer changes across different environments:

Transfer of Reports and Datasources between Environments

Use Case Transporting reports, datasources, datasinks, and files from a development environment to a testing or production environment.

Advantage Ensures that changes can be tested in a controlled environment before being transferred to production.

Backing up Work Results during Development

Use Case Developers can create regular transports to secure the current state of their work.

Advantage Provides a way to secure work steps as snapshots that can be restored or transferred to another system if needed. This is especially useful when multiple developers are working in parallel or when experimental changes are made that need to be rolled back later.

Duplicating Transports for Further Processing

Use Case Creating copies of existing transports to make changes or use content in other contexts.

Advantage Flexibility in managing transports, as existing transports can be duplicated and reused as open transports.

Restoring States after Errors

Use Case Undoing unwanted changes by applying a new transport that restores the original states.

Advantage Allows restoring the original state after an error or misapplication by transporting and applying the corresponding changes again.

Reverting Changes

Use Case Indirectly reverting changes through manual adjustments or repeated transports, as direct rollbacks are not possible in ReportServer.

Advantage Provides a method to undo applied changes by either relying on backups or performing manual corrections.

Conclusion

The transport mechanism of ReportServer offers flexible and powerful tools to securely transport changes across different system environments, secure work steps, and correct errors when necessary. Through careful planning and application of transports, administrators and developers can ensure the stability and consistency of their systems while remaining flexible in responding to changes.

11.2 Configuration and Prerequisites

For the transport mechanism in ReportServer to be used, specific configurations and prerequisites must first be met.

Unique Keys for Objects

Unique Keys All transportable objects, such as reports, datasources, datasinks, and files, require a unique key (per tree) that is automatically generated when the object is created. These keys are necessary to uniquely identify the objects and avoid conflicts during transport.

Configuration Files

`main/transport.cf`

This file contains basic settings for importing and applying transports. It specifies, among other things, the remote server from which transports are imported, the target directory for the transports and the user responsible for applying the transports.

Common Errors in transport.cf

Correctness of the Remote Server Key The the remote server's key must be correctly specified. An incorrect key will prevent the transports from being retrieved.

Directory under `target` The directory specified in the target field must exist. If the directory is not present, it is created by ReportServer automatically while retrieving the transports.

Reloading the Configuration After any changes to the `transport.cf` or other configuration files, the new configuration must be loaded. This is done using the `config reload` terminal command, which ensures that the changes take effect.

`main/mappings.cf`

This file defines the mapping of datasources between the source system and the target system. There are two possible options that determine the priority of the mapping.

Mapping If explicit mapping is required between a datasource in the source system and one in the target system, it is defined in the `mappings.cf` file. This is necessary when the datasources in the systems have different keys.

Same-Key If the datasources in both systems have the same key, explicit mapping can be omitted. ReportServer automatically recognizes that the datasource is identical based on the key and uses it accordingly in the target system.

Priority of Options

- If a mapping is explicitly defined, it takes precedence.
- If no mapping exists and the datasources in both systems have the same key (Same-Key), this datasource will be automatically used without requiring a mapping.
- Note that you can change the order of precedence in the configuration file.

Remote Server Configuration

To exchange transports between different ReportServer instances, the remote server must be configured in the Remote-Servers section. This configuration includes:

Name A name to identify the remote server.

Key A unique key for authentication.

REST-URL The URL of the remote server, which can be found in the system console under General Information.

Username The username authorized to access the remote server.

API Key A special key for authenticating API requests between the servers. This API key must be stored as a user property in the user profile. It uses an apikey analogously as in executing reports via URL (see Section 7.11) for more information on this.

Version Consistency

Transports can only be exchanged and applied if ReportServer version and schema version of both systems are identical. This prerequisite is crucial to avoid compatibility issues and to ensure that the transported objects function correctly in the target system. If there is a version mismatch, data structures or functions might not align correctly, leading to errors during the import or application of the transports.

11.3 Usage

The transport process in ReportServer involves several steps that must be carefully followed to ensure a successful transfer of objects between systems.

Creating a Transport in the Source System

The process begins with the creation of a transport order in the source system (e.g., development). This transport order acts as a container that consolidates the objects and their configurations for transfer between systems.

Objects such as reports and datasources can then be added to the transport via the context menu **Add to Transport** in the respective area of ReportServer. It is important to note that the saved state of the object at the time of addition is what is included in the transport. If an object that is already in the transport is added again, the previous version of the object in the transport is overwritten. This ensures that the most current version of the object is always included in the transport.

Important: Subsequent changes to the object after it has been added to the transport are not automatically included in the transport. If modifications are made to an object that has already been added, the object must be added to the transport again to include the updated version.

As long as a transport is still open, the objects within it can be managed. The management of the object list is done via the “Objects” tab of the transport. Objects can be added or removed before the transport is finalized.

After all relevant objects have been added to the transport and the object list has been managed, the transport order is closed on the source system. This effectively releases the transport. Additionally, only closed transports can be transferred. A transport must therefore be released and thereby closed before it can be imported into another system.

Important: A released transport can no longer be re-opened or modified. All objects and their states contained within are now final.

Retrieving and Importing the Transport in the Target System

The next step is to retrieve the released transports from the source system in the target system. This process is initiated in the Transport Management section of ReportServer by pressing the “Fetch Transports from Remote” button. When retrieving, all closed transports from the remote system are imported into the target system and stored there. However, the content of the transports is not unpacked or applied at this time.

Applying the Transport in the Target System

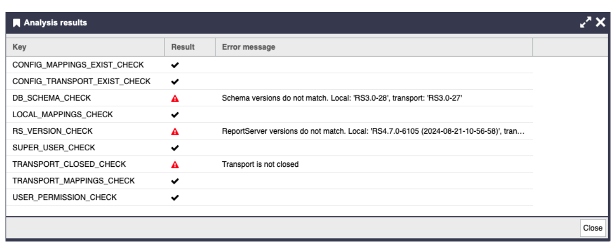
Before the transport is applied, it should be thoroughly verified. This ensures that all settings and prerequisites are met, particularly that the mappings for the datasources are correctly configured. This guarantees that the objects function properly in the target system.

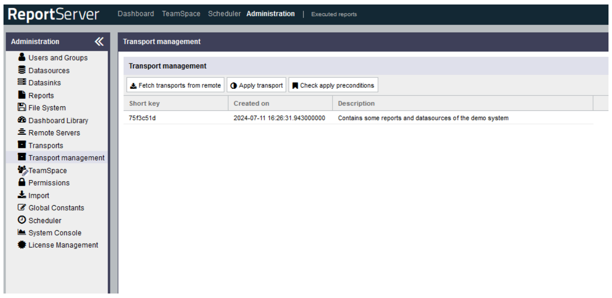
The following preconditions must be met before a transport can be applied:

RS_VERSION_CHECK	The ReportServer version of the transport must exactly match the current ReportServer version in the target system.
DB_SCHEMA_CHECK	The database schema version of the transport must match the current schema version in the target system. Typically, this precondition is satisfied when the RS_VERSION_CHECK precondition is met.
TRANSPORT_CLOSED_CHECK	Only closed (released) transports are eligible for application. This precondition verifies that the transport is indeed closed.
SUPER_USER_CHECK	Only a super user may apply a transport, as transports can alter many parts of the system, which requires superuser privileges. The super user must be specified in the `main/transport.cf` configuration file and must correspond to an existing super user in the system. This precondition checks that these criteria are met.
USER_PERMISSION_CHECK	The current user, i.e., the one applying the transport, must have both Read and Execute permissions on the transport. This precondition ensures that the user has the necessary permissions.
CONFIG_TRANSPORT_EXIST_CHECK	The `main/transport.cf` configuration file must exist in the target system. This precondition checks for the presence of this file.
CONFIG_MAPPINGS_EXIST_CHECK	The `main/mappings.cf` configuration file must be present. This precondition verifies its existence.
LOCAL_MAPPINGS_CHECK	The local keys defined in the mappings within the `main/mappings.cf` configuration file must correspond to an existing datasource in the current system. Local keys refer to datasource keys from the local system, and thus they must point to an existing data source.
TRANSPORT_MAPPINGS_CHECK	All remote keys found in the transport being applied must match a local key that points to an existing local data source. This matching can be done in two ways: either through explicit mappings defined in the mappings file, or by identifying a data source in the local system that shares the same key. The LOCAL_MAPPINGS_CHECK precondition ensures that the mappings file correctly links remote keys to local data sources, while this precondition verifies that any keys not explicitly mapped in the mappings file correspond to an existing data source in the local system. Thus, any keys not covered by an explicit mapping in the mappings file must still point directly to an existing data source within the local system.

A transport can only be applied if all of the above preconditions are met. These preconditions can be checked by clicking the **Check apply preconditions** button.

The following screenshot illustrates the outcome when some preconditions are met, while others are not, indicating that the transport cannot be applied.





The following screenshot depicts a transport that has been imported into the system. The transport can be applied if all preconditions meet as explained above.

When applying a transport in the target system, the contained objects are integrated into the target system using a merge mechanism. The merge process identifies whether the objects are new or modified, based on the unique key of the object.

Case Example for a New Report

What happens If the transport contains a report whose key does not exist in the target system, the report is imported into the target system as a new report. The report is fully transferred and then available in the target system.

Result The new report is inserted into the target system and can be used and further edited like any other report.

Case Example for a Modified Report

What happens If the transport contains a report whose key already exists in the target system, the merge process recognizes it as a modified report. In this case, the existing report in the target system is completely replaced with the changes from the transport.

Result The report in the target system will look exactly like it does in the source system (test system). The original version of the report in the target system will be overwritten. This means that all changes to queries, names, parameters, report properties, and report metadata will be fully transferred to the target system.

Note on Variants Variants of the report that exist in the target system will generally remain intact. However, it is important to note that changes to parameters or the replacement of parameters can affect existing variants. These may become unusable if the new parameters are not compatible with the existing variants or if the structure of the report has been significantly changed.

11.4 Monitoring and Logging

The entire transport process, from creation to retrieval and application of the transports, is thoroughly monitored and logged. These logs provide complete traceability and assist in troubleshooting if issues arise during the import or application of the transports.

After a transport has been successfully imported and applied, the associated logs can be directly accessed within the respective transport. These logs contain detailed information about the transport's progress, including all actions taken, any errors, and warnings. This allows administrators to precisely track what changes were made and whether the transport was executed correctly.

This logging ensures a high level of transparency and traceability for all transport operations performed within ReportServer.

11.5 Frequently Asked Questions FAQ

What are transports and what purpose do they serve?

Transports in ReportServer are packages that contain reports, datasources, datasinks, and files, enabling their secure transfer between different system environments (e.g., development, test, production). They are used to transfer modifications and configurations to other systems in a consistent and controlled way, ensuring a uniform and stable system landscape.

How is a transport created?

A transport is created in the source system (e.g., development) by creating a transport job. This job serves as a container that aggregates the objects and their configurations for transfer between systems. This is typically done via the context menu "Add to Transport" for the respective objects.

What is the encrypted name of a transport, and why is it used?

The encrypted name of a transport is an automatically generated string composed of a timestamp and a unique identifier, (e.g., `2024-08-18T11:10:03-3672330c`). This name ensures that each transport can be uniquely identified and prevents conflicts during the management and transfer of transports between systems.

What are the fields in a transport, and why can't they be edited? What can specifically be edited?

A transport contains fields such as the creation date, creator, the server ID of the source server, the schema version of ReportServer, the import date, importer, import log, and status (e.g., `CREATED`, `CLOSED`, `IMPORTED`). These fields are fixed and cannot be edited after creation to ensure the integrity and traceability of the transport. What can be edited are the objects within an open transport, which can be added or removed before the transport is closed.

Currently, the following statuses are defined:

CREATED This status is set when a transport is newly created. This can happen through a context menu or a terminal command.

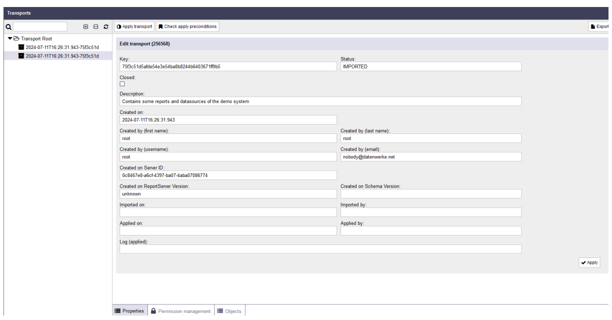
CLOSED This status is set when a transport is released or closed. This means that the transport is no longer editable, as released transports are fixed.

IMPORTED This status will be assigned when fetching transports from an external system (e.g., from a test environment).

APPLIED This status is set when an existing transport is applied. This means that the contained objects have been previously integrated into the system using a merge mechanism.

ERROR This status is set when an attempt is made to apply an existing transport and it is found that the transport is not applicable.

INVALID Internal status, currently not used.



Following is a screenshot of an imported transport. This means that the transport has been fetched from an external system.

What are transport logs, and how are they structured?

Transport logs are detailed records that document all steps and events during the transport of an object. They include information about the transport's progress, such as the time of import, actions performed, and any errors or warnings. These logs are structured to provide a clear and traceable overview of the entire transport process.

How can elements be added to a transport?

Elements such as reports and datasources can be added to a transport via the context menu "Add to Transport" in the respective areas of ReportServer. It is important to note that the state of the object at the time of addition to the transport is captured.

What happens if you add the same element to a transport multiple times?

If the same element is added to a transport multiple times, it is typically stored only once in the transport. The most recently added state of the object overwrites the previous one, ensuring that the transport contains the most current version of the object.

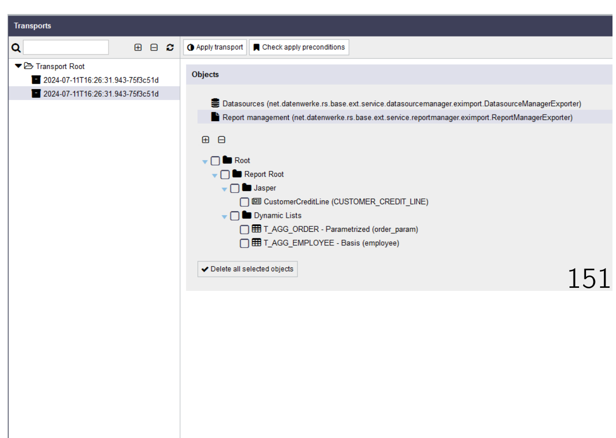
How can elements be removed from a transport?

As long as a transport is open, elements can be removed via the "Objects" tab in the Transport Management. Once removed, these elements are no longer part of the transport and will not be imported into the target system during transfer.

How can you view the elements contained in a transport?

The elements contained in a transport can be viewed in the Transport Management of ReportServer. Through the "Objects" tab, administrators can see and manage a list of all included reports, datasources, datasinks and files.

The following screenshot shows a transport that includes one Jasper report and two dynamic lists. It also contains datasources, which can be viewed by clicking on the "datasources" section.



What happens if you move the original elements? Will these changes be reflected in the transport?

If the original elements are moved or changed after being added to a transport, these changes are not automatically reflected in the transport. The transport captures the state of the object at the time of addition. Any changes require the objects to be re-added to the transport to include the updated version.

How do you transfer a transport, or a list of transports from one server to another?

Released transports are transferred from the source system (e.g., development) to the target system (e.g., production) by importing them in the Transport Management of the target system using the "Retrieve Transports from Remote" button. Here, the transports are stored in the target system but are not yet unpacked or applied.

What are application conditions? What conditions must be met to apply a transport?

Application conditions, also known as "apply preconditions", are specific requirements that must be met before a transport can be applied in the target system. These include the correct configuration of datasources, the presence of all required dependencies, and compatibility with the schema version of the target system. Transports can only be applied if ReportServer version and schema version of both source and target systems are identical.

What can be done with a transferred transport? What does "apply" mean exactly, and what happens when you click "apply"?

