JasperReports® IO Professional Edition

User Guide

Version 4.0.0 | January 2024

Contents

Introduction to JasperReports® IO Professional Edition

JasperReports IO is an HTTP-based reporting service for JasperReports® Library, providing an interface to the JasperReports Library reporting engine through the use of a REST API and a JavaScript API. The REST API provides services for running, exporting, and interacting with reports while the JavaScript API allows you to embed reports and their input controls into your web pages and web applications using JavaScript frameworks for the layout and style sheets (CSS) to control the look and feel. Report templates, data sources, and all report resources are stored in a local repository or in an Amazon Web Services (AWS) S3 bucket and you have the option to create new report templates using Jaspersoft® Studio.

The JasperReports IO service can be deployed in various ways, from a single web application with interactive reports for small-scale deployments, to container-based deployments in the cloud, where specialized services running in separate containers work together to deliver a single, embeddable reporting service for large-scale deployments.

JasperReports IO is available as a downloadable standalone package and as an hourly offering on the AWS Marketplace.

This chapter contains the following sections:

JasperReports IO Professional Edition License Usage and Restrictions

This version of JasperReports IO can simultaneously run up to the licensed number of concurrent report runs, with queuing of additional requests. Usage is restricted to a single machine instance and it may not be deployed into an environment where multiple JasperReports IO instances are used to distribute the workload for a single end use application.

JasperReports IO licensees are entitled to use Jaspersoft Studio Professional to create reports. JasperReports IO Professional for AWS users must register via a link on the AWS Marketplace product page to receive a copy of Jaspersoft Studio Professional. Users that obtain the downloadable copy of JasperReports IO from the `Jaspersoft.com` site are entitled to apply the license file from the JasperReports IO Professional installation to their Jaspersoft Studio installation.

Installing JasperReports IO Using the Standalone Package

System Requirements

The JasperReports IO service can run a maximum of 2, 5, or 10 reports concurrently, depending on your license. The following table contains the recommended system requirements for JasperReports IO based on the maximum number of concurrent report runs:

Maximum Concurrent Report Runs	Processor	RAM
2	1 - 2 cores	512 MB - 2 GB
5	2 - 4 cores	2 GB - 4 GB
10	2 - 4 cores	4 GB - 8 GB

Starting JasperReports IO

JasperReports IO is available as a standalone ZIP package, downloadable from jaspersoft.com.

Different download packages are available for Windows, Linux, and macOS.

This installation package contains all the services and components needed for creating your own client applications and embeddable reports, including the JasperReports IO Professional reporting service, the JavaScript API, a web server, a sample web application,

data source adapters, and a Java Runtime Environment. The reporting service and sample web application are deployed when you start the web server. The sample web application helps you get started with creating your own application by demonstrating the capabilities of the JasperReports IO reporting service and the JasperReports IO JavaScript API.

See Managing JasperReports IO for information on the content and directory structure of JasperReports IO.

To start the JasperReports IO reporting service

1. Download the standalone package for your machine's operating system.
2. Extract the standalone package and open the extracted folder.
3. Run the start script to launch the web server.
   a. If you are using Windows, run the `start.bat` script.
   b. If you are using Linux or macOS, run `start.sh`.

   The script starts the web server. The JasperReports IO web application is ready for use.
4. To test the demo web application, open a browser and go to the following URL: `http://localhost:8080`.

   The browser opens the sample JasperReports IO web application. The sample application displays details about how to work with JasperReports IO.
5. To shut down the web server, run the stop script.
   a. If you are using Windows, run the `stop.bat` script.
   b. If you are using Linux or macOS, run `stop.sh`.

Installing JasperReports IO For Amazon Web Services

This section covers the JasperReports IO For Amazon Web Services (AWS) Hourly offering. You can purchase the product directly on the AWS Marketplace.

Prerequisites

You need a few things before you can install and run JasperReports IO on Amazon Web Services:

- An Amazon Web Services account.
  If you already have an account, log in to AWS.
  To create an AWS account, go to Amazon Web Services sign in page, click the **Create a new AWS account** button, and follow the instructions.

If you have a personal Amazon.com account stored in your browser, AWS uses that account by default. You need to sign out of Amazon or, preferably, use a different browser to set up an AWS account separate from your personal account.

- A valid Amazon key pair in your account. If you do not have a valid key pair, follow the instructions on the AWS documentation site:
  https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html
- The Required Permissions for using our CloudFormation template.

Required Permissions

Using our CF templates typically requires some admin permissions. AWS permissions required to launch a new JasperReports IO instance include:

- CloudFormation create stack and events
- Create and run EC2 instances
- Create EC2 security groups
- Create IAM resources
- Create S3 bucket
- Create CloudWatch log (if selected)

Accepting Terms of Use

You need to accept the terms of use for both the AWS Marketplace and Jaspersoft. This is a single process with multiple steps.

To accept the license agreement

1. Go to Jaspersoft listing on the Amazon Marketplace. You can use the link provided here, or use the Marketplace search function to locate the page.
2. Click the link for the JasperReports IO For AWS product.
   This page shows the total projected charges plus EC2 charges. Simply visiting a page does not place your order.
3. Click **Continue** to go to the Launch page.
4. Verify the information on this page and click **Accept Terms**.
   When your order is processed, you may receive an email confirmation.

Supported Instance Types

The following is a list of the instance types supported for JasperReports IO:

- T2 Micro (t2.micro)
- T2 Medium (t2.medium)

Performance may vary based on system attributes such as network, bandwidth, memory requirements for a given use case, query requirements, and the like.

For more information about EC2 instance types, see the AWS documentation:
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html

Creating a JasperReports IO Instance from a CloudFormation Template

A stack is a collection of AWS resources that you create and delete as a single unit. Our CloudFormation template creates the following resources and bundles them into a usable stack:

- IAM role
- S3 bucket
- EC2 instance with JasperReports IO installed, configured, and using the IAM role for appropriate credentials.

To create a JasperReports IO instance

1. Open the Launching Jaspersoft for AWS web page.
2. Click the **Launch Jaspersoft IO for AWS** tab.
3. Click the URL for your region. The **Select Template** page appears. By default, AWS provides a stack template source URL. Do not change this.
4. Click **Next**. The **Specify Details** page appears.
5. In the **Stack Name** field, give your CloudFormation stack a unique name.
6. Select an **InstanceType** from the dropdown. See Supported Instance Types for more information.
7. In the **KeyName** field, enter an existing key pair name.
8. In the **RemoveSamples** field, select whether to remove the sample web application and reports from your JasperReports IO instance's repository.
9. Choose the **VpcId** from your account.
10. Choose the **SubnetId** from the VPC.
11. Choose whether to create a publicly accessible IP address for the instance using **EnablePublicIp**. Default is set to True. Select **False** to refuse.
12. In the **SecuredIp** field, enter the IP address and mask for SSH access.
13. Choose whether to enable CloudWatch logs for your instance by selecting Yes for **CloudWatchIntegration**.
14. In the **S3BucketName** field, enter the name of the S3 bucket where you want to store your JasperReports IO reports and customizations. The S3 bucket must be in the same region as your JasperReports IO instance. A new S3 bucket is created if you leave the field empty.

If you enter an existing S3 bucket's name incorrectly, you experience errors when using JasperReports IO because the S3 bucket does not exist. See Correcting an Invalid S3 Bucket for instructions on fixing the issue.

15. Click **Next**. The **Options** page appears.
16. Add any tags that you want to use to simplify the administration of your infrastructure. A tag consists of a key/value pair and will flow to resources inside

your stack. You can add up to 10 unique keys to each instance, along with an optional value for each key.

17. If you want all operations for the stack limited to a certain role, use the **Permissions** section to choose the role.
18. In the **Rollback Triggers** section, set alarms you want CloudFormation to use to monitor the creation of the stack. If any alarms are triggered, CloudFormation stops of the creation of the stack and rolls it back.
19. Expand the **Advanced** section and set your notification, timeout, and other options.
20. Click **Next**. The **Review** page appears. Double-check your template, parameter, and option information.
21. Click the acknowledgment checkbox, then click **Create**. You see your stack name listed in a table. While it is being created the Status column displays `CREATE_IN_PROGRESS`. After a few minutes, the status should change to `CREATE_COMPLETE`. If the status changes to `ROLLBACK` instead of `CREATE_COMPLETE`, you may need to accept the Terms of Use. Check the Events tab for more information.
22. Select your complete instance and click the **Outputs** tab. Here you find the name of the S3 bucket for your repository, the link for the CloudWatch log, and the Getting Started URL for logging into the JasperReports IO web application if you enabled a publicly accessible IP address.

Creating a Repository Folder in Your S3 Bucket

When setting up your JasperReports IO instance, you need to create a repository folder in your S3 bucket to store the resources to create and run your reports.

To create a repository folder

1. On the AWS Management Console home page, click **S3**.
2. Click the name of the bucket for your instance or cluster.
3. Click **Create Folder**.
4. Enter `remoteRepository` for the name of the folder.
5. Select **None (use bucket settings)** for the encryption setting.
6. Click **Save**. AWS creates the `remoteRepository` folder.

You can create the repository directories for your report resources in the new `remoteRepository` folder and upload your files. See Managing JasperReports IO for more information on the repository directory structure.

Correcting an Invalid S3 Bucket

If you enter the incorrect name for an existing S3 bucket when creating your instance, you need to update the settings for the instance and associated IAM role to point them to the correct S3 bucket.

To correct the S3 bucket

1. On the AWS Management Console home page, click **EC2**.
2. Click **Instances** in the sidebar.
3. Click the instance with the invalid S3 bucket in the table.
4. Click **Actions > Instance State > Stop** to stop the instance.
5. Click **Actions > Instance Settings > View/Change User Data**.
6. Locate the `s3.repository.bucket` and replace the invalid S3 bucket name with the correct one.
7. Click **Save**.
8. Return to the AWS Management Console home page and click **IAM**.
9. Click **Roles** in the sidebar.
10. Click the name of the IAM role created for your JasperReports IO instance in the table.
11. On the **Permissions** tab, expand the policy and click **S3** under Service. The tab displays a list of S3 actions.
12. Click **Edit Policy**.
13. Click the **JSON** tab.
14. Locate **Resource** and replace the invalid S3 bucket name with the correct one. For example:

```
{
                        "Statement": [
                        {
                        "Action": [
                        "s3:Get*",
                        "s3:List*"
                        ],
                        "Resource": [
                        "arn:aws:s3:::jrio-jrios3bucket-12",
                        "arn:aws:s3:::jrio-jrios3bucket-12/*"
                        ],
                        "Effect": "Allow"
                        }
                        ]
                }
```

15. Click **Review the policy**.
    AWS displays the S3 service that you are updating. You can click **S3** to review the service before committing your changes.
16. Click **Save changes**.
    AWS updates the IAM role with the S3 bucket changes.
17. Return to the AWS Management Console home page and click **IAM**.
18. Restart your instance.

Managing JasperReports IO

After installing JasperReports IO, you will need to create the reports, web applications, and anything else you require for reporting and storing the files in a repository for the reporting service to use. The JasperReports IO installation includes many sample files that you can use for reference. You need to use a file browser on the host machine to view and manage the contents of the JasperReports IO installation.

This chapter covers the basics of managing your JasperReports IO installation, including:

- File directory structure
- The web application servers and web applications
- The repository
- AWS S3 buckets
- Cloud repositories
- Security permissions

Unless noted otherwise, all references to JasperReports IO are for the standalone version, not JasperReports IO for AWS.

For JasperReports IO for AWS, the reporting service is part of an instance hosted on Amazon Web Services. Use the AWS Management Console to manage the JasperReports IO instance hosted on the service. See Managing Amazon Web Services for JasperReports IO for information on managing JasperReports IO for AWS.

This chapter includes the following sections:

- Security

JasperReports IO Directories

The directory where JasperReports IO is installed on the host machine is referred to as **<jrio-install>** in this guide. The <jrio-install> directory contains the start and stops scripts for the server and the license agreement. The contents of JasperReports IO are organized as the following directories when first installed:

JasperReports IO Directories

Directory	Description
docker	Contains a Dockerfile for creating a docker image of JasperReports IO, a start script for the image, and a repository configuration file.
jetty	Contains the Eclipse Jetty web application server that hosts the JasperReports IO web application. See Web Application Server for more information.
jre	Contains the Java Run time Environment for running the JasperReports IO reporting service.
jrio	Contains the files for the JasperReports IO reporting service and web applications. See JasperReports IO Reporting Service and Web Application Directories for more information.
repository	The repository stores all the resources used to run and create reports, including data source definitions, JRXML files, datatypes, and helper files such as images. See JasperReports IO Repository for more information.

JasperReports IO Reporting Service and Web Application Directories

The <jrio-install>/jrio/webapps directory contains the files for the JasperReports IO reporting service, JavaScript API, and the sample web application.

Web Application Directories

Directory	Description
`jrio`	The JasperReports IO web application, including the reporting service and all configuration files.
`jrio-client`	Contains the files for JasperReports IO's JavaScript API, including the `jrio.js` file and UI themes. See the JavaScript API for more information.
`jrio-docs`	Contains the files for the JasperReports IO sample web application, which demonstrates the capabilities of the reporting service, as well as documentation and report samples.
`ROOT`	The sample HTML file in this directory forwards root requests to the `jrio-docs` web application.

Web Application Server

The JasperReports IO reporting service is deployed in a Java web application on an Eclipse Jetty web server included with JasperReports IO. You can create a web application with interactive reports and the web application server handle all HTTP requests from users. The web application server is located in the `<jrio-install>/jetty` directory. For information on the Eclipse Jetty web server, see the Jetty documentation.

On JasperReports IO, there are two scripts in the `<jrio-install>` directory to start and stop the Eclipse Jetty server and the JasperReports IO reporting service. The script to start the web application server is `start.bat` for Windows and `start.sh` for Mac OS and Linux, and the stop script is `stop.bat` for Windows and `stop.sh` for Mac OS and Linux.

Configuring the Web Application Server

The start script specifies several startup configuration parameters for the server that can be changed to better suit your needs.

Java Virtual Machine Heap Memory

There are two parameters for specifying the amount of heap memory allocated to the JasperReports IO reporting service's Java web application when it starts up. The `-Xms<size>` parameter specifies the initial heap memory size for the web application and the `-Xmx<size>` parameter specifies the maximum heap memory size. The following examples show an initial heap memory size of 256 MB and a maximum size of 512 MB:

Linux: `./jre/bin/java -Xms256m -Xmx512m -jar ./jetty/start.jar`

Windows: `jre\bin\java -Xms256m -Xmx512m -jar jetty\start.jar`

Mac OS: `./jre/Contents/Home/bin/java -Xms256m -Xmx512m -jar ./jetty/start.jar`

Web Application Server Port

By default, the web application server starts on port 8080, but if this port is already in use on your host machine, you can edit the start script to change the following setting to another port number:

```
-Djetty.http.port=8080
```

Web Application Server Stop Port and Stop Key

The start and stop scripts define the stop port and stop key settings required for stopping the web application server. The stop port is the number of the port on the host machine that listens for termination requests and the stop key is a key that must be the port of the stop request. These settings must match in both scripts for the web application server to shut down properly. The following is an example of the stop port and key settings:

```
-DSTOP.PORT=8989 -DSTOP.KEY=st0p_J3Tty
```

Web Application

When the web application server is running, the user's web browser accesses the HTML files in the `<jrio-install>/repository/ROOT` folder when they open `http://<jrio>:8080` in their browser. You can store the files for your web application in this folder or create an `index.html` file to redirect the user's browser to a web application in another directory. For example, the sample `index.html` in the directory that was created during installation

redirects the user's browser to `http://<jrio>:8080/jrio-docs`, where the sample web application that comes with JasperReports IO is installed.

JasperReports IO Repository

The JasperReports IO repository is a folder-based structure where all the resources used to run and create reports are stored and from where they are retrieved when reports are run by the JasperReports IO reporting service. You can have the repository on the host machine or an AWS S3 bucket.

The type of repository, its location in the JasperReports IO file structure, and other specific repository implementation properties can be specified in the following configuration file:

```
js-install>/jrio/webapps/jrio/WEB-INF/applicationContext-repository.xml
```

JasperReports IO comes with a repository full of sample reports and resources in the `<js-install>/repository` directory, but you can create your own repository. If you are using JasperReports IO for AWS, you have to create a repository in an S3 bucket. See Creating a Repository Folder in Your S3 Bucket for more information.

Repository Directory Structure

The JasperReports IO repository is structured as follows:

Repository Directories for Sample Reports

Directory	Description
data	Contains the data source adapters and data source files for your reports.
images	Contains image files used in reports.
JR-INF	Contains the configuration files for report execution.
reports	Contains report templates.

Data Sources and Data Adapters

A data adapter is a resource that specifies how and where to obtain data. Specifically, it is an object that contains information about how to connect to or retrieve the data, and the logic to do that. This information includes, URL, user, password, paths, and so on. Data adapters also contain the logic to prepare all parameters for JasperReports IO to run the query and iterate data. All the connections are opened and passed directly to JasperReports during report generation. A data adapter does not contain any data itself, which is stored in data sources.

The sample repository installed with JasperReports IO contains multiple data adapters and data sources in the `<js-install>/repository/data` directory that you can use for your own reports. These data adapters include:

- JDBC connection
- CSV connection
- Excel connection
- Empty connection
- JNDI connection
- Remote XML connection

JasperReports IO can use other types of data adapters that are not included in the sample repository. You can create your own data adapters for JasperReports IO either by using the Data Adapter Wizard in Jaspersoft Studio or by creating a custom data adapter using a JRDAX file.

Reports

The repository resource that aggregates all information needed to run a report is called a JasperReport. A JasperReport is based on a JRXML file that conforms to the JasperReports Library that JasperReports IO uses to render reports. Users can create reports for JasperReports IO using Jaspersoft Studio.

A JasperReport is a complex resource composed of other resources:

• The main JRXML file that defines the report.

• A data source that supplies data for the report.

• A query if none is specified in the main JRXML.

• The query may specify its own data source, which overrides the data source defined in the report.

• Input controls for parameters that users may enter before running the report. Input controls are composed of either a datatype definition or a list of values.

• Any additional file resources, such as images and fonts.

• If the report includes subreports, the JRXML files for the subreports.

End users interact with a JasperReport as a single resource, but report creators must define all the component resources. Refer to the Jaspersoft Studio User Guide for more information on creating reports.

Configuring the Web Application Server to Reference the Repository

On JasperReports IO, the repository containing the sample report templates used by the sample web application is located in the `<jrio-install>/jrio/repository` directory. Since this repository folder is not located within the JasperReports IO web application folder, you need to configure the `<js-install>/jrio/webapps/WEB-INF/applicationContext-repository.xml` file to point the web application to the repository directory.

There are two different ways to point the web application to a repository on the host machine: Using a relative file path using the `WebappRelativeRepositoryFactory` bean or an absolute file path using the `FileRepositoryService` bean.

To use a relative file path, locate the `WebappRelativeRepositoryFactory` bean in the `applicationContext-repository.xml` file and enter the relative file path as the value for the `root` property:

```
<bean
class="com.jaspersoft.jrio.common.repository.WebappRelativeRepositoryFac
tory">
    <property name="jasperReportsContext"
ref="baseJasperReportsContext"/>
    <property name="root" value="../../../repository"/>
</bean>
```

Since the web application is deployed to the `<jrio-install>jrio/webapp/jrio` directory, use "`../../../repository`" as the relative file path to point the web application to the repository in the `<jrio-install>/jrio/repository` directory.

If you want to use an absolute file path to the repository directory, change the repository bean class from `com.jaspersoft.jrio.common.repository.WebappRelativeRepositoryFactory` to `com.jaspersoft.jrio.common.repository.FileSystemRepository` and edit the value for the second `<constructor-arg>` to add the absolute file path:

```
<bean
class="com.jaspersoft.jrio.common.repository.FileSystemRepository">
    <constructor-arg><ref
bean="baseJasperReportsContext"/></constructor-arg>
    <constructor-arg><value>/mnt/jrio-repository</value></constructor-
arg>
</bean>
```

Use a slash (/) at the beginning of the URI for the root directory.

If you are using an AWS S3 bucket for the repository, refer to the AWS S3 Bucket Repository for instructions on configuring the web application server to use the bucket.

Configuring the Web Application Server to Use Multiple Repositories

If you use multiple repository directories to store your report templates and resources, JasperReports IO can treat these separate directories as a single repository through absolute file paths. A report works if its JRXML template is in one repository and its resources are in a second. Add a `FileRepositoryService` bean to the `<js-install>/jrio/webapps/WEB-INF/applicationContext-repository.xml` file for each repository that you want to use.

```
<bean
class="com.jaspersoft.jrio.common.repository.FileSystemRepository">
       <constructor-arg><ref bean="baseJasperReportsContext"/>
</constructor-arg>
       <constructor-arg><value>/mnt/repository1</value></constructor-arg>
</bean>

<bean
```

```
class="com.jaspersoft.jrio.common.repository.FileSystemRepository">
       <constructor-arg><ref bean="baseJasperReportsContext"/>
</constructor-arg>
       <constructor-arg><value>/mnt/repository2</value></constructor-arg>
</bean>
```

Managing Amazon Web Services for JasperReports IO

This section describes how to use an AWS S3 bucket for a repository for JasperReports IO, referring to reports stored in an S3 bucket, and customizations for JasperReports IO for AWS.

AWS S3 Bucket Repository

JasperReports IO comes with a sample configuration setting for connecting your standalone JasperReports IO instance to an AWS S3 bucket as a repository. The S3 bucket can either be public and accessed without credentials or accessed securely using AWS credentials. Locate and the `S3RepositoryService` bean in the `<js-install>/jrio/webapps/WEB-INF/applicationContext-repository.xml` configuration file to implement an AWS S3 bucket for a repository:

```
<bean
class="com.jaspersoft.jrio.common.repository.s3.S3RepositoryService">
    <property name="jasperReportsContext"
ref="baseJasperReportsContext"/>
    <property name="s3Service">
        <bean
class="com.jaspersoft.jrio.common.repository.s3.S3ServiceFactory">
        <property name="region" value="us-east-1"/>
    <!--
        <property name="accessKey" value="put-id-here"/>
        <property name="secretKey" value="put-key-here"/>
    -->
        </bean>
    </property>
    <property name="bucketName" value="jrio-repo-sample"/>
    <property name="pathPrefix" value="jrio-repository/"/>
```

```
    <bean
class="com.jaspersoft.jrio.common.repository.s3.S3RepositoryPersistenceS
erviceFactory" factory-method="instance"/>

</bean>
```

You need to enter the region, S3 bucket name, and the path to the repository.

Use the `accessKey` and `secretKey` properties to enter your AWS ID and key. These credentials are optional if the S3 bucket is public.

If you created your JasperReports IO from the CloudFormation template for AWS, this configuration file appears similar to the following:

```
<bean
class="com.jaspersoft.jrio.common.repository.s3.S3RepositoryService">
    <property name="jasperReportsContext"
ref="baseJasperReportsContext"/>
    <property name="s3Service">
        <bean
class="com.jaspersoft.jrio.common.repository.s3.S3ServiceFactory">
        <property name="region" value="${s3.repository.region:null}"/>
    <!--
        <property name="accessKey" value="put-id-here"/>
        <property name="secretKey" value="put-key-here"/>
    -->
        </bean>
    </property>
    <property name="bucketName" value="${s3.repository.bucket:null}"/>
    <property name="pathPrefix"
value="${s3.repository.path.prefix:null}"/>
</bean>
```

The `${...}` is automatically populated by user data that was generated when the JasperReports IO instance was created.

You do not have to provide your AWS credentials if you created a S3 bucket or selected an existing one as when creating the JasperReports IO instance using the CloudFormation template. An IAM role is automatically created that allows the JasperReports IO instance to connect to the S3 bucket without having to provide the credentials.

Referring to Reports in the AWS S3 Bucket Repository

For storing report resources in an AWS S3 bucket, you need to create a folder in the bucket called `remoteRepository`. See Creating a Repository Folder in Your S3 Bucket for instructions on adding this folder to your S3 bucket.

JasperReports IO accesses the reports in the bucket through the REST and JavaScript APIs using relative URIs with `/remoteRepository` as the root directory. For example, if you have a report stored in the repository at `/remoteRepository/reports/myReport.jrxml`, the reference to the API is `/reports/myReport`. When opening the report in the viewer, the URL is:

```
 http://<JRIO domain>:<JRIO port>/jrio-docs/viewer/viewer.html?jr_
report_uri=/reports/myReport
```

See the REST API and JavaScript API chapters for more information on how to use them.

JasperReports IO for AWS and VPC Security

When creating your JasperReports IO for AWS instance, you select the VPC and a subnet it belongs to. An AWS VPC isolates its resources to a virtual network with advanced security features to protect the user's resources. AWS VPCs include security features such as subnets within availability zones, IP ranges, route tables, and security groups to protect the resources.

To access the services and resources you want to use, your JasperReports IO for AWS instance needs to be on the same VPC as those services and the appropriate subnets across availability zones. If you have issues connecting your JasperReports IO for AWS instance to the resources and services it needs, you may need to update the AWS security features for the VPC to allow access.

Customizations for JasperReports IO for AWS

JasperReports IO and JasperReports IO for AWS allows you to use your S3 bucket to store customized configuration files for your JasperReports IO instance. In the S3 bucket, you must recreate the JasperReports IO directory structure for the configuration files in a folder

called `customizations`, starting with the folders at the `<jrio-install>` root level, such as `jrio` and `repository`.

If you want to remove the customized file from the instance, you need to copy the original configuration file to the S3 bucket and reboot the instance. This will replace the file on the instance and remove the customizations from JasperReports IO. Deleting the customized file from the S3 bucket without adding a replacement will not remove the customizations when the instance is restarted. JasperReports IO for AWS includes a special `service jrio start/stop` command for starting and stopping the web application.

To upload your customization

1. On the AWS Management Console homepage, click **S3**.
2. Find the bucket for your JasperReports IO instance and click the name.
3. Click **Create Folder** and create a folder called **`Customizations`**.
4. Click the name of the **`Customizations`** folder.
5. Click **Create Folder** and recreate the paths to your files.

