JASPERSOFT

JASPERREPORTS SERVER PROGRAMMING GUIDE

RELEASE 5.6

http://www.jaspersoft.com

Copyright © 2014 Jaspersoft Corporation. All rights reserved. Printed in the U.S.A. Jaspersoft, the Jaspersoft logo, Jaspersoft iReport Designer, JasperReports Library, JasperReports Server, Jaspersoft OLAP, Jaspersoft Studio, and Jaspersoft ETL are trademarks and/or registered trademarks of Jaspersoft Corporation in the United States and in jurisdictions throughout the world. All other company and product names are or may be trade names or trademarks of their respective owners.

This is version 0514-JSP56-01 of the *JasperReports Server Programming Guide*.

Table of Contents

Chapter 1 API Reference: Visualize.js

The JavaScript API exposed through Visualize.js allows you to embed and programmatically interact with reports dynamically. 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.

This chapter contains the following sections:

- **Requesting the Visualize.js Script**
- **Contents of the Visualize.js Script**
- **Usage Patterns**

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

- **API Reference: login and logout**
- **API Reference: resourcesSearch**
- **API Reference: report**
- **API Reference: inputControls**

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

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

1.1 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	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.
`_opt`	true|false	When true, the Javascript is optimized (reduced in size). By default, this parameter is false.

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>
```

1.2 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
 *  @returns {Deferred} dfd
 */
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
     * @returns {Deferred} dfd
     */
```

```
    login : function(auth){},

    /**
     * Destroy current auth session
     * @returns {Deferred} dfd
     */
    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.

1.3 Usage Patterns

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

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

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

If you prefer, you can use the deferred pattern instead of the callback:

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

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

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

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

Chapter 2 API Reference: Login and Logout

The initialization of the script sets the authentication method and credentials that you want to use for accessing JasperReports Server. You can then use the `login` and `logout` functions to manage multiple user sessions.

This chapter contains the following sections:

2.1 Authentication Properties

The properties argument to the visualize function has all the fields for specifying various authentication methods.

```
{
    "oneOf": [
        {
            "type": "object",
            "description": "Visualize main properties",
            "properties": {
                "server": {
                    "type": "string",
                    "description": "Url to JRS instance."
                },
                "auth": {
                    "description": "Authentication Properties",
                    "type": "object",
                    "oneOf": [
                        {
                            "properties": {
                                "token": {
                                    "type": "string",
                                    "description": "SSO authentication token"
                                },

                                "headers": {
                                    "type": "object",
                                    "description": "HTTP header parameters"
                                },
                                "queryParams": {
                                    "type": "object",
                                    "description": "HTTP query parameters"
                                }
                            },
                            "additionalProperties" : false,
                            "required": ["token"]
                        },
                        {
                            "properties": {
                                "name": {
                                    "type": "string",
                                    "description": "Name of the user to authenticate"
                                },
                                "password": {
                                    "type": "string",
                                    "description": "Password of the user to authenticate"
                                },
                                "organization": {
                                    "type": "string",
                                    "description": "Organization of the user to authenticate"
                                },
                                "timezone": {
                                    "type": "string",
                                    "description": "Default user timezone to use for this session"
                                },
                                "headers": {
                                    "type": "object",
                                    "description": "HTTP header parameters"
                                },
                                "queryParams": {
                                    "type": "object",
                                    "description": "HTTP query parameters"
                                }
                            },
                            "additionalProperties" : false,
                            "required": ["name", "password"]
                        }
                    ]
                }
            },
            "required": ["server", "auth"]
        },
        {
            "$ref": "#/definitions/func"
        }
    ],
    "definitions": {
        "func" : {
            "title": "Successful callback",
            "type": "object"
        }
    }
}
```

There are several ways to set the user credentials, based on your environment.

2.2 Login With Plain Text Credentials

Specify the username, password, organization if required, and optional parameters in the `auth`structure.

```
visualize({
    auth: {
        name: "jasperadmin",
        password: "jasperadmin",
        organization:"organization_1",
        timezone: "Europe/Helsinki"
    }
}, function (v) {
    ...
}, function () {
    alert("Unexpected error!");
});
```

2.3 Login With SSO Token

If you have a single-sign-on (SSO) provider implemented and have configured JasperReports Server to use it, you can specify the SSO token for Visualize.js login. This example shows a token from a Central Authentication Service (CAS) server.

```
visualize({
    auth : { token : "ST-40-CZeUUnGPxEqgScNbxh9l-sso-cas.prod.jaspersoft.com"}
}, function (v){

    alert("You are now logged into JasperReports Server with your SSO token.");
    ...

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

2.4 Logging Out

To log out and destroy the current user session, call the logout function and optionally specify any action to take when done.

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

    ...
    //destroy session
    $("#logout").click(function () {
        v.logout().done(function () {
            alert("You are now logged out of JasperReports Server.");
        });
    });
});
```

2.5 Login With Hooks

If you have external authentication providers, you can invoke their login and logout URLs.

```
visualize({
    auth: {
        name: "jasperadmin",
        password: "jasperadmin",
        loginFn: function (properties, request) {
            // Use a customLogin function to authenticate
            // It must be on the same domain: 'request' works only with JRS instance
            alert("Sending custom login request to 'http://bi.example.com/customLogin'");
            return request({
                url: "http://bi.example.com/customLogin?username=" + properties.name + "&password=" +
properties.password
            });
        },
        logoutFn: function (properties, request) {
            // Use a customLogout function to destroy the session
            // It must be on the same domain: 'request' works only with JRS instance
            alert("Sending custom logout request to 'http://bi.example.com/customLogout'");
            return request({
                url: "http://bi.example.com/customLogout"
            });
        }
    }
}, function (v) {
    ...
});
```

2.6 UI for Login/Logout

You can define IDs (`#name`) with listeners that perform login and logout functions. In your HTML, you can then assign these IDs to the appropriate buttons or links.

```
visualize(
    function(v){
        $("#selected_resource").change(function () {
            $("#container").html("");
            createReport($("#selected_resource").val(), v);
        });
        $("#login").click(function(){
            v.login(getAuthData()).done(function(){
                createReport($("#selected_resource").val(),v);
                showMessage(".success");
            }).fail(function(){showMessage(".error");});
        });
        $("#logout").click(function(){
            v.logout().done(function(){showMessage(".logout");});
        });
        $(':disabled').prop('disabled', false);
    }
);

//create and render report to specific container
function createReport(uri, v) {
    v("#container").report({
        resource: uri,
        error: function (err) {
            alert(err.message);
            }
     });
};

function showMessage(selector){
    $(".message").hide();
    $(selector).show();
};

function getAuthData(){
    return {name: $("#j_username").val(),
            password: $("#j_password").val(),
            organization:$("#orgId").val(),
            locale:$("#userLocale").val(),
            timezone:$("#userTimezone").val()
           }
};
```

2.7 UI for Login/Logout With SSO Token

The code is slightly different if you have a login/logout UI and use SSO tokens. Note that the logout uses the `.always` event instead of `.done`.