A transferred transport can be applied in the target system. "Apply" means that the objects in the transport are unpacked and integrated into the target system. The reports, datasources, datasinks, and files are implemented according to the defined configurations and mappings in the target system. When applying a transport, the objects are merged into the target system:

New Report / New Datasource If the object is recognized as new (Key does not yet exist in the target system), it is fully imported into the target system.

Changed Report / Changed Datasource If the object is recognized as existing (Key exists in the target system), the existing object is updated with the changes from the transport.

In the following two subsections, you can find some examples for reports.

Changes to Queries, Names, Parameters, Report Properties, Report Metadata

The report will be fully overwritten in the target system after applying the transport, so it appears exactly as it did in the source system (test system).

Variants

Variants of the report remain intact. However, caution is advised as changes to parameters or the exchange of parameters can affect the functionality of these variants. Variants could become unusable if they rely on parameters that have been changed or removed, or if the structure of the report has been significantly altered.

What happens if you click "Apply" multiple times?

Clicking "Apply" multiple times will typically result in the transport being applied only once. Reapplying should not cause additional changes since the transport is processed only once. However, errors may occur if the same objects are attempted to be imported again.

Can a transport be transported again?

Yes, a transport can be recreated at any time, regardless of its current state (open, closed, or already applied), by using the "Duplicate as Open" function in the context menu. This feature allows for copying an existing transport and managing it as an open transport. This is particularly useful for efficiently organizing content and making further changes without affecting the original transport.

What happens if a transport cannot be applied?

If a transport cannot be applied due to errors or missing prerequisites, the transport status remains unchanged, and the corresponding errors are documented in the transport's log. The administrator must resolve the reported issues before the transport can be applied again.

Can multiple transports be applied simultaneously?

Yes, multiple transports can be applied sequentially or simultaneously, provided that the application conditions for each transport are met. It is essential to ensure that the transports do not contain conflicting changes to avoid consistency issues in the target system.

What happens if a transport is interrupted during application?

If a transport is interrupted during application (e.g., due to a server crash), some parts of the transport may have been applied while others were not. It is crucial to check the system after such an interruption and, if necessary, reapply the transport or manually intervene to ensure system consistency.

Can transports be exchanged between different versions of ReportServer?

No, transports can only be exchanged and applied between systems with identical ReportServer and schema versions. Systems must be updated to the same version before a transport can be exchanged.

How are transports handled during a system backup?

Transports that have not yet been completed or applied should be completed or exported before a system backup. This ensures that they are not lost during a system restoration and can be correctly restored.

How is the order of transports determined?

The order of transports is not enforced by ReportServer but is the responsibility of the user. It is important to perform transports in the correct order, especially if later transports depend on or contain changes that build on previous ones. The user should ensure that all necessary dependencies are transported in the correct order to avoid consistency issues in the target system.

What are the risks of applying transports in the wrong order?

Applying transports in the wrong order can lead to missing dependencies or inconsistent data being imported into the target system. This may result in reports, datasources, or other objects not functioning correctly or unexpected errors occurring. Therefore, it is crucial to carefully plan and adhere to the order of transports.

Should intermediate states be transported?

Ideally, only final work states should be transported to ensure that the target system receives a stable and consistent version of the objects. Transporting intermediate states can lead to a disorganized system state and increase the risk of errors and inconsistencies. If it is necessary to transport intermediate states, it should be done with special care, and the order and dependencies should be clearly documented.

How can transports be used as a backup of work results during development?

Transports can be effectively used as backups of work results during development. Developers can regularly create transports to secure the current state of their work. These transports then serve as snapshots that can be restored or transferred to another system if needed. This is especially useful when multiple developers are working in parallel or when experimental changes are made that may need to be rolled back later. To ensure efficient backup, only meaningful work steps should be captured as transports, and these transports should be appropriately named and organized to avoid confusion.

What configurations are necessary?

To successfully perform transports in ReportServer, some configurations are necessary:

Unique Keys All transportable objects such as reports, datasources, datasinks, and files; must have a unique key (Key) that is automatically generated upon creation.

Configuration Files

- `main/transport.cf`: This file contains settings for the import and application of transports, including the specification of the remote server, target directory, and responsible user. It is crucial to ensure that the remote server key is correctly specified, and that any changes made to the `transport.cf` file are reloaded using the `config reload` command.
- `main/mappings.cf`: This file defines the mapping of datasources between the source and target systems. If an explicit mapping is defined, or if keys are identical, mapping can be omitted (Same-Key). Changes to the mappings.cf file must be reloaded with the `config reload` command.

Remote Server Configuration In the Remote Servers section, the name, key, REST URL, username, and API key of the remote server must be correctly configured.

Version and Schema Version ReportServer and schema version must be identical in both source and target systems to enable transport.

These configurations are crucial to ensure that transports are carried out correctly and applied successfully.

Can applied transports be undone?

Applied transports cannot be directly undone via ReportServer. To revert changes made by an applied transport, you must either restore a previously created system backup or manually reset the objects to their previous state. Another option is to create a new transport that reverts the changes to the desired original state. Therefore, it is important to ensure that all changes are reviewed and correct before applying a transport to avoid unintended consequences.

Chapter 12

REST API

ReportServer publishes several REST API endpoints automatically which can be used for accessing given REST services.

12.1 Finding the REST URL

You can find the main REST endpoint of your ReportServer in the "General Info" system console. Here, you can find a "REST URL" entry which displays the complete URL to your REST endpoint. An example is given below.

REST URL: `https://SERVER:PORT/reportserverbasedir/reportserver/rest`

Having this base URL it is easy to append the services described below to access the specific REST services available.

Note that spaces in URLs will be encoded as

12.2 Authentication

REST API authentication uses an apikey analogously as in executing reports via URL (see Section 7.11). Once you set an apikey for a given user, you can access all REST services using this apikey in the URL, e.g. for the "general-info" REST service:

`https://SERVER:PORT/reportserverbasedir/reportserver/rest/general-info?user=myuser&apikey=MYAPIKEY`

In the example above, the user authenticated is myuser and the apikey is MYAPIKEY.

> Note that the apikey can be found as plain text in the URL, so make sure you use HTTPS. Treat the apikey like any password.

12.3 test Service

The `test` REST service issues a test request which allows you to make sure the REST endpoint is correct and available.

The syntax is:

https://SERVER:PORT/reportserverbasedir/reportserver/rest/test?user=myuser&apikey=MYAPIKEY

12.4 general-info Service

The `general-info` REST service returns a JSON containing your system configuration. Basically, it contains all data available in the "General Info" system console.

The syntax is:

https://SERVER:PORT/reportserverbasedir/reportserver/rest/general-info?user=myuser&apikey=MYAPIKEY

12.5 dot-renderer Service

The `dot-renderer` REST service allows you to render a Graphviz DOT file (https://graphviz.org/doc/info/lang.html) into a SVG file which can be displayed directly by your web browser.

The syntax is:

https://SERVER:PORT/reportserverbasedir/reportserver/rest/dot-renderer?user=myuser&apikey=MYAPIKEY

Virtual Filesystem / Internal Files

In order to render internal files you need to make a `GET` request and ***must*** specify an additional query variable `path`, which points to the DOT file in your ReportServer virtual filesystem, in this case to `/fileserver/resources/graph.dot`.

(`GET`) https://SERVER:PORT/reportserverbasedir/reportserver/rest/dot-renderer?path=/fileserver/resources/graph.dot&user=myuser&apikey=MYAPIKEY

External Files

In order to render arbitrary external data you need to make a `POST` request and use the request body to specify the dot-data you want to render in plain text.

(`POST`) https://SERVER:PORT/reportserverbasedir/reportserver/rest/dot-renderer?user=myuser&apikey=MYAPIKEY

Don't forget to include your external DOT file in your POST request body.

Customizing Output

An optional `width` parameter may be used to change the SVG file's width. Default is `1200`.

(`GET`) `https://SERVER:PORT/reportserverbasedir/reportserver/rest/dot-renderer?path=/fileserver/resources/graph.dot&width=900&user=myuser&apikey=MYAPIKEY`

(`POST`) `https://SERVER:PORT/reportserverbasedir/reportserver/rest/dot-renderer?width=900&user=myuser&apikey=MYAPIKEY`

An optional `wrap_html` parameter may be used to wrap the SVG output with an empty `header` and a `body` in which the result is embedded. By default this option is set to `false`. Be aware of the fact that this option changes the content type from **image/svg+xml** to **text/html**!

(`GET`) `https://SERVER:PORT/reportserverbasedir/reportserver/rest/dot-renderer?path=/fileserver/resources/graph.dot&user=myuser&apikey=MYAPIKEY&wrap_html=true`

(`POST`) `https://SERVER:PORT/reportserverbasedir/reportserver/rest/dot-renderer?user=myuser&apikey=MYAPIKEY&wrap_html=true`

12.6 markdown-renderer Service

The `markdown-renderer` REST service allows you to render a markdown file (`https://www.markdownguide.org/`) to an HTML representation. While most markdown commands are supported, there are exceptions which cannot be translated directly to HTML and will not be rendered properly. Refer to Chapter 13 for details.

The syntax is:

`https://SERVER:PORT/reportserverbasedir/reportserver/rest/markdown-renderer?user=myuser&apikey=MYAPIKEY`

Virtual Filesystem / Internal Files

Similar to `dot-renderer`, in order to render internal files you need to make a `GET` request and ***must*** specify an additional query variable `path`, which points to the markdown file in your ReportServer virtual filesystem, in this case to `/fileserver/resources/example.md`.

(`GET`) `https://SERVER:PORT/reportserverbasedir/reportserver/rest/markdown-renderer?path=/fileserver/resources/example.md&user=myuser&apikey=MYAPIKEY`

External Files

In order to render arbitrary external data you need to make a `POST` request and use the request body to specify the markdown-data you want to render in plain text.

(`POST`) `https://SERVER:PORT/reportserverbasedir/reportserver/rest/markdown-renderer?user=myuser&apikey=MYAPIKEY`

Don't forget to include your external markdown file in your POST request body.

Customizing Output

An optional `wrap_html` parameter may be used to wrap the HTML output with an empty `header` and a `body` in which the result is embedded. By default this option is set to `false`.

(`GET`) `https://SERVER:PORT/reportserverbasedir/reportserver/rest/markdown-renderer?path=/fileserver/resources/example.md&user=myuser&apikey=MYAPIKEY&wrap_html=true`

(`POST`) `https://SERVER:PORT/reportserverbasedir/reportserver/rest/markdown-renderer&user=myuser&apikey=MYAPIKEY&wrap_html=true`

12.7 node-exporter Service

The `node-exporter` REST service allows you to export any node to XML in an analogous way as the "export" functionality described in Section 10. Along with the XML export, some metadata is displayed in the JSON file.

The syntax is dependent on the specific node being exported and will be described in the following subsections. All nodes support a `flatten` parameter. If this parameter is set to `true`, all objects exported are flattened, i.e. all folders are stripped off.

As an example, consider exporting all reports with the following syntax:

`https://SERVER:PORT/reportserverbasedir/reportserver/rest/node-exporter/reportmanager/;flatten=true?user=myuser&apikey=MYAPIKEY`

You can always use the terminal for finding out the exact path of the node you want to export. Just navigate to the node you to export and enter the `pwd` command as described in Section 20.50.

Exporting reports

The `node-exporter` service can be used to export reports or report directories using the following syntax:

`https://SERVER:PORT/reportserverbasedir/reportserver/rest/node-exporter/reportmanager/myreports/myreport?user=myuser&apikey=MYAPIKEY`

where `/reportmanager/myreports/myreport` is a path to the report being exported. You can also export a complete report directory, e.g. `/reportmanager/myreports`.

An optional `includeVariants` parameter is used to export variants along with the base reports. Default is `false`:

`https://SERVER:PORT/reportserverbasedir/reportserver/rest/node-exporter/reportmanager/myreports/myreport?includeVariants=true&user=myuser&apikey=MYAPIKEY`

Exporting users

The node-exporter service can be used to export users, groups or organizational units using the following syntax:

https://SERVER:PORT/reportserverbasedir/reportserver/rest/node-exporter/usermanager/ClassicModelCars/myuser?user=myuser&apikey=MYAPIKEY

where /usermanager/ClassicModelCars/myuser is a path to the user or group being exported. You can also export a complete organizational unit, e.g. /usermanager/ClassicModelCars.

Exporting datasources

The node-exporter service can be used to export datasources or datasource directories using the following syntax (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):

https://SERVER:PORT/reportserverbasedir/reportserver/rest/node-exporter/datasources/internal%20datasources/ReportServer%20Data%20Source?user=myuser&apikey=MYAPIKEY

where /datasources/internal datasources/ReportServer Data Source is a path to the datasource being exported. You can also export a complete directory, e.g. /datasources/internal datasources.

Exporting datasinks

The node-exporter service can be used to export datasink or datasink directories using the following syntax:

https://SERVER:PORT/reportserverbasedir/reportserver/rest/node-exporter/datasinks/mydirectory/mydatasink?user=myuser&apikey=MYAPIKEY

where /datasinks/mydirectory/mydatasink is a path to the datasink being exported. You can also export a complete directory, e.g. /datasinks/mydirectory.

Exporting files

The node-exporter service can be used to export files or file directories from the virtual file system using the following syntax:

https://SERVER:PORT/reportserverbasedir/reportserver/rest/node-exporter/fileserver/mydirectory/myfile.groovy?user=myuser&apikey=MYAPIKEY

where /fileserver/mydirectory/myfile.groovy is a path to the file being exported. You can also export a complete directory, e.g. /fileserver/mydirectory.

Exporting remote servers

The node-exporter service can be used to export remote servers or remote server directories using the following syntax:

https://SERVER:PORT/reportserverbasedir/reportserver/rest/node-exporter/remoteservers/mydirectory/myremoteserver?user=myuser&apikey=MYAPIKEY

where /remoteservers/mydirectory/myremoteserver is a path to the remote server being exported. You can also export a complete directory, e.g. /remoteservers/mydirectory.

Exporting transports

The node-exporter service can be used to export transports or transport directories using the following syntax:

https://SERVER:PORT/reportserverbasedir/reportserver/rest/node-exporter/transports/mydirectory/mytransport?user=myuser&apikey=MYAPIKEY

where /transports/mydirectory/mytransport is a path to the transport being exported. You can also export a complete directory, e.g. /transports/mydirectory.

Note that if you are exporting a specific transport, you have to enter its key. E.g.:

https://SERVER:PORT/reportserverbasedir/reportserver/rest/node-exporter/transports/mydirectory/a5f9000b91ea0a85c3ab8a9a35cbec65aa1a609a?user=myuser&apikey=MYAPIKEY

where a5f9000b91ea0a85c3ab8a9a35cbec65aa1a609a is the transport's key.

Exporting dashboards from dashboard library

The node-exporter service can be used to export dashboards or dashboard library directories using the following syntax (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)::

https://SERVER:PORT/reportserverbasedir/reportserver/rest/node-exporter/dashboardlib/mydirectory/Demo%20Dashboard?user=myuser&apikey=MYAPIKEY

where /dashboardlib/mydirectory/Demo Dashboard is a path to the dashboard being exported. You can also export a complete directory, e.g. /dashboardlib/mydirectory.

Chapter 13

Markdown

ReportServer provides robust support for rendering Markdown files (https://www.markdownguide.org/). This allows users to create and preview Markdown documents within the platform.

Following is an overview of the capabilities, along with examples demonstrating various Markdown syntax elements. A comprehensive documentation on Markdown syntax can be found here: https://www.markdownguide.org/ and https://www.markdownguide.org/basic-syntax/.