   For example, if you want to upload a configuration file that goes in the `<jrio-install>/jrio/webapps/jrio/WEB-INF/classes` directory, you have to create a folder for each directory in that file path.
6. After creating the folder paths, browse to the folder for your configuration file.
7. Click **Upload**.
8. Click **Add files** and find the configuration file on your local machine.
9. Click **Upload** to upload the configuration file.

   AWS uploads the file and stores it in the S3 bucket.
10. With the configuration file in place, SSH into your instance using your AWS private key and username.
11. Stop the JasperReports IO instance using the following command:

```
sudo service jrio stop
```

12. Start the JasperReports IO instance:

```
sudo service jrio start
```

When the JasperReports IO instance restarts, the changes based on the configuration file is in place.

Cloud Repositories for JasperReports IO

This section describes how JasperReports IO can use reports and resources stored in the cloud repositories (Google Drive, GitHub, and Dropbox) using OAuth 2.0 standard protocol for authorization.

OAuth2 Repositories

By default, JasperReports IO comes with three preconfigured OAuth2 repositories for Google Drive, GitHub, and Dropbox. Each of these is defined in a separate configuration file as follows:

```
[JRIO_WEB_APP]/WEB-INF/applicationContext-google-drive.xml
```

```
[JRIO_WEB_APP]/WEB-INF/applicationContext-github.xml
```

```
[JRIO_WEB_APP]/WEB-INF/applicationContext-dropbox.xml
```

To use these repositories, each repository configuration file needs to be updated with actual `clientId` and `secretKey` values. These values are obtained from the target cloud storage providers while registering your JasperReports IO instance with them.

The configuration file for the Google Drive repository appears similar to the following:

```
<bean
class="com.jaspersoft.jrio.common.repository.google.GoogleDriveRepositor
yService">
   <property name="jasperReportsContext"
ref="baseJasperReportsContext"/>
   <property name="googleDriveProvider">
      <bean
class="com.jaspersoft.jrio.common.repository.google.RequestTokenGoogleDr
iveProvider">
         <property name="googleDriveFactory">
```

```
            <bean
class="com.jaspersoft.jrio.common.repository.google.GoogleDriveFactory">
               <property name="clientId" value="put-client-id-here"/>
               <property name="secretKey" value="put-secret-key-here"/>
             </bean>
         </property>
         <property name="serviceCache">
            <bean
class="com.jaspersoft.jrio.common.execution.cache.LocalCacheAccessFactor
y">
               <property name="cacheContainer" ref="localCacheManager"/>
               <property name="cacheRegion"
value="googleDriveServices"/>
            </bean>
         </property>
       </bean>
    </property>
</bean>
```

Accessing Cloud Repositories

The sample web application helps you connect to the cloud repositories (Google Drive, GitHub, and Dropbox) by providing a login UI. You can access the sample cloud repository login UI if you have the required OAuth2 credentials, namely `clientId` and `secretKey`. These values need to be specify in both repository configuration files and the client application configuration file for `JRIO_DOCS_WEB_APP]/WEB-INF/classes/jasperreports.properties`.

The sample web application acts as a proxy to the JasperReports IO application and acquires the OAuth2 authorization tokens from the cloud services. Then these access tokens are passed to the JasperReports IO, allowing JasperReports IO to load reporting resources from the remote repositories.

Security

JasperReports IO provides security for your web applications and reports through a protection domain used by the Java security manager. A protection domain defines the security permissions, public keys, and URI for a group of JasperReports IO components, such as report expressions and repository JAR files. You can customize the permissions using the `<jrio-install>/jrio/security.policy` file.

JasperReports IO comes with a preconfigured protection domain that by default gives users all permissions to the files for:

- The Java Virtual Machine.
- The web application server.
- The JasperReports IO reporting service web applications.

The preconfigured protection domain restricts users' permissions to the following:

- Repository JARs.
- Report expressions.

The following shows the preconfigured protection domain settings in the `security.policy` file:

```
grant codeBase "file:${java.home}/lib/-" {
permission java.security.AllPermission;
};
grant codeBase "file:${java.home}/lib/ext/-" {
    permission java.security.AllPermission;
};
grant codeBase "file:${user.dir}/jetty/-" {
    permission java.security.AllPermission;
};
grant codeBase "file:${user.dir}/jrio/webapps/-" {
    permission java.security.AllPermission;
};
//permissions for JRIO repository jars
    grant codeBase "file:/__jrio/repository/jars/" {
//permission java.security.AllPermission;
};
//permissions for JR reports
grant codeBase "file:/__jrio/repository/reports/" {
};
```

This default configuration restricts a user's ability to pass parameters within the path of a report. You can edit the protection domain to customize the security permissions for JasperReports IO to meet your security needs.

More details about the syntax of the `security.policy` file and what permissions are available can be found in the Java Security documentation.

The protection domain and the Java security manager for used by JasperReports IO are not active when you first install the reporting service. To activate the security manager and

protection domain, edit the start script in the <jrio-install> directory to uncomment the following:

```
-Djava.security.manager -Djava.security.policy=jrio/security.policy
```

The Java security manager and protection domain is active when you start the web application server.

The reports Service

The rest_v2/reports service has a simple API for obtaining report output, such as PDF and XLSX. The service also provides functionality to interact with running reports, report options, and input controls.

This chapter includes the following sections:

Running a Report

The reports service allows clients to receive report output in a single request-response. The reports service is a synchronous request, meaning the caller is blocked until the report is generated and returned in the response. For large datasets or long reports, the delay can be significant. If you want to use a non-blocking (asynchronous) request, see The reportExecutions Service

The output format is specified in the URL as a file extension to the report URI.

Method	URL	
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/path/to/report.<format>?<arguments> http://<host>:<port>/jrio/rest_v2/reports/path/to/report.<format>?<arguments>	
Argument	Type/Value	Description
<format>	output type	One of the following: pdf, html, xlsx, rtf, csv, xml, docx, odt, ods, jrprint.

		As of JasperReports Server 6.0, it is also possible to specify json if your reports are designed for data export. For more information, see the JasperReports Library samples documentation. One of the following formats: • Regular output: html, pdf, csv, docx, pptx, xlsx, rtf, odt, ods, xml • Metadata output: data_csv, data_xlsx, data_json
page?	Integer > = 0	An integer value is used to export a specific page. Passing 0 (zero) as the page parameter is accepted, and a blank report is displayed.
anchor?	String	An anchor name in the generated report.
ignore pagination?	Boolean	When set to true, the report is generated as a single page. This can be useful for some formats such as csv. When omitted, this argument's default value is false and the report is paginated normally.
<parameter>	String	Any parameter that is defined for the report. Parameters that are multivalue may appear more than once. See examples below.
<inputControl>	String	Any input control that is defined for the report. Input controls that are multi-select may appear more than once. See examples below. By default, the input values are case sensitive. To change them to case insensitive, set inputControl.handler.values.caseSensitive=false in the jasperserver-pro/WEB-INF/js.config.properties file.
interactive?	Boolean	In the commercial edition of the server, where HighCharts are used in the report, this property determines the generation of the JavaScript required for interaction when exported to HTML. By default it is

		true. If set to false, the chart is generated as a non-interactive image file.
onePage PerSheet?	Boolean	Valid only for the XLS format. When the value is true, each page of the report is on a separate spreadsheet. If false or omitted, the entire report is on a single spreadsheet. For long reports, set this argument to true, otherwise the report does not fit on a single spreadsheet and causes an error.
report Container Width?	Integer	This property specifies the width of the report container. A report specifying this parameter with integer values receives the current screen size width when the report is run.
baseUrl	String	Specifies the base URL that the reportusesuses to load static resources such as JavaScript files. You can also set the deploy.base.url property in the .../WEB-INF/js.config.properties file to set this value permanently. If both are set, the baseUrl parameter in this request takes precedence. Specifies the base URL that the report uses to load static resources such as JavaScript files.
attachments Prefix	attachments	For HTML output, this property specifies the URL path to use for downloading the attachment files (JavaScript and images).

Return Value on Success	Typical Return Values on Failure
200 OK – The content is the requested file.	400 Bad Request – When incorrect format is provided in the Get request. 404 Not Found – When the specified report URI is not found in the repository.

The following examples show various combinations of formats, arguments, and input controls:

http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/AllAccounts.html (all pages)

http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/AllAccounts.html?page=43

http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/AllAccounts.pdf (all pages)

http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/AllAccounts.pdf?page=1

http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/EmployeeAccounts.html?EmployeeID=sarah_id

http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/reports/samples/Cascading_multi_select_report.html?Country_multi_select=USA&Cascading_state_multi_select=WA&Cascading_state_multi_select=CA

http://<host>:<port>/jrio/rest_v2/reports/samples/reports/FirstJasper.html (all pages)

http://<host>:<port>/jrio/rest_v2/reports/samples/reports/FirstJasper.html?page=5

http://<host>:<port>/jrio/rest_v2/reports/samples/reports/FirstJasper.pdf (all pages)

http://<host>:<port>/jrio/rest_v2/reports/samples/reports/FirstJasper.pdf?page=5

http://<host>:<port>/jrio/rest_v2/reports/samples/reports/chartthemes/ChartThemesReport.pdf?chartTheme=aegean

JasperReports Server JasperReports IO does not support exporting Highcharts charts with background images to PDF, ODT, DOCX, or RTF formats. When exporting or downloading reports with Highcharts that have background images to these formats, the background image is removed from the chart. The data in the chart is not affected.

Finding Running Reports

The reports service provides functionality to stop reports that are running. Reports can be running from user interaction, web service calls, or scheduling. The following method provides several ways to find reports that are currently running, in case the client wants to stop them.

This syntax of the reports service is deprecated. See The reportExecutions Service.

Method	URL		
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/path/to/report/ http://<host>:<port>/jasperserver[-pro]/rest_v2/reports?<arguments>		
Argument	Type/Value	Description	
jobID?	String	Find the running report based on its jobID in the scheduler.	
jobLabel?	String	Find the running report based on its jobLabel in the scheduler.	
userName?	String	Name of user who has scheduled a report, in the format <username>%7C<organizationID>. In the commercial editions, %7C<organizationID> is required for all users except system admins (superuser).	
fireTime From?	date/time	Date and time in the following pattern: yyyy-MM-dd'T'HH:mmZ. Together, these arguments create a time range to find when the running report was started. Both of the range limits are inclusive. Either argument may be null to signify an open-ended range.	
fireTimeTo?	date/time		
Return Value on Success			Typical Return Values on Failure
200 OK – The content is a list of execution IDs that can be used for cancellation.			404 Not Found – When the specified report URI is not found in the repository.

For security purposes, the search for running reports has the following restrictions:

- The system administrator (superuser) can see and cancel any report running on the server.
- An organization admin (jasperadmin) can see every running report. But can cancel only the reports that are started by a user of the same organization or its child organizations.

- A regular user can see every running report, but can cancel only the reports that they initiated.

Stopping a Running Report

Use the following method to stop a running report, as found with the previous method.

This syntax of the reports service is deprecated. See The reportExecutions Service.

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/reports/<executionID>/status/

Content-Type	Content
application/xml	Either an empty instance of the ReportExecutionCancellation class or <status>canceled</status>.

Return Value on Success	Typical Return Values on Failure
200 OK – The content also contains: <status>canceled</status>.	204 No Content – When the specified execution ID is not found on the server, and the response body is empty.

The reportExecutions Service

As described in The reports Service , synchronous report execution blocks the client waiting for the response. Synchronous report execution slows down or uses many threads, each waiting for a report, when:

- Managing large reports that may take minutes to complete.
- Running a large number of reports simultaneously

The rest_v2/reportExecutions service provides asynchronous report execution, so that the client does not need to wait for report output. Instead, the client obtains a request ID to check the status of the report periodically. This also called polling. When the report is finished, the client downloads the output. Alternatively, the client can check when specific pages are finished and download available pages. The client can also send an asynchronous request for other export formats (PDF, Excel, and others) of the same report. Again the client can check the status of the export and download the result when the export has been completed.

Reports scheduled on the server also run asynchronously. reportExecutions allows you to access jobs triggered by the scheduler. Finally, the reportExecutions service allows the client to stop and remove any report execution or job that has been triggered.

This chapter includes the following sections:

- Stopping Running Reports and Jobs
- Removing a Report Execution

Running a Report Asynchronously

To run a report asynchronously, the reportExecutions service provides a method to specify all the parameters needed to open a report. Report parameters are all sent as a reportExecutionRequest object. The response from the server contains the request ID needed to track the execution until completion.

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions http://<host>:<port>/jrio/rest_v2/reportExecutions

Content-Type	Content
application/xml application/json	A complete ReportExecutionRequest in either XML or JSON format. See the example and table below for an explanation of its properties. A complete ReportExecutionRequest in JSON format. See the example and table below for an explanation of its properties.

Return Value on Success	Typical Return Values on Failure
200 OK – The content contains a ReportExecution descriptor. See below for an example	403 Forbidden – When the logged-in user does not have permission to access the report in the request. 404 Not Found – When the report URI specified in the request does not exist.

The following example shows the structure of the ReportExecutionRequest:

```
<reportExecutionRequest>

<reportUnitUri>/supermart/details/CustomerDetailReport</reportUnitUri>
```

```
    <async>true</async>
    <reportContainerWidth>900</reportContainerWidth>
    <freshData>false</freshData>
    <saveDataSnapshot>false</saveDataSnapshot>
    <outputFormat>html</outputFormat>
    <interactive>true</interactive>
    <ignorePagination>false</ignorePagination>
    <pages>1-5</pages>
    <parameters>
        <reportParameter name="someParameterName">
            <value>value 1</value>
            <value>value 2</value>
        </reportParameter>
        <reportParameter name="someAnotherParameterName">
            <value>another value</value>
        </reportParameter>
    </parameters>
</reportExecutionRequest>
```

```
{
    "reportUnitUri":"/samples/reports/chartthemes/ChartThemesReport",
    "async":true,
    "interactive":true,
    "pages":"1-5",
    "attachmentsPrefix":"/jrio/rest_v2/reportExecutions/
        {reportExecutionId}/exports/{exportExecutionId}/attachments/",
    "baseUrl":"/jrio",
    "parameters":
    {
        "reportParameter":
        [
            {"name":"chartTheme","value":["aegean"]},
            {"name":"anotherParamName","value":["value 1","value 2"]}
        ]
    }
}
```

The following table describes the properties that you can specify in the ReportExecutionRequest:

Property	Required or Default	Description
reportUnitUri	Required	Repository path (URI) of the report to run. For commercial editions with organizations, the URI is relative to the logged-in user's organization. Repository path (URI) of the report to run.
outputFormat	Required	Specifies the desired output format: pdf, html, xlsx, rtf, csv, xml, docx, odt, ods, jrprint. As of JasperReports Server 6.0, it is also possible to specify json if your reports are designed for data export. For more information, see the JasperReports Library samples documentation. Specifies the desired output format: • Regular output: html, pdf, csv, docx, pptx, xlsx, rtf, odt, ods, xml • Metadata output: data_csv, data_xlsx, data_json
freshData	false	When data snapshots are enabled, it specifies whether the report should get fresh data by querying the data source. If false, use a previously saved data snapshot (if any). By default, if a saved data snapshot exists for the report it is used when running the report.
saveDataSnapshot	false	When data snapshots are enabled, it specifies whether the data snapshot for the report should be written (or overwritten) with data from the report execution.
interactive	true	In a commercial edition of the server where Highcharts are used in the report, this property determines whether the required JavaScript is generated. It is returned as an attachment when exported to HTML. If false, the chart is generated as a non-interactive image file (also as an attachment).

Property	Required or Default	Description
allowInlineScripts	true	Affects HTML export only. If true, then inline scripts are allowed, otherwise no inline script is included in the HTML output.
ignorePagination	Optional	When set to true, the report is generated as a single long page. This can be used with HTML output to avoid pagination. When omitted, the ignorePagination property on the JRXML, if any, is used.
pages	Optional	Specify a page range to generate a partial report. The format is: <startPageNumber>-<endPageNumber>
async	false	Determines whether reportExecution is synchronous or asynchronous. When set to true, the response is provided immediately and the client must poll the report status. They can download the results when ready. By default, this property is false and the operation pauses until the report execution is complete. The client must wait but can download the report immediately after the response.
transformerKey	Optional	Advanced property used when requesting a report as a JasperPrint object. This property can specify a JasperReports Library generic print element transformer of the class net.sf.jasperreports.engine.export.GenericElementTransformer. These transformers are pluggable as JasperReports Library extensions.
attachmentsPrefix	attachments	For HTML output, this property specifies the URL path to use for downloading the attachment files (JavaScript and images). The full path of the default value is: {contextPath}/rest_v2/reportExecutions/{reportExecutionId}/exports/

Property	Required or Default	Description
		{exportExecutionId}/attachments/ You can specify a different URL path using the placeholders {contextPath}, {reportExecutionId}, and {exportExecutionId}.
baseUrl	String	Specifies the base URL that the report uses to load static resources such as JavaScript files. You can also set the deploy.base.url property in the .../WEB-INF/js.config.properties file to set this value permanently. If both are set, the baseUrl parameter in this request takes precedence. Specifies the base URL that the report uses to load static resources such as JavaScript files.
parameters	See example	A list of input control parameters and their values. By default, the parameter values are case-sensitive. To change them to case insensitive, set inputControl.handler.values.caseSensitive=false in the jasperserver-pro/WEB-INF/js.config.properties file.
reportContainerWidth	Optional	This property specifies the width of the report container. A report specifying this parameter with integer values receives the current screen size width when the report is run.

When successful, the reply from the server contains the reportExecution descriptor. This descriptor contains the request ID and status needed for the client to request the output. There are two statuses, one for the report execution itself, and one for the chosen output format.

The following descriptor shows that the report is still running (<status>execution</status>).

The following descriptor shows that the report was placed in the report execution queue ("status":"queued"):

```
<reportExecution>
    <currentPage>1</currentPage>
    <exports>
        <export>
            <id>html</id>
            <status>queued</status>
        </export>
    </exports>
    <reportURI>/supermart/details/CustomerDetailReport</reportURI>
    <requestId>f3a9805a-4089-4b53-b9e9-b54752f91586</requestId>
    <status>execution</status>
</reportExecution>
```

```
{
    "requestId":"9ecf5c6f-b70d-4170-8a3b-b305db4c2253",
    "reportURI":"/samples/reports/chartthemes/ChartThemesReport",
    "status":"queued"
}
```

The value of async in the request determines if the report output is available after receiving response. Your client should implement either synchronous or asynchronous processing of the response depending on the value you set for the async property.

Set the async property

Polling Report Execution

When requesting reports asynchronously, use the following method to poll the status of the report execution. The request ID in the URL is the one returned in the reportExecution descriptor.

This service supports the extended status value that includes an appropriate message.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/status/ http://<host>:<port>/jrio/rest_v2/reportExecutions/requestID/status/

Options	Sample Return Value
accept: application/xml (default)	`<status>ready</status>`

accept: application/status+xml	`<status>`
	`  <errorDescriptor>`
	
	`<errorCode>input.controls.validation.error</errorCode>`
	`  <message>Input controls validation failure</message>`
	`  <parameters>`
	`  <parameter>Specify a valid value for type Integer.`
	`  </parameter>`
	`  </parameters>`
	`  </errorDescriptor>`
	`  <value>failed</value>`
	`</status>`

accept: application/json	`{ "value": "ready" }`

accept: application/status+json	`{`
	`  "value": "failed",`
	`  "errorDescriptor": {`
	`  "message": "Input controls validation failure",`
	`  "errorCode": "input.controls.validation.error",`
	`  "parameters": ["Specify a valid value for type Integer."]`
	`  }`
	`}`

Return Value on Success	Typical Return Values on Failure
200 OK – The content contains the report status, as shown	404 Not Found – When the

above. In the extended format, error reports contain error messages suitable for display.	specified requestID does not exist.

Requesting Page Status

When requesting reports asynchronously, you can also poll the status of a specific page during the report execution. The request ID in the URL is the one returned in the reportExecution descriptor.

When requesting reports asynchronously, you can also poll the status of a specific page during the report execution. The `executionId` in the URL is the one returned in the `reportExecution` descriptor. This service returns a response containing `reportStatus`, `pageFinal`, and `pageTimestamp` attributes.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/pages/pageNumber/status http://<host>:<port>/jrio/rest_v2/reportExecutions/<executionId>/pages/<pageNumber>/status

Options	Sample Response Content
accept: application/status+json	`{ "reportStatus": "ready", "pageTimestamp": "0", "pageFinal": "true" }`

Return Value on Success	Typical Return Values on Failure
200 OK – The content contains the page status, as shown above.	404 Not Found – When the request ID specified in the request does not exist.

Requesting Report Execution Details

Once the report is ready, your client must determine the names of the files to download by requesting the reportExecution descriptor again. Specify the requestID in the URL as follows:

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID http://<host>:<port>/jrio/rest_v2/reportExecutions/requestID
Options	
accept: application/xml accept: application/json	
Return Value on Success	Typical Return Values on Failure
200 OK – The content contains a ReportExecution descriptor. See below for an example.	404 Not Found – When the request ID specified in the request does not exist.

The reportExecution descriptor now contains the list of exports for the report, including the report output itself and any other file attachments. File attachments such as images and JavaScript occur only with HTML export.

```
{
    "status": "ready",
    "totalPages": 47,
    "requestId": "cce63cba-1685-4708-ba4f-b70f277a1cd5",
    "reportURI": "/public/Samples/Reports/AllAccounts",
    "exports": [
        {
        "status": "ready",
        "outputResource": {
            "contentType": "text/html,"
            "output final": true,
            "outputTimestamp": 0
        },
        "id": "db9acf02-8add-4196-9ab5-ce86844bfc2e",
```

```
        "attachments": [

           {
                "contentType": "image/png",
                "fileName": "img_0_0_0"
            }
        ]
    }
```

```
{
    "status": "ready",
    "totalPages": 47,
    "requestId": "b487a05a-4989-8b53-b2b9-b54752f998c4",
    "reportURI": "/reports/samples/AllAccounts",
    "exports": [{
        "id": "195a65cb-1762-450a-be2b-1196a02bb625",
        "options": {
            "outputFormat": "html",
            "attachmentsPrefix": "./images/",
            "allowInlineScripts": false
        },
        "status": "ready",
        "outputResource": {
            "contentType": "text/html"
        },
        "attachments": [{
            "contentType": "image/png",
            "fileName": "img_0_46_0"
        },
        {
            "contentType": "image/png",
            "fileName": "img_0_0_0"
        },
        {
            "contentType": "image/jpeg",
            "fileName": "img_0_46_1"
        }]
    },
    {
        "id": "4bac4889-0e63-4f09-bbe8-9593674f0700",
        "options": {
            "outputFormat": "html",
            "attachmentsPrefix": "{contextPath}/rest_
v2/reportExecutions/{reportExecutionId}/exports/
{exportExecutionId}/attachments/",
            "baseUrl": "http://localhost:8080/jrio",
```

```
            "allowInlineScripts": true
        },
        "status": "ready",
        "outputResource": {
            "contentType": "text/html"
        },
        "attachments": [{
            "contentType": "image/png",
            "fileName": "img_0_0_0"
        }]
    }]
}
```

When exporting a chart report to HTML, the image produced for the chart is a part of HTML, and can be in two formats - JavaScript or SVG:

- When "interactive" is set to *true*, it is embedded as JavaScript in HTML that uses Highcharts js to render the chart.
- When "interactive" is set to *false*, the chart image is embedded as SVG as part of HTML.

When the option *net.sf.jasperreports.force.html.embed.image=false in WEB-INF/classes/jasperreports.properties* in combination with *interactive=false*, this puts the SVG images into attachments instead of HTML.

Requesting Report Output

After requesting a report execution and waiting synchronously or asynchronously for it to finish, your client is ready to download the report output.

Every export format of the report has an ID that is used to retrieve it. For example, the HTML export in the previous example has the ID 195a65cb-1762-450a-be2b-1196a02bb625. To download the main report output, specify this export ID in the following method:

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_ v2/reportExecutions/requestID/exports/ exportID/outputResource http://<host>:<port>/jrio/rest_v2/reportExecutions/requestID/exports/

exportID/outputResource	

Return Value on Success	Typical Return Values on Failure
200 OK – The content is the main output of the report. The format specified by the contentType property of the outputResource descriptor. For example: text/html	400 Bad Request – When invalid values are provided for export options in the request body. 404 Not Found – When the request ID specified in the request does not exist.

For example, to download the main HTML of the report execution response above, use the following URL:

GET http://localhost:8080/jasperserver-pro/rest_v2/reportExecutions/b487a05a-4989-8b53-b2b9-b54752f998c4/exports/195a65cb-1762-450a-be2b-1196a02bb625/outputResource

GET http://localhost:8080/jrio/rest_v2/reportExecutions/b487a05a-4989-8b53-b2b9-b54752f998c4/exports/195a65cb-1762-450a-be2b-1196a02bb625/outputResource

JasperReports Server does not support exporting Highcharts charts with background images to PDF, ODT, DOCX, or RTF formats. When exporting or downloading reports with Highcharts that have background images to these formats, the background image is removed from the chart. The data in the chart is not affected.

JasperReports IO does not support exporting Highcharts charts with background images to PDF, ODT, DOCX, or RTF formats. When exporting or downloading reports with Highcharts that have background images to these formats, the background image is removed from the chart. The data in the chart is not affected.

To download file attachments for HTML output, use the following method. Download all attachments to display the HTML content properly. The given URL is the default path, but it can be modified with the attachmentsPrefix property in the reportExecutionRequest, as described in Running a Report Asynchronously.

Method	URL

GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/exports/exportID/attachments/fileName http://<host>:<port>/jrio/rest_v2/reportExecutions/requestID/exports/exportID/attachments/fileName

Return Value on Success	Typical Return Values on Failure
200 OK – The content is the attachment in the format specified in the contentType property of the attachment descriptor, for example: image/png	404 Not Found – When the request ID specified in the request does not exist.