```
visualize(
    function(v){
        $("#selected_resource").change(function () {
            $("#container").html("");
            createReport($("#selected_resource").val(), v);
        });
        $("#login").click(function(){
            v.login(getAuthData()).done(function(){
                createReport($("#selected_resource").val(),v);
                showMessage(".success");
            }).fail(function(){showMessage(".error");});
        });
        $("#logout").click(function(){
            v.logout().always(function(){showMessage(".logout");});
        });
        $(':disabled').prop('disabled', false);
    }
);

//create and render report to specific container
function createReport(uri, v) {
    v("#container").report({
        resource: uri,
        error: function (err) {
            alert(err.message);
            }
     });
};

function showMessage(selector){
    $(".message").hide();
    $(selector).show();
};

function getAuthData(){
    return {token: $("#token").val()};
};
```

2.8 Sharing Credentials Between Calls

Use the `visualize.config` function to define and store authentication credentials. It uses the same `auth` structure as the `visualize` function. You can then create several containers with separate calls to `visualize`, but using the common credentials.

```
visualize.config({
    auth: {
        name: "jasperadmin",
        password: "jasperadmin",
        organization:"organization_1",
        timezone: "Europe/Helsinki"
    }
});

visualize(function (v) {
    v("#container1").report({
        resource: "/public/Samples/Reports/06g.ProfitDetailReport",
        error: function (err) {
            alert(err.message);
        }
    });
});

visualize(function (v) {
    v("#container2").report({
        resource: "/public/Samples/Reports/State_Performance",
        error: function (err) {
            alert(err.message);
        }
    });
});
```

Chapter 3 API Reference: resourcesSearch

The `resourcesSearch` function performs searches in the repository to find content that can be displayed by visualize.js.

This chapter contains the following sections:

3.1 Search Properties

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

```
{
    "type": "object",
    "properties": {
        "server": {
            "type": "string",
            "description": "Url to JRS instance."
        },
        "q": {
            "type": "string",
            "description": "Query string. Search for occurrence in label or description of resource."
        },
        "folderUri": {
            "type": "string",
            "description": "Parent folder URI.",
            "pattern": "^/\\w*(/\\w+)*$"
        },
        "type": {
            "type": "string",
            "description": "Type of resources to search.",
            "enum": [
                "folder", "dataType", "jdbcDataSource", "awsDataSource", "jndiJdbcDataSource",
                "virtualDataSource", "customDataSource", "beanDataSource", "xmlaConnection",
                "listOfValues", "file", "reportOptions", "dashboard", "adhocDataView",
                "query", "olapUnit", "reportUnit", "domainTopic", "semanticLayerDataSource",
                "secureMondrianConnection", "mondrianXmlaDefinition", "mondrianConnection",
                "inputControl"
            ]
        },
        "offset": {
            "type": "integer",
            "description": "Pagination. Index of first resource to show.",
            "minimum": 0
        },
        "limit": {
            "type": "integer",
            "description": "Pagination. Resources count per page.",
            "minimum": 0
        },
        "recursive": {
            "type": "boolean",
            "description": "Flag indicates if search should be recursive."
        },
        "sortBy": {
            "type": "string",
            "description": "Field to sort on.",
            "enum": [
                "uri",
                "label",
                "description",
                "type",
                "creationDate",
                "updateDate",
                "accessTime",
                "popularity"
            ]
        },
        "accessType": {
            "type": "string",
            "description": "Filtering by type of access, e.g. what was done with resource.",
            "enum": [
                "viewed",
                "modified"
            ]
        },
        "showHiddenItems": {
            "type": "boolean",
            "description": "Flag indicates if hidden items should present in results."
        },
        "forceTotalCount": {
            "type": "boolean",
            "description": "If true, Total-Count header is always set (impact on performance),
                            otherwise - in first page only"
        }
    },
    "required": ["server"]
}
```

3.2 Search Functions

The `resourcesSearch` function exposes the following functions:

```
define(function () {

    /**
     * Constructor. Takes context as argument.
     * @param contextObj - map of properties.
     */
    function ResourcesSearch(contextObj){};

    /**
     * Get/Set 'q' parameter of the query
     * @param contextObj - map of properties.
     * @returns this if 'value' sent to the method,
     *          otherwise returns current value of the parameter
     */
    ResourcesSearch.prototype.q= function(value){};

    /**
     * Get/Set 'folderUri' parameter of the query
     * @param value - new value, optional
     * @returns this if 'value' sent to the method,
     *          otherwise returns current value of the parameter
     */
    ResourcesSearch.prototype.folderUri= function(value){};

    /**
     * Get/Set 'type' parameter of the query
     * @param value - new value, optional
     * @returns this if 'value' sent to the method,
     *          otherwise returns current value of the parameter
     */
    ResourcesSearch.prototype.type= function(value){};

    /**
     * Get/Set 'offset' parameter of the query
     * @param value - new value, optional
     * @returns this if 'value' sent to the method,
     *          otherwise returns current value of the parameter
     */
    ResourcesSearch.prototype.offset= function(value){};

    /**
     * Get/Set 'limit' parameter of the query
     * @param value - new value, optional
     * @returns this if 'value' sent to the method,
     *          otherwise returns current value of the parameter
     */
    ResourcesSearch.prototype.limit= function(value){};


    /**
     * Get/Set 'recursive' parameter of the query
     * @param value - new value, optional
     * @returns this if 'value' sent to the method,
     *          otherwise returns current value of the parameter
     */
    ResourcesSearch.prototype.recursive= function(value){};

    /**
     * Get/Set 'sortBy' parameter of the query
     * @param value - new value, optional
     * @returns this if 'value' sent to the method,
     *          otherwise returns current value of the parameter
     */
    ResourcesSearch.prototype.sortBy= function(value){};

    /**
     * Get/Set 'accessType' parameter of the query
     * @param value - new value, optional
     * @returns this if 'value' sent to the method,
     *          otherwise returns current value of the parameter
     */
    ResourcesSearch.prototype.accessType= function(value){};