> Please note that there is a markdown-renderer REST service. More information on this REST service can be found in Section 12.6. This means, the renderer can be called by URL, e.g. https://SERVER:PORT/reportserverbasedir/reportserver/rest/markdown-renderer?path=/fileserver/resources/example.md&user=myuser&apikey=MYAPIKEY.
>
> Also, the markdown renderer can be used via scripting (`net.datenwerke.rs.markdown.service.markdown.MarkdownService`). Refer to the Scripting Guide for more details.

In ReportServer, it is important to set the content type of the Markdown file (e.g. `markdown.md`) to "text/markdown". This is necessary in order for ReportServer to recognize the file as Markdown files and being able to convert the files to HTML.

> Note that if the file's content type is not set to "text/markdown", you won't see the preview tab or preview buttons.

Features:

Preview Markdown Tab Available in the file system administration window, this tab allows users to preview the Markdown content directly within ReportServer.

Preview Markdown Button Available in the Properties tab of the file system administration window, This button renders the Markdown and exports it as HTML into a new window.

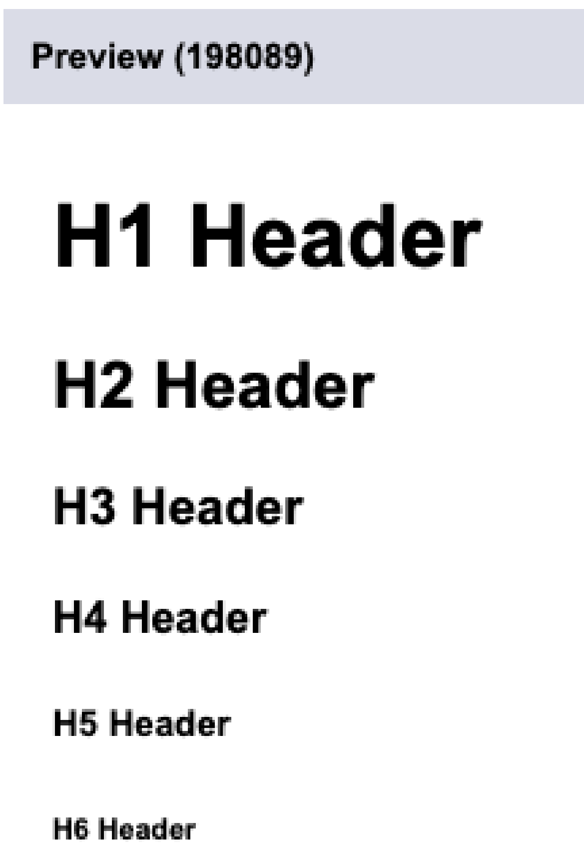
13.1 Markdown Syntax Examples

Header

Markdown supports headers, which are created using the # symbol. The number of # symbols indicates the level of the header.

```
# H1 Header
## H2 Header
### H3 Header
#### H4 Header
##### H5 Header
###### H6 Header
```

This is a preview of the result rendering.



Emphasis

Markdown allows for text emphasis using underscores _ or asterisks *.

```
*Italic* or _Italic_

**Bold** or __Bold__

***Bold Italic*** or ___Bold Italic___
```

This is a preview of the result rendering.

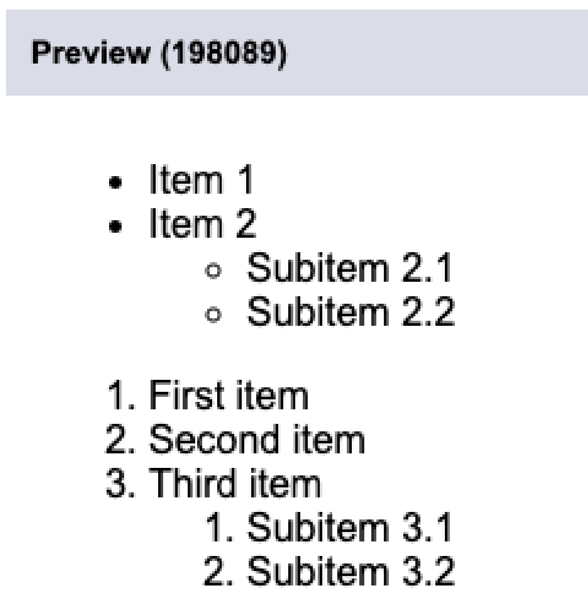
Lists

Markdown supports both ordered and unordered lists.

Unordered List	Created using -, *, or +.
Ordered List	Created using numbers followed by a period.

```
- Item 1
- Item 2
  - Subitem 2.1
  - Subitem 2.2

1. First item
2. Second item
3. Third item
   1. Subitem 3.1
   2. Subitem 3.2
```

This is a preview of the result rendering.



Task List Items

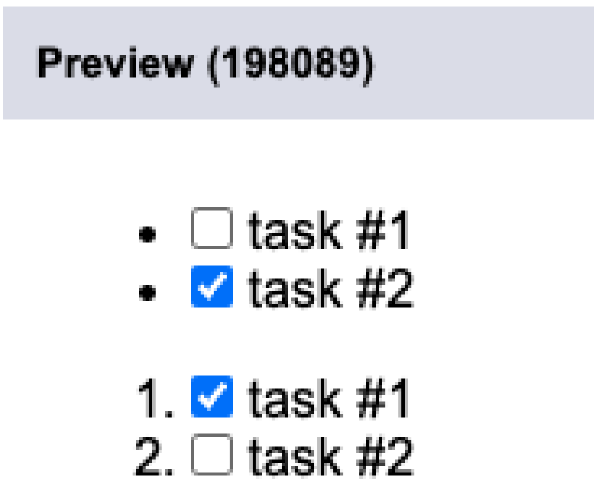
Task lists are rendered in Markdown as checkboxes, either selected or unselected.

A task can be represented as a list item where the first non-whitespace character is a left bracket `[`, followed by a space or an `x` (either lowercase or uppercase), then a right bracket `]` followed by at least one space before any other content. An example is shown below.

Note that you have to include the `-` for creating an unordered task list as explained in Section 13.1, or numbers followed by a period for creating an ordered task list.

```
- [ ] task #1
- [x] task #2

1. [x] task #1
2. [ ] task #2
```

This is a preview of the result rendering.



Comments

Comments can be added as shown in the following example:

```
[//]: # (platform independent comment)
```

To ensure maximum portability, it is essential to insert a blank line before and after this type of comment. This is because some Markdown parsers may not function correctly if definitions are directly adjacent to regular text.

Links

Links can be created using square brackets `[]` for the text and parentheses `()` for the URL.

```
[ReportServer](https://www.reportserver.net)
```

This is a preview of the result rendering.

Images

Images are similar to links but prefixed with an exclamation mark `!`.

```
![Alt Text](
http://localhost:8090/ReportServer/reportserver/fileServerAccess?id=9399)
```

where http://localhost:8090/ReportServer/reportserver/fileServerAccess?id=9399 is the image's URL in ReportServer.

You can also set the height and width of the image as shown below. Both attributes are optional, so you can specify only the width or the height as well.

```
![Alt Text](
http://localhost:8090/ReportServer/reportserver/fileServerAccess?id=9399){width
    =100 height=200}
```

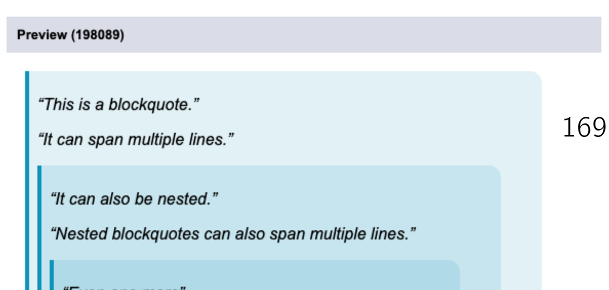
This is a preview of the result rendering.

Blockquotes

Blockquotes are created using the `>` symbol. Note that blockquotes can be nested.

```
> This is a blockquote.
>
> It can span multiple lines.
>
> > It can also be nested.
> >
> > Nested blockquotes can also span multiple lines.
> >
> > > Even one more
> > >
> > > Nested blockquote
```

This is a preview of the result rendering.



Code

Markdown provides two ways to format code: ***inline*** code and ***code blocks***.

Inline Code

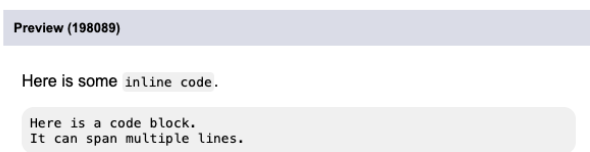
To create inline code, enclose the code snippet in backticks:

```
Here is some `inline code`.
```

Code Blocks

For multiline code blocks, use triple backticksbefore and after the code:

````
```
Here is a code block.
It can span multiple lines.
```
````



This is a preview of the result rendering.

> Important: Ensure that there is an empty line before and after the code block for proper formatting.

Syntax Highlighting Markdown supports syntax highlighting by specifying the language name after the opening triple backticks. For example, to highlight Groovy code:

````
```groovy
// This is a Groovy code block
def example = "Hello, Markdown!"
println(example)
```
````

This helps improve readability and provides language-specific syntax highlighting.

Supported are:

- brainfuck
- c
- clike
- clojure

- cpp
- csharp
- css
- css-extras
- dart
- git
- go
- groovy
- java
- javascript
- typescript
- json
- kotlin
- latex
- makefile
- markdown
- markup
- python
- scala
- sql
- swift
- yaml

When working with DOT code blocks (Section 13.1), be aware that their enclosing characters differ from those used in standard Markdown code blocks, even if they appear similar. Always check the correct syntax to avoid formatting issues.

Embedding Graphviz DOT

ReportServer's Markdown syntax allows you to directly include Graphviz DOT files (https://graphviz.org/) into Markdown by as shown in the example below by enclosing them in ´´´. The DOT files are then rendered to SVG. You can find more information on Graphviz DOT on Section 14. You can find many more examples here: https://graphviz.org/gallery/.

Please ensure that there is a new line after the opening characters and before the closing characters when using DOT code blocks.

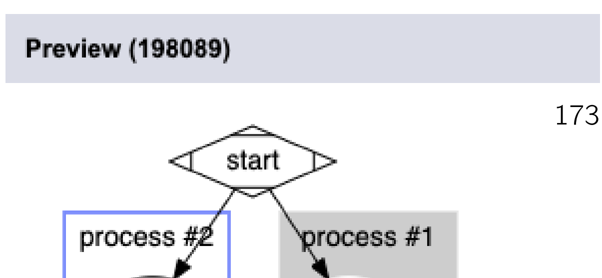
Additionally, note that the enclosing characters for code blocks (Section 13.1) are not identical to those used in DOT code blocks, although they may appear similar.

```
´´´

// https://graphviz.org/Gallery/directed/cluster.html
digraph G {
  fontname="Helvetica,Arial,sans-serif"
  node [fontname="Helvetica,Arial,sans-serif"]
  edge [fontname="Helvetica,Arial,sans-serif"]

  subgraph cluster_0 {
    style=filled;
    color=lightgrey;
    node [style=filled,color=white];
    a0 -> a1 -> a2 -> a3;
    label = "process #1";
  }

  subgraph cluster_1 {
    node [style=filled];
    b0 -> b1 -> b2 -> b3;
    label = "process #2";
    color=blue
  }
  start -> a0;
  start -> b0;
  a1 -> b3;
  b2 -> a3;
  a3 -> a0;
  a3 -> end;
  b3 -> end;

  start [shape=Mdiamond];
  end [shape=Msquare];
}

´´´
```

You can include an optional width or/and height of the rendered SVG by including these after the ´´´ opening tag. You have the flexibility to specify an optional width and/or height for the rendered SVG by including these parameters immediately after the ``` opening tag on the same line as shown below.

````
```width=100 height=200

// https://graphviz.org/Gallery/directed/cluster.html
digraph G {
  fontname="Helvetica,Arial,sans-serif"
  node [fontname="Helvetica,Arial,sans-serif"]
  edge [fontname="Helvetica,Arial,sans-serif"]

  subgraph cluster_0 {
    style=filled;
    color=lightgrey;
    node [style=filled,color=white];
    a0 -> a1 -> a2 -> a3;
    label = "process #1";
  }

  subgraph cluster_1 {
    node [style=filled];
    b0 -> b1 -> b2 -> b3;
    label = "process #2";
    color=blue
  }
  start -> a0;
  start -> b0;
  a1 -> b3;
  b2 -> a3;
  a3 -> a0;
  a3 -> end;
  b3 -> end;

  start [shape=Mdiamond];
  end [shape=Msquare];
}

```
````

This is a preview of the result rendering.



Tables

Tables in Markdown are created using pipes | and hyphens - to define the structure of the table. It uses GitHub's Flavored Markdown, described here: https://docs.github.com/en/get-started/writing-on-github/working-with-advanced-formatting/organizing-information-with-tables. Here's an example:

```
| Header 1 | Header 2 | Header 3 |
|----------|----------|----------|
| Row 1    | Data 1   | Data 2   |
| Row 2    | Data 3   | Data 4   |
| Row 3    | Data 5   | Data 6   |
```

Preview (198089)

Header 1	Header 2	Header 3
Row 1	Data 1	Data 2
Row 2	Data 3	Data 4
Row 3	Data 5	Data 6

This is a preview of the result rendering.

You can align text to the left, right, or center of a column by placing colons : to the left, right, or on both sides of the hyphens in the header row as shown below.

```
| Left-aligned | Center-aligned | Right-aligned |
| :---         |     :---:      |          ---: |
| Row 1        | Data 1         | Data 2        |
| Row 2        | Data 3         | Data 4        |
```

Preview (198089)

Left-aligned	Center-aligned	Right-aligned
Row 1	Data 1	Data 2
Row 2	Data 3	Data 4

This is a preview of the result rendering.

You can include formatting such as links, inline code blocks, and text styling within your table:

```
| Left-aligned     | Center-aligned            | Right-aligned |
| :---             |     :---:                 |          ---: |
| Row1 *important* | `Data 1`                  | Data 2        |
| Row 2            | Data 3 **very important** | Data 4        |
```

Preview (198089)

Left-aligned	Center-aligned	Right-aligned
Row1 *important*	`Data 1`	Data 2
Row 2	Data 3 **very important**	Data 4

This is a preview of the result rendering.

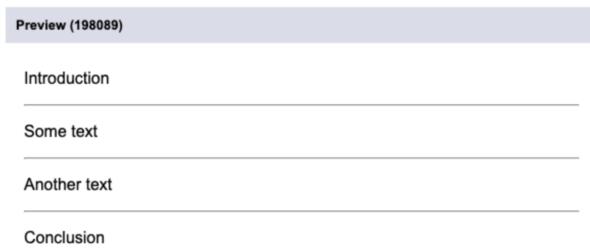
You can find much more information on tables here: https://docs.github.com/en/get-started/writing-on-gith

Horizontal Rules

Horizontal rules can be used to create a visual separation between sections. They are created using three or more hyphens `--`, asterisks `***`, or underscores `___`.

```
Introduction

---

Some text

***

Another text

---

Conclusion
```

This is a preview of the result rendering.



Strikethrough

Strikethrough of text is done in Markdown by enclosing it in tilde as shown below.

```
The ~~old data~~ old files have been removed.
```

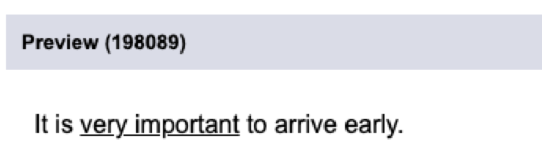
This is a preview of the result rendering.

Underlining

Underlining of text is done im Markdown by enclosing it in `++` as shown below.

```
It is ++very important++ to arrive early.
```

This is a preview of the result rendering.



Chapter 14

Graphviz DOT

Graphviz https://graphviz.org/ is a powerful tool for creating diagrams and visual representations of graphs using the DOT language. The DOT language allows you to describe the nodes and edges of a graph in a plain text format, which Graphviz then uses to render the graph visually.