For example, to download the one of the images for the HTML report execution response above, use the following URL:

GET http://localhost:8080/jasperserver-pro/rest_v2/reportExecutions/912382875_1366638024956_2/exports/html/attachments/img_0_46_0

GET http://localhost:8080/jrio/rest_v2/reportExecutions/912382875_1366638024956_2/exports/html/attachments/img_0_46_0

Requesting Report Bookmarks

Some reports have additional meta-information associated with them, such as bookmarks and indexes of report sections or parts. Clients can use this information to create a table of contents for the report. There are links to bookmarks and parts defined in the report in the table of contents. After running a report, you can request this information using the same request ID.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/info http://<host>:<port>/jrio/rest_v2/reportExecutions/{executionId}/info

Options	Sample Response Content

accept: application/json accept: application/xml	A structure that contains bookmarks and report parts, as shown below.

Return Value on Success	Typical Return Values on Failure
200 OK – The content contains the report meta-information, as shown below.	404 Not Found – When the request ID specified in the request does not exist.

Example of a request URL:

```
https://localhost:8080/jasperserver[-pro]/rest_
v2/reportExecutions/70b9b169-1c0e-431c-b8bc-a6f49328bc75/info
```

JSON:

```
{
  "bookmarks": {
    "id": "bkmrk_1058907116",
    "type": "bookmarks",
    "bookmarks": [
      {
        "label": "USA shipments",
        "pageIndex": 22,
        "elementAddress": "0",
        "bookmarks": [
          {
            "label": "Albuquerque",
            "pageIndex": 22,
            "elementAddress": "4",
            "bookmarks": null
          },
          {
            "label": "Anchorage",
            "pageIndex": 23,
            "elementAddress": "116",
            "bookmarks": null
          },
          ...
        ]
      }
    ]
  },
```

```
  "parts": {
    "id": "parts_533304192",
    "type": "reportparts",
    "parts": [
      {
        "idx": 0,
        "name": "Table of Contents"
      },
      {
        "idx": 3,
        "name": "Overview"
      },
      {
        "idx": 22,
        "name": "USA shipments"
      }
    ]
  }
}
```

Exporting a Report Asynchronously

After running a report and downloading its content in a given format, you can request the same report in other formats. As with exporting report formats through the user interface, the report does not run again because the export process is independent of the report.

Method	URL
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/exports/ http://<host>:<port>/jrio/rest_v2/reportExecutions/requestID/exports/

Content-Type	Content
application/xml application/json	Send an export descriptor in either XML or JSON format to specify the format and details of your request. For example: `<export>`

```
    <outputFormat>html</outputFormat>
    <pages>10-20</pages>
    <attachmentsPrefix>./images/</attachmentsPrefix>
</export>
```

Send an export descriptor in JSON format to specify the format and details of your request. For example:

```
{
    "outputFormat": "html",
    "pages": "10-20",
    "attachmentsPrefix": "./images/"
}
```

Options
accept: application/xml (default) accept: application/json

Return Value on Success	Typical Return Values on Failure
200 OK – The content contains an exportExecution descriptor. See below for an example.	404 Not Found – When the request ID specified in the request does not exist.

The following example shows the exportExecution descriptor that the server sends in response to the export request:

```
{
    "id":"6b7ce8fa-f1d7-4d53-9af6-4569edb05d1b",
    "status":"queued"
}
```

```
<exportExecution>
    <id>html;attachmentsPrefix=./images/</id>
    <status>ready</status>
    <outputResource>
        <contentType>text/html</contentType>
    </outputResource>
</exportExecution>
```

Modifying Report Parameters

You can update the report parameters, also known as input controls, through a separate method before running a report execution again. For more operations with input controls, see The inputControls Service.

You can update the report parameters, also known as input controls, through a separate method before running an existing report execution again. Use the following method to reexecute the report with a different set of parameter values:

Method	URL	
POST	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/parameters http://<host>:<port>/jrio/rest_v2/reportExecutions/requestID/parameters	
Argument	Type/Value	Description
freshData	true	When data snapshots are enabled, you must set this to true to force the server to get fresh data when you change parameters. This overrides the default value of false, as explained in the table of properties in Running a Report Asynchronously.
Media-Type		Content

application/json		[
		{
		"name":"someParameterName",
		"value":["value 1", "value 2"]
		},
		{
		"name":"someAnotherParameterName",
		"value":["another value"]
		}
		]

application/xml	<reportParameters> <reportParameter name="Country_multi_select"> <value>Mexico</value> </reportParameter> <reportParameter name="Cascading_state_multi_select"> <value>Guerrero</value> <value>Sinaloa</value> </reportParameter> </reportParameters>

Return Value on Success	Typical Return Values on Failure
204 No Content – There is no content to return.	404 Not Found – When the request ID specified in the request does not exist.

Polling Export Execution

As with the execution of the main report, you can also poll the execution of the export process. This service supports the extended status value that includes an appropriate message.

As with the execution of the main report, you can also poll the execution of the export process. This service supports the extended status value that includes an appropriate message.

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/exports/exportID/status http://<host>:<port>/jrio/rest_v2/reportExecutions/requestID/exports/exportID/status

Options	Sample Return Value
accept: application/xml (default)	`<status>ready</status>`

accept: application/status+xml	`<status>`
	`  <errorDescriptor>`
	
	`<errorCode>input.controls.validation.error</errorCode>`
	`  <message>Input controls validation failure</message>`
	`  <parameters>`
	`  <parameter>Specify a valid value for type Integer.</parameter>`
	`  </parameters>`
	`  </errorDescriptor>`
	`  <value>failed</value>`
	`</status>`

accept: application/json	`{ "value": "ready" }`

accept: application/status+json	`{`
	`  "value": "failed",`
	`  "errorDescriptor": {`
	`  "message": "Input controls validation failure",`
	`  "errorCode": "input.controls.validation.error",`
	`  "parameters": ["Specify a valid value for type Integer."]`
	`  }`
	`}`

Return Value on Success	Typical Return Values on Failure
200 OK – The content contains the export status, as shown above. In the extended format, error reports contain error messages suitable for display.	404 Not Found – When the specified request ID does not exist.

For example, to get the status of the HTML export in the previous example, use the following URL:

GET http://localhost:8080/jasperserver-pro/rest_v2/reportExecutions/912382875_1366638024956_2/exports/195a65cb-1762-450a-be2b-1196a02bb625/status

GET http://localhost:8080/jrio/rest_v2/reportExecutions/912382875_1366638024956_2/exports/195a65cb-1762-450a-be2b-1196a02bb625/status

When the status is "ready", the client can download the new export output and any attachments as described in Requesting Report Output. For example:

GET http://localhost:8080/jasperserver-pro/rest_v2/reportExecutions/912382875_1366638024956_2/exports/195a65cb-1762-450a-be2b-1196a02bb625/outputResource

GET http://localhost:8080/jasperserver-pro/rest_v2/reportExecutions/912382875_1366638024956_2/exports/195a65cb-1762-450a-be2b-1196a02bb625/images/img_0_46_0

GET http://localhost:8080/jrio/rest_v2/reportExecutions/912382875_1366638024956_2/exports/195a65cb-1762-450a-be2b-1196a02bb625/outputResource

GET http://localhost:8080/jrio/rest_v2/reportExecutions/912382875_1366638024956_2/exports/195a65cb-1762-450a-be2b-1196a02bb625/images/img_0_46_0

Finding Running Reports and Jobs

The reportExecutions service provides a method to search for reports that are running on the server. It includes asynchronous reports that are still running and those that are finished but in cache and available by their request ID.

The search for reports also includes report jobs triggered by the scheduler, both running and finished but still in the cache.

To search for running or finished reports, use the search arguments with the following URL:

Method	URL
GET	http://<host>:<port>/jasperserver[-pro]/rest_

	v2/reportExecutions?<arguments>	
Argument	Type/Value	Description
reportURI	Optional String	This string matches the repository URI of the running report, relative to the currently logged-in user's organization.
jobID	Optional String	For scheduler jobs, this argument matches the ID of the job that triggered the running report.
jobLabel	Optional String	For scheduler jobs, this argument matches the name of the job that triggered the running report.
userName	Optional String	For scheduler jobs, this argument matches the user ID that created the job.
fireTimeFrom	Optional Date/Time	For scheduler jobs, the fire time arguments define a range of time. You can check if the current job was triggered during this time. You can specify either or both of the arguments. Specify the date and time in the following pattern: yyyy-MM-dd'T'HH:mmZ.
fireTimeTo		
Options		
accept: application/xml (default) accept: application/json		
Return Value on Success		Typical Return Values on Failure
200 OK – The content is a descriptor for each of the matching results.		204 No Content – When the search results are empty.

The response contains a list of summary reportExecution descriptors, for example in XML:

```
<reportExecutions>
    <reportExecution>

<reportURI>repo:/supermart/details/CustomerDetailReport</reportURI>
        <requestId>2071593484_1355224559918_5</requestId>
```

```
    </reportExecution>
</reportExecutions>
```

Given the request ID, you can obtain more information about each result by downloading the full reportExecution descriptor, as described in Requesting Report Execution Details.

For security purposes, the search for running reports has the following restrictions:

- The system administrator (superuser) can see and cancel any report running on the server.
- An organization admin (jasperadmin) can see every running report, but can cancel only the reports that were started by a user from the same or child organization.
- A regular user can see every running report, but can cancel only the reports that he initiated.

Stopping Running Reports and Jobs

To stop a running report and cancel its output, use the PUT method and set the status as cancelled.

Method	URL
PUT	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID/status/ http://<host>:<port>/jrio]/rest_v2/reportExecutions/requestID/status/

Content-Type	Content
application/xml application/json	Send a status descriptor in JSON format with the value cancelled. For example: Send a status descriptor in either XML or JSON format with the value cancelled. For example: XML: <status>cancelled</status> JSON: { "value": "cancelled" }

Options	
accept: application/xml (default) accept: application/json	
Return Value on Success	Typical Return Values on Failure
200 OK – When the report execution was successfully stopped, the server replies with the same status: XML: <status>cancelled</status> JSON: { "value": "cancelled" } { "value": "cancelled" } 204 No Content – When the report specified by the request ID is not running, either because it finished running, failed, or was stopped by another process.	404 Not Found – When the request ID specified in the request does not exist.

Removing a Report Execution

Deleting a report that has run removes it from the cache and makes its output no longer available. If the report execution is still running, it is stopped automatically then removed.

Method	URL
DELETE	http://<host>:<port>/jasperserver[-pro]/rest_v2/reportExecutions/requestID http://<host>:<port>/jrio/rest_v2/reportExecutions/<executionID>
Return Value on Success	Typical Return Values on Failure
204 No Content – The report execution was successfully removed.	404 Not Found – When the request ID specified in the request does not exist.

API Reference - Visualize.js

The JavaScript API exposed through Visualize.js allows you to embed and dynamically interact with reports. With Visualize.js, you can create web pages and web applications that seamlessly embed reports and complex interaction. You can control the look and feel of all elements through CSS and invent new ways to merge data into your application. Visualize.js helps you make advanced business intelligence available to your users.

JavaScript API Reference - jrio.js

The JavaScript API exposed through jrio.js allows you to embed reports into your web pages and web applications. The embedded elements are fully interactive, either through the UI or programmatically. Users navigate their data in the context of your app, and you can dynamically compute, update, or render the jrio.js elements to create seamless interaction. You can use JavaScript frameworks for layout and control the look and feel of all elements through style sheets (CSS).

With the JavaScript API, you can invent new ways to merge data into your application, and make advanced business intelligence available to your users.

This chapter contains the following sections:

Each function of Visualize.js is then described in the following chapters:

- JavaScript API Reference - report
- API Reference - inputControls
- API Reference - dashboard
- JavaScript API Reference - Errors

The last chapters demonstrate more advanced usage of Visualize.js:

- JavaScript API Usage - Report Events
- JavaScript API Usage - Hyperlinks
- JavaScript API Usage - Interactive Reports
- Visualize.js Tools

This chapter contains the following sections:

- Loading the jrio.js Script
- Configuring the JasperReports IO Client
- Usage Patterns
- Testing Your JavaScript
- Changing the Look and Feel

Each function of the JasperReports IO JavaScript API is then described in the following chapters:

- JavaScript API Reference - report
- JavaScript API Reference - Errors

The last chapters demonstrate more advanced usage of the JasperReports IO JavaScript API:

- JavaScript API Usage - Report Events
- JavaScript API Usage - Hyperlinks
- JavaScript API Usage - Interactive Reports

Loading the jrio.js Script

The script to include on your HTML page is named jrio.js. It is located on your running instance of the JasperReports® IO JavaScript API distribution, which is available for download and can be deployed in your hosting web application. Later on your page, you also need a container element to display the report from the script.

```
<!-- Provide the URL to jrio.js -->
<script src="http://bi.example.com:8080/jrio-client/optimized-
scripts/jrio/jrio.js"></script>
...
<!-- Provide a container for the report -->
<div id="reportContainer"></div>
```

The content of jrio.js is type='text/javascript', but that is the default so you usually don't need to include it.

Requesting the Visualize.js Script

The script to include on your HTML page is named visualize.js. It is located on your running instance of JasperReports® Server. Later on your page, you also need a container element to display the report from the script.

```
<!-- Provide the URL to visualize.js -->
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>
...
<!-- Provide a container for the report -->
<div id="container"></div>
```

The content of visualize.js is type='text/javascript', but that is the default so you usually don't need to include it.

You can specify several parameters when requesting the script:

Parameter	Type or Value	Description
userLocale	locale string	Specify the locale to use for display and running reports. It must be one of the locales supported by JasperReports Server. The default is the locale configured on the server.
logEnabled	true|false	Enable or disable logging. By default, it is enabled (true).
logLevel	debug|info|warn|error	Set the logging level. By default the level is error.
baseUrl	URL	The URL of the JasperReports Server that will respond to visualize requests. By default, it is the same server instance that provides the script.

The following request shows how to use script parameters:

```
<script src="http://bi.example.com:8080/jasperserver-pro/client/
             visualize.js?userLocale=fr&logLevel=debug"></script>
```

Configuring the JasperReports IO Client

Loading the jrio.js script as above gives you access to the JasperReports IO JavaScript API in your web page.

But this JasperReports IO client-side API needs to be configured to point to an existing JasperReports IO REST reporting service URL which delivers the actual reports output to be displayed on the current page, the location of the API scripts (either optimized or non-optimized) and UI theme.

This is achieved by calling the config() function on the jrio object made available globally on the page by the loading of the jrio.js script:

```
jrio.config({
    server : "http://bi.example.com:8080/jrio",
    scripts : "http://bi.example.com:8080/jrio-client/optimized-scripts",
    theme: {
        href: "http://bi.example.com:8080/jrio-client/themes/default"
    },
    locale: "en_US"
});
```

You can specify several parameters when requesting the script:

Parameter	Type or Value	Description
server	URL	The URL to the JasperReports IO service that responds to the report generating REST requests. This parameter is required.
scripts	URL	The URL to the folder containing the JasperReports IO JavaScript API files, either in optimized or non-optimized format. This parameter is required.
theme	URL	The URL to the folder containing the JasperReports IO JavaScript UI theme files.
locale	locale string	Specify the locale to use for display and running reports. It must be one of the locales supported by JasperReports IO. The default is the locale configured on the server. This parameter is required.
logEnabled	true|false	Enable or disable logging. By default, it is disabled (false).
logLevel	debug|info|warn|error	Set the logging level. By default the level is error.

The server, scripts, and locale parameters are required. The jrio object may produce errors if they are not set.

The scripts making up the JasperReports IO JavaScript API are available in two formats: optimized and non-optimized. They are placed in separate folders in the JasperReports IO JavaScript API distribution under /optimized-scripts and /scripts subfolders respectively.

If you notice undesirable side-effects when including the JasperReports IO JavaScript library, change the client configuration to use the optimized scripts to provide better protection, also known as encapsulation. For example, the JasperReports IO JavaScript API functions might interfere with collapse functions on your menus. Non-optimized scripts are preferred when you want to perform some runtime debugging for the JavaScript code.

If you want to used the optimized jrio.js script, use the following URL to load it: <script src="http://bi.example.com:8080/myapp/jriojsapi/optimized-scripts/jrio/jrio.js"></script>

If you want to use the non-optimized jrio.js script, you will have to use all of the following scripts:

```
<script src="http://bi.example.com:8080/myapp/jriojsapi/scripts/bower_
components/requirejs/require.js"></script>
<script
src="http://bi.example.com:8080/myapp/jriojsapi/scripts/require.config.js"
></script>
<script src="http://bi.example.com:8080/jrio-
client/scripts/jrio/loader/jasper.js"></script>
<script src="http://bi.example.com:8080/jrio-
client/scripts/jrio/jrio.js"></script>
<script>
    require.config({
        baseUrl: "http://bi.example.com:8080/myapp/jriojsapi/scripts"
    });
</script>
```

Contents of the Visualize.js Script

The Visualize.js script itself is a factory function for an internal JrsClient.

```
/**
 *  Establish connection with JRS instance and generate
 *  ready to use client
 *  @param {Object} properties - configuration to connect to JRS instance
 *  @param {Function} callback - optional, successful callback
 *  @param {Function} errorback - optional, invoked on error
 *  @param {Function} always - optional, invoked always
 */
function visualize(properties, callback, errorback, always){}

/**
 *  Store common configuration, to share them between visualize calls
 *  @param {Object} properties - configuration to connect to JRS instance
 */
function visualize.config(properties);
```

You write JavaScript in a callback that controls what the client does. The following code sample shows the functions of the JrsClient that are available to you:

```
{    /**
     * Perform authentification with provided auth object
     * @param auth {object}  - auth properties
     */
    login : function(auth){},
```

```
    /**
     * Destroy current auth session
     */
    logout : function() {},

    /**
     * Create and run report component with provided properties
     * @param properties {object}  - report properties
     * @returns {Report} report - instance of Report
     */
    report : function(properties){},

    /**
     * Create and run controls for provided controls properties
```

```
     * @param properties {object}  - input controls properties
     * @returns {Options} inputControls instance
     *
     */
    inputControls : function(properties){},

    /**
     * Create and run resource search component for provided properties
     * @param properties {object}  - search  properties
     * @returns {Options} resourcesSearch instance
     *
     */
    resourcesSearch : function(properties){}
}
```

These functions are described in the remaining API reference chapters.

Usage Patterns

After configuring the JasperReports IO client object, you write the callback that will execute inside this client provided by jrio.js.

```
jrio.config({
    server : "http://bi.example.com:8080/jrio",
    scripts : "http://bi.example.com:8080/myapp/jriojsapi/optimized-
scripts",
    theme: {
        href: "http://bi.example.com:8080/myapp/jriojsapi/themes/default"
    },
    locale: "en_US"
});
jrio(function(jrioClient) {
    jrioClient.report({
        resource: "/samples/reports/highcharts/HighchartsChart",
        container: "#reportContainer",
        error: function(err) {
            alert(err);
        },
```

```
    });
});
```

After specifying the authentication information, you write the callback that will execute inside the client provided by visualize.js.

```
visualize({
    server: "http://bi.example.com",
    auth: {
        name : "joeuser",
        password: "joeuser"
    }
}, function(v){

    //'v' is a client to JRS instance under  "http://bi.example.com"
    //session established for joeuser/joeuser

    var report = v.report(...);

}, function(err){
    alert(err.message);
});
```

The following sample shows how to use the always callback:

```
visualize({
    server: "http://bi.example.com",
    auth: {
        name : "joeuser",
        password: "joeuser"
    }
    }).always(function(err, v) {

        if (err) {
        alert(err.message);
        return;
        }
```

```
    v.report(); // use v here
});
```

As of JasperReports Server 7.9, the server can provide the code to embed any report, dashboard, or Ad Hoc view. This means you can browse your server's repository, choose the dashboard or Ad Hoc View that you want to embed, preview it, and then select the </> icon in the toolbar to view the sample code. For more information, see:

- Getting the Embed Code of a Report
- Getting the Embed Code for a Dashboard
- Getting the Embed Code for an Ad Hoc View

Testing Your JavaScript

As you learn to use Visualize.js the JasperReports IO JavaScript API and write the JavaScript that embeds your reports into your web app, you should have a way to run and view the output of your script.

In order to load Visualize.js jrio.js, your HTML page containing your JavaScript must be accessed through a web server. Opening a static file with a web browser does not properly load the iframes needed by the script.

One popular way to view your Visualize.js JasperReports IO output, is to use the jsFiddle online service. You specify your HTML, JavaScript, and optional CSS in 3 separate frames, and the result displays in the fourth frame.

Another way to test your JavaScript is to use the app server bundled with JasperReports Server. If you deploy the server from the installer with the Apache Tomcat app server, you can create an HTML file at the root of the server web app, for example:

<js-install>/apache-tomcat/webapps/jasperserver-pro/testscript.html

Another way to test your JavaScript is to use the app server bundled with JasperReports IO. If you deploy the server from the installer with the Jetty web application server, you can create an HTML file at the root of one of the web apps shipped with it by default, for example:

<jrio-install>/jrio/webapps/jrio-docs/testscript.html

Write your HTML and JavaScript in this file, and then you can run Visualize.js by loading the file through the following URL:

http://mydomain.com:8081/jasperserver-pro/testscript.html

Write your HTML and JavaScript in this file, and then you can run jrio.js by loading the file through the following URL:

http://mydomain.com:8081/jrio-docs/testscript.html

Changing the Look and Feel

When you create a web application that embeds Visualize.jsJasperReports IO content, you determine the look and feel of your app through layout, styles, and CSS (Cascading Style Sheets). Most of the content that you embed consists of reports and dashboards that you create in JasperReports Server with JasperReports IO or Jaspersoft® Studio, where you set the appearance of colors, fonts, and layout to match your intended usage.

But some Visualize.jsJasperReports IO JavaScript API elements also contain UI widgets that are generated by the server in a default style, for example the labels, buttons, and selection boxes for the input controls of a report. In general, the default style is meant to be neutral and embeddable in a wide range of visual styles. If the default style of these UI widgets does not match your app, there are two approaches described in the following sections:

- Customizing the UI with CSS – You can change the appearance of the UI widgets through CSS in your app.
- Customizing the UI with Themes – You can redefine the default appearance of the UI widgets in themes on the server.

Customizing the UI with CSS

The UI widgets generated by the server have CSS classes and subclasses, also generated by the server, that you can redefine in your app to change their appearance. To change the appearance of the generated widgets, create CSS rules that you would add to CSS files in your own web app. To avoid the risk of unintended interference with other CSS rules, you should define your CSS rules with both a classname and a selector, for example:

```
#inputContainer .jr-mInput-boolean-label {
    color: #218c00;
}
```

To change the style of specific elements in the server's generated widgets, you can find the corresponding CSS classes and redefine them. To find the CSS classes, write the JavaScript display the UI widgets, for example input controls, then test the page in a browser. Use your browser's code inspector to look at each element of the generated widgets and locate the CSS rules that apply to it. The code inspector shows you the classes and often lets you modify values to preview the look and feel that you want to create. For more information, see Embedded Input Control Styles.

Customizing the UI with Themes

You can redefine the default appearance of the UI widgets in themes on the server.

Themes are CSS in the JasperReports IO JavaScript API. The UI widgets in JasperReports IO elements are generated on the server and their look and feel is ultimately determined by themes.

The UI widgets in Visualize.js elements are generated on the server and their look and feel is ultimately determined by themes. Themes are CSS files defined for organizations, and thus appearances are determined by the user credentials submitted by your Visualize.js instance. Certain visual properties of the UI widgets embedded in your app by Visualize.js can be modified by changing themes on the server. For more information about themes, see the JasperReports Server Administrator Guide.

However, not all properties of a UI widget can be modified through themes. If you want to create a fully custom visualization, you might need to access raw data through the Visualize.js APIs and write your own elements and CSS to display them. For more information, see Exporting Data From a Report.

Cross-Domain Security

After developing your web application and embedding Visualize.js, you will deploy it to your users through some web domain, for example bi-enabled.myexample.com. For testing, JasperReports Server accepts Visualize.js requests from any domain, and your web

app will display embedded reports and dashboards. However, once you go into production, you should add security and make sure that the server responds only to Visualize.js requests from your web app. This is called cross-site request forgery (CSRF) protection.

JasperReports Server includes a configurable whitelist of domains, and it only responds to Visualize.js requests from those domains. Add your web domain to this whitelist, and no other web apps on malicious domains can access your server. In addition to whitelisted domains, JasperReports Server always responds to requests that originate from the same domain as the server itself.

To configure the cross-domain whitelist

1. Log in as system administrator (superuser).
2. Select Manage > Server Settings then Server Attributes.
3. The server attribute named domainWhitelist contains a regular expression that matches allowed domains. Set it as follows:
    - When your Visualize.js web app is on another domain, such as in this example, create a regular expression to match the protocol, domain name and port numbers. You can also match multiple subdomains or several port numbers as in this example:
      domainWhitelist = http://*.myexample.com:80\d0
      The server translates this simplified expression into the proper regular expression ^http://.*\.myexample\.com:80\d0$. If you want to avoid the translation, put ^ $ around your value.
    - When your Visualize.js web app is on the same domain as your JasperReports Server set the value to <blank> (no value) so that no other domain has access:
      domainWhitelist = <blank>
    - It is also possible to limit the domains accessing the JasperReports Server for each organization. For advanced configuration details, see the section on cross-domain whitelists in the JasperReports Server Security Guide.

For more information about setting attributes see the JasperReports Server Administrator Guide.

Cross-Site Errors

Even when the domain whitelist is configured, some browsers may limit cross-site access for security reasons. In particular, the Safari browser often blocks access to the Visualize.js script because it uses third-party cookies to enable cross-site access.