    /**
     * Get/Set 'showHiddenItems' parameter of the query
     * @param value - new value, optional
     * @returns this if 'value' sent to the method,
     *          otherwise returns current value of the parameter
     */
    ResourcesSearch.prototype.showHiddenItems= function(value){};

    /**
     * Get/Set 'forceTotalCount' parameter of the query
     * @param value - new value, optional
     * @returns this if 'value' sent to the method,
     *          otherwise returns current value of the parameter
     */
    ResourcesSearch.prototype.forceTotalCount= function(value){};

    return ResourcesSearch;
});
```

3.3 Finding Resources

The following code examples show two different ways of handling results after making a simple repository search in the Public folder.

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

```
var search = v.resourcesSearch({
    folderUri: "/public",
    recursive: false
    success: function(repo) {
        console.log(repo.data()); // resourceLookups
    }
});
```

3.4 Reusing a Search Instance

If you make multiple searches, for example in different folders, you can create a function to do that using the `ResourcesSearch` function.

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

// call 1
folderContentQuery.folderUri("/uri1").run(doSomethingWithResultFunction);
...
// call 2 after some time
folderContentQuery.folderUri("/uri2").run(doSomethingWithResultFunction);
```

3.5 Reusing Search Results

Code example:

```
var call = new ResourcesSearch({
    server:"http://localhost:8080/jasperserver-pro",
    folderUri: "/public",
    recursive: false
})).run(function(resourceLookups){
    // data() available here
});
// at this point call.data() will return null until the run callback is called.
call.data() === null // -> true

.....
// if some data was obtained earlier, it accessible via data()
var resourceLookups = call.data();
```

3.6 Discovering Available Types

You can write code to discover and display the types that can be searched and types of sorting that can be specified.

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

    buildControl("Resources types", v.resourcesSearch.types);
    buildControl("Resources search sort types", v.resourcesSearch.sortBy);

});

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));
}
```

Chapter 4 API Reference: report

The `report` function runs reports on JasperReports Server and displays the result in a container that you provide.

This chapter contains the following sections:

4.1 Report Properties

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

```
{
    "title": "Report Properties",
    "description": "A JSON Schema describing a Report Properties",
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
        "server": {
            "type": "string",
            "description": "URL of JRS instance."
        },
        "resource": {
            "type": "string",
            "description": "Report resource URI.",
            "pattern": "^/[^/~!#\\$%^|\\s`@&*()\\+={}\\[\\]:;\"'<>,?/\\|\\\\]+(/[^/~!#\\$%^|\\s`@&*()
\\+={}\\[\\]:;\"'<>,?/\\|\\\\]+)*$"
        },

        "container": {
            "type": "string",
            "description": "CSS selector for container to render report to."
        },
        "params": {
            "$ref": "#/definitions/params"
        },
        "pages": {
            "type": ["string", "integer"],
            "description": "Range of report's pages or single report page",
            "pattern": "^\\d+(\\-\\d+)?$",
            "default": 1
        },
        "linkOptions": {
            "type": "object",
            "description": "Customization for report's links",
            "properties": {
                "beforeRender": {
                    "$ref": "#/definitions/Function",
                    "description": "Allows to change any link representation before report would be
                                    rendered"
                },
                "events": {
                    "$ref": "#/definitions/eventsMapper",
                    "description": "Allow to add listener by specific events to links representations
                                    by link name"
                }
            }
        },
        "defaultJiveUi" : {
            "type": "object",
            "description": "Control default JIVE UI in report",
            "properties": {
                "enabled": {
                    "type": "boolean",
                    "default": false
                },
                "onError":{
                    "$ref": "#/definitions/Function",
                    "description": "Jive UI error listener"
                }
            }
        },
        "isolateDom": {
           "type": "boolean",
           "description": "Isolate report's DOM from third-party page CSS. Can't be set while default
                           JIVE UI is enabled",
           "default": false
        }
    },
    "required": ["server", "resource"],
    "definitions": {
        "params": {
            "type": "object",
            "description": "Report's parameters values",
            "additionalProperties": {
                "type": "array"
            }
        },

        "ExportOptions": {
            "title": "Report export options",
            "type": "object",
            "properties": {
                "outputFormat": {
                    "enum": [
                        "pdf",
                        "xlsx",
                        "xls",
                        "rtf",
                        "csv",
                        "xml",
                        "odt",
                        "ods",
                        "docx"
                    ]
                },
                "pages": {
                    "type": ["string", "integer"],
                    "description": "Exports all pages if this property was not specified. Range of
                                    report's pages or single report page",
                    "pattern": "^\\d+(\\-\\d+)?$"
                },
                "paginated": {
                    "type": "boolean",
                    "description": "Control 'pagination' feature. Only 'xls' and 'xlsx' support it",
                    "default": "false"
                }
            },
            "required": ["outputFormat"]
        },
        "Function": {
            "type": "object",
            "description": "JavaScript Function"
        },
        "eventsMapper": {
            "type": "object",
            "description": "Map events by name to user defined handler. For example: 'click',
                            'focus', etc ",
            "additionalProperties": {
                "$ref": "#/definitions/Function"
            }
        },
        "chartComponent": {
            "description": "JIVE chart component schema",
            "properties": {
                "id": {
                    "type": "string",
                    "description": "Chart component identifier"
                },
                "componentType": {
                    "enum": ["chart"]
                },
                "chartType": {
                    "description": "Special chart's type",
                    "enum": [
                        "Bar", "Column", "Line", "Area", "Spline",
                        "AreaSpline", "StackedBar", "StackedColumn", "StackedLine", "StackedArea",
                        "StackedSpline", "StackedAreaSpline", "StackedPercentBar",
                        "StackedPercentColumn", "StackedPercentLine", "StackedPercentArea",
                        "StackedPercentSpline", "StackedPercentAreaSpline", "Pie", "DualLevelPie",
                        "TimeSeriesLine", "TimeSeriesArea", "TimeSeriesSpline",
                        "TimeSeriesAreaSpline", "ColumnLine", "ColumnSpline", "StackedColumnLine",
                        "StackedColumnSpline", "MultiAxisLine", "MultiAxisSpline", "MultiAxisColumn",
                        "Scatter", "Bubble", "SpiderColumn", "SpiderLine", "SpiderArea"
                    ]
                }
            },
            "required": ["id"]
        }
    }
}
```

4.2 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
     */

    //Special  getters

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

    // Special setters

    /**
     * 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
 * @return {Deferred} dfd
 */
Report.prototype.run = function(callback, errorback, 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){};

/**
 * 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){};


/**
 * Update report's component
 * @param {Object} component - jive component to update, should have id field
 * @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.updateComponent = function(component, callback, errorback, always){};

/**
 * Update report's component
 * @param {String} id - jive component id
 * @param {Object} properties - jive component's properties to 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.updateComponent = function(id, properties, 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.prototype.undoAll = function(callback, errorback, always){};

     /**
     * Redo next 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.redo = function(callback, errorback, always){};

    /**
     * Export report to specific format, execute only after report run action is finished
     * @param {ExportOptions} exportOptions - export options
     * @param {Function} callback - optional, invoked with link object
     * @param {Function} errorback - optional, invoked in case of failed export
     * @param {Function} always - optional, invoked optional, invoked always
     * @return{Deferred} dfd
     */
     Report.prototype.export = function(exportOptions, callback, errorback, always){};

    /**
     * Cancel all execution, destroy report representation if any, leave only
     * properties
     * @param {Function} callback - optional, invoked in case of successful cleanup
     * @param {Function} errorback - optional, invoked in case of failed cleanup
     * @param {Function} always - optional, invoked optional, invoked always
     * @return {Deferred} dfd
     */
    Report.prototype.destroy = function(callback, errorback, always){};

    return Report;
});
```