Below is an overview of the features, along with examples demonstrating various elements of DOT syntax. For complete documentation on DOT syntax, visit the Graphviz documentation: https://graphviz.org/. You can find many more examples here: https://graphviz.org/gallery/.

Please note the availability of a dot-renderer REST service. Detailed information about this REST service is provided in Section 12.5. This service can be accessed via URL, for example: https://SERVER:PORT/reportserverbasedir/reportserver/rest/dot-renderer?path=/fileserver/resources/graph.dot&user=myuser&apikey=MYAPIKEY.

Additionally, the DOT renderer can be utilized through scripting (`net.datenwerke.rs.dot.service.dot.DotService`). Refer to the Scripting Guide for more details.

Please note that you can embed DOT files directly within Markdown syntax. You can also specify the width and height of the rendered SVG. For more details, refer to Section 13.1.

Please be aware that the DOT renderer creates a temporary file in your `user.home` directory. Therefore, this directory must have read and write permissions for your Tomcat server. You can verify if this directory is readable in the "General Information" section of your system console under the "User home" entry. Additionally, ensure that this directory is writable.

In ReportServer, it's crucial to set the content type of your DOT file (e.g., `myfile.dot`) to "text/vnd.graphviz". This ensures that ReportServer recognizes the file as a DOT file and can convert it to SVG.

> Note that if the file's content type is not set to "text/vnd.graphviz", the preview tab and preview buttons will not be available.

Features

Preview SVG Tab Located in the file system administration window, this tab allows users to preview DOT content directly within ReportServer, rendered as SVG.

Preview SVG Button Found in the Properties tab of the file system administration window, this button renders the DOT content and exports it as an SVG in a new window.

14.1 Basic DOT Syntax

A DOT file consists of a graph definition, which includes nodes, edges, and their attributes. The basic structure of a DOT file is as follows:

Graph Represents an undirected graph.

Digraph Represents a directed graph.

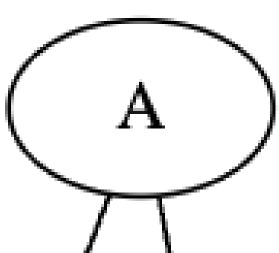
Node Represents a node in the graph.

Edge Represents an edge between two nodes

Example of an Undirected Graph:

```
graph G {
    A -- B;
    B -- C;
    C -- D;
    D -- A;
}
```

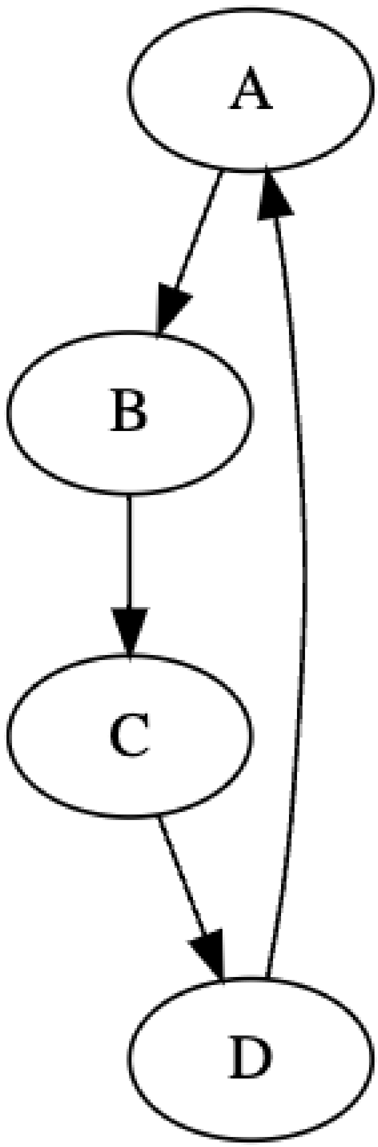
This is a preview of the result rendering.



Example of a Directed Graph:

```
digraph G {
    A -> B;
    B -> C;
    C -> D;
    D -> A;
}
```

This is a preview of the result rendering.



Nodes and Edges

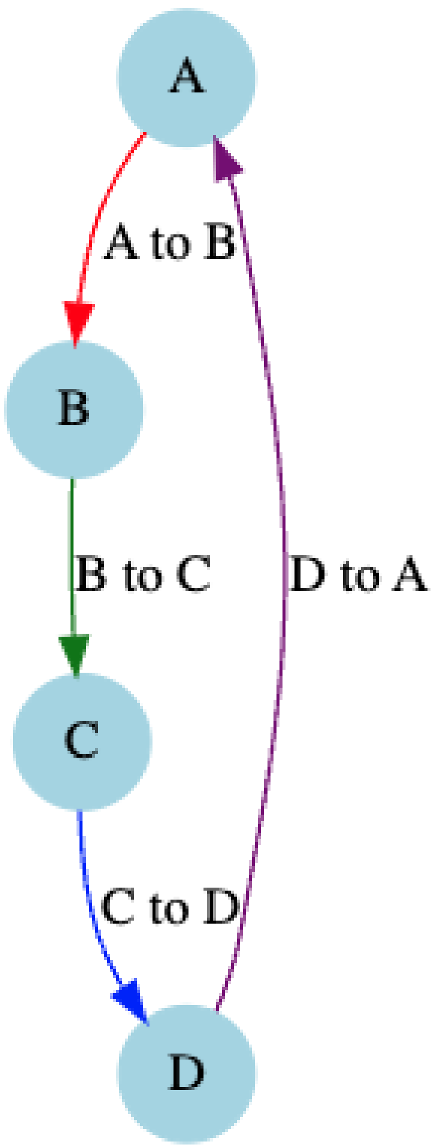
Nodes and edges can be customized with various attributes such as labels, colors, shapes, and styles.

Example with node and edge attributes:

```
digraph G {
    node [shape=circle, color=lightblue, style=filled];
    A -> B [label="A to B", color=red];
    B -> C [label="B to C", color=green];
```

```
    C -> D [label="C to D", color=blue];
    D -> A [label="D to A", color=purple];
}
```

This is a preview of the result rendering.



Below is an overview of node attributes:

shape Defines the shape of the node (e.g., `circle`, `box`, `ellipse`).

color Defines the color of the node.

style Defines the style of the node (e.g., `filled`, `dashed`, `dotted`).

Below is an overview of edge attributes

label Defines a label for the edge.

color Defines the color of the edge.

style Defines the style of the edge (`solid`, `dashed`, `dotted`).

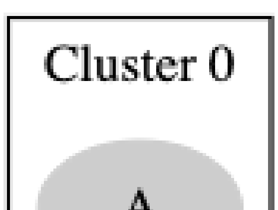
Subgraphs

Subgraphs are used to group nodes and edges together, which can help in organizing complex graphs or applying common attributes to a group of nodes and edges.

Example of Subgraphs

```
digraph G {
    subgraph cluster_0 {
        node [style=filled, color=lightgrey];
        A -> B;
        B -> C;
        label = "Cluster 0";
    }

    subgraph cluster_1 {
        node [style=filled, color=lightblue];
        D -> E;
        E -> F;
        label = "Cluster 1";
    }

    C -> D;
    F -> A;
}
```

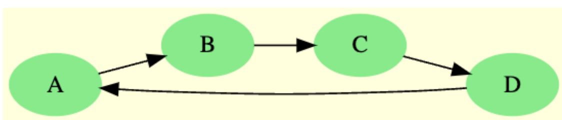
This is a preview of the result rendering.



Graph Attributes

Graph attributes define the overall appearance and layout of the graph.

```
digraph G {
    graph [rankdir=LR, bgcolor=lightyellow];
    node [shape=ellipse, style=filled, color=lightgreen];
    edge [color=black];

    A -> B;
    B -> C;
    C -> D;
    D -> A;
}
```



This is a preview of the result rendering.

An overview of the graph attributes is shown below.

rankdir Defines the direction of the graph layout (`TB` for top to bottom, `LR` for left to right).

bgcolorr Defines the background color of the graph.

Chapter 15

Remote RS Servers

ReportServer 4.6.0 introduces a new kind of node called "remote server". This is used to defined remote ReportServer servers, which can be then used to copy external data, e.g. reports or users from another ReportServer installation (e.g. PROD) into your local ReportServer installation (e.g. TEST). For this, refer to Section 20.56 rpull.

Currently, remote RS REST servers are supported, which use the ReportServer REST API.

The REST URL can be found in the "General info" system console of your remote ReportServer installation. The remote REST RS server uses an apikey analogously as in executing reports via URL (see Section 7.11). Once you set an apikey for a given user, you can access all REST services, including remote RS REST servers.

The "Test remote RS REST server" button allows to test the remote server and make sure all data entered on its definition works correctly.

Note that the apikey can be found as plain text in the URL, so make sure you use HTTPS. Treat the apikey like any password.

Chapter 16

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 if the user is not an owner.

During Scheduling three different Scheduling Actors are important:

The executor
: The executor is the user as which the report will be executed. Thus, this determines the ***permissions and the data available during report execution***. Allowing to change the executor is useful e.g. if the original executor leaves the organization: the scheduler administrator is able to simply change the executor by a couple of clicks. The executor may only be changed by the Scheduler Administrator.

The owners
: The owners are users that are able to change a scheduled job. The job owners may change the complete job definition, but they can not change the job executor if they are not Scheduler Administrators. Allowing a job to have more than one owner is useful if the original job owner is out of office for some weeks, e.g. in vacation, and the job has to be changed by someone else during this time. Defining multiple owners allows all owners to change a job without having to give them Scheduler Administrator permissions.

The recipients
: Recipients are the users that get the scheduled report e.g. by e-mail.

16.1 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 datasource 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.
```

16.2 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. ReportServer 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.

16.3 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.

16.4 Terminal Commands

The terminal allows you to control the scheduler, e.g. to manually terminate or restart it. For further information refer to Section 20.58 (command **scheduler**).

16.5 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.

16.6 Creating and Using a Condition Report

In this section, we will give an example of how to create a conditional report, That is, the corresponding variant can be set to be executed in a conditional manner, i.e. only if the given condition is met during the scheduled execution time. Note that only variants may be scheduled.

First of all, a condition report must be created. The condition report is a variant of a dynamic list which we will use during conditional scheduling.

The demo data comes with the table T_AGG_PRODUCT, which contains information about products. As an example, we create a variant of the T_AGG_PRODUCT as a condition report. We select the PRO_PRODUCTNAME column and we will set the condition to be met if this column contains at least 10 entries.

For this purpose, open the T_AGG_PRODUCT report (Administration - Reports). Select the column PRO_PRODUCTNAME and set the aggregate function "Count". You can enter an alias for your column, e.g. NUMBER_OF_PRODUCTS. Save the report as a variant and call this "min10_Condition". Note that you need to ensure that your condition report only always returns a single row. In the above example, the report contains only the column PRO_PRODUCTNAME.

Creating a condition

Now that you have defined a condition report, it is time to create the condition.

In this example, you will create a condition that uses the condition report "min10_Condition" you previously created. To do this, open the terminal (CTRL + ALT + T) and enter the following command:

```
rcondition create id:Report:152649 min10_Condition myCondition "Min 10 entries"
```

The easiest to provide the report is via id. In the example the id is 152649 so you can use the above command. You can find more information on the syntax of the rcondition command in Section 20.51.

Note: The id for the report can be found in the heading line of the variant. In our example, the id of the condition report "min10_Condition" is 152649.

ReportServer confirms the creation of the condition with a "Condition created" message. You can check the existing conditions via the `rcondition list` command, and remove conditions with `rcondition remove`.

Using the condition while scheduling

Finally, we have a condition and can now use it when scheduling. For this, when you schedule a report, make sure to check the box "advanced options" on the first page of the scheduling wizard. Coose the dynamic list report T_AGG_PRODUCT for scheduling and select two columns: PRO_PRODUCTNAME and Y_AVG_PRICE. Save this report as a variant under the name "Productname_avgPrice". This is the variant being scheduled using the condition report "min10_Condition" previously created.

Open the variant "Productname_avgPrice" and click on "Schedule". Check "advanced options" and follow all steps until you reach the "Conditional Scheduling" dialog. Click on "Add condition" and select the condition "min10_Condition".

After clicking on the "Submit" button you can define the actual condition using a condition formula. In our example, you can use the formula `PRO_PRODUCTNAME >= 10`, which means that the condition holds if the number of products (`PRO_PRODUCTNAME`) is at least 10. You can test the condition for validity with the "test condition" button.

With this, the conditional scheduling is ready. Before the report is executed while scheduling, the condition is checked for validity. If the condition is true, the report is executed and sent.

Note: Setting the option "If the conditions do not hold" you can define the behavior if the condition is not true. Either skip the execution (which is probably the usual case) or retry the execution.

16.7 Predefined Conditions

The ReportServer administrator is also able to create predefined conditions by scripting. The scheduling users can select the desired condition(s) from the set of conditions predefined. In such a way, the users do not have to type formulas, which improves user experience and avoids common errors. As a standard predefined scheduler condition, ReportServer has a "not empty" condition. It allows to prevent sending a report if it is empty.

It is important to emphasize that completely new conditions may be created by scripting. As an example, please take a look at the following simple script. It allows report execution during working days, while during the weekend, the report execution is disabled.

```
import net.datenwerke.rs.condition.service.condition.hooks.ConditionProviderHook
import net.datenwerke.rs.condition.client.condition.dto.SimpleCondition
import java.util.Calendar

def HOOK_NAME = "IS_WORKINGDAY_HOOK"

def callback = [
  provideConditionFor: { report ->
   SimpleCondition cond = new SimpleCondition();
    cond.setKey(HOOK_NAME);
    cond.setName("Is working day");
    cond.setDescription("Actions are executed if today is a working day");
    return cond;
  },
  consumes: { key -> return HOOK_NAME.equals(key); },
  execute: { key, expression, user, rjob ->
   Calendar c1 = Calendar.getInstance();
   c1.setTime(new Date());
   return ( c1.get(Calendar.DAY_OF_WEEK) == Calendar.SATURDAY ||
           c1.get(Calendar.DAY_OF_WEEK) == Calendar.SUNDAY ) ? false: true;
  },
  isBeforeActions: { -> return true; },
  isBeforeJob: { -> return true; },
] as ConditionProviderHook

GLOBALS.services.callbackRegistry.attachHook(HOOK_NAME, ConditionProviderHook.
      class, callback)
```

This script may be saved in the startup.d folder for execution at ReportServer startup. The predefined condition "Is working day" is then available for users during scheduling definition.

16.8 Defining a Simple Condition via Scripting

Alternatively, the conditions can also be defined via a small script. The basic outline of the script is:

```
import net.datenwerke.scheduler.service.scheduler.hooks.SchedulerExecutionHook
import net.datenwerke.scheduler.service.scheduler.hooks.adapter. ↙
    ↳ SchedulerExecutionHookAdapter
import net.datenwerke.rs.scheduler.service.scheduler.jobs.report.ReportExecuteJob
import net.datenwerke.rs.base.service.reportengines.table.entities.TableReport
import net.datenwerke.rs.base.service.reportengines.table.output.object. ↙
    ↳ CompiledTableReport
import net.datenwerke.scheduler.service.scheduler.helper.SkipJobExecution
def HOOK_NAME = "SkipEmptyListSchedulerHook";
def callback = [
  doesVetoExecution : { job, logEntry ->
    if(job should not be executed)
        return new SkipJobExecution("No data")
  }
  ] as SchedulerExecutionHookAdapter
GLOBALS.services.callbackRegistry.attachHook(HOOK_NAME, SchedulerExecutionHook. ↙
    ↳ class, callback)
```

So, basically, you implement the method doesVetoExecution as part of hook SchedulerExecutionHook. One example implementation, which checks whether or not the report contains any data (note that this only works with dynamic lists) would be:

```
import net.datenwerke.scheduler.service.scheduler.hooks.SchedulerExecutionHook
import net.datenwerke.scheduler.service.scheduler.hooks.adapter. ↙
    ↳ SchedulerExecutionHookAdapter
import net.datenwerke.rs.scheduler.service.scheduler.jobs.report.ReportExecuteJob
import net.datenwerke.rs.base.service.reportengines.table.entities.TableReport
import net.datenwerke.rs.base.service.reportengines.table.output.object. ↙
    ↳ CompiledTableReport
import net.datenwerke.scheduler.service.scheduler.helper.SkipJobExecution
def HOOK_NAME = "SkipEmptyListSchedulerHook";
def callback = [
  doesVetoExecution : { job, logEntry ->
    if(job instanceof ReportExecuteJob && job.getReport() instanceof TableReport){
       job.doExecute()
       if(job.getExecutedReport() instanceof CompiledTableReport && !job. ↙
    ↳ getExecutedReport().hasData()){
         return new SkipJobExecution("No data")
       }
    }
  }
  ] as SchedulerExecutionHookAdapter
GLOBALS.services.callbackRegistry.attachHook(HOOK_NAME, SchedulerExecutionHook. ↙
    ↳ class, callback)
```

For a general introduction to scripting, refer to the Scripting Guide.

Chapter 17

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>

  <saikuCharts>
    <color>#1f77b4</color>
    <color>#aec7e8</color>
    <color>#ff7f0e</color>
    <color>#ffbb78</color>
    <color>#2ca02c</color>
    <color>#98df8a</color>
```

```
      <color>#d62728</color>
      <color>#ff9896</color>
      <color>#9467bd</color>
      <color>#c5b0d5</color>
      <color>#8c564b</color>
      <color>#c49c94</color>
      <color>#e377c2</color>
      <color>#f7b6d2</color>
      <color>#7f7f7f</color>
      <color>#c7c7c7</color>
      <color>#bcbd22</color>
      <color>#dbdb8d</color>
      <color>#17becf</color>
      <color>#9edae5</color>
    </saikuCharts>
  </theme>
</configuration>
```

The configuration file consists of five 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.