There are several workarounds in this case:

- Use a different browser. Other browsers may allow access when Safari does not. However, in the future, other browsers may also begin to restrict access to third-party cookies that enable cross-site access.
- Use a proxy to make JasperReports Server appear to be on the same domain as your website. If you have already embedded Visualize.js in a cross-site way, changing to proxying will require changes to your environment (implementing and configuring a proxy service) and to your application.
- Use the same domain but different sub-domains for your application and JasperReports Server, for example:
    - www.myapp.com for your application
    - jaspersoft.myapp.com for JasperReports Server serving the Visualize.js script

When accessing JasperReports Server application resources from another domain, Chrome will treat it as a CORS request. To make this work, both applications (requestor and JasperReports Server) should communicate via HTTPS protocol:

- Server should use HTTPS.
- Client-side application should also use HTTPS.
- For XMLHttpRequest, you need to set *withCredentials=true*

JasperReports Server will send *Secure;HttpOnly Cookies* attributes.

The Chrome browser accepts cookies only with these attributes (ignoring Path=/jasperserver-pro, so it works for CORS requests).

Enabling Private Network Access

When private network access is blocked for AJAX calls from public networks, you can enable private network access by configuring JasperReports Server to send an Access-Control-Allow-Private-Network: true header back to the browser.

To enable private network access

1. Find tomcat/webapps/jasperserver-pro/WEB-INF/applicationContext-security-web.xml
2. Find <bean id="corsProcessor" class="com.jaspersoft.jasperserver.api.security.csrf.JSCorsProcessor">
3. In that bean under <property name="headerUrlPatterns">, uncomment:

```
<entry key="Access-Control-Allow-Private-Network">
       <util:list>
               <value>.*</value>
       </util:list>
</entry>
```

4. And under <property name="headerValues">, uncomment:

```
<<entry key="Access-Control-Allow-Private-Network">
       <util:list>
               <value>true</value>
       </util:list>
</entry>
```

Note that in the property *headerUrlPatterns*, you can set regex which will decide to which URL requests these headers should be set, by default there is .* which means that this header will be set for every request.

Serving Visualize.js from a CDN

This feature allows you to serve Visualize.js scripts from the static host or a CDN.

Setup

To set up:

1. Copy the following folder: *<tomcat>/webapps/jasperserver-pro/scripts/visualize/* (or if you are building Visualize from the source code: *<jasperserver-ui>/pro/jrs-ui-pro/build/overlay/scripts/visualize/*) to the location which will be used as a static resources source.
2. Set up your static resources server or CDN to serve content of this folder.

Using Visualize.js from a CDN

Assumuptions:

- The content of the visualize.js folder described above is available at the following URL: *https://somehost.com/visualize*
- The JasperReports Server instance is available at the URL: *https://jrshost.bi/jasperserver-pro*

The following sample shows how to use static Visualize.js:

```
<script src="https://somehost.com/visualize/visualize.js"></script>
 <div id="container"></div>
<script>
  visualize({
    publicPath: "https://somehost.com/visualize/",
    server: "https://jrshost.bi/jasperserver-pro",
    auth: {
      name: "superuser",
      password: "superuser"
    }
  }, function (v) {
```

```
    v.dashboard({
      resource: "/public/Samples/Dashboards/1._Supermart_Dashboard",
      container: "#container",
      error: function (e) {
        console.log(e);
      }
    });
  });
</script>
```

- **publicPath** - this property MUST be present if Visualize.js scripts are served from some separate server (not from JRS instance). Its value should be the URL where folder described in a Set Up section. Value should ends with /
- **server** - this property is mandatory when Visualize.js scripts are served from separate server.

You cannot use URL parameters to configure Visualize.js when it is served from a separate server. So in the above example, this will not work:
https://somehost.com/visualize/visualize.js?baseUrl=https://jrshost.bi/jasperserver-pro

How to Avoid Using publicPath

If you are building Visualize.js from the source code, it is possible to hardcode the value of the publicPath property into the Visualize.js scripts during the build, so that the **publicPath** property becomes optional.

To do this, build Visualize.js from the source code (you should set up your build environment by reading the README.md in the source code):

1. Create **.env.loca**l file in *<jasperserver-ui>/pro/jrs-ui-pro* folder.
2. Add the following lines to this file:

```
WEBPACK_INCLUDE_CONFIGS=visualize
WEBPACK_PUBLIC_PATH=https://somehost.com/visualize/
```

3. Build:

```
cd <jasperserver-ui>/pro/jrs-ui-pro
yarn run build
```

4. Use build result from this folder *<jasperserver-ui>/pro/jrs-ui-pro/build/overlay/scripts/visualize/* to serve Visualize.js.

Serving CSS from a CDN

Visualize.js requires a special CSS which is loaded by default from the JRS instance (which is configured by server property).

This allows you to use the themeability feature because the CSS is loaded after authentication process, so the loaded CSS respects the logged in user theme.

But sometimes you might want to serve CSS from the static server (CDN).

Assuming that the CSS is accessible at this url: *https://somehost.com/visualize/css*, then the following CSS files should be accessible:

https://somehost.com/visualize/css/jasper-ui/jasper-ui.css

https://somehost.com/visualize/css/jquery-ui/jquery-ui.css

https://somehost.com/visualize/css/dashboard/canvas.css

https://somehost.com/visualize/css/panel.css

https://somehost.com/visualize/css/webPageView.css

https://somehost.com/visualize/css/pagination.css

https://somehost.com/visualize/css/menu.css

https://somehost.com/visualize/css/simpleColorPicker.css

https://somehost.com/visualize/css/notifications.css

In this case, use the following configuration to use custom CSS files:

```
visualize({
//... other config props
theme: {
href: "https://somehost.com/visualize/css"
```

```
}
}, function (v) {
//.....
});
```

JavaScript API Reference - report

The report function runs reports on JasperReports Server on the JasperReports IO reporting service and displays the result in a container that you provide. This chapter describes how to render a report in Visualize.js. using the JasperReports IO JavaScript API.

The report function also supports more advanced customizations of hyperlinks and interactivity that are described in subsequent chapters:

- JavaScript API Usage - Hyperlinks
- JavaScript API Usage - Interactive Reports

This chapter contains the following sections:

Report Properties

The properties structure passed to the report function is defined as follows:

```
{
    "title": "Report Properties",
    "type": "object",
    "description": "A JSON Schema describing a Report Properties",
    "$schema": "http://json-schema.org/draft-04/schema#",
    "properties": {
        "server": {
            "type": "string",
            "description": "URL of JRS instance."
        },
        "resource": {
            "type": "string",
            "description": "Report resource URI.",
            "pattern": "^/[^/~!#\\$%^|\\s`@&*()\\-+={}\\
[\\]:;\"'<>,?/\\|\\\\]+(/[^/~!#\\$%^|\\s`@&*()\\-+={}\\
[\\]:;\"'<>,?/\\|\\\\]+)*$"
        },
        "container": {
            "oneOf": [
                {
                    "type": "object",
                    "additionalProperties" : true,
                    "description": "DOM element to render report to"
                },
                {
                      "type": "string",
                      "description": "CSS selector for container to render
report to."
                }
            ]
        },
        "params": {
            "type": "object",
            "description": "Report's parameters values",
            "additionalProperties": {
                "type": "array"
            }
        },
        "pages": {
```

```
            "type": ["string", "integer", "object"],
            "description": "Range of report's pages or single report page",
            "pattern": "^[1-9]\\d*(\\-\\d+)?$",
            "properties": {
                "pages": {
                    "type": ["string", "integer"],
                    "description": "Range of report's pages or single
report page",
                    "pattern": "^[1-9]\\d*(\\-\\d+)?$",
                    "default": 1,
                    "minimum": 1
                },
                "anchor": {
                    "type": ["string"],
                    "description": "Report anchor"
                }
            },
            "default": 1,
            "minimum": 1
        },
```

```
        "scale" : {
            "default": "container",
            "oneOf" : [
                {
                    "type": "number",
                    "minimum" : 0,
                    "exclusiveMinimum": true,
                    "description" : "Scale factor"
                },
                {
                    "enum": ["container", "width", "height"],
                    "default": "container",
                    "description" : "Scale strategy"
                }
            ]
        },
        "defaultJiveUi": {
            "type": "object",
            "description": "Default JIVE UI options.",
```

```
            "properties": {
                "enabled": {
                    "type": "boolean",
                    "description": "Enable default JIVE UI.",
                    "default": "true"
                }
            }
        },
        "isolateDom": {
            "type": "boolean",
            "description": "Isolate report in iframe.",
            "default": "false"
        },
        "linkOptions": {
            "type": "object",
            "description": "Report's parameters values",
            "properties": {
                "beforeRender": {
                    "type": "function",
                    "description": "A function to process link - link
element pairs."
                },
                "events": {
                    "type": "object",
                    "description": "Backbone-like events object to be
applied to JR links",
                    "additionalProperties" : true
                }
            }
        },
        "ignorePagination": {
            "type": "boolean",
            "description": "Control if report will be split into separate
pages",
            "default": false
        },
```

```
        "autoresize": {
            "type": "boolean",
            "description": "Automatically resize report on browser window
```

```
resize",
            "default": true
        },
        "chart": {
            "type": "object",
            "additionalProperties": false,
            "description": "Properties of charts inside report",
            "properties": {
                "animation": {
                    "type": "boolean",
                    "description": "Enable/disable animation when report is
rendered or resized. Disabling animation may increase performance in some
cases. For now works only for Highcharts-based charts."
                },
                "zoom": {
                    "enum": [false, "x", "y", "xy"],
                    "description": "Control zoom feature of chart reports.
For now works only for Highcharts-based charts."
                }
            }
        },
        "loadingOverlay": {
            "type": "boolean",
            "description": "Enable/disable report loading overlay",
            "default": true
        },
        "scrollToTop": {
            "type": "boolean",
            "description": "Enable/disable scrolling to top after report
rendering",
            "default": true
        },
        "showAdhocChartTitle": {
            "type": "boolean",
            "description": "Enable/disable showing Ad Hoc chart reports
title",
            "default": true
        }
    },
    "required": ["server", "resource"]
}
```

When setting a container for Visualize.js, the container element must not be a script element, or an element must not contain script tags for the security reasons.

The properties structure passed to the report function is defined as follows:

```
    {
    "title": "Report Properties",
    "type": "object",
    "description": "A JSON Schema describing a Report Properties",
    "$schema": "http://json-schema.org/draft-04/schema#",
    "properties": {
        "server": {
            "type": "string",
            "description": "URL of JRS instance."
        },
        "resource": {
            "type": "string",
            "description": "Report resource URI."
        },
        "container": {
            "oneOf": [
                {
                    "type": "object",
                    "additionalProperties" : true,
                    "description": "DOM element to render report to"
                },
                {
                      "type": "string",
                      "description": "CSS selector for container to render
report to."
                }
            ]
        },
        "params": {
            "type": "object",
            "description": "Report's parameters values",
            "additionalProperties": {
                "type": "array"
            }
        },
        "pages": {
            "type": ["string", "integer", "object"],
            "description": "Range of report's pages or single report page",
```

```
            "pattern": "^[1-9]\\d*(\\-\\d+)?$",
            "properties": {
                "pages": {
                    "type": ["string", "integer"],
                    "description": "Range of report's pages or single
report page",
                    "pattern": "^[1-9]\\d*(\\-\\d+)?$",
                    "minimum": 1
                },
                "anchor": {
                    "type": ["string"],
                    "description": "Report anchor"
                }
            },
            "default": 1,
            "minimum": 1
        },
         "scale" : {
            "default": "container",
            "oneOf" : [
                {
                    "type": "number",
                    "minimum" : 0,
                    "exclusiveMinimum": true,
                    "description" : "Scale factor"
                },
                {
                    "enum": ["container", "width", "height"],
                    "default": "container",
                    "description" : "Scale strategy"
                }
            ]
         },
         "defaultJiveUi": {
            "type": "object",
            "description": "Default JIVE UI options.",
            "properties": {
                "enabled": {
                    "type": "boolean",
                    "description": "Enable default JIVE UI.",
                    "default": "true"
                },
```

```
        "type": "boolean",
            "description": "Control if report will be split into separate
pages"
        },
        "reportLocale": {
            "type": "string",
            "description": "The locale to use for the report"
        },
        "reportTimeZone": {
            "type": "string",
            "description": "The time zone to use for the report"
```

```
        },
        "autoresize": {
            "type": "boolean",
            "description": "Automatically resize report on browser window
resize",
            "default": true
        },
        "chart": {
            "type": "object",
            "additionalProperties": false,
            "description": "Properties of charts inside report",
            "properties": {
                "animation": {
                    "type": "boolean",
                    "description": "Enable/disable animation when report is
rendered or resized. Disabling animation may increase performance in some
cases. For now works only for Highcharts-based charts."
                },
                "zoom": {
                    "enum": [false, "x", "y", "xy"],
                    "description": "Control zoom feature of chart reports.
For now works only for Highcharts-based charts."
                }
            }
        },
        "loadingOverlay": {
            "type": "boolean",
            "description": "Enable/disable report loading overlay",
            "default": true
        },
```

```
        "scrollToTop": {
            "type": "boolean",
            "description": "Enable/disable scrolling to top after report
rendering",
            "default": true
        },
        "showAdhocChartTitle": {
            "type": "boolean",
            "description": "Enable/disable showing Ad Hoc chart reports
title",
            "default": true
        }
    },
    "required": ["server", "resource"]
}
```

Report Functions

The report function exposes the following functions:

```
define(function () {

    /**
     * @param {Object} properties - report properties
     * @constructor
     */
```

```
    function Report(properties){}

    /**
     * Setters and Getters are functions around
     * schema for bi component at ./schema/ReportSchema.json
     * Each setter returns pointer to 'this' to provide chainable API
     */

    /**
     * Get any result after invoking run action, 'null' by default
     * @returns any data which supported by this bi component
     */
    Report.prototype.data = function(){};

    /**
     * Attaches event handlers to some specific events.
     * New events overwrite old ones.
     * @param {Object} events - object containing event names as keys and
event handlers as values
     * @return {Report} report - current Report instance (allows chaining)
     */
    Report.prototype.events = function(events){};

    //Actions

    /**
     * Perform main action for bi component
     * Callbacks will be attached to deferred object.
     * @param {Function} callback - optional, invoked in case of successful
run
     * @param {Function} errorback - optional, invoked in case of failed
run
     * @param {Function} always - optional, invoked always
```

```
 /**
     * Render report to container, previously specified in property.
     * Clean up all content of container before adding Report's content
     * @param {Function} callback - optional, invoked in case successful
export
     * @param {Function} errorback - optional, invoked in case of failed
export
     * @param {Function} always - optional, optional, invoked always
     * @return {Deferred} dfd
     */
    Report.prototype.render = function(callback, errorback, always){};

    /**
     * Refresh report execution
     * @param {Function} callback - optional, invoked in case of successful
refresh
     * @param {Function} errorback - optional, invoked in case of failed
refresh
     * @param {Function} always - optional, invoked optional, invoked
always
     * @return {Deferred} dfd
     */
    Report.prototype.refresh = function(callback, errorback, always){};

    /**
     * Cancel report execution
     * @param {Function} callback - optional, invoked in case of successful
cancel
     * @param {Function} errorback - optional, invoked in case of failed
cancel
     * @param {Function} always - optional, invoked optional, invoked
always
     * @return {Deferred} dfd
     */
    Report.prototype.cancel = function(callback, errorback, always){};
```

```
return{Deferred} dfd
     */
    Report.prototype.save = function(callback, errorback, always){};

    /**
     * Save JIVE components state as new report
     * @param {Object} options - resource information (i.e. folderUri,
label, description, overwrite flag)
     * @param {Function} callback - optional, invoked in case of successful
update
     * @param {Function} errorback - optional, invoked in case of failed
update
     * @param {Function} always - optional, invoked optional, invoked
always
     * @return{Deferred} dfd
     */
    Report.prototype.save = function(options, callback, errorback, always)
{};

    /**
     * Undo previous JIVE component update
     * @param {Function} callback - optional, invoked in case of successful
update
     * @param {Function} errorback - optional, invoked in case of failed
update
     * @param {Function} always - optional, invoked optional, invoked
always
     * @return{Deferred} dfd
     */
    Report.prototype.undo = function(callback, errorback, always){};

    /**
     * Reset report to initial state
     * @param {Function} callback - optional, invoked in case of successful
update
     * @param {Function} errorback - optional, invoked in case of failed
update
     * @param {Function} always - optional, invoked optional, invoked
always
     * @return{Deferred} dfd
```

Report Structure

The Report Data structure represents the rendered report object manipulated by the report function. Even though it is named "data", it does not contain report data, but rather the data about the report. For example, it contains information about the pages and bookmarks in the report.

The report structure also contains other components described elsewhere:

- The definitions of hyperlinks and how to work with them is explained in Customizing Links
- Details of the Jaspersoft Interactive Viewer and Editor (JIVE UI) are explained in Interacting With JIVE UI Components.

```
{
    "title": "Report Data",
    "description": "A JSON Schema describing a Report Data",
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
        "totalPages": {
            "type": "number",
            "description": "Report's page total count"
        },
        "links": {
            "type": "array",
            "description": "Links extracted from markup, so their quantity
depends on pages you have requested",
            "items": {
                "$ref": "#/definitions/jrLink"
            }
        },
```

```
        "bookmarks": {
            "type": "array",
            "description": "Report's bookmarks. Quantity depends on current
page",
            "items": {
                "$ref": "#/definitions/bookmark"
            }
        },
        "reportParts": {
            "type": "array",
            "description": "Report's parts. Quantity depends on current
page",
            "items": {
                "$ref": "#/definitions/reportPart"
            }
        },
        "components": {
            "type": "array",
            "description": "Components in report, their quantity depends on
pages you have requested",
            "items": {
                "type": "object",
                "description": "JIVE components data"
            }
        }
    },
    "definitions": {
        "bookmark":{
            "type": "object",
            "properties":{
                "page": "number",
                "anchor": "string",
                "bookmarks": {
                    "type": "array",
                    "items": {
                        "$ref": "#/definitions/bookmark"
                    }
                }
            }
        },
        "reportPart":{
            "type": "object",
            "properties":{
                "page": "number",
                "name": "string"
```

```

```

The Report Data structure represents the rendered report object manipulated by the report function. Even though it is named "data," it does not contain report data, but rather the data about the report. For example, it contains information about the pages and bookmarks in the report.

The report structure also contains other components described elsewhere:

- The definitions of hyperlinks and how to work with them is explained in Customizing Links
- Details of the Jaspersoft Interactive Viewer and Editor (JIVE UI) are explained in Interacting With JIVE UI Components.

```
{
    "title": "Report Data",
    "description": "A JSON Schema describing a Report Data",
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
        "totalPages": {
            "type": "number",
            "description": "Report's page total count"
        },
        "links": {
            "type": "array",
            "description": "Links extracted from markup, so their quantity
depends on pages you have requested",
            "items": {
                "$ref": "#/definitions/jrLink"
            }
        },
        "bookmarks": {
            "type": "array",
            "description": "Report's bookmarks. Quantity depends on current
page",
            "items": {
                "$ref": "#/definitions/bookmark"
```

```
            }
        },
        "reportParts": {
            "type": "array",
            "description": "Report's parts. Quantity depends on current
page",
            "items": {
                "$ref": "#/definitions/reportPart"
            }
        },
        "components": {
            "type": "array",
            "description": "Components in report, their quantity depends on
pages you have requested",
            "items": {
                "type": "object",
                "description": "JIVE components data"
            }
        }
    },
    "definitions": {
        "bookmark":{
            "type": "object",
            "properties":{
                "page": "number",
                "anchor": "string",
                "bookmarks": {
                    "type": "array",
                    "items": {
                        "$ref": "#/definitions/bookmark"
                    }
                }
            }
        },
        "reportPart":{
            "type": "object",
            "properties":{
                "page": "number",
                "name": "string"
            }
        },
        "jrLink": { // see chapter on hyperlinks
        }
    }
}
```

Rendering a Report

To run a report on the server and render it in Visualize.js, create a report object and set its properties. The server and resource properties determine which report to run, and the container property determines where it appears on your page.

```
var report = v.report({
    server: "http://bi.example.com:8080/jasperserver-pro",
    resource: "/public/Sample/MyReport",
    container: "#container"
});
```

To run a report on the server and render it with JasperReports IO Javascript API, load the jrio.js script, configure the JasperReports IO client object to point to the JasperReports IO REST service and to the needed Javascript files and theme, and then call the report function providing the URI of the report to run, and the container where it should be rendered on your page.

The following code example shows how to display a report that the user selects from a list.

```
jrio.config({
    server : "http://bi.example.com:8080/jrio",
    scripts : "https://bi.example.com/jrio-client/optimized-scripts",
    theme: {
        href: "https://bi.example.com/jrio-client/themes/default"
    },
    locale: "en_US"
});
jrio(function(jrioClient) {
    var report,
        selector = document.getElementById("selected_resource");
    selector.addEventListener("change", function() {
        report = createReport(selector.value);
     });
    report = createReport(selector.value);
    function createReport(uri) {
       return jrioClient.report({
           resource: uri,
           container: "#reportContainer",
```

```
            error: failHandler
        });
    }
    function failHandler(err) {
        alert(err);
    }
});
```

The following code example shows how to display a report that the user selects from a list.

```
visualize({
    auth: { ...
    }
}, function (v) {

    //render report from provided resource
    v("#container").report({
        resource: $("#selected_resource").val(),
        error: handleError
    });

    $("#selected_resource").change(function () {
        //clean container
        $("#container").html("");
        //render report from another resource
        v("#container").report({
            resource: $("#selected_resource").val(),
            error:handleError
        });
    });

     //enable report chooser
    $(':disabled').prop('disabled', false);

    //show error
    function handleError(err){
        alert(err.message);
    }

});
```

The HTML page that displays the report uses a static list of reports in a drop-down selector, but otherwise needs only a container element.

```
<!--Provide the URL to visualize.js-->
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>
<select id="selected_resource" disabled="true" name="report">
    <option value="/public/Samples/Reports/1._Geographic_Results_by_
Segment_Report">Geographic Results by Segment</option>
    <option value="/public/Samples/Reports/2_Sales_Mix_by_Demographic_
Report">Sales Mix by Demographic</option>
    <option value="/public/Samples/Reports/3_Store_Segment_Performance_
Report">Store Segment Performance</option>
    <option value="/public/Samples/Reports/04._Product_Results_by_Store_
Type_Report">Product Results by Store Type</option>
</select>
<!--Provide a container to render your visualization-->
<div id="container"></div>
```

The HTML page that displays the report uses a static list of reports in a drop-down selector, but otherwise needs only a container element.

```
<!-- Provide the URL to jrio.js -->
<script src="http://bi.example.com:8080/jrio-client/optimized-
scripts/jrio.js"></script>
<select id="selected_resource" name="report">
 <option value="/samples/reports/TableReport">Table Report</option>
 <option
value="/samples/reports/highcharts/HighchartsChart">Highcharts</option>
</select>
<!--Provide container to render your visualization-->
<div id="reportContainer"/>
```

Getting the Embed Code of a Report

As of JasperReports Server 7.9, the server can provide the code to embed any report displayed in the report viewer. This lets you browse the repository, preview a report, and get its basic embed code. You can paste this code directly in your application, and then edit it for your needs. You can also open the report's embed code in JSFiddle to see the effect of your edits in real time, then copy the final code from JSFiddle.

To copy the embed code of a report

1. Log into JasperReports Server and browse the repository to find the report you want to embed.
2. Run the report to view its output in the server's report viewer.
3. Click the </> icon in the menu bar to view the embed code.
4. The Report Embed Code dialog shows you the Visualize.js code and a preview of the report as it is currently saved. Select Copy Code to copy the entire code block to your clipboard. You can also highlight selected parts of the code and use Ctrl-C (Command-C on Mac OS), for example if you want only the main function.

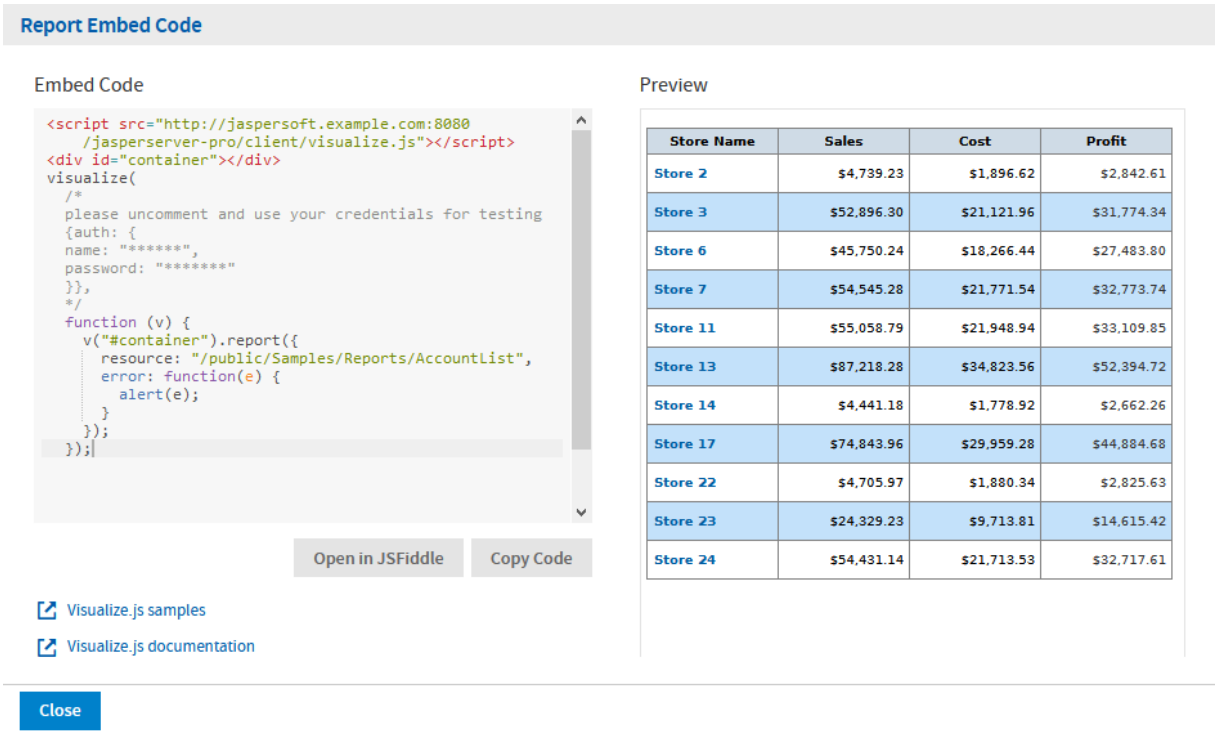

Figure 1: The Embed Code of a Report

The code sample includes comments where you can enter credentials for authentication. You should also change the name of the container to match the one in your application.

