REVOLUTION ANALYTICS

RevoDeployR™ API Reference Guide

Version 1.0

We want our documentation to be useful, and we want it to address your needs. If you have comments on this or any Revolution document, send e-mail to doc@revolutionanalytics.com. We'd love to hear from you.

Contents

Chapter 1 Introduction

RevoDeployR supports the development of Web-enabled R applications. Web-enabled R applications integrate with RevoDeployR services exposed on a public R Web services API.

RevoDeployR services support two distinct categories of client application:

- Stateful R session clients. In a stateful environment, the client controls when and how an R session is created. The client also has knowledge of R code being executed, and can directly manipulate the R objects in the session.
- Stateless R session clients. In a stateless environment, the client passes a collection of parameters that invokes a predefined R script (defined in the RevoDeployR management console) that is executed and some values are returned.

1.1 Stateful Services

For those clients that have a trust relationship with the server and need to interact in an open-ended way with R on the server, a collection of Session Management services are provided. These services allow for such things as:

- Creating an R session
- Executing arbitrary blocks of R code
- Retrieving R objects and artifacts resulting from code execution, such as plots
- Destroying R sessions.

The details of this *Session Management* service layer are described in Chapters 3 and 4.

1.2 R Script Services

Clients that need to invoke a predefined R script in a controlled manner will use a specialized *R Script* layer. This layer sits on top of the *Session Manager* and other support services.

R scripts are typically invoked in a stateless environment, but support for stateful execution is also provided. This *R Script* service layer is described in Chapter 8.

1.3 Supporting Services

In order to effectively support both stateful and stateless operating environments, a collection of support services are also defined. These include:

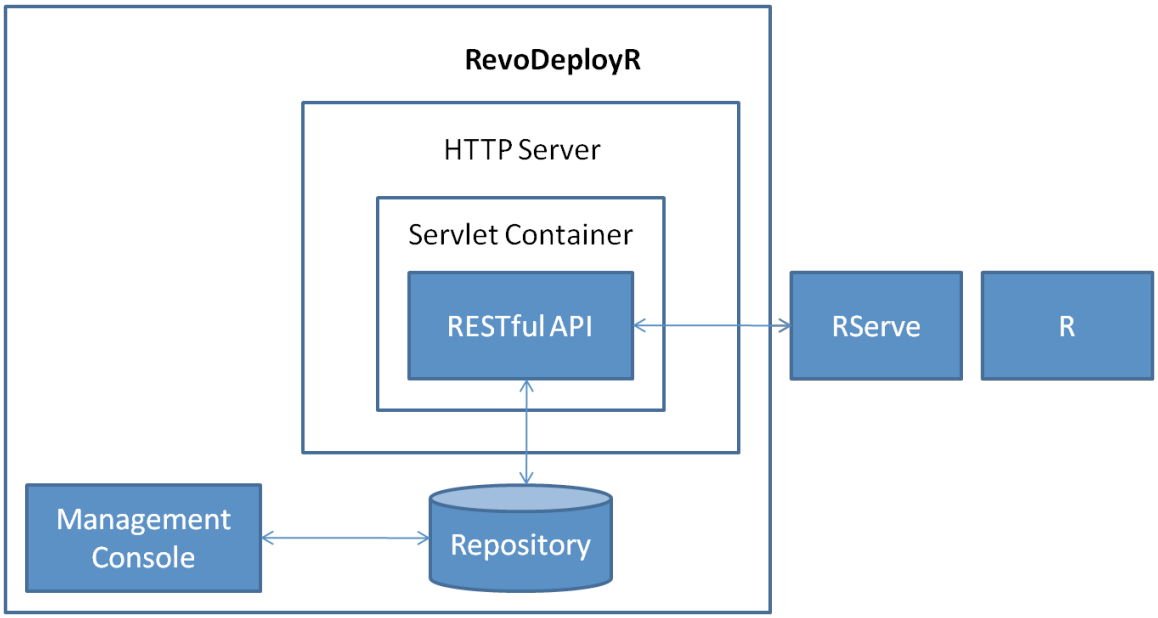
- *User* services, which control access to API calls, R scripts and R objects.
- *File Management* services, which upload files into the HTTP Session and manage plots generated from R code execution.
- *Project Management* services, which store and retrieve R sessions with RevoDeployR state, including the R workspace, command history and generated outputs such as plots.

1.4 Architectural Components

RevoDeployR exposes a RESTful API that delivers R Web services to any authenticated client using HTTP.