Next, in the “css” 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.

Finally, in the “saikuCharts” section, you can change the color theme used in Saiku/Mondrian report charts. You can use either hexadecimal color notation as in the example above, or named colors from the “colors” list, e.g. “purple-light”.

Chapter 18

Terminal

The ReportServer Terminal is a mighty tool for administrators. You open it with the keyboard shortcut `CTRL+ALT+T` or using the "Admin Panel" system console (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.

18.1 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 20.

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.

18.2 The Virtual File System

ReportServer integrates various objects (reports, datasources, 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     Datasource management
reportmanager    Report management
dashboardlib    Dashboard 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 subfolder.

```
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
```

18.3 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
```

Thus, the cache which stores all configuration files will be reloaded by entering the command above. As an alternative, you can restart your ReportServer for reloading config files.

18.4 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 7.9). 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.

18.5 Object Resolver

The following describes how to locate entities. Entities are stored objects such as reports, users or TeamSpaces. You can find entities by searching for classes annotated with `@Entity`. You can also find a list of all entities in our ReportServer SourceForge project https://sourceforge.net/projects/dw-rs/. Download the latest apidocs file from the src directory for this. Further, you can also find all entities for the current ReportServer version here: https://reportserver.net/api/latest/entities.html.

Many terminal commands provide you with an object resolver to find a specific entity or a group of entitities. Currently, there are three object resolvers:

ID Resolver	Allows you to locate an entity by its entity ID.
Path Resolver	Allows you to find an entity by its Virtual File System path.
HQL Resolver	Allows you to find entities by a HQL (Hibernate Query Language) query.

We will illustrate these object resolvers by using the locate command. Any other command supporting object resolvers works analogously to the examples provided below.

The locate command allows you, as its name suggests, to locate entities by using an object resolver query. Refer to the locate command documentation for more information on this.

The ID object resolver uses an entity ID for locating an entity. Since entity IDs are unique, the query resolves to zero or one entity. The syntax is: `id:EntityType:entityId` where `EntityType` is the specific entity class, e.g. `TableReport`. Refer to the entity list mentioned above for the specific entity types. `entityId` is the specific entity ID. E.g.:

```
reportserver$ locate id:TableReport:123
Report Root/Dynamic Lists/myReport (Reportmanager)
```

The path resolver requires a Virtual File System path for locating the specific entity. Note that the path requires quotation marks if it contains spaces. E.g.:

```
reportserver$ locate "/reportmanager/Dynamic Lists/myReport"
Report Root/Dynamic Lists/myReport (Reportmanager)
```

You can also insert a relative path to the current location, e.g.:

```
reportserver$ locate myReport
Report Root/Dynamic Lists/myReport (Reportmanager)
```

Further, you can use an HQL query (Hibernate Query Language) for locating the needed entity or group of entities. Refer to https://docs.jboss.org/hibernate/orm/5.0/userguide/en-US/html/ch13.html for the HQL documentation. The syntax is: `hql:query` where the query is a valid select HQL query. Since HQL queries have blank spaces, quotation marks are needed. E.g.:

```
reportserver$ locate "hql:from TableReport where id=123"
Report Root/Dynamic Lists/myReport (Reportmanager)
```

You can also search for entity attributes, e.g.:

```
reportserver$ locate "hql:from TableReport where name like '%myReport%'"
Report Root/Dynamic Lists/myReport (Reportmanager)
Report Root/myReport2 (Reportmanager)
```

Following you can find a list of some important entity types, which can be used with the object resolvers described above. The complete list can be found here: https://reportserver.net/api/latest/entities.html

Report entities:

Report General report entity. Example: `locate id:Report:123`

TableReport Dynamic list. Example: `locate id:TableReport:123`

BirtReport BIRT report. Example: `locate id:BirtReport:123`

CrystalReport Crystal report. Example: `locate id:CrystalReport:123`

GridEditorReport Grid editor report. Example: `locate id:GridEditorReport:123`

JasperReport Jasper report. Example `locate id:JasperReport:123`

JxlsReport JXLS report. Example `locate id:JxlsReport:123`

SaikuReport Saiku report. Example: `locate id:SaikuReport:123`

ScriptReport Script report. Example: `locate id:ScriptReport:123`

ReportFolder Report folder. Example: `locate id:ReportFolder:123`

User entities:

User User. Example: `locate id:User:123`

Group Group. Example: `locate id:Group:123`

OrganisationalUnit Organisational unit. Example: `locate id:OrganisationalUnit:123`

Datasource entities:

DatasourceDefinition General datasource definition entity. Example: `locate id:DatasourceDefinition:123`

DatabaseDatasource Relational database. Example: `locate id:DatabaseDatasource:123`

CsvDatasource CSV datasource. Example: `locate id:CsvDatasource:123`

MondrianDatasource Mondrian datasource. Example: `locate id:MondrianDatasource:123`

BirtReportDatasourceDefinition BIRT report datasource. Example: `locate id:BirtReportDatasourceDefiniti`

ScriptDatasource Script datasource. Example: `locate id:ScriptDatasource:123`

DatabaseBundle Database bundle. Example: `locate id:DatabaseBundle:123`

DatasourceFolder Datasource folder. Example: `locate id:DatasourceFolder:123`

Datasink entities:

DatasinkDefinition General datasink definition entity. Can always be used instead of a specific datasink entity below. Example: `locate id:DatasinkDefinition:123`

AmazonS3Datasink Amazon S3 datasink. Example: `locate id:AmazonS3Datasink:123`

BoxDatasink Box datasink. Example: `locate id:BoxDatasink:123`

DropboxDatasink Dropbox datasink. Example: `locate id:DropboxDatasink:123`

EmailDatasink Email - SMTP datasink. Example: `locate id:EmailDatasink:123`

FtpDatasink FTPS datasink. Example: `locate id:FtpDatasink:123`

FtpsDatasink FTPS datasink. Example: `locate id:FtpsDatasink:123`

GoogleDriveDatasink Google Drive datasink. Example: `locate id:GoogleDriveDatasink:123`

LocalFileSystemDatasink Local filesystem datasink. Example: `locate id:LocalFileSystemDatasink:123`

OneDriveDatasink OneDrive - SharePoint (O365) datasink. Example: `locate id:OneDriveDatasink:123`

PrinterDatasink Printer datasink. Example: `locate id:PrinterDatasink:123`

SambaDatasink Samba - SMB/CIFS datasink. Example: `locate id:SambaDatasink:123`

ScpDatasink SCP datasink. Example: `locate id:ScpDatasink:123`

ScriptDatasink Script datasink. Example: `locate id:ScriptDatasink:123`

SftpDatasink SFTP datasink. Example: `locate id:SftpDatasink:123`

TableDatasink Table datasink. Example: `locate id:TableDatasink:123`

DatasinkFolder Datasink folder. Example: `locate id:DatasinkFolder:123`

Filesystem entities:

FileServerFile File. Example: `locate id:FileServerFile:123`

FileServerFolder Folder. Example: `locate id:FileServerFolder:123`

Chapter 19

Terminal Operators

In the following we describe all terminal operators currently supported.

19.1 Write-into-file operators

Analogously as in Linux/Unix, the > operator allows you to send the command results to a given file in the ReportServer virtual filesystem. The >> operator allows you to append the command results into a given file.

For example, the following would write the output of the "`columnsMetadata id:DatasourceDefinition:8005 T_AGG_CUSTOMER`" command into the "/fileserver/results.txt" file.

```
columnsMetadata id:DatasourceDefinition:8005 T_AGG_CUSTOMER > /fileserver/results.txt
```

The following would append the output of the "`columnsMetadata id:DatasourceDefinition:8005 T_AGG_CUSTOMER`" command into the "/fileserver/results.txt" file.

```
columnsMetadata id:DatasourceDefinition:8005 T_AGG_CUSTOMER >> /fileserver/results.txt
```

19.2 Write-into-datasink operator

Analogously as the write-into-file operators, the >>> operator allow you to send the command results to a given datasink. The datasink can be fetched using object resolver queries. Refer to Section 18.5 Object Resolver for more details of object resolver queries.

For example, the following would write the output of the "`columnsMetadata id:DatasourceDefinition:8005 T_AGG_CUSTOMER`" command into the datasink with id 123. Depending on the datasink type, the results will be send per Email, OneDrive - SharePoint (O365), Dropbox, etc.

```
columnsMetadata id:DatasourceDefinition:8005 T_AGG_CUSTOMER >>> id:DatasinkDefinition:123
```

```
Use: command >>> datasink
```

Chapter 20

Terminal Commands

In the following we will describe all terminal commands currently supported.

20.1 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
```

20.2 cat

Allows to display the specified file on the terminal window.

```
Use: cat file
```

20.3 cd

Allows to change the current directory.

```
Use: cd directory
```

20.4 clearInternalDbCache

Clears the cache of the internal database. This is used to optimise the performance for CSV and script datasources (refer to „datasources”).

```
Use: clearInternalDbCache
```

20.5 clearInternalScriptCache

Clears the cache of the internal scripting cache, e.g. compiled scripts.

```
Use: clearInternalScriptCache
```

20.6 columnsExist

Checks if a given column list exists in a given table.

Here, datasource is an object resolver query that returns exactly one datasource. Refer to Section 18.5 Object Resolver for more information on object resolver queries.

The following example checks if the "CUS_CITY" column exists in the "T_AGG_CUSTOMER" table of the datasource with id 123.

```
columnsExist id:DatasourceDefinition:123 T_AGG_CUSTOMER CUS_CITY
```

The following example checks if the "CUS_CITY" and "myColumn" columns exists in the "T_AGG_CUSTOMER" table of the datasource with id 123.

```
columnsExist id:DatasourceDefinition:123 T_AGG_CUSTOMER CUS_CITY myColumn
```

```
Use: columnsExist datasource table columns
```

20.7 columnsMetadata

Allows to fetch column metadata of a given table.

Here, datasource is an object resolver query that returns exactly one datasource. Refer to Section 18.5 Object Resolver for more information on object resolver queries.

The following example prints the metadata of the "T_AGG_CUSTOMER" table in the Datasource with id 123:

```
columnsMetadata id:DatasourceDefinition:123 T_AGG_CUSTOMER
```

The default metadata printed is the following:

- COLUMN_NAME
- TYPE_NAME
- COLUMN_SIZE
- DECIMAL_DIGITS
- ORDINAL_POSITION
- IS_NULLABLE
- IS_AUTOINCREMENT

Metadata documentation of the columns above can be found here: https://docs.oracle.com/en/java/javase/11/docs/api/java.sql/java/sql/DatabaseMetaData.html#getColumns(java.lang.String,java.lang.String,java.lang.String,java.lang.String).

You may append, additionally to the default columns listed above, any number of the available metadata columns by passing them as arguments. E.g. you may choose to append TABLE_SCHEM and CHAR_OCTET_LENGTH as such:

> Note that you can use the >>> operator for sending the command results to a given datasink. This may be useful for long command outputs for better result analysis. You can also use > for new file creation or >> for file append. Details of all terminal operators can be found in Chapter 19 Terminal Operators.

```
columnsMetadata id:DatasourceDefinition:123 T_AGG_CUSTOMER TABLE_SCHEM CHAR_OCTET_LENGTH
```

```
Use: columnsMetadata datasource table [column] [column...]
```

20.8 config

Configuration files are cached in order to optimize the performance. When a configuration file is changed, the cache must be emptied for loading the changes into the cache and thus activating the changes made.

```
Use: config reload
```

In order to read the current active value of a configuration parameter, you can use "config echo", e.g. for reading the default charset in the main.cf configuration file:

```
config echo main/main.cf default.charset
```

would return you e.g. "UTF-8".

For reading an attribute in the form:

```
<mailaction html="false">
```

you can write: `config echo scheduler/scheduler.cf scheduler.mailaction[@html]`.

More details on the syntax can be found in the Apache Commons Configuration documentation: https://commons.apache.org/proper/commons-configuration/userguide/quick_start.html

20.9 connPoolStats

Prints connection pool statistics. For details on all parameters and configuration check https://www.mchange.com/projects/c3p0/ and the Connection Pool's Section on the Configuration Guide.

This command prints the following information:

Datasource	Datasource name and id.
Max pool size	Maximum number of connections a pool will maintain at any given time.
Number of connections	Current total number of connections in the pool (both busy and idle).
Busy connections	Number of busy connections in the pool. These connections are already checked out from the pool.
Idle connections	Idle of busy connections in the pool. These connections can be checked out from the pool in order to be used.
Threads awaiting connection checkout	Number of threads currently waiting for a connection from the connection pool.
Unclosed orphaned connections	Number of checked out connections from the pool but not longer being managed by the connection pool.

In order to monitor the connection pool usage, two important parameters are "numBusyConnections" and "numThreadsAwaitingCheckoutDefaultUser". If the "numBusyConnections" reaches the "maxPoolSize", this means that all the connections in the connection pool have exhausted and you will see "numThreadsAwaitingCheckoutDefaultUser" increasing. This means that the number of connections in the connection pool is not enough for the current load.

20.10 copy

The superordinate `copy` command includes the following commands to copy ReportServer objects and entities.

copy parameterDefinitions

Copies all parameter definitions from an origin report to a target report. If the reports are variants, their parent base reports are used. Note that dependencies on other parameters are not copied, so these have to be copied manually.

In order to select the reports, you can use object resolver queries. Refer to Section 18.5 Object Resolver for more details on this. Note that the queries must each resolve to exactly one report.

If the `replaceExistingParameters` is `true`, the command replaces parameter definitions in the target report having the same key as in the origin report. If `false`, it ignores these.

> Note that you can also copy parameter definitions with the "copy" and "paste" context menu items of the parameter management panel.

The following example copies all parameter definitions of the origin report with ID `123` into the target report with ID `456`. It replaces parameter definitions in the target report having the same key as in the origin report.

```
copy parameterDefinitions id:Report:123 id:Report:456 true
```

```
Use: copy parameterDefinitions origin target replaceExistingParameters
```

copy tableContents

Copies the contents of a given table to another table, which may reside in another database type, so allows you to copy table contents database-independently. E.g. you may copy the contents of a table from a MSSQL database to a table in an Oracle database. All datasource types supported by ReportServer are supported.