5. Alternatively, select Open in JSFiddle to load the same code into a new Fiddle, an online JavaScript viewer and interactive editor. This lets you modify the JavaScript or HTML, and add CSS if desired, then see the results in real time.

Before or after copying the embed code, you can fine-tune the report within the viewer, for example sorting a table colum or changing the chart type. Save your changes to make them available to others. Visualize.js always displays the latest saved version of a report, as determined by the repository URL in the embed code.

When displayed through Visualize.js, a report with a chart includes the  icon in the top left corner that allows users to switch between chart types, though not all chart types work with the data in a given report. This selector changes the chart in the user's current session, but the changes can't be saved. For every new session, Visualize.js displays the default appearance of the chart in the report.

Setting Report Parameters

To set or change the parameter values, update the params object of the report properties and invoke the run function again.

```
    // update report with new parameters
    report
        .params({ "Country": ["USA"] })
        .run();
    ...
    // later in code
    console.log(report.params()); // console log output: {"Country":
["USA"] }
```

The example above is trivial, but the power of Visualize.js the JasperReports IO JavaScript API comes from this simple code. You can create any number of user interfaces, database lookups, or your own calculations to provide the values of parameters. Your parameters could be based on 3rd party API calls that get triggered from other parts of the page or other pages in your app. When your reports can respond to dynamic events, they are seamlessly embedded and much more relevant to the user.

Here are further guidelines for setting parameters:

- If a report has required parameters, you must set them in the report object of the initial call, otherwise you'll get an error. For more information, see Catching Report Errors.
- Parameters are always sent as arrays of quoted string values, even if there is only one value, such as ["USA"] in the example above. This is also the case even for single value input such as numerical, boolean, or date/time inputs. You must also use the array syntax for single-select values as well as multi-select parameters with only one selection. No matter what the type of input, always set its value to an array of quoted strings.
- The following values have special meanings:
    - "" – An empty string, a valid value for text input and some selectors.
    - "~NULL~" – Indicates a NULL value (absence of any value), and matches a field that has a NULL value, for example if it has never been initialized.
    - "~NOTHING~" – Indicates the lack of a selection. In multi-select parameters, this is equivalent to indicating that nothing is deselected, thus all are selected. In a single-select non-mandatory parameter, this corresponds to no selection (displayed as ---). In a single-select mandatory parameter, the lack of selection makes it revert to its default value.

Saving a Report

Once you change the report parameters, you can save the new report in the repository. You can invoke the report.save function without parameters to overwrite the current report. You can also specify a new name or a new folder to save as a different report. The authenticated user must have write permission to the report or to the folder.

```
report.save();

report.save({folderUri:"/public",
            label:"Some report label",
            description:"Some report description",
            overwrite:true});
```

The following schema describes the parameters to the report.save function:

```
{
    "title": "Report save options",
    "type": "object",
    "properties": {
        "folderUri": {
            "type": "string",
            "description": "The URI of a folder to save a report",
            "pattern": "^/.+"
        },
        "label": {
            "type": "string",
            "description": "Report resource label"
        },
        "description": {
            "type": "string",
            "description": "Report resource description"
        },
        "overwrite": {
            "type": "boolean",
            "description": "If true and there is a resource with the same
name, then the resource will be overwritten",
            "default": "false"
        }
    },
    "required": ["folderUri"]
}
```

Rendering Multiple Reports

JavaScript Example:

```
visualize({
    auth: { ...
    }
}, function (v) {

    var reportsToLoad = [
        "/public/Samples/Reports/AllAccounts",
```

```
        "/public/Samples/Reports/01._Geographic_Results_by_Segment_Report",
        "/public/Samples/Reports/Cascading_Report_2_Updated",
        "/public/Samples/Reports/07g.RevenueDetailReport"
    ];

    $.each(reportsToLoad, function (index, uri) {
        var container = "#container" + (index + 1);
        v(container).report({
            resource: uri,
            success: function () {
                console.log("loaded: " + (index + 1));
            },
            error: function (err) {
                alert(err.message);
            }
        });
    });
});
```

JavaScript Example:

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    var reportsToLoad = [
        "/samples/reports/TableReport",
        "/samples/reports/highcharts/HighchartsChart",
        "/samples/reports/cvc/Figures",
        "/samples/reports/OrdersTable"
    ];
    $.each(reportsToLoad, function (index, uri) {
        var container = "#container" + (index + 1);
        jrioClient(container).report({
            resource: uri,
            success: function () {
                console.log("loaded: " + (index + 1));
            },
            error: function (err) {
                alert(err.message);
            }
        });
```

```
    });
});
```

Associated HTML:

```
<script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.js"></sc
ript>
<script src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.10.0/jquery-
ui.min.js"></script>
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>
<table class="sample">
    <tr>
        <td id="container1"></td>
        <td id="container2"></td>
    </tr>
    <tr>
        <td id="container3"></td>
        <td id="container4"></td>
    </tr>
</table>
```

Associated HTML:

```
<script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.js"></sc
ript>
<script src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.10.0/jquery-
ui.min.js"></script>
<script src="http://bi.example.com:8080/jriosjapi/client/jrio.js"></script>
<table class="sample">
    <tr>
        <td id="container1"></td>
        <td id="container2"></td>
    </tr>
    <tr>
        <td id="container3"></td>
        <td id="container4"></td>
```

```
    </tr>
</table>
```

Associated CSS:

```
html, body {
}
table.sample {
    width: 100%;
}
td#c1, td#c2, td#c3, td#c4 {
    width: 50%;
}
```

Resizing a Report

When rendering a report, by default it is scaled to fit in the container you specify. When users resize their window, reports will change so that they fit to the new size of the container. This section explains several ways to change the size of a rendered report.

To set a different scaling factor when rendering a report, specify its scale property:

- container – The report is scaled to fully fit within the container, both in width and height. If the container has a different aspect ratio, there will be white space in the dimension where the container is larger. This is the default scaling behavior when the scale property is not specified.
- width – The report is scaled to fit within the width of the container. If the report is taller than the container, users will need to scroll vertically to see the entire report.
- height – The report is scaled to fit within the height of the container. If the report is wider than the container, users will need to scroll horizontally to see the entire report.
- Scale factor – A decimal value greater than 0, with 1 being equivalent to 100%. A value between 0 and 1 reduces the report from its normal size, and a value greater than 1 enlarges it. If either or both dimensions of the scaled report are larger than the container, users will need to scroll to see the entire report.

In every case, the entire report is scaled in both directions by the same amount, you cannot change the aspect ratio of tables and crosstab elements.

For example, to initialize the report to half-size (50%), specify the following scale:

```
var report = v.report({
    resource: "/public/Sample",
    container: "#reportContainer",
    scale: 0.5
});
```

For example, to initialize the report to half-size (50%), specify the following scale:

```
var report = jrioClient.report({
    resource: "/public/Sample",
    container: "#reportContainer",
    scale: 0.5
});
```

You can also change the scale after rendering, in this case to more than double size (250%):

```
report
    .scale(2.5)
    .run();
```

Alternatively, you can turn off the container resizing and modify the size of the container explicitly:

```
var report = v.report({
    resource: "/public/Sample",
    container: "#reportContainer",
    scale: "container",
    autoresize: false
```

```
});

$("#reportContainer").width(500).height(500);
report.resize();
```

```
var report = jrioClient.report({
    resource: "/public/Sample",
    container: "#reportContainer",
    scale: "container",
    autoresize: false
});

$("#reportContainer").width(500).height(500);
report.resize();
```

To make the Report Viewer responsive, an additional property is introduced in Visualize.js that is reportContainerWidth. You need to set the value of this property to the width of the report container. Then this value is passed to the report execution process as `REPORT_CONTAINER_WIDTH` built-in parameter value.

```
visualize({
    auth: {...
    }
}, function(v) {
     let reportConfig = {
         resource: "/public/ResponsiveReport",
         container: "#container",
         reportContainerWidth: getContainerWidth(),
         events: {
               responsiveBreakpointChanged: function(error) {
                    if (error) {
                        console.log(error);
                     } else {
                         report.destroy();
                         // rerun report
                         reportConfig.reportContainerWidth =
getContainerWidth();
                         report = v.report(reportConfig);
```

```
                        }
                    }
                },
                error: (e) => console.error(e.message || e)
            };
            let report = v.report(reportConfig);

            function getContainerWidth() {
                  return document.getElementById("container").clientWidth;

            }
});
```

Setting Report Pagination

To set or change the pages displayed in the report, update the pages object of the report properties and invoke the run function again.

```
report
    .pages(5)
    .run(); // re-render report with page 5 into the same container

report
    .pages("2") // string is also allowed
    .run();

report
    .pages("4-6") // a range of numbers as a string is also possible
    .run();

report
    .pages({     // alternative object notation
        pages: "4-6"
    })
    .run();
```

The pages object of the report properties also supports bookmarks by specifying the anchor property. You can also specify both pages and bookmarks as shown in the example below. For more information about bookmarks, see Providing Bookmarks in Reports.

```
    report
        .pages({    // bookmark inside report to navigate to
            anchor: "summary"
        })
        .run();

    report
        .pages({    // set bookmark to scroll report to in scope of
provided pages
            pages: "2-5",
            anchor: "summary"
        })
        .run();
```

Creating Pagination Controls (Next/Previous)

Again, the power of Visualize.js comes from these simple controls that you can access programmatically. You can create any sort of mechanism or user interface to select the page. In this example, the HTML has buttons that allow the user to choose the next or previous pages.

```
visualize({
    auth: { ...
    }
}, function (v) {

    var report = v.report({
        resource: "/public/Samples/Reports/AllAccounts",
        container: "#container"
    });

    $("#previousPage").click(function() {
        var currentPage = report.pages() || 1;

        report
            .pages(--currentPage)
            .run()
                .fail(function(err) { alert(err); });
```

```
    });

    $("#nextPage").click(function() {
        var currentPage = report.pages() || 1;

        report
            .pages(++currentPage)
            .run()
                .fail(function(err) { alert(err); });
    });
});
```

Again, the power of the JasperReports IO JavaScript API comes from these simple controls that you can access programmatically. You can create any sort of mechanism or user interface to select the page. In this example, the HTML has buttons that allow the user to choose the next or previous pages.

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    var report = jrioClient.report({
        resource: "/samples/reports/TableReport",
        container: "#reportContainer",
        error: function(err) { alert(err); },
    });

    $("#previousPage").click(function() {
        var currentPage = report.pages() || 1;

        report
            .pages(--currentPage)
            .run()
            .fail(function(err) { alert(err); });
    });

    $("#nextPage").click(function() {
        var currentPage = report.pages() || 1;

        report
            .pages(++currentPage)
```

```
            .run()
            .fail(function(err) { alert(err); });
    });
});
```

Associated HTML:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>

<button id="previousPage">Previous Page</button><button id="nextPage">Next
Page</button>

<div id="container"></div>
```

Associated HTML:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="http://bi.example.com:8080/jriosjapi/client/jrio.js"></script>

<button id="previousPage">Previous Page</button><button id="nextPage">Next
Page</button>

<div id="reportContainer"></div>
```

Creating Pagination Controls (Range)

JavaScript Example:

```
visualize({
    auth: { ...
```

```
    }
}, function (v) {
    var report = v.report({
        resource: "/public/Samples/Reports/AllAccounts",
        container: "#container"
    });

    $("#pageRange").change(function() {
        report
            .pages($(this).val())
            .run()
                .fail(function(err) { alert(err); });
    });
});
```

JavaScript Example:

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    var report = jrioClient.report({
        resource: "/samples/reports/TableReport",
        container: "#reportContainer",
        error: function(err) { alert(err); },
    });
    $("#pageRange").change(function() {
        report
            .pages($(this).val())
            .run()
            .fail(function(err) { alert(err); });
    });
});
```

Associated HTML:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="http://bi.example.com:8080/jasperserver-
```

```
pro/client/visualize.js"></script>

Page range: <input type="text" id="pageRange"></input>

<div id="container"></div>
```

Associated HTML:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="http://bi.example.com:8080/jriosjapi/client/jrio.js"></script>

Page range: <input type="text" id="pageRange"></input>

<div id="reportContainer"></div>
```

Exporting From a Report

To export a report, invoke its export function and specify the outputFormat property. You MUST wait until the run action has completed before starting the export. The following export formats are supported:

"pdf", "xlsx", "rtf", "csv", "xml", "odt", "ods", "docx", "json", "pptx"

For CSV and JSON output, see Exporting Data From a Report. Note that the HTML output of a report is not available through Visualize.js.

To export a report, invoke its export function and specify the outputFormat property. You MUST wait until the run action has completed before starting the export. The following export formats are supported:

"pdf", "docx", "pptx", "csv", "xlsx", "rtf", "odt", "ods", "html", "xml", "data_csv", "data_json," "data_xlsx"

The last three are for pure data output, also known as "metadata" exporters in the JR Library, and you can learn more about them in Exporting Data From a Report.

```
report.run(exportToPdf);

function exportToPdf() {
    report
        .export({
            outputFormat: "pdf"
        })
        .done(function (link) {
            window.open(link.href); // open new window to download report
        })
        .fail(function (err) {
            alert(err.message);
        });
}
```

The following sample exports 10 pages of the report to a paginated Excel spreadsheet:

```
report.run(exportToPaginatedExcel);

function exportToPaginatedExcel() {
    report
        .export({
            outputFormat: "xlsx",
            pages: "1-10",
            ignorePagination: false
        })
        .done(function(link){
            window.open(link.href); // open new window to download report
        })
        .fail(function(err){
            alert(err.message);
        });
}
```

The following sample exports the part of report associated with a named anchor:

```
report.run(exportPartialPDF);
```

```
function exportPartialPDF() {
    report
        .export({
            outputFormat: "pdf",
            pages: {
                anchor: "summary"
            }
        })
        .done(function(link){
            window.open(link.href); //open new window to download report
        })
        .fail(function(err){
            alert(err.message);
        });
}
```

The following example creates a user interface for exporting a report:

```
visualize({
    auth: { ...
    }
}, function (v) {

    var $select = buildControl("Export to: ", v.report.exportFormats),
        $button = $("#button"),
        report = v.report({
            resource: "/public/Samples/Reports/5g.AccountsReport",
            container: "#container",

            success: function () {
                button.removeAttribute("disabled");
            },

            error: function (error) {
                console.log(error);
            }
        });

    $button.click(function () {
```

```
        console.log($select.val());

        report.export({
            //export options here
            outputFormat: $select.val(),
            //exports all pages if not specified
            //pages: "1-2"
        }, function (link) {
           var url = link.href ? link.href : link;
           window.location.href = url;
        }, function (error) {
            console.log(error);
        });
    });

    function buildControl(name, options) {

        function buildOptions(options) {
            var template = "<option>{value}</option>";
            return options.reduce(function (memo, option) {
                return memo + template.replace("{value}", option);
            }, "")
        }

        var template = "<label>{label}</label><select>
{options}</select><br>",
            content = template.replace("{label}", name)
                .replace("{options}", buildOptions(options));

        var $control = $(content);
        $control.insertBefore($("#button"));
        //return select
        return $($control[1]);
    }

});
```

The following example creates a user interface for exporting a report:

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    var $select = buildControl("Export to: ",

["pdf","xlsx","docx","pptx","csv","rtf","odt","ods","html","xml","data_
csv","data_json","data_xlsx"]),
        $button = $("#button"),
        report = jrioClient.report({
            resource: "/samples/reports/OrdersTable",
            container: "#reportContainer",
            success: function () {
                button.removeAttribute("disabled");
            },
            error: function (error) {
                console.log(error);
            }
       });

    $button.click(function () {

        console.log($select.val());

        report.export({
            //export options here
            outputFormat: $select.val(),
            //exports all pages if not specified
            //pages: "1-2"
        }, function (link) {
            var url = link.href ? link.href : link;
            window.location.href = url;
        }, function (error) {
            console.log(error);
        });
    });

    function buildControl(name, options) {

        function buildOptions(options) {
            var template = "<option>{value}</option>";
            return options.reduce(function (memo, option) {
                return memo + template.replace("{value}", option);
            }, "")
```

```
        }

        var template = "<label>{label}</label><select>
{options}</select><br>",
            content = template.replace("{label}", name)
               .replace("{options}", buildOptions(options));

        var $control = $(content);
        $control.insertBefore($("#button"));
        //return select
        return $($control[1]);
    }
});
```

Associated HTML:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<!-- Provide the URL to visualize.js -->
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>

<button id="button" disabled>Export</button>
<!-- Provide a container for the report -->
<div id="container"></div>
```

Associated HTML:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>

<!-- Provide the URL to jrio.js -->
<script src="http://bi.example.com:8080/jriosjapi/client/jrio.js"></script>

<button id="button" disabled>Export</button>
<!-- Provide a container for the report -->
<div id="reportContainer"></div>
```

Exporting Data From a Report

You can also request the raw data of the report in CSV or JSON format.

The following example shows how to export data in CSV format. CSV output is plain text that you must parse to extract the values that you need:

```
report.run(exportToCsv);

function exportToCsv() {
    report
        .export({
            outputFormat: "csv"
        })
        .done(function(link, request){
            request()
                .done(function(data) {
                    // use data here, data is CSV format in plain text
                });
                .fail(function(err){
                    //handle errors here
                });
        })
        .fail(function(err){
            alert(err.message);
        });
}
```

You can also request the raw data of the report in CSV, XLSX or JSON format.

The following example shows how to export pure data in CSV format using the metadata CSV exporter. CSV output is plain text that you must parse to extract the values that you need.

```
report.run(exportToCsv);

function exportToCsv() {
    report
        .export({
```

```
            outputFormat: "data_csv"
        })
        .done(function(link, request){
            request()
                .done(function(data) {
                    // use data here, data is CSV format in plain text
                })
                .fail(function(err){
                    //handle errors here
                });
        })
        .fail(function(err){
            alert(err.message);
        });
}
```

The following example shows how to export data in JSON format. By its nature, JSON format can be used directly as data within your JavaScript.

```
report.run(exportToJson);

function exportToJson() {
    report
        .export({
            outputFormat: "json"
        })
        .done(function(link, request){
            request({
                dataType: "json"
            })
            .done(function(data) {
                // use JSON data as objects here
            })
            .fail(function(err){
                    //handle errors here
            });

        })
        .fail(function(err){
            alert(err.message);
        });
}
```

The following example shows how to export data in JSON format. By its nature, JSON format can be used directly as data within your JavaScript.

```
report.run(exportToJson);

function exportToJson() {
    report
        .export({
            outputFormat: "data_json"
        })
        .done(function(link, request){
            request({
                dataType: "json"
            })
            .done(function(data) {
                // use JSON data as objects here
            })
            .fail(function(err){
                    //handle errors here
            });

        })
        .fail(function(err){
            alert(err.message);
        });
}
```

Refreshing a Report

JavaScript Example:

```
visualize({
    auth: { ...
    }
}, function (v) {

    var alwasyRefresh = false;
```

```
    var report = v.report({
        //skip report running during initialization
        runImmediately: !alwasyRefresh,
        resource: "/public/viz/usersReport",
        container: "#container1",
     });

    if (alwasyRefresh){
        report.refresh();
    }

    $("button").click(function(){
        report
            .refresh()
            .done(function(){console.log("Report Refreshed!");})
            .fail(function(){alert("Report Refresh Failed!");});
    });

});
```

Associated HTML:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>
<button>Refresh</button>
<div id="container1"></div>
```

JavaScript Example:

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    var alwaysRefresh = false;

    var report = jrioClient.report({
        //skip report running during initialization
```

```
        runImmediately: !alwaysRefresh,
        resource: "/samples/reports/FirstJasper",
        container: "#reportContainer",
     });

    if (alwaysRefresh){
        report.refresh();
    }

    $("button").click(function(){
        report
            .refresh()
            .done(function(){console.log("Report Refreshed!");})
            .fail(function(){alert("Report Refresh Failed!");});
    });

});
});
```

Associated HTML:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="http://bi.example.com:8080/jriosjapi/client/jrio.js"></script>
<button>Refresh</button>
<div id="reportContainer"></div>
```

Canceling Report Execution

To stop a running report, call its cancel function:

```
    ...
    report
    .cancel()
    .done(function(){
        alert("Report Canceled");
    })
```

```
    .fail(function(){
        alert("Report Failed");
    });
```

The following example is more complete and creates a UI for a spinner and cancel button for a long-running report.

```
var spinner = createSpinner();

visualize({
    auth: { ...
    }
}, function (v) {

    var button = $("button");

    var report = v.report({
        resource: "/public/Reports/Slow_Report",
        container: "#container",
        events: {
            changeTotalPages : function(){
               spinner.remove();
            }
        }
    });

    button.click(function () {
        report
            .cancel()
            .then(function () {
                spinner.remove();
                alert("Report Canceled!");
            })
            .fail(function () {
                alert("Can't Cancel Report");
            });
    });
});

function createSpinner() {
    var opts = {
```

```
        lines: 17, length: 3, width: 2, radius: 3, corners: 0.6, rotate: 0,
direction: 1,
        color: '#000', speed: 1, trail: 60, shadow: false, hwaccel: false,
zIndex: 2e9,
        top: 'auto', left: 'auto', className: 'spinner'
    };
    var container = $("#spinner");
    var spinner = new Spinner(opts).spin(container[0]);
    return container;
}
```

Associated HTML:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="http://fgnass.github.io/spin.js/spin.js"></script>
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>
<div id="spinner"></div>
<button>Cancel</button>
<div id="container"></div>
```

The following example is more complete and creates a UI for a cancel button for a long-running report.

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    var button = $("button");

    var report = jrioClient.report({
        resource: "/samples/reports/SlowReport",
        container: "#reportContainer",
        events: {
            changeTotalPages : function(){
               button.remove();
            }
        }
```

```
    });

    button.click(function () {
        report
            .cancel()
            .then(function () {
                button.remove();
                alert("Report Canceled!");
            })
            .fail(function () {
                alert("Can't Cancel Report");
            });
    });
});
```

Associated HTML:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="http://bi.example.com:8080/jriosjapi/client/jrio.js"></script>
<button>Cancel</button>
<div id="reportContainer"></div>
```

Discovering Available Charts and Formats

You can write code to discover and display the types of charts and export formats that can be specified. The following example reads the exportFormats, chart.types, and table.column.types of the given report and dynamically creates a selection dialog for each:

```
visualize({
    auth: {
        name: "jasperadmin",
        password: "jasperadmin",
        organization: "organization_1"
```

```
    }
}, function (v) {

    buildControl("Report export formats", v.report.exportFormats);
    buildControl("Chart types", v.report.chart.types);
    buildControl("Report table column types", v.report.table.column.types);

});

function buildControl(name, options) {

        function buildOptions(options) {
            var template = "<option>{value}</option>";
            return options.reduce(function (memo, option) {
                return memo + template.replace("{value}", option);
            }, "")
        }

    console.log(options);

    if (!options.length){
        console.log(options);
    }

    var template = "<label>{label}</label><select>{options}</select><br>",
        content = template.replace("{label}", name)
                .replace("{options}", buildOptions(options));

    $("#container").append($(content));
}
```

JavaScript API Reference - Errors

This chapter describes common errors and explains how to handle them with Visualize.js.

This chapter describes common errors and explains how to handle them with the JasperReports IO Javascript API.

This chapter contains the following sections:

Error Properties

The properties structure for Generic Errors is defined as follows:

```
{
    "title": "Generic Errors",
    "description": "A JSON Schema describing Visualize Generic Errors",
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
        "errorCode": {
           "type": "string"
        },
```

```
        "message": {
            "type": "string"
        },
        "parameters":{
            "type": "array"
        }
    },
    "required": ["errorCode", "message"]
}
```

Common Errors

The following table lists common errors, their messages, and causes.

Error	Message - Description
Page or app not responding	{no_message} - If your page or web application has stopped working without notification or errors, check that the server providing visualize.js JasperReports IO JavaScript API is accessible and returning scripts.
unexpected.error	An unexpected error has occurred - In most of cases this is either a JavaScript exception or an HTTP 500 (Internal Server Error) response from server.
schema.validation.error	JSON schema validation failed: {error_message} - Validation against schema has failed. Check the validationError property in object for more details.
unsupported. configuration.error	{unspecified_message} - This error happens only when isolateDom = true and defaultJiveUi.enabled = true. These properties are mutually exclusive.
authentication.error	Authentication error - Credentials are not valid or session has expired.
container.not.found.error	Container was not found in DOM - The specified container was

Error	Message - Description
report.execution.failed	Report execution failed - The report failed to run on the server.
report.execution.cancelled	Report execution was canceled - Report execution was canceled.
report.export.failed	Report export failed - The report failed to export on the server.
licence.not.found	JRS missing appropriate license JRIO missing appropriate license- The server's license was not found.
licence.expired	JRS missing appropriate license JRIO missing appropriate license - The server's license has expired.
resource.not.found	Resource not found in Repository - Either the resource doesn't exist in the repository or the user doesn't have permissions to read it.
export.pages.out.range	Requested pages {0} out of range - The user requested pages that don't exist in the current export.
input.controls. validation.error	{server_error_message} - The wrong input control params were sent to the server.