The API is implemented internally using RServe, which is an R package supporting remote client connections to live R sessions. For more information on RServe, refer to **`http://www.rforge.net/Rserve/doc.html`**.

The components are shown in the following drawing:



For more information on REST, refer to **`http://en.wikipedia.org/wiki/Representational_State_Transfer`**.

1.5 RevoDeployR Management Console

The RevoDeployR management console provides administrative functions to support the integration of Web or desktop client applications with the RevoDeployR Web services API.

Authorized users with administrator or script manager permissions can use the management console to create, edit, and deploy R scripts and R objects. These R scripts and R objects can then be used by client applications on the API. Additionally, administrators can create and manage user roles, R boundaries (CPU and memory), and IP access filters in order to define the context in which the R scripts are accessed and executed.

The console is also used to manage user accounts and grant permissions to some or all of the RevoDeployR R Web services APIs. There are three user accounts that are preconfigured in the management console:

- The default system administrator, **`admin`**. We recommend changing the password immediately to prevent unauthorized access.
- A default script manager, **`testmanager`**. We strongly recommend that you disable this account in production deployments.
- A default test account for developers, **`testuser`**. We strongly recommend that you disable this account in production deployments.

For more information on the management console, refer to the RevoDeployR **Management Console User's Guide**.

Chapter 2 R Web Services API

All of the services described in the API Reference Guide are available via a RESTful API using HTTP.

You can send HTTP GET or POST methods to call exposed API methods.

- HTTP GET method is required when using methods to retrieve data on the API.
- HTTP POST method is required when using methods that submit, change, or destroy data.

In return, you can obtain either JSON or XML output.

Additionally, each request will return a meaningful HTTP response status code. For a list of response codes, refer to section 2.2.

Parameters

For each service call, there is a list of required and optional parameters.

Parameter values are expected to be UTF-8 compliant and URL-encoded.

The RevoDeployR API currently supports both JSON and XML formats for data exchange. Consequently, each method requires the format parameter in order to specify how to format the request and response data on the call.

While all parameters are specified as a name/value pair, some parameter values require complex data. Whenever complex data is required, a XML/JSON schema defines how these values are to be constructed. For more information on the schema definitions and specific examples on the service calls where appropriate, refer to Appendix A and Appendix B in this document.

2.1 HTTP Response Status Codes

The RevoDeployR R Web Services API responds using HTTP status codes as described in the table. Client applications designed around this API should handle these response codes in their implementation.

Table 1: HTTP response codes

Code	Description	Type
200 OK	Successful	Success
400 Bad Request	Invalid data was passed on the call	Error
401 Unauthorized Access	Authentication credentials were invalid	Error
403 Forbidden	Caller is not authorized for call	Error
405 HTTP Method Disallowed	Incorrect HTTP method used on call	Error
409 Conflict	R session is currently busy executing call. Concurrent call rejected	Error
503 Service Temporarily Unavailable	Logout temporarily invalidated HTTP session	Error

2.2 Summary of API Call

Session Services

Call	Description
/r/session/create	Creates an R session
/r/session/close	Closes and deletes the specified R session
/r/session/ping	Pings the specified R session
/r/session/execute/code	Execute a block of R code in the specified R session
/r/session/execute/script	Executes an R script in the specified R session
/r/session/output	Retrieves the console output from the specified R session
/r/session/history	Retrieves the history from the specified R session
/r/session/workspace/save	Saves the R workspace of the specified R session to the RevoDeployR repository
/r/session/project/save	Saves the specified R session and associated server state as a RevoDeployR project

Note: The */r/session/create* call returns a session identifier. All other */r/session/** calls take a session identifier as a call parameter.

Live Object Management

Call	Description
/r/session/object/list	Lists the R objects in the specified R session
/r/session/object/get	Gets the value of a named R object in the speicifed R session
/r/session/object/save	Saves the named R object from the specified R session to the RevoDeployR repository
/r/session/object/delete	Deletes the named R object from the specified R session

/r/session/object/load	Loads an R object from the repository into the specified R session

Repository Object Management

Call	Description
/r/repository/object/list	Lists the R objects stored in the RevoDeployR repository
/r/repository/object/delete	Deletes an R object from the RevoDeployR repository

File Management

Call	Description
/r/file/upload	Uploads a file into the RevoDeployR repository

Project Management

Call	Description
/r/project/list	Lists the RevoDeployR projects for a user
/r/project/load	Loads a RevoDeployR project into a new session
/r/project/delete	Deletes a RevoDeployR project

User Services

Call	Description
/r/user/login	Supports user login using basic authentication
/r/user/logout	Supports user log out
/r/user/whoami	Returns currently logged-in user
/r/user/autosave	Enables/disables autosave for all R sessions in the user's HTTP session

RScript Services

Call	Description
/r/script/list	Lists the R scripts in RevoDeployR repository that are available for execution
/r/script/execute	Executes an R script in a stateless environment

Chapter 3 Session Services

3.1 Sessions

R sessions are created in response to explicit requests on the R Web services API such as `Create Session` or created implicitly to facilitate the stateless execution of R scripts.