4.3 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, the Report Data structure contains information about the links in the report, as explained in **"Customizing Links" on**

page 55, and components of the JIVE UI, as explained in **“Interacting With JIVE UI Components” on page 59**.

```
{
    "title": "Report Data",
    "description": "A JSON Schema describing a Report Data",
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
        "totalPages": {
            "type": "number",
            "description": "Report's page total count"
        },
        "links": {
            "type": "array",
            "description": "Links extracted from markup, so their quantity depends from pages you have
requested",
            "items": {
                "$ref": "#/definitions/jrLink"
            }
        },
        "components": {
            "type": "array",
            "description": "Components in report, their quantity depends from pages you have
requested",
            "items": {
                "type": "object",
                "description": "JIVE components data"
            }
        }
    },
    "definitions": {
        "jrLink": {
            "title": "JR Hyperlink",
            "type": "object",
            "properties": {
                "type": {
                    "oneOf": [
                        {
                            "$ref": "#/definitions/linkTypeReference"
                        },
                        {
                            "$ref": "#/definitions/linkTypeReportExecution"
                        }
                    ]
                },
                "tooltip": {
                    "type": "string",
                    "description": "Hyperlink tooltip"
                },
                "href": {
                    "type": "string",
                    "description": "Hyperlink reference"
                },
                "parameters": {
                    "type": "object"
                }
            },
            "required": ["type"],
            "definitions": {
                "linkTypeReference": {
                    "enum": ["Reference"],
                    "description": "The  hyperlink points to an external resource."
                },
                "linkTypeReportExecution": {
                    "enum": ["ReportExecution"],
                    "description": "The hyperlink points to JR report"
                }
            }
        }
    }
}
```

4.4 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"
});
```

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 Dem-
ographic</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>
```

4.5 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"] }
```

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" on page 48**.

The example above is trivial, but the power of Visualize.js 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 become truly embedded and much more relevant to the user.

4.6 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);
            }
        });
    });
});
```

Associated HTML:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.js"></script>
<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 CSS:

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

4.7 Setting Report Pagination

To set or change the pages of the report that are displayed, 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 in a string is also possible
        .run();
```

4.8 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); });
    });
});
```

Associated HTML:

```
<script src="http://code.jquery.com/jquery-2.1.0.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>
```

4.9 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); });
    });
});
```

Associated HTML:

```
<script src="http://code.jquery.com/jquery-2.1.0.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>
```

4.10 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", "xls", "rtf", "csv", "xml", "odt", "ods", "docx"
```

```
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: "xls",
            pages: "1-10",
            paginated: true
        })
        .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]);
    }

});
```

Associated HTML:

```
<script src="http://code.jquery.com/jquery-2.1.0.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>
```

4.11 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="http://code.jquery.com/jquery-2.1.0.js"></script>
<script src="http://bi.example.com:8080/jasperserver-pro/client/visualize.js"></script>
<button>Refresh</button>
<div id="container1"></div>
```

4.12 Canceling Report Execution

To stop a report that is executing, 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="http://code.jquery.com/jquery-2.1.0.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>
```

4.13 Discovering Available Charts and Formats

You can write code to discover and display the types of charts and export formats that can be specified.

```
visualize({
    auth: {
        name: "jasperadmin",
        password: "jasperadmin",
        organization: "organization_1"
    }
}, function (v) {
    buildControl("Chart types", v.report.chart.types);
    buildControl("Report export formats", v.report.exportFormats);
    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));
}
```

CHAPTER 5 API REFERENCE: INPUTCONTROLS

The `inputControls` function prepares and displays input controls for reports that your users interact with.

This chapter contains the following sections:

5.1 Input Control Properties

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

```
{
    "type": "object",
    "properties": {
        "server": {
            "type": "string",
            "description": "Url to JRS instance."
        },
        "resource": {
            "type": "string",
            "description": "URI of resource with input controls.",
            "pattern": "^/\\w*(/\\w+)*$"
        },
        "params": {
            "type": "object",
            "description": "Parameters for input controls.",
            "additionalProperties": {
                "type": "array"
            }
        }
    },
    "required": ["server", "resource"]
}
```

5.2 Input Control Functions

The `InputControls` function exposes the following functions:

```
define(function () {

    /**
     * Constructor. Takes properties as argument.
     * @param properties - map of properties.
     */
    function InputControls(properties){};

    /**
     * Get/Set 'resource' property - URI of resource with input controls.
     * @param value - new value, optional
     * @returns this if 'value' sent to the method,
     *          otherwise returns current value of the parameter
     */
    InputControls.prototype.resource = function(value){};

    /**
     * Get/Set 'params' property - Parameters for input controls.
     * @param value - new value, optional
     * @returns this if 'value' sent to the method,
     *          otherwise returns current value of the parameter
     */
    InputControls.prototype.params = function(value){};

    return InputControls;
});
```