Catching Initialization and Authentication Errors

Visualize.js is designed to have many places where you can catch and handle errors. The visualize function definition, as shown in Contents of the Visualize.js Script, is:

```
function visualize(properties, callback, errorback, always){}
```

During initialization and authentication, you can handle errors in the third parameter named errorback (an error callback). Your application would then have this structure:

```
visualize({
    auth : { ...
    }
}, function(){

    // your application logic

}, function(err){

    // handle all initialization and authentication errors here

})
```

Catching Search Errors

One way to handle search errors is to specify an error handler as the second parameter of run:

```
new ResourcesSearch({
    server:"http://localhost:8080/jasperserver-pro",
    folderUri: "/public",
    recursive: false
})).run( usefulFunction, function(error){

    alert(error);

})
```

Another way to handle search errors is to specify a function as the third parameter of run. This function is an always handler that runs every time when operation ends.

```
new ResourcesSearch({
    server:"http://localhost:8080/jasperserver-pro",
    folderUri: "/public",
    recursive: false
})).run(usefulFunction, errorHandler, function(resultOrError){

    alert(resultOrError);

})
```

Validating Search Properties

You can also validate the structure of the search properties without making an actual call to the search function:

```
var call = new ResourcesSearch({
    server:"http://localhost:8080/jasperserver-pro",
    folderUri: "/public",
    recursive: false
});

var error = call.validate();

if (!error){
    // valid
} else {
    // invalid, read details from error
}
```

Catching Report Errors

To catch and handle errors when running reports, define the contents of the err function as shown in the following sample:

```
visualize({
    auth : { ...
    }
}, function(v){

    var report = v.report({
        error: function(err){
            // invoked once report is initialized and has run
        }
    });

    report
        .run()
        .fail(function(err){
            // handle errors here
        });

)
```

To catch and handle errors when running reports, define the contents of the err function as shown in the following sample:

```
jrio.config({
    ...
});
jrio(function(jrioClient) {

    var report = jrioClient.report({
        error: function(err){
            // invoked once report is initialized and has run
        }
    });

    report
        .run()
        .fail(function(err){
            // handle errors here
        });

)
```

Catching Input Control Errors

Catching and handling input control errors is very similar to handling report errors. Define the contents of the err function that gets invoked in error conditions, as shown in the following sample:

```
visualize({
    auth : { ...
    }
}, function(v){

    var ic = v.inputControls({
        error: function(err){
            // invoked once input control is initialized
        }
    });

    inputControls
        .run()
        .fail(function(err){
            // handle errors here
        });

)
```

Validating Input Controls

You can also validate the structure of your input controls without making an actual call. However, the values of the input controls and their relevance to the named resource are not checked.

```
var ic = new InputControls({
    server: "http://localhost:8080/jasperserver-pro",
    resource: "/public/my_report",
    params: {
        "Country_multi_select":["Mexico"],
```

```
        "Cascading_state_multi_select":["Guerrero", "Sinaloa"]
    }
});

var error = ic.validate();

if (!error){
    // valid
} else {
    // invalid, read details from error
}
```

JavaScript API Usage - Report Events

Depending on the size of your data, the report function can run for several seconds or minutes, just like reports in the JasperReports Server UI. You can listen for events that give the status of running reports and display pages sooner.

This chapter contains the following sections:

Tracking Completion Status

By listening for the reportCompleted event, you can give information or take action when a report finishes rendering.

```
visualize({
    auth: { ...
    }
}, function (v) {
    var report = v.report({
        // run example with a very long report
        resource: "/public/Samples/Reports/RevenueDetailReport",
        container: "#container",
        events: {
            reportCompleted: function(status) {
                alert("Report status: "+ status+ "!");
            }
        },
        error: function(error) {
            alert(error);
```

```
        },
    });
});
```

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    var report = jrioClient.report({
        // run example with a very long report
        resource: "/samples/reports/SlowReport",
        container: "#reportContainer",
        events: {
            reportCompleted: function(status) {
                alert("Report status: "+ status + "!");
            }
        },
        error: function(error) {
            alert(error);
        },
    });
});
```

Tracking Report Container Size

By listening for the responsiveBreakpointChanged event, you can track when the container size has been passed from one interval to another.

```
visualize({
    auth: {...
    }
}, function(v) {
     let reportConfig = {
         resource: "/public/ResponsiveReport",
         container: "#container",
```

```
          reportContainerWidth: getContainerWidth(),
          events: {
                responsiveBreakpointChanged: function(error) {
                     if (error) {
                         console.log(error);
                      } else {
                          report.destroy();
                       // rerun report
                          reportConfig.reportContainerWidth =
getContainerWidth();
                          report = v.report(reportConfig);
                             }
                       }
                     },
                     error: (e) => console.error(e.message || e)
             };
             let report = v.report(reportConfig);

             function getContainerWidth() {
                   return document.getElementById("container").clientWidth;

            }
 });
```

Listening for Page Totals

By listening for the changeTotalPages event, you can track the filling of the report.

```
visualize({
    auth: { ...
    }
}, function (v) {
       var report = v.report({
           resource: "/public/Samples/Reports/AllAccounts",
           container: "#container",
           error: function(error) {
               alert(error);
               },
           events: {
```

```
                changeTotalPages: function(totalPages) {
                    alert("Total Pages:" + totalPages);
                }
            }
        });
});
```

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    var report = jrioClient.report({
        // run example with a very long report
        resource: "/samples/reports/SlowReport",
        container: "#reportContainer",
        events: {
            changeTotalPages: function(totalPages) {
                alert("Total Pages:" + totalPages);
            }
        },
        error: function(error) {
            alert(error);
        },
    });
});
```

Listening for the Last Page

Listening for the pageFinal event, lets you know when the last page of a running report has been generated.

```
visualize({
    auth: { ...
    }
}, function (v) {
```

```
    var report = v.report({
        // run example with a very long report
        resource: "/public/Samples/Reports/RevenueDetailReport",
        container: "#container",
        events: {
            pageFinal: function(el) {
                console.log(el);
                alert("Final page is rendered!");
            },
            reportCompleted: function(status) {
                alert("Report status: "+ status+ "!");
            }
        },
        error: function(error) {
            alert(error);
        },
    });
});
```

Customizing a Report's DOM Before Rendering

By listening for the beforeRender event, you can access the Document Object Model (DOM) of the report to view or modify it before it is displayed. In the example the listener finds span elements and adds a color style and an attribute my-attr="test" to each one.

```
visualize({
    auth: { ...
    }
}, function (v) {
    // enable report chooser
    $(':disabled').prop('disabled', false);

    //render report from provided resource
    startReport();

    $("#selected_resource").change(startReport);

    function startReport () {
```

```
        // clean container
        $("#container").html("");
        // render report from another resource
        v("#container").report({
            resource: $("#selected_resource").val(),
            events:{
                beforeRender: function(el){
                    // find all spans
                    $(el).find(".jrPage td span")
                         .each(function(i, e){
                               // make them red
                               $(e).css("color","red")
                                   .attr("data-my-attr", "test");
                          });
                    console.log($(el).find(".jrPage").html());
                }
            }
        });
    };
});
```

By listening for the beforeRender event, you can access the Document Object Model (DOM) of the report to view or modify it before it is displayed. In the example the listener finds span elements and adds a color style and an attribute my-attr="test" to each one.

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    // enable report chooser
    $(':disabled').prop('disabled', false);

    //render report from provided resource
    startReport();

    $("#selected_resource").change(startReport);

    function startReport () {
        // clean container
        $("#reportContainer").html("");
        // render report from another resource
```

```
        jrioClient("#reportContainer").report({
            resource: $("#selected_resource").val(),
            events:{
                beforeRender: function(el){
                    // find all spans
                    $(el).find(".jrPage td.jrcolHeader span")
                        .each(function(i, e){
                            // make them red
                            $(e).css("color","red")
                                .attr("data-my-attr", "test");
                        });
                    console.log($(el).find(".jrPage").html());
                }
            }
        });
    };
});;
```

The HTML page that displays the report uses a static list of reports in a drop-down selector, but otherwise needs only a container element. This is similar to the basic report example in Rendering a Report, except that the JavaScript above will change the report before it's displayed.

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<!-- Provide the URL to visualize.js -->
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>
<select id="selected_resource" disabled="true" name="report">
    <option value="/public/Samples/Reports/1._Geographic_Results_by_
Segment_Report">Geographic Results by Segment</option>
    <option value="/public/Samples/Reports/2_Sales_Mix_by_Demographic_
Report">Sales Mix by Demographic</option>
    <option value="/public/Samples/Reports/3_Store_Segment_Performance_
Report">Store Segment Performance</option>
    <option value="/public/Samples/Reports/04._Product_Results_by_Store_
Type_Report">Product Results by Store Type</option>
</select>
<!-- Provide a container to render your visualization -->
<div id="container"></div>
```

The HTML page that displays the report uses a static list of reports in a drop-down selector, but otherwise needs only a container element. This is similar to the basic report example in

Rendering a Report, except that the JavaScript above will change the report before it's displayed.

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>

<!-- Provide the URL to jrio.js -->
<script src="http://bi.example.com:8080/jriojsapi/client/jrio.js"></script>
<select id="selected_resource" disabled="true" name="report">
    <option value="/samples/reports/TableReport">Table Report</option>
    <option value="/samples/reports/OrdersTable">Orders Table</option>
</select>
<!-- Provide a container to render your visualization -->
<div id="reportContainer"></div>
```

JavaScript API Usage - Hyperlinks

Both reports and dashboards include hyperlinks (URLs) that link to websites or other reports. The JasperReports IO JavaScript API Visualize.js gives you access to the links so that you can customize them or open them differently. For links generated in the report, you can customize both the appearance and the container where they are displayed.

This chapter contains the following sections:

Structure of Hyperlinks

The following JSON schema describes all the parameters on links, although not all are present in all cases.

```
"jrLink": {
    "title": "JR Hyperlink",
    "description": "A JSON Schema describing JR hyperlink",
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties":{
        "id": {
            "type": "string",
            "description": "Hyperlink id, reflected in corresponding
attribute in DOM. Is not used for AdHocExecution hyperlink type."
        },
        "type": {
            "type": "string",
```

```
            "description": "Hyperlink type. Default types are LocalPage,
LocalAnchor, RemotePage, RemoteAnchor, Reference, ReportExecution,
AdHocExecution. Custom hyperlink types are possible"
        },
        "target": {
            "type": "string",
            "description": "Hyperlink target. Default targets are Self,
Blank, Top, Parent. Custom hyperlink targets are possible"
        },
        "tooltip": {
            "type": "string",
            "description": "Hyperlink tooltip"
        },
        "href": {
            "type" : "string",
            "description": "Hyperlink reference. Is an empty string for
LocalPage, LocalAnchor and ReportExecution hyperlink types"
        },
        "parameters": {
            "type": "object",
            "description": "Hyperlink parameters. Any additional parameters
for hyperlink"
        },
        "resource": {
            "type": "string",
            "description": "Repository resource URI of resource mentioned
in hyperlink. For LocalPage and LocalAnchor points to current report, for
ReportExecution - to _report parameter"
        },
        "pages": {
            "type": ["integer", "string"],
            "description": "Page to which hyperlink points to. Is actual
for LocalPage, RemotePage and ReportExecution hyperlink types"
        },
        "anchor": {
            "type": "string",
            "description": "Anchor to which hyperlink points to. Is actual
for LocalAnchor, RemoteAnchor and ReportExecution hyperlink types"
        }
    },
    "required": ["type", "id"]
}
```

Customizing Links

You can customize the appearance of link elements in a generated report in two ways:

- The linkOptions exposes the beforeRender event to which you can add a listener with access to the links in the document as element pairs.
- The normal click event lets you add a listener that can access to a link when it's clicked.

```
visualize({
    auth: {
        name: "jasperadmin",
        password: "jasperadmin",
        organization: "organization_1"
    }
},function (v) {
    v("#container1").report({
        resource: "/AdditionalResourcesForTesting/Drill_Reports_with_
Controls/main_report",
        linkOptions: {
            beforeRender: function (linkToElemPairs) {
                linkToElemPairs.forEach(function (pair) {
                    var el = pair.element;
                    el.style.backgroundColor = "red";
                });
            },
            events: {
                "click": function(ev, link){
                    if (confirm("Change color of link id " + link.id + " to
green?")){
                        ev.currentTarget.style.backgroundColor = "green";
                        ev.target.style.color = "#FF0";
                    }
                }
            }
        },
        error: function (err) {
            alert(err.message);
        }
    });
});
```

You can customize the appearance of link elements in a generated report in two ways:

- The linkOptions exposes the beforeRender event to which you can add a listener with access to the links in the document as element pairs.
- The normal click event lets you add a listener that can access to a link when it's clicked.

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    jrioClient("#reportContainer").report({
        resource: "/samples/reports/TableReport",
        linkOptions: {
            beforeRender: function (linkToElemPairs) {
                linkToElemPairs.forEach(function (pair) {
                    var el = pair.element;
                    el.style.backgroundColor = "red";
                });
            },
            events: {
                "click": function(ev, link){
                    if (confirm("Change color of link id " + link.id + " to
green?")){
                        ev.currentTarget.style.backgroundColor = "green";
                        ev.target.style.color = "#FF0";
                    }
                }
            }
        },
        error: function (err) {
            alert(err.message);
        }
    });
});
```

Drill-Down in Separate Containers

By using the method of listing for clicks on hyperlinks, you can write a JasperReports IO JavaScript API script that sets the destination of drill-down report links to another container. This way, you can create display layouts or overlays for viewing drill-down links

embedded in your reports. This sample code also changes the cursor for the embedded links, so they are more visible to users.

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    jrioClient("#main").report({
        resource: "/samples/reports/TableReport",
        linkOptions: {
            beforeRender: function (linkToElemPairs) {
                linkToElemPairs.forEach(showCursor);
            },
            events: {
                "click": function(ev, link){
                    if (link.type == "ReportExecution"){
                        jrioClient("#drill-down").report({
                            resource: link.parameters._report,
                            params: {
                                latitude: [link.parameters.latitude],
                                longitude: [link.parameters.longitude],
                                zoom: [link.parameters.zoom]
                            },
                        });
                    }
                    console.log(link);
                }
            }
        },
        error: function (err) {
            alert(err.message);
        }
    });

    function showCursor(pair){
        var el = pair.element;
        el.style.cursor = "pointer";
    }
});
```

By using the method of listing for clicks on hyperlinks, you can write a visualize.js script that sets the destination of drill-down report links to another container. This way, you can create display layouts or overlays for viewing drill-down links embedded in your reports. This sample code also changes the cursor for the embedded links, so they are more visible to users.

```
visualize({
    auth: {
        name: "jasperadmin",
        password: "jasperadmin",
        organization: "organization_1"
    }
}, function (v) {

    v("#main").report({
        resource: "/MyReports/Drill_Reports_with_Controls/main_report",
        linkOptions: {
            beforeRender: function (linkToElemPairs) {
                linkToElemPairs.forEach(showCursor);
            },
            events: {
                "click": function(ev, link){
                   if (link.type == "ReportExecution"){
                        v("#drill-down").report({
                            resource: link.parameters._report,
                            params: {
                                city: [link.parameters.city],
                                country: link.parameters.country,
                                state: link.parameters.state
                            },
                        });
                     }
                      console.log(link);
                }
            }
        },
        error: function (err) {
            alert(err.message);
        }
    });
```

```
    function showCursor(pair){
        var el = pair.element;
        if (typeof(el) != "undefined") {
            el.style.cursor = "pointer";
        }
    }

});
```

Associated HTML:

```
<script src="http://underscorejs.org/underscore.js"></script>
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<!-- Provide the URL to visualize.js -->
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>
<!-- Provide a container for the main report and one for the drill-down -->
<div>
    <div style="width:830px;" id="main"></div>
    <div style="width:500px;" id="drill-down"></div>
</div>
```

Associated HTML:

```
<script src="http://underscorejs.org/underscore.js"></script>
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>

<!-- Provide the URL to jrio.js -->
<script src="http://bi.example.com:8080/jriojsapi/client/jrio.js"></script>
<!-- Provide a container for the main report and one for the drill-down -->
<div>
    <div  id="main"></div>
    <div  id="drill-down"></div>
</div>
```

Associated CSS:

```
#main{
   float: left;
}

#drill-down{
    float: left;
}
```

Accessing Data in Links

In this example, we access the hyperlinks through the data.links structure after the report has successfully rendered. From this structure, we can read the tooltips that were set in the JRXML of the report. The script uses the information in the tooltips of all links in the report to create a drop-down selector of city name options.

By using link tooltips, your JRXML can create reports that pass runtime information to the display logic in your JavaScripts.

```
visualize({
    auth: {
        name: "jasperadmin",
        password: "jasperadmin",
        organization: "organization_1"
    }
}, function (v) {

    var $select = $("#selectCity"),
        report = v.report({
            resource: "/MyReports/Drill_Reports_with_Controls/main_report",

            container: "#main",
            success: refreshSelect,
            error: showError
    });
```

```
    function refreshSelect(data){
        console.log(data);
        var options = data.links.reduce(function(memo, link){
                console.log(link);
                return memo + ""+link.tooltip+"";
            },"");
        $select.html(options);
    }

    $("#previousPage").click(function() {
        var currentPage = report.pages() || 1;
        goToPage(--currentPage);
    });
```

```
    $("#nextPage").click(function() {
        var currentPage = report.pages() || 1;
        goToPage(++currentPage);
    });

    function goToPage(numder){
        report
            .pages(numder)
            .run()
                .done(refreshSelect)
                .fail(showError);
    }

    function showError(err){
        alert(err.message);
    }

});
```

By using link tooltips, your JRXML can create reports that pass runtime information to the display logic in your JavaScripts.

```
jrio.config({
    ...
});
jrio(function(jrioClient) {

    var $select = $("#selectCity"),
        report = jrioClient.report({
            resource: "/samples/reports/TableReport",
            container: "#main",
            success: refreshSelect,
            error: showError
    });
    function refreshSelect(data){
        console.log(data);
        $.each(data.links, function (i, item) {
            $select.append($('<option>', {
               value: item.id,
               text : item.tooltip
            }));
```

```
        });
    }
```

```
    $("#previousPage").click(function() {
        var currentPage = report.pages() || 1;
        goToPage(--currentPage);
    });

    $("#nextPage").click(function() {
        var currentPage = report.pages() || 1;
        goToPage(++currentPage);
    });

    function goToPage(numder){
        report
            .pages(numder)
            .run()
                .done(refreshSelect)
                .fail(showError);
    }

    function showError(err){
        alert(err.message);
    }

});
```

Associated HTML:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<!-- Provide the URL to visualize.js -->
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>
<select id="selectCity"></select>
<button id="previousPage">Previous Page</button>
<button id="nextPage">Next Page</button>
<!-- Provide a container for the main report -->
<div>
    <div style="width:20px;"></div>
    <div style="width:500px;" id="main"></div>
</div>
```

Associated HTML:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<!-- Provide the URL to jrio.js -->
<script src="http://bi.example.com:8080/jriojsapi/client/jrio.js"></script>
<select id="selectCity"></select>
<button id="previousPage">Previous Page</button>
<button id="nextPage">Next Page</button>
<!-- Provide a container for the main report -->
<div>
    <div ></div>
    <div  id="main"></div>
</div>
```

Associated CSS:

```
#main{
   float: left;
}
```

JavaScript API Usage - Interactive Reports

Most reports rendered in the JasperReports Server native interface have interactive abilities such as column sorting provided by a feature called JIVE: Jaspersoft Interactive Viewer and Editor. The JIVE UI is the interface of the report viewer in JasperReports Server, and the same JIVE UI is replicated on reports generated in clients using Visualize.js.

Most reports rendered by the JasperReports IO service have interactive abilities such as column sorting provided by a feature called JIVE: Jaspersoft Interactive Viewer and Editor. The JIVE UI is the interface of the report viewer which can be implemented in client applications using the JasperReports IO JavaScript API.

Not only does the JIVE UI allow users to sort and filter regular reports, it also provides many opportunities for you to further customize the appearance and behavior of your reports through Visualize.js the JasperReports IO JavaScript API.

This chapter contains the following sections:

Interacting With JIVE UI Components

The visualize.report JasperReports IO report interface exposes the updateComponent function that gives your script access to the JIVE UI. Using the updateComponent function, you can programmatically interact with the JIVE UI to do such things as set the sort order on a specified column, add a filter, and change the chart type. In addition, the undoAll function acts as a reset.

For the API reference of the visualize.report JasperReports IO report interface, see Report Functions.

First, your script must enable the default JIVE UI to make its components available after running a report:

```
var report = v.report({
    resource: "/public/SampleReport",
    defaultJiveUi : {
        enabled: true
    }
});
...
var components = report.data().components;
```

First, your script must enable the default JIVE UI to make its components available after running a report:

```
var report = jrioClient.report({
    resource: "/samples/reports/TableReport",
    defaultJiveUi : {
        enabled: true
    }
});
...
var components = report.data().components;
```

The components that can be modified are columns and charts. These components of the JIVE UI have an ID, but it may change from execution to execution. To refer to these components, create your report in JRXML and use the net.sf.jasperreports.components.name property to name them. In the case of a column,

this property should be set on the column definition in the table model. In Jaspersoft Studio, you can select the column in the Outline View, then go to Properties > Advanced, and under Misc > Properties you can define custom properties.

Then you can reference the component by this name, for example a column named sales, and use the updateComponent function to modify it.

```
report.updateComponent("sales", {
    sort : {
        order : "asc"
    }
});
```

Or:

```
report.updateComponent({
    name: "sales",
    sort : {
        order : "asc"
    }
});
```

We can also get an object that represents the named component of the JIVE UI:

```
var salesColumn = report
                    .data()
                    .components
                    .filter(function(c){ return c.name === "sales"})
                    .pop();
```

The following example shows how to create buttons whose click events modify the report through the JIVE UI. This example assumes you have a report whose components already have names, in this case, columns named my_accounts and my_dept, and a chart named revenue:

```
visualize({
    auth: { ...
    }
}, function (v) {
```

```
    //render report from provided resource
    var report = v.report({
        resource: "/public/SampleReport",
        container: "#container",
        success: printComponentsNames,
        error: handleError
    });

    $("#resetAll").on("click", function(){
        report.undoAll();
    });

    $("#changeAccounts").on("click", function () {
        report.updateComponent("my_accounts", {
            sort: {
                order: "asc"
            },
            filter: {
                operator: "greater_or_equal",
                value: 15000
            }
        }).fail(handleError);
    });

    $("#changeDept").on("click", function () {
        report.updateComponent("my_dept", {
            sort: {
                order: "desc"
            }
        }).fail(handleError);
    });

    $("#changeChart").on("click", function () {
        report.updateComponent("revenue", {
             chartType: "Pie"
        }).fail(handleError);
    });
```

```
    //show error
    function handleError(err){
        alert(err.message);
    }

    function printComponentsNames(data){
        data.components.forEach(function(c){
            console.log("Component Name: " + c.name, "Component Label: "+
c.label);
        });
    }

});
```

This example assumes you have a report whose components already have names, in this case, columns named ORDERID and SHIPNAME:

```
jrio.config({
...
});
jrio(function(jrioClient) {
       //render report from provided resource
       var report = jrioClient.report({
               resource: "/samples/reports/OrdersTable",
               container: "#reportContainer",
               success: printComponentsNames,
               error: handleError
       });

       $("#resetAll").on("click", function() {
               report.undoAll();
       });

       $("#changeOrders").on("click", function() {
               report.updateComponent("ORDERID", {
                       sort: {
                               order: "asc"
                       },
                       filter: {
```

```
                                operator: "greater_or_equal",
                                value: 10900
                        }
                }).fail(handleError);
        });

        $("#sortCustomers").on("click", function() {
                report.updateComponent("SHIPNAME", {
                        sort: {
                                order: "desc"
                        }
                }).fail(handleError);
        });

        //show error

        function handleError(err) {
                alert(err.message);
        }

        function printComponentsNames(data) {
                data.components.forEach(function(c) {
                        console.log("Component Name: " + c.name, "Component Label: " +
c.label);
                });
        }
});
```

The associated HTML has buttons that will invoke the JavaScript actions on the JIVE UI:

```
<!-- Provide the URL to visualize.js -->
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>
<button id="resetAll">Reset All</button>
<button id="changeAccounts">View Top Accounts</button>
<button id="changeDept">Sort Departments</button>
<button id="changeChart">Show Pie Chart</button>
<!-- Provide a container for the report -->
<div id="container"></div>
```

The associated HTML has buttons that will invoke the JavaScript actions on the JIVE UI:

```
<!-- Provide the URL to jrio.js -->
<script src="http://bi.example.com:8080/jriojsapi/client/jrio.js"></script>
<button id="resetAll">Reset All</button>
<button id="changeOrders">View Top Orders</button>
<button id="sortCustomers">Sort Customers</button>
<!-- Provide a container for the report -->
<div id="reportContainer"></div>
```

Using Floating Headers

One feature of the JIVE UI for tables and crosstabs is the floating header. When you turn on floating headers, the header rows of a table or crosstab float at the top of the container when you scroll down. The report container must allow scrolling for this to take effect. This means that CSS property overflow with values like scroll or auto must be specifically set for the report container.

To turn on floating headers for your interactive reports, set the following parameters when you enable the JIVE UI:

```
var report = v.report({
    resource: "/public/SampleReport",
    defaultJiveUi : {
        floatingTableHeadersEnabled: true,
        floatingCrosstabHeadersEnabled: true
    }
});
```

To turn on floating headers for your interactive reports, set the following parameters when you enable the JIVE UI:

```
var report = jrioClient.report({
    resource: "/samples/reports/TableReport",
    defaultJiveUi : {
        floatingTableHeadersEnabled: true,
        floatingCrosstabHeadersEnabled: true
```

```
    }
});
```

Changing the Chart Type

If you have the name of a chart component, you can easily set a new chart type and redraw the chart.

```
var mySalesChart = report
                    .data()
                    .components
                    .filter(function(c){ return c.name === "salesChart"})
                    .pop();

mySalesChart.chartType = "Bar";

report
    .updateComponent(mySalesChart)
    .done(function(){
        alert("Chart type changed!");
    })
    .fail(function(err){
        alert(err.message);
    });
```

Or:

```
report
    .updateComponent("salesChart", {
          chartType: "Bar"
    })
    .done(function(){
        alert("Chart type changed!");
    })
    .fail(function(err){
        alert(err.message);
    });
```

The following example creates a drop-down menu that lets users change the chart type. You could also set the chart type according to other states in your client.

This code also relies on the report.chart.types interface described in Discovering Available Charts and Formats.