The destination table must exist and must contain the same fields as the source table. Their field types must be compatible.

You can locate your datasources by using an object resolver query. Refer to Section 18.5 Object Resolver for more details of object resolver queries.

Refer to the following example:

```
copy tableContents id:DatabaseDatasource:58 RS_SCHEMAINFO id:DatabaseDatasource:60 ↙
    ↳   myschemainfo entity_id false 200
```

The example copies the contents of the RS_SCHEMAINFO table of your internal datasource with id 58 into a table with name `myschemainfo` in the datasource with id 60.

Again, note this is database-independent. In the example above the datasource with id 58 is a MSSQL datasource, while the datasource with id 60 is an Oracle datasource.

The command prints some information for you to be able to see what is happening in the background. In the example above the information printed is:

Status	OK
All columns	[ENTITY_ID, KEY_FIELD, VALUE]
Datatypes	[-5, 12, 12]
Indexes of primary keys	[0]
SELECT statement	SELECT ENTITY_ID,KEY_FIELD,VALUE FROM RS_SCHEMAINFO
INSERT statement	INSERT INTO myschemainfo (KEY_FIELD,VALUE) values (?,?)
Total duration	0.987 seconds

If your table contains a compound primary key, you can enter it in a ;-separated list. E.g.:

```
copy tableContents id:DatabaseDatasource:58 RS_SCHEMAINFO id:DatabaseDatasource:60 ↙
    ↳   myschemainfo myKey1;myKey2 false 200
```

As both source and destination tables should contain the same fields, the `primaryKeys` should exist in both.

The `copyPrimaryKeys` should be true if primary key values should be copied to the destination table, else false. For primary key fields in the destination table which are calculated by your database (auto-increment, etc), this should be `false` in order for the database to calculate them instead of the command copying them.

The batch size is the packet size in the batch-insert. This is optional. Default is `100`.

```
Use: copyTableContents sourceDatasource sourceTable destinationDatasource destinationTable ↙
 ↳ primaryKeys copyPrimaryKeys [batchSize]
```

20.11 cp

Enables to copy one or more files to a new folder. The "-r" flag marks the process as being recursive. In this case subfolders will also be copied.

```
Use: cp [-r] sourcefiles targetfolder
```

Note that report variants can also be copied into another report. By using wildcards (e. g. prefix*) you can copy several objects.

20.12 createTextFile

Creates a new text file in the fileserver file system and opens a window for editing the new file.

```
Use: createTextFile file
```

20.13 datasourceMetadata

Allows to dynamically call any method from the `DatabaseMetaData` interface found here: https://docs.oracle.com/en/java/javase/11/docs/api/java.sql/java/sql/DatabaseMetaData.html.

The call fails if argument count and name does not match exactly one method of said interface. The call fails as well if the args cannot be converted into the needed parameter types. `null` may be passed as a String in the args if necessary: it will be evaluated to a `null` object. As you will see it is a very powerful and versatile tool at your disposal.

All results will be displayed as a table if their return type is a `ResultSet`.

Examples:

```
datasourceMetadata id:DatasourceDefinition:123 getDriverMajorVersion
```

```
datasourceMetadata id:DatasourceDefinition:123 getDriverName
```

```
datasourceMetadata id:DatasourceDefinition:123 getDatabaseMajorVersion
```

```
datasourceMetadata id:DatasourceDefinition:123 getColumns null null T_AGG_CUSTOMER ↙
 ↳ null
```

In the last example we call the following method `https://docs.oracle.com/en/java/javase/11/docs/api/java.sql/java/sql/DatabaseMetaData.html#getColumns(java.lang.String,java.lang.String,java.lang.String,java.lang.String)` and choose to supply the table name to identify which column's metadata will be fetched.

So we pass the method name: `getColumns` and four parameters: `null null T_AGG_CUSTOMER null`. As mentioned above, `null` evaluates to the `null` object which means we call the method `getColumns(null, null, ‘‘T_AGG_CUSTOMER’’, null)`.

> Note that you can use the >>> operator for sending the command results to a given datasink. This may be useful for long command outputs for better result analysis. You can also use > for new file creation or >> for file append. Details of all terminal operators can be found in Chapter 19 Terminal Operators.

```
Use: datasourceMetadata datasource methodName [arg] [arg ...]
```

20.14 deployReport

Allows to analyze a deployment attempt of a given report (left report) into an destination report (right report). Both reports have to exist already in ReportServer.

analyze   Create and download a document containing deployment analysis. This analysis lists conflicts (including context) that would occur during a deployment attempt of the left report into the right report.

Note that if an entry does not cause a conflict, e.g. if the corresponding column is not used in any variant, the entry is not listed in the analysis result.

The `-i` option can be used to ignore case-sensitivity of field names.

In order to select the reports, you can use object resolver queries. Refer to Section 18.5 Object Resolver for more details on this. Note that the queries must resolve to exactly one basic report.

Example:

```
deployReport analyze id:Report:123 id:Report:456
```

A PDF containing the analysis of deploying Report with id 123 (left report) into Report with id 456 (right report) is created and downloaded automatically.

The current sections in the analysis are:

- Columns contained in left report but not in right report
- Columns contained in both reports but which different definitions
- Variants of right report using columns not available in left report
- Variants of right report using columns with different definitions as in left report

```
Use: analyze [-i] leftReport rightReport
```

20.15 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]
```

20.16 diffconfigfiles

Default configuration files are created on first run of ReportServer. Later, when upgrading ReportServer to a newer version, it is probable that newly added configuration files will be missing (i.e. all configuration files added between the version originally installed and the version upgraded to). This command helps you to find out which configuration files are missing without having to search all release notes between these versions. Default config files can also be created with help of this command.

showmissing	allows you to compare the current set of configuration files with the expected set of configuration files of the currently installed version. Then it lists all missing files.
createmissing	allows you to create the default missing configuration files found with "showmissing" into the appropriate location inside your /fileserver/etc path.
createall	copies all default missing configuration files into a given folder. This allows you to compare configuration file contents/fields, etc.

```
Use: diffconfigfiles (showmissing | createmissing | createall folder)
```

20.17 dirmod

Enables to modify fileserver directories. The following subcommand is available:

webaccess	Modify the web access property of a given fileserver directory.

The syntax for modifying a directory's web access is:

```
Use: dirmod webaccess directory access
```

Here, directory is an object resolver query that returns one or more `FileServerFolders`. Refer to Section 18.5 Object Resolver for more information on object resolver queries. `access` may be `true` or `false`.

Usage examples:

Removes web access from `fileserver/resources/` directory:

```
dirmod webaccess fileserver/resources/ false
```

Removes web access from directory with id 123:

```
dirmod webaccess id:FileServerFolder:123 false
```

Adds web access to all filesystem directories containing "report" in their name:

```
dirmod webaccess "hql:from FileServerFolder where name like '%report%'" true
```

20.18 echo

Outputs the given text on the console.

```
Use: echo hello world
```

20.19 editTextFile

Opens an existing text file in the fileserver file system for editing.

```
Use: editTextFile file
```

20.20 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
```

20.21 entitymerge

This command merges two entities transfering values from one to the other (with the exception of the key and id). No entity is deleted after merging. In general, only entities of the same type can be merged.

```
Use: entitymerge target values
```

The following example transfers all values of the origin report `originReport` into the target report `targetReport`.

```
entitymerge "/reportmanager/Dynamic Lists/originReport" "/reportmanager/Dynamic Lists/targetReport"
```

20.22 env

Prints environment information of the current installation including relevant environment variables.

`Use: env`

20.23 exec

Allows to execute scripts from the fileserver's file system.

By entering the `-c` flag (for `commit`) you control whether changes to the script persist in the database.

With the `-d` (for `silence`) you can suppress all the script's output.

The `-t` flag allows you to output the complete stacktrace in case of an exception.

By entering the `-n` (for `non-monitored mode`) flag you prevent the script to run in an own monitored thread. With other words, it runs the script in the server thread instead of in its own thread.

Finally, the `-w` flag displays the script output in a new window.

For further information on scripts refer to the Script Guide.

`Use: exec [-c] [-s] [-t] [-n] [-w] script`

20.24 export all

Allows to export all metadata to a file. This file can be then imported by the command "import all". Usually you should directly zip and save the export data in the File System to save memory, as it will be filed in an XML dialect.

For exporting and zipping the export file, use `export all | zip > myExportFile.zip`. You will get a zipped "data" file, which can be renamed to "data.xml" before importing again.

If you don't need to zip the file, you can use `export all > myExportFile.xml`.

Note that the export file has to be created inside the internal file system: `cd fileserver` and the "import all" command needs an unzipped XML file.

`Use: export all > myExportFile.xml`

20.25 groupmod

Enables to change groups. The following subcommands are available:

addmembers   Add or remove members to a group. These can be users, OUs, or another groups.

The syntax for adding/removing members to/from a group is

```
groupmod addmembers [-c] group [members] [members...]
```

Here, group is an object resolver query that returns exactly one group. Refer to Section 18.5 Object Resolver for more information on object resolver queries. If the optional parameter -c is given, the given members are removed instead of added. If no members are given, all members are being deleted from the group member list. Finally, the members list parameters refer to one or more object resolver queries that return a user, a group, or an OU. Usage examples:

Deletes all members from the group with id 123:

```
groupmod addmembers -c id:Group:123
```

Deletes one user from the group with id 123:

```
groupmod addmembers -c id:Group:123 id:User:456
```

Adds three members (one user, one group, one OU) to the group with id 123:

```
groupmod addmembers id:Group:123 id:User:456 "hql:from Group where id=789" id:OrganisationalUnit
    :987
```

20.26 haspermission

Allows to check if a given user has a given permission on a given target. Returns `true` if the user has the permission, else `false`.

The `-g` flag allows to check generic permissions. Documentation of these including the exact target types you can enter can be found in Section 3.2 Permission Management.

For other objects, e.g. Users, Datasources, etc., you can check the entity types here: `https://reportserver.net/api/latest/entities.html`.

All objects can be fetched using object resolver queries. Refer to Section 18.5 Object Resolver for more details of object resolver queries.

Valid permissions are:

- Read
- Write
- Execute
- Delete
- GrantAccess
- TeamSpaceAdministrator

The following example checks if the user with id `123` has `Execute` permission on the `AccessRsSecurityTarget` generic target, i.e. if the user is allowed to log in into ReportServer.

```
reportserver$ haspermission -g id:User:123 net.datenwerke.rs.core.service.
    genrights.access.AccessRsSecurityTarget Execute
reportserver$ true
```

The following example checks if the user with id `123` has `Read` permission on the datasource with id `456`.

```
reportserver$ haspermission id:User:123 id:DatasourceDefinition:456 Read
reportserver$ false
```

Use: `haspermission [-g] user target right`

20.27 hello

Says hello

Use: `hello`

20.28 hql

Executes HQL (Hibernate Query Language) database queries and displays the results. HQL is used to write database-independent queries. More information here: https://docs.jboss.org/hibernate/orm/5.6/userguide/html_single/Hibernate_User_Guide.html#hql and here: https://docs.jboss.org/hibernate/core/3.3/reference/en/html/queryhql.html.

Note you can find all entities in ReportServer here: https://reportserver.net/api/latest/entities.html and all javadocs here: https://reportserver.net/api/latest/javadoc/index.html.

The results can be displayed in a new window with the `-w` flag.

Example uses are shown next.

List all reports:

```
reportserver$ hql "from Report"
```

List all users:

```
reportserver$ hql "from User"
```

List all dynamic lists with name like 'MyReport':

```
reportserver$ hql "from TableReport t where t.name like '%MyReport%'
```

List all report properties from report with id 123:

```
reportserver$ hql "select r.reportProperties from Report r where r.id = 123"
```

Use: `hql [-w] query`

20.29 id

Allows to print information of a given username.

This includes group information and organizational unit information of a user.

Group information includes direct and indirect groups (via another groups or via organizational units).

`Use: id username`

20.30 info

Displays information about ReportServer objects. The following subcommands are available:

info datasource

Displays general information of a given datasource. For relational databases, displays additional metadata information.

You can use an object resolver query to locate the specific datasource. Refer to Section 18.5 Object Resolver for more details of object resolver queries.

Example for displaying information of the datasource with id 123:

```
info datasource id:DatasourceDefinition:123
```

`Use: info datasource datasource`

20.31 import all

Allows to import an export file which was created with "export all".

`Use: import all export.xml`

20.32 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`

20.33 ldapfilter

Allows you to analyze the installed LDAP filter in the `sso/ldap.cf` configuration file. The LDAP filter is parsed and shown it in a multi-line form that makes it easier to understand its hierarchy and embedded components. The command also tries to simplify the LDAP filter in certain ways (for example, by removing unnecessary levels of hierarchy, like an AND embedded in another AND).

The optional `i` (indentation) flag is used to indicate the number of spaces for indentation. Default is 2. The optional `n` (no-simplification) flag indicates that no simplification should be done, i.e., the filter should not be futher analyzed. Note that you have to reload your configuration changes with `config reload` or restart your ReportServer when you change your filter in the `sso/ldap.cf` configuration file.

As the output of this terminal command is usually long, you can use the >>> operator for sending the output to a given datasink: `ldapfilter >>> id:DatasinkDefinition:123`.

```
Use: ldapfilter [-i] [-n]
```

20.34 ldapguid

Makes a best-effort guess of the appropriate GUID needed for your specific LDAP server. The GUID is needed in the in the LDAP configuration file: `sso/ldap.cf`.

```
Use: ldapguid
```

20.35 ldapimport

Imports LDAP users, groups and organizational units as configured in sso/ldap.cf. Configuration options are described in the Configuration Guide.

For scheduling the functionality periodically, you can use the script available here: `https://github.com/infofabrik/reportserver-samples/blob/main/src/net/datenwerke/rs/samples/admin/ldap/ldapimport.groovy` and schedule it via “scheduleScript”.

```
Use: ldapimport
```

20.36 ldapinfo

Displays some information about your installed LDAP server in the LDAP configuration file: `sso/ldap.cf`.

```
Use: ldapinfo
```

20.37 ldapschema

The superordinate `ldapschema` command includes the following commands which allow you to browse and analyze your LDAP schema. This may be useful for finding out the values needed in the

LDAP configuration file: `sso/ldap.cf`.

> Note that you can use the >>> operator for sending the command results to a given datasink. This may be useful for long command outputs for better result analysis. You can also use > for new file creation or >> for file append. Details of all terminal operators can be found in Chapter 19 Terminal Operators.
>
> For example, in order to send a list of all your LDAP attributes to a datasink with ID 123, you can enter the following command: `ldapschema attributeList >>> id:DatasinkDefinition:123`

ldapschema attributeInfo

Displays schema information of a given attribute. This information includes the attribute's OID, all its names, description, superior and subattributes, syntax, matching rules, usage, among others. You can use the `ldapschema attributeList` for listing all available attributes, which you can further analyze with `ldapschema attributeInfo`.

`Use: ldapschema attributeInfo attribute`

ldapschema attributeList

Displays a list of all attributes found in your LDAP server.

`Use: ldapschema attributeList`

ldapschema entry

Displays a text representation of the complete LDAP schema entry. As this output is usually a long output, you can use the >>> operator for sending the output to a given datasink as noted above: `ldapschema entry >>> id:DatasinkDefinition:123`.

`Use: ldapschema entry`

ldapschema matchingRuleInfo

Displays schema information of a given matching rule. This information includes the matching rule's OID, all its names, description, usage, among others. You can use the `ldapschema matchingRuleList` for listing all available matching rules, which you can further analyze with `ldapschema matchingRuleInfo`.

`Use: ldapschema matchingRuleInfo matchingRule`

ldapschema matchingRuleList

Displays a list of all matching rules found in your LDAP server.