5.3 Input Control Structure

InputControls data() is an array of InputControl objects, with the structure shown in this example:

```
[
  {
    "id":"Cascading_name_single_select",
    "label":"Cascading name single select",
    "mandatory":"true",
    "readOnly":"false",
    "type":"singleSelect",
    "uri":"repo:/reports/samples/Cascading_multi_select_report_files/Cascading_name_single_select",
    "visible":"true",
    "masterDependencies": {
      "controlId": [
        "Country_multi_select",
        "Cascading_state_multi_select"
      ]
    },
    "slaveDependencies":null,
    "validationRules": [
      {
        "mandatoryValidationRule" : {
          "errorMessage" : "This field is mandatory so you must enter data."
        }
      }
    ],

    "state": {
      "uri": "/reports/samples/Cascading_multi_select_report_files/Cascading_name_single_select",
      "id": "Cascading_name_single_select",
      "value": null,
      "options": [
        {
          "selected": false,
          "label": "A & U Jaramillo Telecommunications, Inc",
          "value": "A & U Jaramillo Telecommunications, Inc"
        }
      ]
    }
  },
  ....
]
```

5.4 Fetching Input Control Data

The data being output here has the input control structure shown in the previous section:

```
visualize(function(v){
    var ic = v.inputControls({
        resource: "/public/ReportWithControls",
        success: function(data) {
           console.log(data);
        }
    });
});
```

This example shows an alternate way of fetching input controls:

```
(new InputControls({
    server: "http://localhost:8080/jasperserver-pro",
    resource: "/public/my_report",
    params: {
        "Country_multi_select":["Mexico"],
        "Cascading_state_multi_select":["Guerrero", "Sinaloa"]
    }
})).run(function(inputControlsArray){
    // results here
})
```

5.5 Creating Input Control Widgets

This example retrieves the input controls of a report and parses the structure to create drop-down option menus of values for each control:

```
visualize({
    auth: {
        name: "superuser",
        password: "superuser"
    }
},function(v) {

    v.inputControls({
        resource: "/public/Samples/Reports/16g.InteractiveSalesReport",
        success: function (controls) {
            controls.forEach(buildControl);
        },
        error: function (err) {
            alert(err);
        }
    });

    function buildControl(control) {

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

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

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

});
```

5.6 Cascading Input Controls

In order to implement cascading input controls, you must implement a change listener on the parent control and use it to trigger an update on the dependent control:

```
var reportUri = "/public/Samples/Reports/Cascading_Report_2_Updated";

visualize({
    auth: {
        name: "superuser",
        password: "superuser"
    }
}, function (v) {
    var inputControls = v.inputControls({
        resource: reportUri,
        success: renderInputControls
    });

    var report = v.report({ resource: reportUri, container: "#container" });

    $("#productFamilySelector").on("change", function() {
        report.params({ "Product_Family": [$(this).val()] }).run();
    });
});

function renderInputControls(data) {
    var productFamilyInputControl = _.findWhere(data, {id: "Product_Family"});
    var select = $("#productFamilySelector");
    _.each(productFamilyInputControl.state.options, function(option) {
        select.append("<option " + (option.selected ? "selected" : "") + " value='" +
                        option.value + "'>" + option.label + "</option>");
    });
}
```

Associated HTML:

```
<script src="http://code.jquery.com/jquery-2.1.0.js"></script>
<script src="http://underscorejs.org/underscore-min.js"></script>
<script src="http://bi.example.com:8080/jasperserver-pro/client/visualize.js"></script>
<select id="productFamilySelector"></select>
<div id="container"></div>
```

5.7 Reusing Input Control Instances

Input controls are meant to be dynamic and modified by users. By using the inputControls.params function, you can update the values of input controls and then update the corresponding report.

```
var inputControls = new InputControls({
    server: "http://localhost:8080/jasperserver-pro",
    resource: "/public/my_report"
});

// call 1
inputControls.params({ "Country_multi_select": ["Mexico"] }).run(doSomethingWithResultFunction);
...
// call 2 after some time
inputControls.params({ "Country_multi_select": ["USA"] }).run(doSomethingWithResultFunction);
```

5.8 Reusing Input Control Data

It is possible to store the data from the `inputControls` function and access the data() structure at a later time:

```
var call = (new InputControls({
    server: "http://localhost:8080/jasperserver-pro",
    resource: "/public/my_report"
})).run(function(inputControlsArray){
    // data() available here
});

// at this point call.data() will return null until the run callback is called.
call.data() === null // -> true
...
// if some data was obtained earlier, it accessible via data()
var inputControlsArray = call.data();
```

Chapter 6 API Reference: Errors

This chapter lists the errors that commonly occur and describes how to handle them with Visualize.js.

This chapter contains the following sections:

6.1 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"]
}
```

6.2 Common Errors

The following table lists common errors, their message, and their cause.

Error	***Message* - Description**
Page or app not responding	*{no_message}* - If your page or web application has stopped working without any notifications or errors, then check that the server that provides visualize.js 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 not found in the DOM:error.
`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 licence* - The server's license was not found.
`licence.expired`	*JRS license expired* - The server's license has expired
`resource.not.found`	*Resource not found in Repository* - Either the resource does not 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 which do not exist in the current export.
`input.controls.validation.error`	*{server_error_message}* - The wrong input control params were sent to the server.

6.3 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" on page 6**, is:

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

During intitialization 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

})
```

6.4 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 run every time when operation ends.

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

    alert(resultOrError);

}))
```

6.5 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
}
```

6.6 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
        });

)
```

6.7 Catching Input Control Errors

Catching and handling input control errors is very similar to reports. 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
        });

)
```

6.8 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
}
```

Chapter 7 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 let you display pages sooner.

This chapter contains the following sections:

7.1 Tracking Completion Status

By listening to 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);
        },
    });
});
```

7.2 Listening to Page Totals

By listening to 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);
            }
        }
    });
});
```

7.3 Listening for the Last Page

By listening to the `pageFinal` event, you can 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);
        },
    });
});
```

7.4 Customizing a Report's DOM Before Rendering

By listening to 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());
                }
            }
        });
    };
});
```

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” on page 28**, except that the JavaScript above will change the report before it is displayed.

```
<script src="http://code.jquery.com/jquery-2.1.0.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 Dem-
ographic</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>
```

CHAPTER 8 API USAGE: REPORT HYPERLINKS

Many reports include hyperlinks (URLs) that link to websites or other reports. The `report` function gives you access to the links generated in the report, so that you can customize both the appearance and the container where they are displayed.

This chapter contains the following sections:

- **Customizing Links**
- **Drill-Down in Separate Containers**
- **Accessing Data In Links**

8.1 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 that has access to the links in the document as element pairs.
- The normal click event lets your add a listener that has access to a link when it is 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);
        }
    });
});
```

8.2 Drill-Down in Separate Containers

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.

```
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;
               el.style.cursor = "pointer";
    }

});
```

Associated HTML:

```
<script src="http://underscorejs.org/underscore.js"></script>
<script src="http://code.jquery.com/jquery-2.1.0.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 CSS:

```
#main{
   float: left;
}

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

8.3 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);
    }

});
```

Associated HTML:

```
<script src="http://code.jquery.com/jquery-2.1.0.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 CSS:

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

Chapter 9 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.