```
visualize({
    auth: { ...
    }
}, function (v) {

    //persisted chart name
    var chartName = "geo_by_seg",
        $select = buildControl("Chart types: ", v.report.chart.types),
        report = v.report({
            resource: "/public/Reports/1._Geographic_Results_by_Segment_
Report",
            container: "#container",
            success: selectDefaultChartType
        });

    $select.on("change", function () {
        report.updateComponent(chartName, {
                chartType: $(this).val()
            })
            .done(function (component) {
                chartComponent = component;
            })
            .fail(function (error) {
                alert(error);
            });
    });

    function selectDefaultChartType(data) {
        var component = data.components
                            .filter(function (c) {
                                   return c.name === chartName;
                            })
                            .pop();
        if (component) {
            $select.find("option[value='" + component.chartType + "']")
                   .attr("selected", "selected");
        }
    }
```

```
    function buildControl(name, options) {

        function buildOptions(options) {
            var template = "<option>{value}</option>";
            return options.reduce(function (memo, option) {
                return memo + template.replace("{value}", option);
            }, "")
        }

        console.log(options);

        if (!options.length) {
            console.log(options);
        }

        var template = "<label>{label}</label><select>
{options}</select><br>",
            content = template.replace("{label}", name)
                .replace("{options}", buildOptions(options));

        var $control = $(content);
        $control.insertBefore($("#container"));
        return $control;
    }

});
```

This code also relies on the report.chart.types interface.

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    //persisted chart name
    var chartName = "chartOne",
        $select = buildControl("Chart types: ",
jrioClient.report.chart.types),
        report = jrioClient.report({
            resource: "/samples/reports/highcharts/HighchartsChart",
```

```
            container: "#reportContainer",
            success: selectDefaultChartType
        });

    $select.on("change", function () {
        report.updateComponent(chartName, {
            chartType: $(this).val()
        })
        .done(function (component) {
            chartComponent = component;
        })
        .fail(function (error) {
            alert(error);
        });
    });

    function selectDefaultChartType(data) {
        var component = data.components
            .filter(function (c) {
                return c.name === chartName;
            })
            .pop();
        if (component) {
            $select.find("option[value='" + component.chartType + "']")
                .attr("selected", "selected");
        }
    }

    function buildControl(name, options) {

        function buildOptions(options) {
            var template = "<option>{value}</option>";
            return options.reduce(function (memo, option) {
                return memo + template.replace("{value}", option);
            }, "")
        }

        console.log(options);

        if (!options.length) {
            console.log(options);
        }

        var template = "<label>{label}</label><select>
```

```
```

As shown in the following HTML, the control for the chart type is created dynamically by the JavaScript:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<!-- Provide the URL to visualize.js -->
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>
<!-- Provide a container for the report -->
<div id="container"></div>
```

As shown in the following HTML, the control for the chart type is created dynamically by the JavaScript:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<!-- Provide the URL to jrio.js -->
<script src="http://bi.example.com:8080/jriojsapi/client/jrio.js"></script>
<!--Provide a container for the report -->
<div id="reportContainer"></div>
```

Changing the Chart Properties

Those chart components that are based on Highcharts have a lot of interactivity such as built-in zooming and animation. The built-in zooming lets users select data, for example columns in a chart, but it can also interfere with touch interfaces. With Visualize.js the JasperReports IO JavaScript API, you have full control over these features and you can choose to allow your users access to them or not. For example, animation can be slow on mobile devices, so you could turn off both zooming and animation. Alternatively, if your users have a range of mobile devices, tablets, and desktop computers, then you could give users the choice of turning on or off these properties themselves.

The following example creates buttons to toggle several chart properties and demonstrates how to control them programmatically. First the HTML to create the buttons:

```
<script src="http://localhost:8080/jasperserver-
pro/client/visualize.js"></script>

<button id="disableAnimation">disable animation</button>
<button id="enableAnimation">enable animation</button>
<button id="resetAnimation">reset animation to initial state</button>

<button id="disableZoom">disable zoom</button>
<button id="zoomX">set zoom to 'x' type</button>
<button id="zoomY">set zoom to 'y' type</button>
<button id="zoomXY">set zoom to 'xy' type</button>
<button id="resetZoom">reset zoom to initial state</button>

<div id="container"></div>
```

The following example creates buttons to toggle several chart properties and demonstrates how to control them programmatically. First the HTML to create the buttons:

```
<script src="http://bi.example.com:8080/jriojsapi/client/jrio.js"></script>

<button id="disableAnimation">disable animation</button>
<button id="enableAnimation">enable animation</button>
<button id="resetAnimation">reset animation to initial state</button>

<button id="disableZoom">disable zoom</button>
<button id="zoomX">set zoom to 'x' type</button>
<button id="zoomY">set zoom to 'y' type</button>
<button id="zoomXY">set zoom to 'xy' type</button>
<button id="resetZoom">reset zoom to initial state</button>

<div id="reportContainer"></div>
```

Here are the API calls to set the various chart properties:

```
visualize({
    auth: {
        name: "jasperadmin",
        password: "jasperadmin",
```

```
        organization: "organization_1"
    }
}, function (v) {

    var report = v.report({
        resource: "/public/Samples/Reports/01._Geographic_Results_by_
Segment_Report",
        container: "#container",
        error: function(e) {
            alert(e);
        }
    });
```

```
    function changeChartProperty(prop, value) {
        var chartProps = report.chart();

        if (typeof value === "undefined") {
            delete chartProps[prop];
        } else {
            chartProps[prop] = value;
        }

        report.chart(chartProps).run().fail(function(e) { alert(e); });
    }

    $("#disableAnimation").on("click", function() {
        changeChartProperty("animation", false);
    });
```

```
    $("#enableAnimation").on("click", function() {
        changeChartProperty("animation", true);
    });

    $("#resetAnimation").on("click", function() {
        changeChartProperty("animation");
    });

    $("#disableZoom").on("click", function() {
        changeChartProperty("zoom", false);
    });
```

```
    $("#zoomX").on("click", function() {
        changeChartProperty("zoom", "x");
    });

    $("#zoomY").on("click", function() {
        changeChartProperty("zoom", "y");
    });

    $("#zoomXY").on("click", function() {
        changeChartProperty("zoom", "xy");
    });

    $("#resetZoom").on("click", function() {
        changeChartProperty("zoom");
    });
});
```

Here are the API calls to set the various chart properties:

```
jrio.config({
    ...
});
jrio(function(jrioClient) {

    var report = jrioClient.report({
        resource: "/samples/reports/highcharts/HighchartsChart",
        container: "#reportContainer",
        error: function(e) {
            alert(e);
        }
    });
    function changeChartProperty(prop, value) {
        var chartProps = report.chart();

        if (typeof value === "undefined") {
            delete chartProps[prop];
        } else {
            chartProps[prop] = value;
        }
```

```
        report.chart(chartProps).run().fail(function(e) { alert(e); });
    }

    $("#disableAnimation").on("click", function() {
        changeChartProperty("animation", false);
    });
    $("#enableAnimation").on("click", function() {
        changeChartProperty("animation", true);
    });

    $("#resetAnimation").on("click", function() {
        changeChartProperty("animation");
    });

    $("#disableZoom").on("click", function() {
        changeChartProperty("zoom", false);
    });

    $("#zoomX").on("click", function() {
        changeChartProperty("zoom", "x");
    });

    $("#zoomY").on("click", function() {
        changeChartProperty("zoom", "y");
    });

    $("#zoomXY").on("click", function() {
        changeChartProperty("zoom", "xy");
    });

    $("#resetZoom").on("click", function() {
        changeChartProperty("zoom");
    });
});
```

Undo and Redo Actions

As in JasperReports Server, the JIVE UI supports undo and redo actions that you can access programmatically with Visualize.js. As in many applications, undo and redo actions act like a stack, and the canUndo and canRedo events notify your page you are at either end of the stack.

```
visualize({
    auth: { ...
    }
}, function (v) {

    var chartComponent,
        report = v.report({
            resource: "/public/Samples/Reports/1._Geographic_Results_by_
Segment_Report",
            container: "#container",
            events: {
                canUndo: function (canUndo) {
                    if (canUndo) {
                        $("#undo, #undoAll").removeAttr("disabled");
                    } else {
                        $("#undo, #undoAll").attr("disabled", "disabled");
                    }
                },
                canRedo: function (canRedo) {
                    if (canRedo) {
                        $("#redo").removeAttr("disabled");
                    } else {
                        $("#redo").attr("disabled", "disabled");
                    }
                }
```

```
            },
            success: function (data) {
                chartComponent = data.components.pop();
                $("option[value='" + chartComponent.chartType + "']").attr
("selected", "selected");
            }
        });

    var chartTypeSelect = buildChartTypeSelect(report);

    chartTypeSelect.on("change", function () {
        report.updateComponent(chartComponent.id, {
            chartType: $(this).val()
        })
            .done(function (component) {
```

```
            chartComponent = component;
        })
            .fail(function (error) {
            console.log(error);
            alert(error);
        });
    });

    $("#undo").on("click", function () {
        report.undo().fail(function (err) {
            alert(err);
        });
    });

    $("#redo").on("click", function () {
        report.redo().fail(function (err) {
            alert(err);
        });
    });

    $("#undoAll").on("click", function () {
        report.undoAll().fail(function (err) {
            alert(err);
        });
    });
});
```

```
function buildChartTypeSelect(report) {

    var chartTypes = report.schema("chart").properties.chartType.enum,
        chartTypeSelect = $("#chartType");

    $.each(chartTypes, function (index, type) {
        chartTypeSelect.append("" + type + "");
    });

   return chartTypeSelect;
}
```

The JIVE UI supports undo and redo actions that you can access programmatically with the JasperReports IO JavaScript API. As in many applications, undo and redo actions act like a stack, and the canUndo and canRedo events notify your page you are at either end of the stack.

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    var chartComponent,
        report = jrioClient.report({
            resource: "/samples/reports/highcharts/HighchartsChart",
            container: "#reportContainer",
            events: {
                canUndo: function(canUndo) {
                    if (canUndo) {
                        $("#undo, #undoAll").removeAttr("disabled");
                    } else {
                        $("#undo, #undoAll").attr("disabled", "disabled");
                    }
                },
                canRedo: function(canRedo) {
                    if (canRedo) {
                        $("#redo").removeAttr("disabled");
                    } else {
                        $("#redo").attr("disabled", "disabled");
                    }
                }
            },
            success: function(data) {
                chartComponent = data.components.pop();
                $("option[value='" + chartComponent.chartType + "']").attr
("selected", "selected");
            }
        });
    var chartTypeSelect = buildChartTypeSelect(jrioClient.report);
    chartTypeSelect.on("change", function() {

        report.updateComponent(chartComponent.id, {
            chartType: $(this).val()
        })
            .done(function(component) {
            chartComponent = component;
            console.log("ttttt:" + $(this).val());
        })
            .fail(function(error) {
            console.log(error);
            alert(error);
```

```
        });
    });

    $("#undo").on("click", function() {
        report.undo().fail(function(err) {
            alert(err);
        });
    });

    $("#redo").on("click", function() {
        report.redo().fail(function(err) {
            alert(err);
        });
    });

    $("#undoAll").on("click", function () {
        report.undoAll().fail(function (err) {
            alert(err);
        });
    });
});
function buildChartTypeSelect(report) {
    chartTypeSelect = $("#chartType");
    var chartTypes = report.chart.types;
    chartTypeSelect = $("#chartType");
    $.each(chartTypes, function (index, type) {
        chartTypeSelect.append("<option value=\"" + type + "\">" + type +
"</option>");
    });
   return chartTypeSelect;
}
```

Associated HTML:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<!-- Provide the URL to visualize.js -->
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>

<select id="chartType"></select>
<button id="undo" disabled="disabled">Undo</button>
<button id="redo" disabled="disabled">Redo</button>
```

```
<button id="undoAll" disabled="disabled">Undo All</button>
<!-- Provide a container for the report -->
<div id="container"></div>
```

Associated HTML:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<!-- Provide the URL to jrio.js -->
<script src="http://bi.example.com:8080/jriojsapi/client/jrio.js"></script>

<select id="chartType"></select>
<button id="undo" disabled="disabled">Undo</button>
<button id="redo" disabled="disabled">Redo</button>
<button id="undoAll" disabled="disabled">Undo All</button>
<!-- Provide a container for the report -->
<div id="reportContainer"></div>
```

Sorting Table Columns

This code example shows how to set the three possible sorting orders on a column in the JIVE UI: ascending, descending, and no sorting.

```
visualize({
    auth: { ...
    }
}, function (v) {
    var report = v.report({
        resource: "/public/Samples/Reports/5g.AccountsReport",
        container: "#container",
        error: showError
    });

    $("#sortAsc").on("click", function () {
        report.updateComponent("name", {
```

```
            sort: {
                order: "asc"
            }
        })
        .fail(showError);
    });

    $("#sortDesc").on("click", function () {
        report.updateComponent("name", {
            sort: {
                order: "desc"
            }
        })
        .fail(showError);
    });

    $("#sortNone").on("click", function () {
        report.updateComponent("name", {
            sort: {}
        }).fail(showError);
    });

    function showError(err) {
        alert(err);
    }
});
```

This code example shows how to set the three possible sorting orders on a column in the JIVE UI: ascending, descending, and no sorting.

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    var report = jrioClient.report({
        resource:"/samples/reports/TableReport",
        container: "#reportContainer",
        error: showError
    });
```

```
    $("#sortAsc").on("click", function () {
        report.updateComponent("name", {
            sort: {
                order: "asc"
            }
        })
        .fail(showError);
    });

    $("#sortDesc").on("click", function () {
        report.updateComponent("name", {
            sort: {
                order: "desc"
            }
        })
        .fail(showError);
    });

    $("#sortNone").on("click", function () {
        report.updateComponent("name", {
            sort: {}
        }).fail(showError);
    });

    function showError(err) {
        alert(err);
    }
});
```

Associated HTML:

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="http://underscorejs.org/underscore-min.js"></script>
<!-- Provide the URL to visualize.js -->
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>

<button id="sortAsc">Sort NAME column ASCENDING</button>
<button id="sortDesc">Sort NAME column DESCENDING</button>
<button id="sortNone">Reset NAME column</button>

<!-- Provide a container for the report -->
<div id="container"></div>
```

Associated HTML:

```
<script src="http://code.jquery.com/jquery-2.1.0.js"></script>
<script src="http://underscorejs.org/underscore-min.js"></script>
<!-- Provide the URL to jrio.js -->
<script src="http://bi.example.com:8080/jriojsapi/client/jrio.js"></script>

<button id="sortAsc">Sort NAME column ASCENDING</button>
<button id="sortDesc">Sort NAME column DESCENDING</button>
<button id="sortNone">Reset NAME column</button>

<!-- Provide a container for the report -->
<div id="reportContainer"></div>
```

Filtering Table Columns

This code example shows how to define filters on columns of various data types (dates, strings, numeric) in the JIVE UI. It also shows several filter operator such as equal, greater, between, contain (for string matching), and before (for times and dates).

```
visualize({
    auth: { ...
    }
}, function (v) {
    var report = v.report({
              resource:"/public/viz/report_with_different_column_types",
            container: "#container",
            error: function(err) {
                alert(err);
            }
        });

    $("#setTimestampRange").on("click", function() {
        report.updateComponent("column_timestamp", {
            filter: {
                operator: "between",
                value: [$("#betweenDates1").val(), $("#betweenDates2").val
()]
```

```
        }
    }).fail(handleError);
});
```

```
$("#resetTimestampFilter").on("click", function() {
    report.updateComponent("column_timestamp", {
        filter: {}
    }).fail(handleError);
});
```

```
$("#setBooleanTrue").on("click", function() {
    report.updateComponent("column_boolean", {
        filter: {
            operator: "equal",
            value: true
        }
    }).fail(handleError);
});

$("#resetBoolean").on("click", function() {
    report.updateComponent("column_boolean", {
        filter: {}
    }).fail(handleError);
});

$("#setStringContains").on("click", function() {
    report.updateComponent("column_string", {
        filter: {
            operator: "contain",
            value: $("#stringContains").val()
        }
    }).fail(handleError);
});

$("#resetString").on("click", function() {
    report.updateComponent("column_string", {
        filter: {}
    }).fail(handleError);
});
```

```
    $("#setNumericGreater").on("click", function() {
        report.updateComponent("column_double", {
            filter: {
                operator: "greater",
                value: parseFloat($("#numericGreater").val(), 10)
            }
        }).fail(handleError);
    });

    $("#resetNumeric").on("click", function() {
        report.updateComponent("column_double", {
            filter: {}
        }).fail(handleError);
    });

    $("#setTimeBefore").on("click", function() {
        report.updateComponent("column_time", {
            filter: {
                operator: "before",
                value: $("#timeBefore").val()
            }
        }).fail(handleError);
    });

    $("#resetTime").on("click", function() {
        report.updateComponent("column_time", {
            filter: {}
        }).fail(handleError);
    });
});

function handleError(err) {
     console.log(err);
     alert(err);
}
```

This code example shows how to define filters on columns of various data types (dates, strings, numeric) in the JIVE UI. It also shows several filter operator such as equal, greater, between, contain (for string matching), and before (for times and dates).

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    var report = jrioClient.report({
             resource:"/samples/reports/OrdersTable",
            container: "#reportContainer",
            error: function(err) {
                alert(err);
            }
        });

    $("#setTimestampRange").on("click", function() {
        report.updateComponent("ORDERDATE", {
            filter: {
                operator: "between",
                value: [$("#betweenDates1").val(), $("#betweenDates2").val
()]
            }
        }).fail(handleError);
    });
    $("#resetTimestampFilter").on("click", function() {
        report.updateComponent("ORDERDATE", {
            filter: {}
        }).fail(handleError);
    });

    $("#setStringContains").on("click", function() {
        report.updateComponent("SHIPNAME", {
            filter: {
                operator: "contain",
                value: $("#stringContains").val()
            }
        }).fail(handleError);
    });

    $("#resetString").on("click", function() {
        report.updateComponent("SHIPNAME", {
            filter: {}
        }).fail(handleError);
    });

    $("#setNumericGreater").on("click", function() {
```

```
        report.updateComponent("ORDERID", {
            filter: {
                operator: "greater",
                value: parseFloat($("#numericGreater").val(), 10)
            }
        }).fail(handleError);
    });

    $("#resetNumeric").on("click", function() {
        report.updateComponent("ORDERID", {
            filter: {}
        }).fail(handleError);
    });
});

function handleError(err) {
     console.log(err);
     alert(err);
}
```

Associated HTML:

```
<script src="http://code.jquery.com/jquery-2.1.0.js"></script>
<script src="http://underscorejs.org/underscore-min.js"></script>
<!-- Provide the URL to visualize.js -->
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>


<input type="text" value="2014-04-10T00:00:00" id="betweenDates1"/> -
<input type="text" id="betweenDates2" value="2014-04-24T00:00:00"/>
<button id="setTimestampRange">Set timestamp range</button>
<button id="resetTimestampFilter">Reset timestamp filter</button>
<br/><br/>
<button id="setBooleanTrue">Filter boolean column to true</button>
<button id="resetBoolean">Reset boolean filter</button>
<br/><br/>
<input type="text" value="hou" id="stringContains"/>
<button id="setStringContains">Set string column contains</button>
<button id="resetString">Reset string filter</button>
<br/><br/>
<input type="text" value="40.99" id="numericGreater"/>
```

```
<button id="setNumericGreater">Set numeric column greater than</button>
<button id="resetNumeric">Reset numeric filter</button>
<br/><br/>
<input type="text" value="13:15:43" id="timeBefore"/>
<button id="setTimeBefore">Set time column before than</button>
<button id="resetTime">Reset time filter</button>

<!-- Provide a container for the report -->
<div id="container"></div>
```

Associated HTML:

```
<script src="http://code.jquery.com/jquery-2.1.0.js"></script>
<script src="http://underscorejs.org/underscore-min.js"></script>
<!-- Provide the URL to jrio.js -->
<script src="http://bi.example.com:8080/jriojsapi/client/jrio.js"></script>

<input type="text" value="1997-01-10T00:00:00" id="betweenDates1"/> -
<input type="text" id="betweenDates2" value="1997-10-24T00:00:00"/>
<button id="setTimestampRange">Set timestamp range</button>
<button id="resetTimestampFilter">Reset timestamp filter</button>
<br/><br/>
<input type="text" value="ctu" id="stringContains"/>
<button id="setStringContains">Set string column contains</button>
<button id="resetString">Reset string filter</button>
<br/><br/>
<input type="text" value="10500" id="numericGreater"/>
<button id="setNumericGreater">Set numeric column greater than</button>
<button id="resetNumeric">Reset numeric filter</button>

<!-- Provide a container for the report -->
<div id="reportContainer"></div>
```

Formatting Table Columns

The JIVE UI allows you to format columns by setting the alignment, color, font, size, and background of text in both headings and cells. You can also set the numeric format of cells, such as the precision, negative indicator, and currency. Note that the initial appearance of

any numbers also depends on the locale set either by default on JasperReports Server, or specified in your script request, as described in Requesting the Visualize.js Script.

```
visualize({
    auth: { ...
    }
}, function (v) {
    var columns,
    report = v.report({
        resource: "/public/viz/report_with_different_column_types",
        container: "#container",
        events: {
```

```
            reportCompleted: function (status, error) {
                if (status === "ready") {
                    columns = _.filter(report.data().components, function
(component) {
                        return component.componentType == "tableColumn";
                    });

                    var column4 = columns[4];

                    $("#label").val(column4.label);

                    $("#headingFormatAlign").val
(column4.headingFormat.align);
                    $("#headingFormatBgColor").val
(column4.headingFormat.backgroundColor);
                    $("#headingFormatFontSize").val
(column4.headingFormat.font.size);
                    $("#headingFormatFontColor").val
(column4.headingFormat.font.color);
                    $("#headingFormatFontName").val
(column4.headingFormat.font.name);
                    if (column4.headingFormat.font.bold) {
```

```
                        $("#headingFormatFontBold").attr("checked",
"checked");
                    } else {
                        $("#headingFormatFontBold").removeAttr("checked");
                    }
                    if (column4.headingFormat.font.italic) {
                        $("#headingFormatFontItalic").attr("checked",
"checked");
                    } else {
                        $("#headingFormatFontItalic").removeAttr
("checked");
                    }
                    if (column4.headingFormat.font.underline) {
                        $("#headingFormatFontUnderline").attr("checked",
"checked");
                    } else {
                        $("#headingFormatFontUnderline").removeAttr
("checked");
                    }

                    $("#detailsRowFormatAlign").val
(column4.detailsRowFormat.align);
                    $("#detailsRowFormatBgColor").val
(column4.detailsRowFormat.backgroundColor);
                    $("#detailsRowFormatFontSize").val
(column4.detailsRowFormat.font.size);
                    $("#detailsRowFormatFontColor").val
(column4.detailsRowFormat.font.color);
                    $("#detailsRowFormatFontName").val
(column4.detailsRowFormat.font.name);
                    if (column4.detailsRowFormat.font.bold) {
                        $("#detailsRowFormatFontBold").attr("checked",
"checked");
                    } else {
                        $("#detailsRowFormatFontBold").removeAttr
("checked");
                    }
                    if (column4.detailsRowFormat.font.italic) {
                        $("#detailsRowFormatFontItalic").attr("checked",
```

```
mat.pattern.percentage) {
                        $("#detailsRowFormatPatternPercentage").attr
("checked", "checked");
                    } else {
                        $("#detailsRowFormatPatternPercentage").removeAttr
("checked");
                    }

                    if (column4.detailsRowFormat.pattern.grouping) {
                        $("#detailsRowFormatPatternGrouping").attr
("checked", "checked");
                    } else {
                        $("#detailsRowFormatPatternGrouping").removeAttr
("checked");
                    }
                }
            }
        },
        error: function (err) {
            alert(err);
        }
    });

    $("#changeHeadingFormat").on("click", function () {
        report.updateComponent(columns[4].id, {
            headingFormat: {
                align: $("#headingFormatAlign").val(),
                backgroundColor: $("#headingFormatBgColor").val(),
                font: {
                    size: parseFloat($("#headingFormatFontSize").val()),
                    color: $("#headingFormatFontColor").val(),
                    underline: $("#headingFormatFontUnderline").is
(":checked"),
                    bold: $("#headingFormatFontBold").is(":checked"),
                    italic: $("#headingFormatFontItalic").is(":checked"),
                    name: $("#headingFormatFontName").val()
                }
            }
        }).fail(function (e) {
            alert(e);
        });
    });
```

```
            label: $("#label").val()
        }).fail(function (e) {
            alert(e);
        });
    });
});
```

The JIVE UI allows you to format columns by setting the alignment, color, font, size, and background of text in both headings and cells. You can also set the numeric format of cells, such as the precision, negative indicator, and currency.