`Use: ldapschema matchingRuleList`

ldapschema objectClassInfo

Displays schema information of a given object class. This information includes the object class OID, all its names, description, super and subclasses, required and optional attributes, among others. You can use the `ldapschema objectClassList` for listing all available object classes, which you can further analyze with `ldapschema objectClassInfo`.

```
Use: ldapschema objectClassInfo objectClass
```

ldapschema objectClassList

Displays a list of all object classes found in your LDAP server.

```
Use: ldapschema objectClassList
```

ldapschema syntaxRuleInfo

Displays schema information of a given syntax rule. This information includes the syntax rule's OID, description, usage, among others. Note that different as the rest of the `ldapschema` subcommands, the OID is required for the `syntaxRuleInfo` subcommand. You can use the `ldapschema syntaxRuleList` for listing all available syntax rules together with their OIDs.

```
Use: ldapschema syntaxRuleInfo syntaxRule
```

ldapschema syntaxRuleList

Displays a list of all syntax rules found in your LDAP server.

```
Use: ldapschema syntaxRuleList
```

20.38 ldaptest

Tests LDAP filter, GUID, users, groups and organizational units as configured in sso/ldap.cf. Configuration options are described in the Configuration Guide.

> Note that you can use the >>> operator for sending the command results to a given datasink. This may be useful for long command outputs for better result analysis. You can also use > for new file creation or >> for file append. Details of all terminal operators can be found in Chapter 19 Terminal Operators.

When troubleshooting your LDAP configuration, you should run the commands shown next in the order shown below, as some of them are based on correct configuration. E.g. `ldaptest users` needs a correct filter installed, so `ldaptest filter` should be checked first.

```
ldaptest filter
ldaptest guid
ldaptest groups
ldaptest organizationalUnits
ldaptest users
ldaptest orphans
```

ldaptest filter

Allows you to test the installed filter and prints the results.

If the `-a` flag is entered, requests and displays additional LDAP attributes. These must be separated by semicolon (`;`).

E.g., in order to display the `mail`, `member` and `ou` attribute values of each entry, you can enter the following:

```
ldaptest filter -a mail;member;ou
```

```
Use: ldaptest filter [-a]
```

ldaptest guid

Allows you to test the installed GUID and prints the results.

Additionally this command also scans your usertree for duplicate uses of the same GUID. If duplicates are detected you resolve them before importing users using ldapimport.

```
Use: ldaptest guid
```

ldaptest groups

Allows you to show the LDAP groups together with their attributes (in the sso/ldap.cf configuration file) that would be imported in an `ldapimport` execution.

If the `-s` (schema) flag is entered, the schema of the groups' object class is shown. This may be useful for finding out other group properties that can be entered into the `ldap.cf` configuration file. You can also use the `ldapschema` command for further exploring your object class attributes (refer to 20.37 ldapschema).

If the `-a` flag is entered, requests and displays additional LDAP attributes. These must be separated by semicolon (`;`).

E.g., in order to display the `instanceType` and `groupType` attribute values of each group, you can enter the following:

```
ldaptest groups -a instanceType;groupType
```

`Use: ldaptest groups [-s] [-a]`

ldaptest organizationalUnits

Allows you to show the LDAP organizational units together with their attributes (in the sso/ldap.cf configuration file) that would be imported in an `ldapimport` execution.

If the `-s` (schema) flag is entered, the schema of the organizational units' object class is shown. This may be useful for finding out other organizational unit properties that can be entered into the `ldap.cf` configuration file. You can also use the `ldapschema` command for further exploring your object class attributes (refer to 20.37 ldapschema).

If the `-a` flag is entered, requests and displays additional LDAP attributes. These must be separated by semicolon (`;`).

E.g., in order to display the `distinguishedName` and `commonName` attribute values of each group, you can enter the following:

```
ldaptest groups -a distinguishedName;commonName
```

`Use: ldaptest organizationalUnits [-s] [-a]`

ldaptest users

Allows you to show the LDAP users together with their attributes (in the sso/ldap.cf configuration file) that would be imported in an `ldapimport` execution.

If the `-s` (schema) flag is entered, the schema of the users' object class is shown. This may be useful for finding out other user properties that can be entered into the `ldap.cf` configuration file. You can also use the `ldapschema` command for further exploring your object class attributes (refer to 20.37 ldapschema).

If the `-a` flag is entered, requests and displays additional LDAP attributes. These must be separated by semicolon (`;`).

E.g., in order to display the `memberOf` and `nickname` attribute values of each user, you can enter the following:

```
ldaptest user -a memberOf;nickname
```

> When retrieving information about LDAP users, internal users are checked for possible username collisions when importing. If detected you will notice an additional `conflicts with` column which holds the id of the user with the same username. Resolve these conflicts bevor importing.

`Use: ldaptest users [-s] [-a]`

ldaptest orphans

Your LDAP filter should exclusively retrieve all users, groups, and organizational units. If additional nodes are returned, or if the mappings in ldap.cf are incorrect, nodes may be retrieved that cannot be mapped to a user, group, or organizational unit. These are referred to as LDAP orphans. In a properly installed and configured system, LDAP orphans should not exist. Therefore, LDAP orphans occur when your LDAP filter returns too much data. You can easily list all LDAP orphans using this terminal command.

If the `-a` flag is entered, requests and displays additional LDAP attributes. These must be separated by semicolon (`;`).

```
Use: ldaptest orphans [-a]
```

20.39 listlogfiles

Displays a list of the log files in the `catalina.home` path. If you need to explicitly set the log file path, you can use the `logdir` setting in the `main.cf` configuration file.

You can specify the sorting column(s) by a semicolon-separated list of column numbers in the `-s` option. Allowed are values `1`, `2`, and `3` for the first (filename), second (last modified) and third (size) columns, respectively. If you need to sort a given column in descending order, you can enter a `-` prefix in front of the column's index. Default sorting is by filename (ascending order).

Further, you can use Java regular expressions (https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/regex/Pattern.html) for filtering files.

The example below lists all log files starting with "reportserver" and sorting them by size in descending order.

```
listlogfiles -s -3 -f "reportserver.*"
```

The example below lists all log files starting with "reportserver" and sorting them by size in descending order, followed by filename in ascending order.

```
listlogfiles -s -3;1 -f "reportserver.*"
```

Further, you can use the `-e` option if you want to send the (filtered) log files via e-mail. For example, the following allows you to ZIP and send all log files starting with "reportserver" to the current user via e-mail.

```
listlogfiles -s -3;1 -f "reportserver.*" -e
```

If you need to send the (filtered) log files to any datasink, you can use the `-d` option for this. You can use any object resolver query to locate the specific datasink. Refer to Section 18.5 Object Resolver for more details of object resolver queries. For example, the following allows you to ZIP and send all log files starting with "reportserver" to the datasink with id `123`.

```
listlogfiles -s -3;1 -f "reportserver.*" -d id:DatasinkDefinition:123
```

Note that you can display the last n lines of a given log file with the `viewlogfile` command described in Section 20.69 viewlogfile.

`Use: listlogfiles [-s] [-f] [-e] [-d]`

20.40 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`

20.41 locate

The locate command searches for objects matching a given expression / object resolver query. The optional argument `-t` allows to filter the results to a given object type.

Note that, in order to find an object, your user has to have at least read permissions on that object and module read permissions on the management module where this object resides. E.g.: to find a specific report, the user has to have at least read permissions on the report and read permissions on the report management module. Check Chapter 3 User and Permission Management for more information on permissions.

Expressions can be an object id or object name / key. The object name / key may contain wildcards.

E.g.:	locate 123	Locates the object with id 123.
	locate my*Report	Locates all objects matching the "my*Report" expression. E.g. it matches both "myJasperReport" and "myTableReport".
	locate -t TableReport my*Report	Locates all objects matching the "my*Report" expression of type "TableReport". I.e. it filters the results to return only TableReports.

You can further use an object resolver query to locate the specific entity or group of entities. Refer to Section 18.5 Object Resolver for more details of object resolver queries. Note that when you use an object resolver query, the `-t` option is ignored. Quotation marks are needed when the object resolver query contains spaces. E.g.:

```
reportserver$ locate "hql:from TableReport where id=123"
Report Root/Dynamic Lists/myReport (Reportmanager)
```

`Use: locate [-t] expression`

20.42 ls

Displaying files in the respective folder By entering `ls -l` additional information per file will be displayed.

`Use: ls [-l] path`

20.43 meminfo

Displays the storage utilization of ReportServer.

Used Memory	your current memory usage
Free Memory	the amount of free memory in the Java Virtual Machine
Total Memory	the total amount of memory in the Java Virtual Machine. The value returned by this method may vary over time, depending on the host environment.
Max Memory	the maximum amount of memory that the Java virtual machine will attempt to use.

`Use: meminfo`

20.44 mkdir

Enables to create a new folder.

`Use: mkdir folder name`

20.45 mv

Enables to move files to a target folder. By using wildcards (e. g. prefix*) you can move several objects.

Note that report variants can also be moved into another report.

`Use: mv source file target folder`

20.46 onedrive

Provides easy access to the Microsoft graph API. `onedrive` is a three-tiered command. Subcommands loosely group subsubcommands depending on which kind of onedrive-object those commands deal with. Subsubcommands can be of simple or more complex nature depending on their purpose. All commands use a OneDrive - SharePoint (O365) datasink for configuration and require certain permissions to use the graph API. Should permissions granted by the accesstoken of the onedrive datasink be insufficient you can supply an optional accesstoken which will be used instead of the default one.

Here, datasink is an an object resolver query that returns exactly one onedrive datasink object. Refer to Section 18.5 Object Resolver for more information on object resolver queries.

`group getmygroups`	fetches and displays all OneDrive groups you belong to. This requires the permission `Group.Read.All`
`group getdrivesof`	fetches and displays information of all available drive objects of a given group. This requires the permission `Sites.Read.All`

`Use: onedrive group getmygroups datasink [accesstoken]`

Use: `onedrive group getdrivesof groupid datasink [accesstoken]`

20.47 pkg

Command to install ReportServer packages.

pkg install

Installs a given package. Use flag `-d` to install package from local file system. Otherwise, installs package from ReportServer virtual file system.

Use: `pkg install [-d] packageName`

pkg list

Lists all available packages in the local filesystem.

Use: `pkg list`

20.48 properties

Enables to observe and modify the properties map. The following subcommands are available.

properties clear

Removes all entries from the properties.

Use: `properties clear`

properties contains

Returns true if the properties map contains a mapping for the specified key.

Use: `properties contains key`

properties list

Lists all entries in the properties map.

Use: `properties list`

properties put

Adds a new entry to the properties map. If the key specified already exists in the map, it modifies the corresponding entry.

Use: `properties put key value`

properties remove

Removes a specific entry from the properties map.

`Use: properties remove key`

20.49 ps

Displays scripts which are currently running (and monitored). By entering the kill command you can interrupt executions.

`Use: ps`

20.50 pwd

Displays the current path. By entering the `-i` flag, the path will be given in ID presentation.

`Use: pwd [-i]`

20.51 rcondition

`rcondition` determines dynamic lists as a basis for conditional scheduling (refer also to Chapter 16 Scheduling of Reports). The following subcommands are available:

create Creates a new condition.

The syntax for creating a new condition is

```
rcondition create conditionReport name [key] [description]
```

Here, conditionReport can be either the id or an object resolver query that returns exactly one dynamic report variant. Refer to Section 18.5 Object Resolver for more information on object resolver queries. Name and description identify the same, while key is unique to the condition. Please observe to set enclosing quotation marks if spaces are included in object resolver queries, names, keys and descriptions, e.g. "This is a description".

list Displays a list of the reports marked as conditions.

remove Allows to remove a condition.

`Use: rcondition remove condition`

Here, `condition` can be either the id or an object resolver query of the condition to remove. E.g.: `rcondition remove id:ReportCondition:123`

20.52 registry

Enables to observe and modify the registry. The following subcommands are available.

registry clear

Removes all entries from the registry.

```
Use: registry clear
```

registry contains

Returns true if the registry contains a mapping for the specified key.

```
Use: registry contains key
```

registry list

Lists all entries in the registry.

```
Use: registry list
```

registry put

Adds a new entry to the registry. If the key specified already exists in the registry, it modifies the corresponding entry.

```
Use: registry put key value
```

registry remove

Removes a specific entry from the registry.

```
Use: registry remove key
```

20.53 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.

20.54 rev

By entering the rev command you will have access to saved object versions.The list subcommand 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

20.55 rm

Enables to delete files/objects. To recursively delete folders (which are not empty), the `-r` has to be added.

Note that report variants can also be removed. By using wildcards (e. g. prefix*) you can remove several objects.

`Use: rm [-r] object`

20.56 rpull

Allows to pull entities from a remote RS server into a local RS installation, e.g. from PROD to TEST.

You can always use the terminal for finding out the exact path of the node you want to import. Just navigate to the node you want to export in the remote RS and enter the `pwd` command as described in Section 20.50.

rpull copy

Allows to copy the fetched entities from a remote RS server into a local RS, e.g. from PROD to TEST. The arguments required are the following:

remoteServer	Object resolver query of the remote server, e.g. `/remoteservers/PROD`. Refer to Section 18.5 Object Resolver for more information on object resolver queries and to Section 15 Remote RS Servers for more information on remote RS servers.
remoteEntityPath	The entity path in the remote RS server, e.g. `/usermanager/ClassicModelCars`. This path can be a directory or a single entity.
localTarget	The target in the local RS where the remote entities should be copied to, e.g. `/usermanager/import`
-c	optional flag to indicate that a check has to be performed instead of the real import. It displays all problems found, i.e. it allows to pre-check the real entity import and look for potential errors.
-v	optional flag to indicate that report variants should also be imported from the remote RS server. While importing other objects other than reports, this flag is ignored.

You can always use the terminal for finding out the exact path of the node you want to import. Just navigate to the node you want to export in the remote RS and enter the `pwd` command as described in Section 20.50.

In order to be able to import remote entities, all keys must be set in the remote entities.

In order to be able to import remote reports, their datasources must be able to be mapped to local datasources. This mapping is defined via the /etc/main/rssync.cf configuration file. Details can be found in the Configuration Guide.

The example below imports users from the remote RS REST server named PROD. The entities are being imported from the remote location /usermanager/ClassicModelCars into the locale location /usermanager/import.

```
rpull copy /remoteservers/PROD /usermanager/ClassicModelCars /usermanager/import
```

The example below imports reports, including variants, from the remote RS REST server named PROD. The entities are being imported from the remote location /reportmanager/myreports into the locale location /reportmanager/import.

```
rpull copy -v /remoteservers/PROD /reportmanager/myreports /reportmanager/import
```

The example below performs a check without performing the real import of the example above:

```
rpull copy -c -v /remoteservers/PROD /reportmanager/myreports /reportmanager/import
```

The example below imports files from the remote RS REST server named PROD. The entities are being imported from the remote location /fileserver/myfiles into the locale location /fileserver/import.

```
rpull copy /remoteservers/PROD /usermanager/ClassicModelCars /usermanager/import
```

```
Use: rpull copy [-c] [-v] remoteServer remoteEntityPath localTarget
```

20.57 scheduleScript

Enables to execute timer controlled scripts. "scheduleScript list" delivers a list showing the currently scheduled scripts. scheduleScript execute allows to enter further dispositions.

Note that the command scheduler list shows all scheduler jobs, and scheduler remove jobid allows you to remove current jobs. Refer to the scheduler documentation in Section 20.58 for more details on this command.