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.

This chapter contains the following sections:

9.1 Interacting With JIVE UI Components

The `visualize.report` interface exposes the `updateComponent` function that gives your script access to the JIVE UI. Using the structures exposed by `updateComponent`, 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` interface, see **"Report Functions" on page 24**.

First enable the default JIVE UI, then the components of the JIVE UI are available after running a report:

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

...

var components = report.data().components;
```

Each component of the JIVE UI has an id, but it may change from execution to execution. To refer to components of the UI, create your report in JRXML and use the `net.sf.jasperreports.components.name` property to name each component you want to reference, such as a column in a table. Then you can reference the object by this name, for example `"sales"`, and use the `updateComponent` function.

```
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 event modify the report through the JIVE UI:

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

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

    $("#resetAll").on("click", function(){
        report.undoAll();
    });
    $("#changeFemale").on("click", function () {
        //component's name generated by default from field name
        report.updateComponent("femalesales", {
            sort: {
                order: "asc"
            },
            filter: {
                operator: "greater_or_equal",
                value: 15000
            }
        }).fail(handleError);
    });

    $("#changeDep").on("click", function () {
        //custom compoent's name
        report.updateComponent("my_dep", {
            sort: {
                order: "desc"
            }
        }).fail(handleError);
    });

    $("#changeChart").on("click", function () {
        //custom compoent's name
        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 Lable: "+ 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="changeFemale">Filter And Sort Female</button>
<button id="changeDep">Sort Departments</button>
<button id="changeChart">Show in Pie</button>
<!-- Provide a container for the report -->
<div id="container"></div>
```

9.2 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 programmatically, according to other states in your client.

This code also relies on the `report.chart.types` interface described in **"Discovering Available Charts and Formats" on page 36**.

```
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;
    }

});
```

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

```
<script src="http://code.jquery.com/jquery-2.1.0.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>
```

9.3 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;
}
```

Associated HTML:

```
<script src="http://code.jquery.com/jquery-2.1.0.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>
```

9.4 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);
    }
});
```

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>

<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>
```

9.5 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);
}
```

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>
```

9.6 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" on page 5**.

```
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", "checked");
                    } else {
                        $("#detailsRowFormatFontItalic").removeAttr("checked");
                    }
                    if (column4.detailsRowFormat.font.underline) {
                        $("#detailsRowFormatFontUnderline").attr("checked", "checked");
                    } else {
                        $("#detailsRowFormatFontUnderline").removeAttr("checked");
                    }

                    $("#detailsRowFormatPatternNegativeFormat").val(
                        column4.detailsRowFormat.pattern.negativeFormat);
                    $("#detailsRowFormatPatternPrecision").val(
                        column4.detailsRowFormat.pattern.precision);
                    $("#detailsRowFormatPatternCurrency").val(
                        column4.detailsRowFormat.pattern.currency || "");

                    if (column4.detailsRowFormat.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"),
```

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>
                    <option value="USD">USD</option>
                    <option value="EUR">EUR</option>
                    <option value="GBP">GBP</option>
                    <option value="YEN">YEN</option>
                    <option value="LOCALE_SPECIFIC">LOCALE_SPECIFIC</option>
            </select>
    <br/>
    Thousands grouping: <input type="checkbox" id="detailsRowFormatPatternGrouping" value="true"/>
    <br/>
    Percentage: <input type="checkbox" id="detailsRowFormatPatternPercentage" value="true"/>
    <br/><br/>
    <button id="changeDetailsRowFormat">Change details row format</button>
</div>
<div style="float: left;">
    <h3>Change label of 5th column</h3>
    <br/>
    Label <input type="text" id="label"/>
    <br/>
    <button id="changeLabel">Change label</button>
</div>
<div style="clear: both;"></div>

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

9.7 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);
    }
});
```

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>
```

9.8 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” on page 65**.

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

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>
```

9.9 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);
        });
    });
});
```

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>
```

9.10 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);
        }
    });
});
```

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>
```

CHAPTER 10 VISUALIZE.JS TOOLS

This chapter provides two extended code examples that you can use to test various parts of your own Visualize.js scripts.

This chapter contains the following sections:

10.1 Checking the Scope in Visualize.js

This example reads and displays the properties in the scope after `visualize.report` finishes rendering a report (success).

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

    createReport();

    $("#selected_resource").change(function () {
        //clean container
        $("#container").html("");
        createReport();
    });

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

     function createReport(){
        //render report from another resource
        v("#container").report({
            resource: $("#selected_resource").val(),
            success: function(){
                 setTimeout(function () {
                    console.log("------------------Scope Check Results---------------------------");
                    console.log(scopeChecker.compareProperties(propertiesNames));
                    console.log("----------------------------------------------------------------");
                }, 5000);
            },
            error:handleError

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

The ScopeChecker is another JavaScript used in this example. It can either be a separate .js file or included in your HTML file as shown in this example:

```
<!-- JavaScript for ScopeChecker -->
<script>
    function ScopeChecker(scope) {
        this.scope = scope;
    }

    ScopeChecker.prototype.getPropertiesCount = function() {
        return this.getPropertiesNames().length;
    };

    ScopeChecker.prototype.getPropertiesNames = function() {
        return Object.keys(this.scope);
    };