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
  var columns,
    report = jrioClient.report({
      resource: "/samples/reports/TableReport",
      container: "#reportContainer",
      events: {
        reportCompleted: function(status, error) {
          if (status === "ready") {
            columns = _.filter(report.data().components, function
(component) {
              return component.componentType == "tableColumn";
            });
            var column4 = columns[4];
            $("#label").val(column4.label);
            $("#headingFormatAlign").val(column4.headingFormat.align);
            $("#headingFormatBgColor").val
(column4.headingFormat.backgroundColor);
            $("#headingFormatFontSize").val
(column4.headingFormat.font.size);
            $("#headingFormatFontColor").val
(column4.headingFormat.font.color);
            $("#headingFormatFontName").val
(column4.headingFormat.font.name);
```

```
            if (column4.headingFormat.font.bold) {
              $("#headingFormatFontBold").attr("checked", "checked");
            } else {
              $("#headingFormatFontBold").removeAttr("checked");
            }
            if (column4.headingFormat.font.italic) {
              $("#headingFormatFontItalic").attr("checked", "checked");
            } else {
              $("#headingFormatFontItalic").removeAttr("checked");
            }
            if (column4.headingFormat.font.underline) {
              $("#headingFormatFontUnderline").attr("checked", "checked");
            } else {
              $("#headingFormatFontUnderline").removeAttr("checked");
            }
            $("#detailsRowFormatAlign").val
(column4.detailsRowFormat.align);
            $("#detailsRowFormatBgColor").val
(column4.detailsRowFormat.backgroundColor);
            $("#detailsRowFormatFontSize").val
(column4.detailsRowFormat.font.size);
            $("#detailsRowFormatFontColor").val
(column4.detailsRowFormat.font.color);
            $("#detailsRowFormatFontName").val
(column4.detailsRowFormat.font.name);
            if (column4.detailsRowFormat.font.bold) {
              $("#detailsRowFormatFontBold").attr("checked", "checked");
            } else {
              $("#detailsRowFormatFontBold").removeAttr("checked");
            }
            if (column4.detailsRowFormat.font.italic) {
              $("#detailsRowFormatFontItalic").attr("checked", "checked");
            } else {
              $("#detailsRowFormatFontItalic").removeAttr("checked");
            }
            if (column4.detailsRowFormat.font.underline) {
              $("#detailsRowFormatFontUnderline").attr("checked",
"checked");
            } else {
              $("#detailsRowFormatFontUnderline").removeAttr("checked");
            }
          }
        }
```

```
      },
      error: function(err) {
        alert(err);
      }
    });
   $("#changeHeadingFormat").on("click", function() {
    report.updateComponent(columns[4].id, {
      headingFormat: {
        align: $("#headingFormatAlign").val(),
        backgroundColor: $("#headingFormatBgColor").val(),
        font: {
          size: parseFloat($("#headingFormatFontSize").val()),
          color: $("#headingFormatFontColor").val(),
          underline: $("#headingFormatFontUnderline").is(":checked"),
          bold: $("#headingFormatFontBold").is(":checked"),
          italic: $("#headingFormatFontItalic").is(":checked"),
          name: $("#headingFormatFontName").val()
        }
                                                }
```

```
    }).fail(function(e) {
      alert(e);
    });
  });

  $("#changeDetailsRowFormat").on("click", function() {
    report.updateComponent(columns[4].id, {
      detailsRowFormat: {
        align: $("#detailsRowFormatAlign").val(),
        backgroundColor: $("#detailsRowFormatBgColor").val(),
        font: {
          size: parseFloat($("#detailsRowFormatFontSize").val()),
          color: $("#detailsRowFormatFontColor").val(),
          underline: $("#detailsRowFormatFontUnderline").is(":checked"),
          bold: $("#detailsRowFormatFontBold").is(":checked"),
          italic: $("#detailsRowFormatFontItalic").is(":checked"),
          name: $("#detailsRowFormatFontName").val()
        }
      }
    }).fail(function(e) {
      alert(e);
    });
```

```
  });

  $("#changeLabel").on("click", function() {
    report.updateComponent(columns[4].id, {
      label: $("#label").val()
    }).fail(function(e) {
      alert(e);
    });
  });
});
```

The associated HTML has static controls for selecting all the formatting options that the script above can modify in the report.

```
<script src="http://code.jquery.com/jquery-2.1.0.js"></script>
<script src="http://underscorejs.org/underscore-min.js"></script>
<!-- Provide the URL to visualize.js -->
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>

<div style="float: left;">
    <h3>Heading format for 5th column</h3>
    Align:  <select id="headingFormatAlign">
                <option value="left">left</option>
                <option value="center">center</option>
                <option value="right">right</option></select>
    <br/>
    Background color: <input type="text" id="headingFormatBgColor"
value=""/>
    <br/>
    Font size: <input type="text" id="headingFormatFontSize" value=""/>
    <br/>
    Font color: <input type="text" id="headingFormatFontColor" value=""/>
    <br/>
    Font name: <input type="text" id="headingFormatFontName" value=""/>
    <br/>
    Bold: <input type="checkbox" id="headingFormatFontBold" value="true"/>
    <br/>
```

```
    Italic: <input type="checkbox" id="headingFormatFontItalic"
value="true"/>
    <br/>
    Underline: <input type="checkbox" id="headingFormatFontUnderline"
value="true"/>
    <br/><br/>
    <button id="changeHeadingFormat">Change heading format</button>
</div>
<div style="float: left;">
    <h3>Details row format for 5th column</h3>
    Align: <select id="detailsRowFormatAlign">
        <option value="left">left</option>
        <option value="center">center</option>
        <option value="right">right</option></select>
    <br/>
    Background color: <input type="text" id="detailsRowFormatBgColor"
value=""/>
    <br/>
    Font size: <input type="text" id="detailsRowFormatFontSize" value=""/>
    <br/>
    Font color: <input type="text" id="detailsRowFormatFontColor"
value=""/>
    <br/>
    Font name: <input type="text" id="detailsRowFormatFontName" value=""/>
    <br/>
    Bold: <input type="checkbox" id="detailsRowFormatFontBold"
value="true"/>
    <br/>
    Italic: <input type="checkbox" id="detailsRowFormatFontItalic"
value="true"/>
    <br/>
    Underline: <input type="checkbox" id="detailsRowFormatFontUnderline"
value="true"/>
    <br/><br/>
    <b>Number pattern:</b>
    <br/>
    Negative format: <input type="text"
id="detailsRowFormatPatternNegativeFormat"/>
    <br/>
    Precision: <input type="text" id="detailsRowFormatPatternPrecision"/>
    <br/>
    Currency: <select id="detailsRowFormatPatternCurrency">
                    <option value="">----</option>
```

The associated HTML has static controls for selecting all the formatting options that the script above can modify in the report.

```
<script src="http://code.jquery.com/jquery-2.1.0.js"></script>
<script src="http://underscorejs.org/underscore-min.js"></script>
<!-- Provide the URL to jrio.js -->
<script src="http://bi.example.com:8080/jriojsapi/client/jrio.js"></script>

<div >
    <h3>Heading format for 5th column</h3>
    Align:  <select id="headingFormatAlign">
                <option value="left">left</option>
                <option value="center">center</option>
                <option value="right">right</option></select>
    <br/>
    Background color: <input type="text" id="headingFormatBgColor"
value=""/>
    <br/>
    Font size: <input type="text" id="headingFormatFontSize" value=""/>
    <br/>
    Font color: <input type="text" id="headingFormatFontColor" value=""/>
    <br/>
    Font name: <input type="text" id="headingFormatFontName" value=""/>
    <br/>
    Bold: <input type="checkbox" id="headingFormatFontBold" value="true"/>
    <br/>
    Italic: <input type="checkbox" id="headingFormatFontItalic"
value="true"/>
    <br/>
    Underline: <input type="checkbox" id="headingFormatFontUnderline"
value="true"/>
    <br/><br/>
    <button id="changeHeadingFormat">Change heading format</button>
</div>
<div >
    <h3>Details row format for 5th column</h3>
    Align: <select id="detailsRowFormatAlign">
        <option value="left">left</option>
```

```
        <option value="center">center</option>
        <option value="right">right</option></select>
    <br/>
    Background color: <input type="text" id="detailsRowFormatBgColor"
value=""/>
    <br/>
    Font size: <input type="text" id="detailsRowFormatFontSize" value=""/>
    <br/>
    Font color: <input type="text" id="detailsRowFormatFontColor"
value=""/>
    <br/>
    Font name: <input type="text" id="detailsRowFormatFontName" value=""/>
    <br/>
    Bold: <input type="checkbox" id="detailsRowFormatFontBold"
value="true"/>
    <br/>
    Italic: <input type="checkbox" id="detailsRowFormatFontItalic"
value="true"/>
    <br/>
    Underline: <input type="checkbox" id="detailsRowFormatFontUnderline"
value="true"/>
    <br/><br/>
    <button id="changeDetailsRowFormat">Change details row format</button>
</div>
<div >
    <h3>Change label of 5th column</h3>
    <br/>
    Label <input type="text" id="label"/>
    <br/>
    <button id="changeLabel">Change label</button>
</div>
<div ></div>

<!-- Provide a container for the report -->
<div id="reportContainer"></div>
```

Conditional Formatting on Table Columns

The JIVE UI also supports conditional formatting so that you can change the appearance of a cell's contents based on its value. This example highlights cells in a given column that have a certain value by changing their text color and the cell background color. Note that the column name must be known ahead of time, for example by looking at your JRXML.

```
visualize({
    auth: { ...
    }
}, function (v) {
    // column name from JRXML (field name by default)
    var salesColumnName = "sales_fact_ALL.sales_fact_ALL__store_sales_
2013",
        report = v.report({
            resource: "/public/Samples/Reports/04._Product_Results_by_
Store_Type_Report",
            container: "#container",
            error: showError
        });

    $("#changeConditions").on("click", function() {
            report.updateComponent(salesColumnName, {
                conditions: [
                    {
                        operator: "greater",
                        value: 10,
                        backgroundColor: null,
                        font: {
                            color: "FF0000",
                            bold: true,
                            underline: true,
                            italic: true
                        }
                    },
                    {
                        operator: "between",
                        value: [5, 9],
                        backgroundColor: "00FF00",
                        font: {
                            color: "0000FF"
                        }
                    }
                ]
            })
            .then(printConditions)
            .fail(showError);
    });

    function printConditions(component){
        console.log("Conditions: "+ component.conditions);
```

```
    }

    function showError(err) {
           alert(err);
    }
});
```

The JIVE UI also supports conditional formatting so that you can change the appearance of a cell's contents based on its value. This example highlights cells in a given column that have a certain value by changing their text color and the cell background color. Note that the column name must be known ahead of time, for example by looking at your JRXML.

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    // column name from JRXML (field name by default)
    var report = jrioClient.report({
            resource: "/samples/reports/OrdersTable",
            container: "#reportContainer",
            error: showError
        });

    $("#changeConditions").on("click", function() {
            report.updateComponent("ORDERID", {
                conditions: [
                    {
                        operator: "greater",
                        value: 10500,
                        backgroundColor: null,
                        font: {
                            color: "FF0000",
                            bold: true,
                            underline: true,
                            italic: true
                        }
                    },
                    {
                        operator: "between",
                        value: [10900, 11000],
```

```
                        backgroundColor: "00FF00",
                        font: {
                            color: "0000FF"
                        }
                    }
                ]
            })
            .then(printConditions)
            .fail(showError);
    });

    function printConditions(component){
        console.log("Conditions: "+ component.conditions);
    }

    function showError(err) {
           alert(err);
    }
});
```

This example has a single button that allows the user to apply the conditional formatting when the report is loaded:

```
<script src="http://code.jquery.com/jquery-2.1.0.js"></script>
<script src="http://underscorejs.org/underscore-min.js"></script>
<!-- Provide the URL to visualize.js -->
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>

<button id="changeConditions">Change conditions for numeric column</button>

<!-- Provide a container for the report -->
<div id="container"></div>
```

This example has a single button that allows the user to apply the conditional formatting when the report is loaded:

```
<script src="http://code.jquery.com/jquery-2.1.0.js"></script>
```

```
<script src="http://underscorejs.org/underscore-min.js"></script>
<!-- Provide the URL to jrio.js -->
<script src="http://bi.example.com:8080/jriojsapi/client/jrio.js"></script>

<button id="changeConditions">Change conditions for numeric column</button>

<!-- Provide a container for the report -->
<div id="reportContainer"></div>
```

Sorting Crosstab Columns

Crosstabs are more complex and do not have as many formatting options. This example shows how to sort the values in a given column of a crosstab (the rows are rearranged). Note that the code is slightly different than Sorting Table Columns.

```
visualize({
    auth: {
        name: "superuser",
        password: "superuser"
    }
}, function (v) {
    var column2,
    report = v.report({
        resource: "/public/MyReports/crosstabReport",
        container: "#container",
        events: {
            reportCompleted: function (status, error) {
                if (status === "ready") {
                    var columns = _.filter(report.data().components,
function (component) {
                        return component.componentType ==
"crosstabDataColumn";
                    });

                    column2 = columns[1];
                    console.log(columns);
                }
```

```
            }
        },
        error: function (err) {
            alert(err);
        }
    });

    $("#sortAsc").on("click", function () {
        report.updateComponent(column2.id, {
            sort: {
                order: "asc"
            }
        }).fail(function (e) {
            alert(e);
        });
    });

    $("#sortDesc").on("click", function () {
        report.updateComponent(column2.id, {
            sort: {
                order: "desc"
            }
        }).fail(function (e) {
            alert(e);
        });
    });

    $("#sortNone").on("click", function () {
        report.updateComponent(column2.id, {
            sort: {}
        }).fail(function (e) {
            alert(e);
        });
    });
});
```

Crosstabs are more complex and do not have as many formatting options. This example shows how to sort the values in a given column of a crosstab (the rows are rearranged). Note that the code is slightly different than Sorting Table Columns.

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    var column2,
    report = jrioClient.report({
        resource: "/samples/reports/crosstabs/OrdersReport",
        container: "#reportContainer",
        events: {
            reportCompleted: function(status, error) {
                if (status === "ready") {
                    var columns = _.filter(report.data().components,
function(component) {
                        return component.componentType ==
"crosstabDataColumn";
                    });

                    column2 = columns[1];
                    console.log(columns);
                }
            }
        },
        error: function(err) {
            alert(err);
        }
    });

    $("#sortAsc").on("click", function () {
        report.updateComponent(column2.id, {
            sort: {
                order: "asc"
            }
        }).fail(function(e) {
            alert(e);
        });
    });

    $("#sortDesc").on("click", function() {
        report.updateComponent(column2.id, {
            sort: {
                order: "desc"
            }
        }).fail(function(e) {
            alert(e);
```

```
        });
    });

    $("#sortNone").on("click", function() {
        report.updateComponent(column2.id, {
            sort: {}
        }).fail(function(e) {
            alert(e);
        });
    });
});
```

The associated HTML has the buttons to trigger the sorting:

```
<script src="http://code.jquery.com/jquery-2.1.0.js"></script>
<script src="http://underscorejs.org/underscore-min.js"></script>
<!-- Provide the URL to visualize.js -->
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>

<button id="sortAsc">Sort 2nd column ascending</button>
<button id="sortDesc">Sort 2nd column descending</button>
<button id="sortNone">Do not sort on 2nd column</button>

<!-- Provide a container for the report -->
<div id="container"></div>
```

The associated HTML has the buttons to trigger the sorting:

```
<script src="http://code.jquery.com/jquery-2.1.0.js"></script>
<script src="http://underscorejs.org/underscore-min.js"></script>
<!-- Provide the URL to jrio.js -->
<script src="http://bi.example.com:8080/jriojsapi/client/jrio.js"></script>

<button id="sortAsc">Sort 2nd column ascending</button>
<button id="sortDesc">Sort 2nd column descending</button>
<button id="sortNone">Do not sort on 2nd column</button>

<!-- Provide a container for the report -->
<div id="reportContainer"></div>
```

Sorting Crosstab Rows

This example shows how to sort the values in a given row of a crosstab (the columns are rearranged).

```
visualize({
    auth: { ...
    }
}, function (v) {
    var row,
    report = v.report({
        resource: "/public/MyReports/crosstabReport",
        container: "#container",
        events: {
            reportCompleted: function (status, error) {
                if (status === "ready") {
                    row = _.filter(report.data().components, function
(component) {
                        return component.componentType ==
"crosstabRowGroup";
                    })[0];
                }
            }
        },
        error: function (err) {
            alert(err);
        }
    });

    $("#sortAsc").on("click", function () {
        report.updateComponent(row.id, {
            sort: {
                order: "asc"
            }
        }).fail(function (e) {
            alert(e);
        });
    });

    $("#sortDesc").on("click", function () {
        report.updateComponent(row.id, {
            sort: {
                order: "desc"
```

```
            }
        }).fail(function (e) {
            alert(e);
        });
    });

    $("#sortNone").on("click", function () {
        report.updateComponent(row.id, {
            sort: {}
        }).fail(function (e) {
            alert(e);
        });
    });
});
```

This example shows how to sort the values in a given row of a crosstab (the columns are rearranged).

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    var row,
    report = jrioClient.report({
        resource: "/samples/reports/crosstabs/OrdersReport",
        container: "#reportContainer",
        events: {
            reportCompleted: function(status, error) {
                if (status === "ready") {
                    row = _.filter(report.data().components, function
(component) {
                        return component.componentType ==
"crosstabRowGroup";
                    })[0];
                }
            }
        },
        error: function(err) {
            alert(err);
        }
    });
```

```
    $("#sortAsc").on("click", function() {
        report.updateComponent(row.id, {
            sort: {
                order: "asc"
            }
        }).fail(function(e) {
            alert(e);
        });
    });

    $("#sortDesc").on("click", function() {
        report.updateComponent(row.id, {
            sort: {
                order: "desc"
            }
        }).fail(function (e) {
            alert(e);
        });
    });

    $("#sortNone").on("click", function () {
        report.updateComponent(row.id, {
            sort: {}
        }).fail(function(e) {
            alert(e);
        });
    });
});
```

The associated HTML has the buttons to trigger the sorting:

```
<script src="http://code.jquery.com/jquery-2.1.0.js"></script>
<script src="http://underscorejs.org/underscore-min.js"></script>
<!-- Provide the URL to visualize.js -->
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>

<button id="sortAsc">Sort rows ascending</button>
<button id="sortDesc">Sort rows descending</button>
<button id="sortNone">Do not sort rows</button>

<!-- Provide a container for the report -->
<div id="container"></div>
```

The associated HTML has the buttons to trigger the sorting:

```
<script src="http://code.jquery.com/jquery-2.1.0.js"></script>
<script src="http://underscorejs.org/underscore-min.js"></script>
<!-- Provide the URL to jrio.js -->
<script src="http://bi.example.com:8080/jriojsapi/client/jrio.js"></script>

<button id="sortAsc">Sort rows ascending</button>
<button id="sortDesc">Sort rows descending</button>
<button id="sortNone">Do not sort rows</button>

<!-- Provide a container for the report -->
<div id="reportContainer"></div>
```

Implementing Search in Reports

The JIVE UI supports a search capability within the report. The following example relies on a page with a simple search input.

```
<input id="search-query" type="input" />
<button id="search-button">Search</button>
<!--Provide container to render your visualization-->
<div id="container"></div>
```

The JIVE UI supports a search capability within the report. The following example relies on a page with a simple search input.

```
<input id="search-query" type="input" />
<button id="search-button">Search</button>
<!--Provide container to render your visualization-->
<div id="reportContainer"></div>
```

Then you can use the search function to return a list of matches in the report. In this example, the search button triggers the function and passes the search term. It uses the

console to display the results, but you can use them to locate the search term in a paginated report.

```
visualize({
    auth: {
        name: "jasperadmin",
        password: "jasperadmin",
        organization: "organization_1"
    }
}, function (v) {

    //render report from provided resource
    var report = v.report({
        resource: "/public/Samples/Reports/AllAccounts",
        error: handleError,
        container: "#container"
    });

    $("#search-button").click(function(){
        report
        .search($("#search-query").val())
            .done(function(results){
                !results.length && console.log("The search did not return
any results!");
                for (var i = 0; i < results.length; i++) {
                    console.log("found " + results[i].hitCount + " results
on page: #" +
                                    results[i].page);
                }
            })
            .fail(handleError);
    });


    //show error
    function handleError(err){
        alert(err.message);
    }

});
```

Then you can use the search function to return a list of matches in the report. In this example, the search button triggers the function and passes the search item. It uses the console to display the results, but you can use them to locate the search term in a paginated report.

```
jrio.config({
    ...
});
jrio(function(jrioClient) {

    //render report from provided resource
    var report = jrioClient.report({
        resource: "/samples/reports/TableReport",
        error: handleError,
        container: "#reportContainer"
    });

    $("#search-button").click(function(){
        report
        .search($("#search-query").val())
            .done(function(results){
                !results.length && console.log("The search did not return
any results!");
                for (var i = 0; i < results.length; i++) {
                    console.log("found " + results[i].hitCount + " results
on page: #" +
                                    results[i].page);
                }
            })
            .fail(handleError);
    });


    //show error
    function handleError(err){
        alert(err.message);
    }

});
```

The search function supports several arguments to refine the search:

```
    $("#search-button").click(function(){
        report
        .search({
            text: $("#search-query").val(),
            caseSensitive: true,
```

```
            wholeWordsOnly: true
        })
           ...
```

Providing Bookmarks in Reports

The JIVE UI also supports bookmarks that are embedded within the report. You must create your report with bookmarks, but then Visualize.js can make them available on your page. The following example has a container for the bookmarks and one for the report:

```
<div>
    <h4>Bookmarks</h4>
    <div id="bookmarksContainer"></div>
</div>
<!--Provide container to render your visualization-->
<div id="container"></div>
```

The JIVE UI also supports bookmarks that are embedded within the report. You must create your report with bookmarks, but then the JasperReports IO JavaScript API can make them available on your page. The following example has a container for the bookmarks and one for the report:

```
<div>
    <h4>Bookmarks</h4>
    <div id="bookmarksContainer"></div>
</div>
<!--Provide container to render your visualization-->
<div id="reportContainer"></div>
```

Then you need a function to read the bookmarks in the report and place them in the container. A handler then responds to clicks on the bookmarks.

```
visualize({
    auth: {
        name: "jasperadmin",
        password: "jasperadmin",
        organization: "Organization_1"
    }
}, function (v) {

    //render report from provided resource
    var report = v.report({
        // resource: "/public/Samples/Reports/AllAccounts",
        resource: "/reports/interactive/TableReport",
        error: handleError,
        container: "#container",
        events: {
            bookmarksReady: handleBookmarks
        }
    });

    //show error
    function handleError(err){
        alert(err.message);
    }

    $("#bookmarksContainer").on("click", ".jr_bookmark", function(evt) {
        report.pages({
            anchor: $(this).data("anchor")
        }).run();
    });

    // handle bookmarks
    function handleBookmarks(bookmarks, container) {
        var li, ul = $("<ul/>");
        !container && $("#bookmarksContainer").empty();
        container = container || $("#bookmarksContainer");

        $.each(bookmarks, function(i, bookmark) {
            li = $("<li><span class='jr_bookmark' title='Anchor: " +
bookmark.anchor + ", page: " + bookmark.page + "' data-anchor='" +
bookmark.anchor + "' data-page='" + bookmark.page + "'>" + bookmark.anchor
+ "</span></li>");
            bookmark.bookmarks && handleBookmarks(bookmark.bookmarks, li);
            ul.append(li);
        });
```

```
        container.append(ul);
    }
});
```

Then you need a function to read the bookmarks in the report and place them in the container. A handler then responds to clicks on the bookmarks.

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    //render report from provided resource
    var report = jrioClient.report({
        resource: "/samples/reports/TableReport",
        error: handleError,
        container: "#reportContainer",
        events: {
            bookmarksReady: handleBookmarks
        }
    });

    //show error
    function handleError(err){
        alert(err.message);
    }

    $("#bookmarksContainer").on("click", ".jr_bookmark", function(evt) {
        report.pages({
            anchor: $(this).data("anchor")
        }).run();
    });

    // handle bookmarks
    function handleBookmarks(bookmarks, container) {
        var li, ul = $("<ul/>");
        !container && $("#bookmarksContainer").empty();
        container = container || $("#bookmarksContainer");

        $.each(bookmarks, function(i, bookmark) {
            li = $("<li><span class='jr_bookmark' title='Anchor: " +
```

```
bookmark.anchor + ", page: " + bookmark.page + "' data-anchor='" +
bookmark.anchor + "' data-page='" + bookmark.page + "'>" + bookmark.anchor
+ "</span></li>");
            bookmark.bookmarks && handleBookmarks(bookmark.bookmarks, li);
            ul.append(li);
        });

        container.append(ul);
    }
});
```

Disabling the JIVE UI

The JIVE UI is enabled by default on all reports that support it. When the JIVE UI is disabled, the report is static and neither users nor your script can interact with the report elements. You can disable it in your visualize.report call as shown in the following example:

```
visualize({
    auth: { ...
    }
}, function (v) {
    v.report({
        resource: "/public/Samples/Reports/RevenueDetailReport",
        container: "#reportContainer",
        defaultJiveUi: { enabled: false },
        error: function (err) {
            alert(err.message);
        }
    });
});
```

The JIVE UI is enabled by default on all reports that support it. When the JIVE UI is disabled, the report is static and neither users nor your script can interact with the report elements. You can disable it in your jrioClient.report call as shown in the following example:

```
jrio.config({
    ...
});
jrio(function(jrioClient) {
    jrioClient.report({
        resource: "/samples/reports/TableReport",
        container: "#reportContainer",
        defaultJiveUi: { enabled: false },
        error: function (err) {
            alert(err.message);
        }
    });
});
```

Associated HTML:

```
<script src="http://bi.example.com:8080/jasperserver-
pro/client/visualize.js"></script>
<p>JIVE UI is disabled on this Visualize.js report:</p>
<div id="reportContainer">Loading...</div>
```

Associated HTML:

```
<script src="http://bi.example.com:8080/jriojsapi/client/jrio.js"></script>
<p>JIVE UI is disabled on this report:</p>
<div id="reportContainer">Loading...</div>
```

Jaspersoft Documentation and Support Services

For information about this product, you can read the documentation, contact Support, and join Jaspersoft Community.

How to Access Jaspersoft Documentation

Documentation for Jaspersoft products is available on the Product Documentation website, mainly in HTML and PDF formats.

The Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The documentation for this product is available on the JasperReports® IO Professional Edition Product Documentation page.

How to Access Related Third-Party Documentation

When working with JasperReports® IO Professional Edition, you may find it useful to read the documentation of the following third-party products:

How to Contact Support for Jaspersoft Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our product Support website.
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the product Support website. If you do not have a username, you can request one by clicking **Register** on the website.

How to Join Jaspersoft Community

Jaspersoft Community is the official channel for Jaspersoft customers, partners, and employee subject matter experts to share and access their collective experience. Jaspersoft Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from Jaspersoft products. In addition, users can submit and vote on feature requests from within the Jaspersoft Ideas Portal. For a free registration, go to Jaspersoft Community.

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

Jaspersoft, JasperReports, Visualize.js, and TIBCO are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (https://www.cloud.com/legal) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: https://scripts.sil.org/OFL

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at https://www.tibco.com/patents.