3.2 Request Handling

Because R is neither thread-safe nor re-entrant, the RevoDeployR implementation prevents the concurrent execution of R commands within a specific R session.

Each request on an R session must complete before the next request is accepted on the service interface.

Whenever requests are made on a "busy" R session, the call will be rejected by RevoDeployR with a `HTTP 409` status code.

3.3 Errors and Warning

Distinct from the HTTP response codes returned on a request, R errors and warnings can also be generated, particularly when executing code.

Errors and warning messages from the R session are indicated in the response markup on each call and should be handled by the client application accordingly.

3.4 Preconfiguring Sessions

In some cases, it may be useful to start each R session with a default set of packages and objects preloaded. This can be accomplished by modifying the appropriate entries in RServe configuration file Rserv.conf. This preload setting will affect every session. For more information on this file, refer to `http://www.rforge.net/Rserve/doc.html`.

3.5 Create Session

Creates a new R session for the currently logged in user, and returns a session identifier

End Point

/r/session/create

HTTP Method

POST

Request Format

Content-Type: application/x-www-form-urlencoded; charset=utf-8

Parameters

format – specifies the format for the output as either JSON or XML

Example

```
POST /r/session/create
format=json
```

Return (XML)

```
<deployr>
  <response success="true" call="/r/session/create" session="LIVE-1daa502b-
   a68c-48b7-9e54-c3983c2f1cbd"/>
</deployr>
```

Return (JSON)

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/session/create",
            "session": "LIVE-1daa502b-a68c-48b7-9e54-c3983c2f1cbd"
        }
    }
}
```

3.6 Close Session

Closes the R session specified by the session identifier, and cleans up all of the resources associated with that session. The specific cleanup semantics are determined by the autosave configuration set on the user's HTTP session. For more information, refer to the description in section 7.6, AutoSave.

End Point

/r/session/close

HTTP Method

POST

Request Format

Content-Type: application/x-www-form-urlencoded; charset=utf-8

Parameters

format – specifies the format for the output as either JSON or XML

session – specifies a valid session identifier

Example

```
POST /r/session/close
format=json
session=LIVE-1daa502b-a68c-48b7-9e54-c3983c2f1cbd
```

Return (XML)

```
<deployr>
  <response success="true" call="/r/session/close" session="LIVE-1daa502b-
   a68c-48b7-9e54-c3983c2f1cbd"/>
</deployr>
```

Return (JSON)

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/session/close",
            "session": "LIVE-1daa502b-a68c-48b7-9e54-c3983c2f1cbd"
        }
    }
}
```

3.7 Ping Session

Sends a ping to the specified R session to ensure that it is still alive.

End Point

/r/session/ping

HTTP Method

GET

Request Format

Content-Type: application/x-www-form-urlencoded; charset=utf-8

Parameters

format – specifies the format for the output as either JSON or XML

session – specifies a valid session identifier

Examples

```
GET /r/session/ping
format=json
session=LIVE-92d9c643-5620-40a1-8626-47ded19970cc
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/session/ping" session="LIVE-92d9c643-
   5620-40a1-8626-47ded19970cc"/>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/session/ping",
            "session": "LIVE-92d9c643-5620-40a1-8626-47ded19970cc"
        }
    }
}
```

3.8 Execute Code

Executes a specific block of R code within the specified R session.

The following is returned, if generated:

- Any errors or warnings
- Any console output
- References to any plots

Plots can be either named or unnamed. Named plots must be specified by their short file name, such as myplot.png, using the files parameter. Unnamed plots are any other plots generated during execution.

Note: In order for R to produce console output, use the "**`print()`**" command.