    ScopeChecker.prototype.compareProperties = function(scope1PropertiesNames, scope2PropertiesNames)
{
        if (!scope1PropertiesNames) {
            throw "Properties for scope 1 not specified";
        }
        if (!scope2PropertiesNames) {
            scope2PropertiesNames = this.getPropertiesNames();
        }

        var comparisonResult = {
            added: [],
            removed: [],
            madeUndefined: [],
            pollution: []
        };

        var i, j;
        for (i = 0; i < scope1PropertiesNames.length; i++) {
            comparisonResult.removed.push(scope1PropertiesNames[i]);
            for (j = 0; j < scope2PropertiesNames.length; j++) {
                if (scope1PropertiesNames[i] === scope2PropertiesNames[j]) {
                    comparisonResult.removed.pop();
                    break;
        }   }   }

        for (i = 0; i < scope2PropertiesNames.length; i++) {
            comparisonResult.added.push(scope2PropertiesNames[i]);
            for (j = 0; j < scope1PropertiesNames.length; j++) {
                if (scope2PropertiesNames[i] === scope1PropertiesNames[j]) {
                    comparisonResult.added.pop();
                    break;
        }   }   }

        for (i = 0; i < comparisonResult.added.length; i++) {
            if (this.scope[comparisonResult.added[i]] === undefined) {
                comparisonResult.madeUndefined.push(comparisonResult.added[i]);
            } else {
                comparisonResult.pollution.push(comparisonResult.added[i]);
        }   }

        return comparisonResult;
    };


    var propertiesNames = [];
    var scopeChecker = new ScopeChecker(window);
    propertiesNames = scopeChecker.getPropertiesNames();
</script>
<!-- Provide the URL to visualize.js -->
<script src="http://bi.example.com:8080/jasperserver-pro/client/visualize.js?_opt=true"></script>
<select id="selected_resource" 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>
```

10.2 CSS Diagnostic Tool

The CSS diagnostic tool lets you load various CSS libraries and see how they interact or interfere with the CSS that Visualize.js uses to render reports. It has lets you choose your JasperReports Server from a static list, so that you may try different themes on different servers. After you load a report using visualize.report, you can choose to load a variety of popular CSS libraries and see if they affect your report. The list of reports to choose from is in the HTML code below.

The key feature of this tool is the ability to set the `isolateDOM`property on the visualize.report function call. This property modifies the CSS of the report so that it does not conflict with other CSS libraries. The downside is that you cannot use the `defaultJiveUi` property in conjunction with `isolateDOM`, and the tool enforces this by clearing the former if you select the latter.

Save the Javascript, HTML, and CSS for the CSS Diagnostic Tool to your environment, and edit the files to use your server instances, your reports, and your Visualize.js code.

```
// ************ SETTINGS **********
var serverUrls = [
    "http://test.example.com:8080/jasperserver-pro",
    "http://cust.example.com:8080/jasperserver-pro",
    "http://localhost:8080/jasperserver-pro",
    "http://bi.example.com:8080/jasperserver-pro"
];
urlUsed = 3; // default used (one css loads from this server before visualize)
var reportsList = {};
// ********************************
function setupLoaderV() {
    var html,
        radioTpl = '<input id="#id" type="radio" value="#value" name="confServer" /><label for="#id"
title="#title" >#label</label><br/>';
    for (var i = 0, l = serverUrls.length; i < l; i++) {
        html = radioTpl.replace(/#id/g, "serverUrl_" + i)
            .replace(/#value/g, i)
            .replace(/#title/g, serverUrls[i])
            .replace("#label", serverUrls[i].split("/")[2]);

        $("#serverUrlsDiv").append(html);
        $("#serverUrl_" + i).prop("checked", i === urlUsed);
    }
}
function onLoad() {
    setupLoaderV();
    $( "#buttons_ui button:first" ).button({
      icons: {
        primary: "ui-icon-locked"
      },
      text: false
    }).next().button({
      icons: {
        primary: "ui-icon-locked"
      }
    }).next().button({
      icons: {
        primary: "ui-icon-gear",
        secondary: "ui-icon-triangle-1-s"
      }
    }).next().button({
      icons: {
        primary: "ui-icon-gear",
        secondary: "ui-icon-triangle-1-s"
      },
      text: false
    });

    var availableTags = [
      "ActionScript",
      "AppleScript",
      "Asp",
      "BASIC",
      "C",
      "C++",
      "Clojure",
      "COBOL",
      "ColdFusion",
      "Erlang",
      "Fortran",
      "Groovy",
      "Haskell",
      "Java",
      "JavaScript",
      "Lisp",
      "Perl",
      "PHP",
      "Python",
      "Ruby",
      "Scala",
      "Scheme"
    ];

    $( "#tags" ).autocomplete({
      source: availableTags
    });
    $("#datepicker-user").datepicker();
    $("#datepicker-user2").datepicker();
    $( "#tabs" ).tabs();

    $("#loadV").click(loadV);
    fillSheetList();
    loadCSS();
    $(window).on("keypress", function(e){
        var char = e.charCode - 49;
        if (char < 1 && char > 9) return;
        var input = $("#sheetList > li > input")[char];
        if (!input) return;
        $(input).trigger( "click" )
    });
    $("#isolateDOM").change(function() {
        $("#defaultJiveUi").attr({
            "disabled": $(this).is(':checked') ? "disabled" : null,
            "checked": false
        });
    });
}

function loadCSS() {

    var CSSlibs = [
        { disable: true, href: serverUrls[urlUsed] + "/themes/reset.css" },
        { disable: true, href: "//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css" },
        { disable: true, href: "//cdnjs.cloudflare.com/ajax/libs/normalize/3.0.1/normalize.min.css" },
        { disable: true, href: "//cdnjs.cloudflare.com/ajax/libs/meyer-reset/2.0/reset.css" },
        { disable: true, href: "http://yui.yahooapis.com/3.16.0/build/cssreset/cssreset-min.css" },
        { disable: true, href: "http://tantek.com/log/2004/undohtml.css" },
        { disable: true, href: "//cdnjs.cloudflare.com/ajax/libs/sanitize.css/2.0.0/sanitize.css" },
        { disable: false, href: "http://ajax.-
googleapis.com/ajax/libs/jqueryui/1.10.4/themes/vader/jquery-ui.css" }
    ];
    var head = $("head"),
        link;

    for (var i = 0, l = CSSlibs.length; i < l; i++) {
        link = $("<link rel='stylesheet' type='text/css' href='" + CSSlibs[i].href + "' />");
        head.append(link);
        if (CSSlibs[i].disable) {
            link.on("load", (function(link){
                return function() {
                    $(link)[0].disabled = true;
                    fillSheetList();
                }
            })(link));
        }
        link.on("error", (function(link){
            return function() {
                $(link)[0].href += "-LOAD_ERROR"
                fillSheetList();
            }
        })(link));
    }
}

function loadV() {
    var locale = $('#confLocale').val() || "en";
    urlUsed = $('input[name="confServer"]:checked', '#containerLoadV').val();
    var useOptimize = $('#confOptimized').is(":checked");
    $.getScript(serverUrls[urlUsed] + "/client/visualize.js?_opt=" + useOptimize.toString(), function
() {
        visualize({
            auth: {
                name: "superuser",
                password: "superuser",
                locale: locale
            }
        }, function (v) {
            fillSheetList();
            $("#loadV").remove();
            $("#loadReports").show();