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.groovy " " today at 15:23
```

```
scheduleScript execute myScript.groovy " " every day at 15:23
scheduleScript execute myScript.groovy " " at 23.08.2012 15:23
scheduleScript execute myScript.groovy " " every workday at 15:23 starting on 15.03.2011
   for 10 times
scheduleScript execute myScript.groovy " " every hour at 23 for 10 times
scheduleScript execute myScript.groovy " " today between 16:00 and 23:00 every 10 minutes
scheduleScript execute myScript.groovy " " every week on monday and wednesday at 23:12
   starting on 27.09.2011 until 28.11.2012
scheduleScript execute myScript.groovy " " every month on day 2 at 12:12 starting on
   27.09.2011 11:25 for 2 times
```

20.58 scheduler

The superordinate `scheduler` command 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.

```
Use: scheduler daemon [start, stop, restart, enable, disable, status, wdstatus, wdshutdown,
  wdstart, wdrestart]
```

list Lists jobid, type and nextFireTime.

```
Use: scheduler list
```

listFireTimes Lists the upcoming fire times for a given `jobid` for the next `numberofFireTimes`. If `numberofFireTimes` is not specified default is 10.

```
Use: scheduler listFireTimes jobid numberofFireTimes
```

remove Deletes a job with given `jobid` from the dispositions.

```
Use: scheduler remove jobid
```

replaceUser Replaces an old user with a new user in all owners, executors, scheduled-by and recipients of all active scheduler jobs. The old and new users can be addressed with an object resolver query. Refer to Section 18.5 Object Resolver for more information on object resolver queries.

The following example replaces the old user with ID 123 with the new user with ID 456.

```
scheduler replaceUser id:User:123 id:User:456
```

```
Use: scheduler replaceUser oldUser newUser
```

unschedule Cancels a job with given `jobid` from the dispositions.

```
scheduler unschedule jobid
```

20.59 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.

Here, datasource is an object resolver query that returns exactly one datasource of type `DatabaseDatasource`. Refer to Section 18.5 Object Resolver for more information on object resolver queries.

Example

```
reportserver$ cd "/datasources/internal datasources/"
reportserver$ sql "ReportServer Data Source"
> SELECT COUNT(*) FROM RS_AUDIT_LOG_ENTRY
COUNT(*)
27783
> bye
Good Bye
```

```
Use: sql datasource
```

20.60 tableExists

Checks if a given table exists in a given datasource. Here, datasource is an object resolver query that returns exactly one datasource. Refer to Section 18.5 Object Resolver for more information on object resolver queries.

The following example checks if the “T_AGG_CUSTOMER” table exists in the datasource with id 123.

```
tableExists id:DatasourceDefinition:123 T_AGG_CUSTOMER
```

```
Use: tableExists datasource table
```

20.61 transport

Provides methods to manage transport objects.

transport create

Allows to create a transport object into the target location specified. The arguments required are the following:

target	The target directory location where the transport should be saved into, e.g. `/transports/mydir`. Refer to Section 18.5 Object Resolver for more information on object resolver queries.
description	A description of the transport to be created.

```
Use: transport create target description
```

20.62 ssltest

Allows you to test your SSL configuration. For example, the following allows you to test a HTTPS connection to `www.google.com`:

```
ssltest www.google.com 443
```

In case you installed a server's certificate, for example for LDAPS or LDAP StartTLS, this command is useful for testing the installed certificate analogously as shown below:

```
ssltest ipOrHostOfYourServer 10389
```

```
Use: ssltest host port
```

20.63 teamspacemod

Enables to change TeamSpaces. The following subcommands are available:

addmembers Add or removes members to or from a TeamSpace. These can be either users or groups.

The syntax for adding/removing members to/from a TeamSpace is

```
teamspacemod addmembers [-c] teamspace [members] [members...]
```

Here, teamspace is an object resolver query that returns exactly one TeamSpace. Refer to Section 18.5 Object Resolver for more information on object resolver queries. If the optional parameter -c is given, the given members are removed instead of added. If no members are given, all members are being deleted from the TeamSpace member list. Finally, the members list parameters refer to one or more object resolver queries that return a user or a group. All these users and groups are being added to the given TeamSpace as guests. Usage examples:

Deletes all members from the TeamSpace with id 123:

```
teamspacemod addmembers -c id:TeamSpace:123
```

Deletes one member from the TeamSpace with id 123:

```
teamspacemod addmembers -c id:TeamSpace:123 id:User:456
```

Adds three members (two users and one group) to the TeamSpace with id 123:

```
teamspacemod addmembers id:TeamSpace:123 id:User:456 "hql:from Group where id=789" ↙
 ↳ "/usermanager/myOU/myUser"
```

setrole Change a users role in the TeamSpace

20.64 unzip

Enables to unpack files presented in ZIP format.

```
Use: unzip file
```

20.65 updateAlias

Reloads the alias configuration. Refer also to section „Terminal".

Use: `updateAlias`

20.66 updatedb

Updates the search index. This may take some minutes depending on the data volume.

Use: `updatedb`

20.67 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`

20.68 variantTest

Allows to create a PDF document containing a test analysis of an execution of a given variant or a given report. In case of errors, it shows the error details in the result.

For base reports, the report query is shown in the test results if the user has "Administration" and "Report management" generic permissions.

For reports using database bundles, the command also allow to specify which datasources should be tested. All datasources given in the command must be valid for the given report, i.e. it is not allowed to test a variant with a datasource not used in the corresponding report.

In order to select the reports and the datasources, you can use object resolver queries. Refer to Section 18.5 Object Resolver for more details on this. Note that the report queries must resolve to exactly one `Report` entity and the datasource queries to one or more `DatasourceDefinition`s.

Examples:

```
variantTest id:Report:123
```

The report or variant with ID 123 is being tested.

```
variantTest id:DatasourceDefinition:456 id:DatasourceDefinition:789 id:Report:123
```

The report or variant with ID 123 is being tested with datasources 456 and 789. Both datasources must be part of the datasource bundle the report uses.

```
variantTest "hql:from DatasourceDefinition where name='PROD'" id:Report:123
```

The report or variant with ID 123 is being tested with the datasource named `PROD`. This datasource must be part of the datasource bundle the report uses.

Use: variantTest [datasource] [datasource...] report

20.69 viewlogfile

Displays the last n lines of a given log file in the `catalina.home` path. If you need to explicitly set the log file path, you can use the `logdir` setting in the `main.cf` configuration file.

The example below shows the "reportserver.log" file.

```
viewlogfile reportserver.log
```

Note that you can list, filter and send via e-mail or any datasink the complete log files using the `listlogfiles` command described in Section 20.39 listlogfiles.

Use: viewlogfile logFilename

20.70 xslt

Enables to perform an XSL transformation. Here, stylesheet input and output are FileServer files.

Example: `xslt T_AGG_EMPLOYEE.xsl T_AGG_EMPLOYEE_input.html T_AGG_EMPLOYEE_result.xml`

You can find the result and the example files here: https://github.com/infofabrik/reportserver-samples/tree/main/src/net/datenwerke/rs/samples/templates/xslt.

Use: xslt stylesheet input output

20.71 zip

Enables to compress and pack files into a zip archive. The input list is a space-separated list of files or/and directories.

Example:

```
zip myfile.zip 1.groovy 2.groovy etc 3.groovy
```

Zips three files and one directory into myfile.zip.

Use: zip outputFile.zip inputList

Chapter 21

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 datasources, 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 `https://groovy-lang.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.groovy</login>
      <rs>fileserver/bin/onstartup.groovy</rs>
    </startup>
    <cache>
      <maxsize>10000</maxsize>
      <expiresafter>30</expiresafter> <!-- in minutes -->
```

```
      </cache>
  </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 next section (`startup`) allows 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.

The `cache` section controls the internal script caching mechanism. This is used for performance reasons, e.g. for not recompiling the same script twice when used frequently. The `maxsize` setting specifies the maximum number of entries the cache may contain. The `expiresafter` specifies that each entry should be automatically removed from the cache once the given fixen duration has elapsed after the entry's creation, or the most recent replacement of its value.

> Note that you can manually clear script cache using the `clearInternalScriptCache` terminal command.

21.1 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.groovy`. The extension `.groovy` has no relevance here, although, it has established as a standard for scripts (besides `.groovy`, `.rs` is frequently used):

```
reportserver$ createTextFile hello1.groovy
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.groovy
Hello World
```

Note: Groovy scripts can also be executed directly from the fileserver tree via the context menu. This feature supports two execution modes:

- Simple Execution (`exec`): Runs the script without saving changes.
- Execution with Commit (`exec -c`): Runs the script and persists any changes made.

This option is available only for files with the content type `text/groovy` and the `.groovy` file extension.

21.2 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.groovy`) you can redirect the script output (or the error output) to a separate window.

21.3 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.groovy
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.groovy
No argument stated
reportserver$ exec searchReportByQuery.groovy 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 (datasource, 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 datasource can be addressed via the field **datasourceContainer.datasource**, and the relational datasources 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 datasource 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}
```

21.4 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.groovy
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.groovy
```

Note: Groovy scripts can also be executed directly from the fileserver tree via the context menu. This feature supports two execution modes:

- Simple Execution (`exec`): Runs the script without saving changes.
- Execution with Commit (`exec -c`): Runs the script and persists any changes made.

This option is available only for files with the content type `text/groovy` and the `.groovy` file extension.

21.5 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.

21.6 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 20.

21.7 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.groovy`
- 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 6) 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 22

Integrating ReportServer with an Active Directory using LDAP

In the following we will outline the necessary steps to connect ReportServer to an Active Directory using LDAP. As there are many valid ways to organize a company's directory (may it be AD or another vendors product) ReportServer allows you to customize all relevant LDAP options. This on one hand means, that the configuration might seem rather complex, but on the other hand it provides you with a maximum of flexibility.

To connect ReportServer to the Active Directory Service we will use ReportServer's integrated configuration. The whole process can be divided into two, mostly separate parts. One part is the synchronization of the user objects: we will automatically copy Users, Organizational Units and Groups from the directory to ReportServer and keep them updated. The second part is a mechanism that authenticates the previously imported users when they log into ReportServer.

22.1 Synchronizing Users

The current `ldapimport.groovy` script is available here: `https://github.com/infofabrik/reportserver-samples/blob/main/src/net/datenwerke/rs/samples/admin/ldap/ldapimport.groovy`.

The script reads its configuration from the `sso/ldap.cf` configuration file. As you may want to schedule the LDAP-import process, e.g. to import LDAP users every night, you can use the above script for this purpose. Otherwise, you can also use the `ldapimport` terminal command together with the `sso/ldap.cf` configuration file for manually importing LDAP users. With other words: for manually importing users use the `ldapimport` terminal command. For all other purposes use the `ldapimport.groovy` script above. Both basically achieve the same, but the script is scheduleable:

For scheduling this functionality periodically, schedule the `ldapimport.groovy` script via the `scheduleScript` terminal command as described in Section 20.57.

Note that you can (and should) use the `ldaptest` terminal commands for checking your LDAP configuration before letting the real import to happen. Details and example uses can be found in Section 20.38. The `ldapschema` (Section 20.37), `ldapguid` (Section 20.34), `ldapfilter` (Section 20.38) and `ldapinfo` (Section 20.36) terminal commands may also be useful for exploring your LDAP server and also the extended the `ldaptest users`, `ldaptest groups` and `ldaptest organizationalUnits` with a `-s` (schema) flag (refer to 20.38 ldaptest)

While the `-s` flag allows you to explore the installed object class types of your users', OUs' and groups' object classes, the `ldapschema` allows you to explore any object class.

For example, you may execute `ldaptest users -s` for printing the schema of the users' object class. You should get a list of optional attributes, required attributes, and the parent object class. Suppose the parent's object class is "organizationalPerson". You may then explore this object class with `ldapschema objectClassInfo organizationalPerson`.

You may continue exploring the LDAP schemas until the top-most object class: "top".

Refer to the Configuration Guide for a detailed description of all configurable values of the `ldap.cf` configuration file: https://reportserver.net/en/guides/config/chapters/SSO-related-properties/#LDAP

22.2 Authenticating Users

As of ReportServer 4.3.0, LDAP authentication is supported out-of-the-box. For using it, you have to install the `net.datenwerke.rs.ldap.service.ldap.pam.LdapPAM` or `net.datenwerke.rs.ldap.service.ldap.pam.LdapPAMAuthoritative` PAM in your `reportserver.properties` configuration file as described here: https://reportserver.net/en/guides/config/chapters/configfile-reportserverproperties/

It reads your `ldap.cf` configuration file together with metadata of your previously imported users, and authenticates the given user against your LDAP server.

Now that you should have a basic understanding how the LDAP mechanism works, let's give it a try. Download the two files `ldapimport.groovy` and `sso/ldap.cf` to your computer.

Open the `sso/ldap.cf` with a text editor and change the configuration options to match your configuration. Details on these can be found in the Configuration Guide: https://reportserver.net/en/guides/config/chapters/SSO-related-properties/#LDAP

After you modified the file, open ReportServer in your browser and go to the fileserver section in the admin module.

Upload the `ldapimport.groovy` to a location below the bin directory. Open the terminal by pressing `CTRL+ALT+T`. Upload the `sso/ldap.cf` to the `/etc` directory and type the `config reload` terminal command in order to reload your configuration.

As noted above, it is important to test your LDAP configuration first, so try to execute the following commands in order, as they are based on previous configuration. For example, the `ldaptest users` assumes the filter is correct, so you should run `ldaptest filter` first.

```
ldaptest filter
ldaptest guid
ldaptest groups
ldaptest organizationalUnits
ldaptest users
ldaptest orphans
```

Check the output of the above commands. If you get the correct output for all test commands, congratulations, your LDAP is configured correctly and you are ready to proceed. If you have any error in the commands above, you have to check and correct your configuration in the `ldap.cf` configuration file. Don't forget to run `config reload` after each configuration change.

Now you are ready to import your users from your LDAP server.

For this purpose, change your current directory to the location where you put the script file using the cd command and execute the import script.

```
cd /fileserver/bin
exec -c ldapimport.groovy
```

The `-c` (commit) flag is important because otherwise changes to the data model made by the script would be reverted after execution.

If you now change over to the user manager section you can view the results of the import. Also some statistics were written to the server's logfile/console.

Now you have to give "ReportServer access" generic permission to your imported users in order for them to be able to log-in. This of course depends on your desired configuration, but the easiest way would be to give the required permission to the parent directory where your LDAP users reside. Details can be found here in Section 3.2.

After you have verified that the import was successful, and you set the required permissions, it's time to load the authenticator module. Edit your `reportserver.properties` and set your LdapPAM as shown below:

```
rs.authenticator.pams = net.datenwerke.rs.ldap.service.ldap.pam. ↙
    ↳ LdapPAMAuthoritative
```

Restart your ReportServer after saving your `reportserver.properties` configuration.

Now you should be able to log in with your LDAP users.

Note you can also use the legacy LdapPAM script available here: https://github.com/infofabrik/reportserver-samples/blob/main/src/net/datenwerke/rs/samples/admin/ldap/legacyHookldappam.groovy

22.3 Possible Improvements

Using the scheduler to refresh users periodically To keep ReportServer's user database in sync with your company directory you would probably like to run the `ldapimport.groovy` script automatically from time to time. To do this, you can use the `scheduleScript` terminal command.

Chapter 23

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 **Dashboard 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.

23.1 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
    }
```

```
});
```

23.2 Embedding Dashboards via the URL

Similarly to reports (see Section 7.11) 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 dashboard 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 24

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.

Note that if you don't need the SFTP server, you can disable it via configuration. After a ReportServer restart, the SFTP server will not be started if it was disabled previously.

Once the connection is established you are in the root directory. Here, analogously 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 25

Maintenance

In this chapter we consider general administrative maintenance tasks.

25.1 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.

25.2 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.

An example query for looking into your audit log tables is the following:

```
SELECT E.*, P.KEY_FIELD, P.VALUE
```

```
FROM RS_AUDIT_LOG_ENTRY E join RS_AUDIT_LOG_PROPERTY P on E.ENTITY_ID = P. ↙
    ↳ LOG_ENTRY_ID
ORDER BY DATE_FIELD desc
```

25.3 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)