            $("#addReport").on("click", function(){
                var uri = $(),
                    defaultJiveUi,
                    isolateDOM
                createReport(
                    v,
                    $("#selected_resource").val(),
                    $("#defaultJiveUi").is(':checked'),
                    $("#isolateDOM").is(':checked')
                );
            });
        });
    });
    $("#loadV").html("Loading...").attr("disabled", "disabled");
       $("#containerLoadV").addClass("disabled").children("input").attr("disabled", "disabled");
}

function createReport(v, uri, defaultJiveUi, isolateDOM) {
    var reportIndex = (+new Date() + "").substr(-5);
    console.log(reportIndex)
    reportsList[reportIndex] = "";
    fillReportsList();
    $("#reportContainer").append("<div id='vis_" + reportIndex + "'></div>");
    $("#vis_" + reportIndex).addClass("qwe");
    v.report({
        server: serverUrls[urlUsed],
        resource: uri,
        container: "#vis_" + reportIndex,
        error: function (err) {
            alert(err.message);
        },
        defaultJiveUi: { enable: defaultJiveUi },
        isolateDOM: isolateDOM || false,
        success: function () {
            fillSheetList();
            reportsList[reportIndex] = uri;
            fillReportsList();
            //processIC(v, uri);
        }
    });

}

function processIC (v, reportUri) {
    var inputControls = v.inputControls({
        resource: reportUri,
        success: function(data) {
            console.log(data);
        }
    });
}

function fillReportsList() {
    $("#reportsList").html("");
    for (var reportIndex in reportsList) {
        if (!reportsList.hasOwnProperty(reportIndex)) continue;
        var uri = reportsList[reportIndex];
        //if (uri)
        var li = $("<li>" + (/[^\/][\w\.]+$/g.exec(reportsList[reportIndex]) || "Loading...") + " (<a 
href='#'>remove</a>)</li>");
        $("#reportsList").append(li);
        li.children("a").click((function (reportIndex) {
            return function (e) {
                e.preventDefault();
                $("#vis_" + reportIndex).remove();
                delete reportsList[reportIndex];
                $(e.target).parent().remove();
            };
        })(reportIndex));

    }

}

function fillSheetList() {
    var sheets = $("link");
    var checkboxLI = '<li>#index: <input type="checkbox" id="#id" checked="checked"><label for="#id" 
title="#title">#label</label></li>',
        sheetPath = '',
        sheetPathSplitted = '',
        html = "";

    $("#sheetList").html("");

    for (var i = 0; i < sheets.length; i++) {

        if (sheets[i].href === null) continue;
        sheetPath = sheets[i].href;

        sheetPathSplitted = sheetPath.split("/");
        html = checkboxLI.replace(/#id/g, "sheetItem_" + i)
            .replace(/#index/g, i+1)
            .replace("#title", sheetPath)
            .replace("#label", sheets[i].label || sheetPathSplitted[sheetPathSplitted.length - 1]);

        $("#sheetList").append(html);
        $("#sheetItem_" + i).change(function (e) {
            var id = this.id.split("_")[1];
            $("link")[id].disabled = !$(this).is(':checked');
        });
        $("#sheetItem_" + i)[0].checked = !$("link")[i].disabled;
    }
}
```

The HTML for the CSS diagnostic tool contains a static list of reports to load. Add your own reports to this list.

```
<script type="text/javascript">
    window.addEventListener("load", onLoad);
</script>
<div style="width:327px;float: left;border-right:1px solid #333;margin-right:2px">
     <h4>Settings</h4>

    <div>
        <div id="containerLoadV">
            <div id="serverUrlsDiv"></div>
            <label for="confLocale">Locale: </label>
            <input id="confLocale" value="en" />
            <!-- options -->
            <br/>
            <input id="confOptimized" type="checkbox" />
            <label for="confOptimized" >- use optimized javascript </label>
            <br/>
            <button id="loadV">Load visualize</button>
            <br/>

        </div>
        <div id="loadReports" style="display:none;">
            <div>Add report:</div>
            <input id="defaultJiveUi" type="checkbox" checked="checked" />
            <label for="defaultJiveUi" >- default JIVE UI </label>
            <br/>
            <input id="isolateDOM" type="checkbox" />
            <label for="isolateDOM" >- isolate DOM </label>
            <br/>

            <select id="selected_resource" name="report" style="width:195px">
                <option value="">-</option>
                <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>
            <button id="addReport">Add</button>
        </div>
        <p>Loaded reports list:</p>
        <ul id="reportsList" style="overflow-wrap: break-word;display:inline;"></ul>
        <div style="border-top:1px solid #333;width:100%"></div>
        <h4>Stylesheets list:</h4>
        <em>Use 1-9 keys to enable\disable css libs.</em>

        <ul id="sheetList" style="display:inline;"></ul>
    </div>

    <div style="border-top:1px solid #333;width:100%"></div>

    <div style="width:320px;">
         <h4>User components</h4>

        <!-- detepicker -->
        <div style="height: 240px;">
            <div id="datepicker-user"></div>
        </div>
        <input id="datepicker-user2"/>
        <!-- /datepicker -->
        <!-- autocomplete -->
        <div class="ui-widget">
            <label for="tags">Autocomplete</label>
            <input id="tags"/>
        </div>
        <!-- autocomplete -->
        <!-- tabs -->
        <div id="tabs">
          <ul>
            <li><a href="#tabs-1">Nunc</a></li>
            <li><a href="#tabs-2">Pr dor</a></li>
            <li><a href="#tabs-3">A laia</a></li>
          </ul>
          <div id="tabs-1">
            <p>Proin elit arcu Aliquam sodales tortor vitae ipsum. Aliquam nulla. Duis aliquam moles-
tie erat. Ut et mauris vel pede varius sollicitudin. Sed ut dolor nec orci tincidunt interdum. Pha-
sellus ipsum. Nunc tristique tempus lectus.</p>
          </div>
          <div id="tabs-2">
            <p>Morbi tincidunt, tellus pellentesque pretium posuere, felis lorem euismod felis, eu
```

Associated CSS:

```
.qwe {
    height: 100%;
}
#reportContainer {
    height: 100%;
}
body, html {
    height: 100%;
}
#containerLoadV.disabled {
    color: #666;
}

/**
  * Break something: modify the CSS here to something visibly wrong.
  * Add more elements, classes, or IDs to see if they affect Visualize.js content.
  */

table {
    font-size: 25px;

}
```

INDEX