End Point

`/r/session/execute/code`

HTTP Method

`POST`

Request Format

`Content-Type: application/x-www-form-urlencoded; charset=utf-8`

Parameters

`format –` specifies the format for the output as either JSON or XML

`session –` specifies a valid session identifier

`code` – specifies some R code to execute (make sure it is URL-encoded)

`robjects` – Optional. Comma-separated list of R objects to be returned

`files` – Optional. Comma-separated list of files such as plots and text files to be returned

Examples

Important! The code must be URL-encoded before it can be posted to the server to ensure proper escaping of reserved characters.

```
POST /r/session/execute/code
format=xml
session=LIVE-44e81abe-46fa-4a7b-9c47-9931c81f3619
robjects=myVector
files=myplot.png
code=myVector <- rnorm(100); png("myplot.png"); plot(myVector); dev.off();
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/session/execute/code" session="LIVE-
    44e81abe-46fa-4a7b-9c47-9931c81f3619">
     <objects>
       <object name="test.png"
    value="http://[server]/deployr/r/file/get/FILE-62313dff-487b-4a47-978f-
    2741f381589b/test.png"/>
```

```
    </objects>
    <robjects>
      <robject name="myVector" type="vector">
        <numeric
   value="0.3164758010404593,1.1669658074158804,1.2471139556272761,1.28536
   97180130437,0.05985921935971333,-0.2775597023418813,-
   0.7652265340363482,1.7714992088525292,-0.3266298170428679,-
   0.0742444722443912"/>
      </robject>
    </robjects>
  </response>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/session/execute/code",
            "session": "LIVE-d14b331c-b9e7-4e88-a46a-8fadbe383094",
            "objects": {
                "test.png": {
                    "value": "http://[server]/deployr/r/file/get/FILE-
   c2432c95-6be7-45e1-b7d3-ac96975c88ee/test.png"
                }
            },
            "robjects": {
                "myVector ": {
                    "type": "vector",
                    "rclass": "numeric",
                    "value": [
                        -0.39290406400314515,
                        1.72026513680149,
                        0.41944517687483074,
                        0.35081350824251734,
                        -2.663078299306171,
                        1.4088335105668601,
                        0.4282611059992903,
                        -2.456163897318624,
                        -0.6516229630069713,
                        -1.0225354054529743
                    ]
                }
            }
        }
    }
}
```

3.9 Execute Script

Executes the specified R script in a specified R session. The following is returned, if generated:

- Specified R objects
- Any errors or warnings
- References to any generated plots. Plots can be either named or unnamed. Named plots must be specified by their short file name, such as myplot.png, using the files parameter. Unnamed plots are any other plots generated during execution.
- Any console output

This call is executed using the access privileges of the caller. If the caller has not authenticated with RevoDeployR, then the call is considered to have anonymous privileges, which means only R scripts deployed for anonymous access can be executed. If the caller has authenticated with RevoDeployR then this call is executed with the caller's access privileges, which are used to verify access to any secured R script before execution.

Note: In order for R to produce console output, use the "print()" command.

End Point

```
/r/session/execute/script
```

HTTP Method

```
POST
```

Request Format

```
Content-Type: application/x-www-form-urlencoded; charset=utf-8
```

Parameters

`format` – specifies the format for the output as either JSON or XML

`session`– specifies a valid session identifier

`rscript` – specifies the name of a valid R Script

`preload` – optional. Specifies the id of the object to preload in R session

`inputs` – optional. Specifies the XML/JSON encoded inputs for R Script

`robjects` – optional. Specifies the comma-separated list of R objects to be returned

`files` – optional. Specifies the comma-separated list of files (e.g. plots, text files) to be returned

`saveworkspace` – optional. When true, the R workspace is saved after executing R Script

Examples

```
POST /r/session/execute/script
format=json
session=LIVE-f4500f7a-8d84-4d10-91d7-c892b06e0f0e
rscript=Sample Script
inputs={
    "df" : {
        "type": "dataframe",
        "value": {
            "a": {
                "type": "vector",
                "value": [
                    1,
                    2,
                    3
                ]
            },
            "b": {
                "type": "vector",
                "value": [
                    4,
                    5,
                    6
                ]
            }
        }
    },
    "x": {
        "value": "df$b"
    },
    "y": {
        "value": "df$a"
    }
}
```

RScript

This is the server-deployed R script that will be executed.

```
##The inputs to this script are "df", "x" and "y".
myformula<-formula(paste(x,"~",y))
print(myformula)
lm1<-lm(myformula)
plot(lm1)
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/session/execute/script" session="LIVE-
   544b1a49-839d-406f-aa5c-3c2e8be1ae79">
    <robjects>
    </robjects>
    <repository>
    </repository>
    <files>
```

```
      <file name="unnamedplot001.png"
   value="http://[server]/deployr/r/file/get/FILE-3d850e2c-864a-44f2-8188-
   811f31110962/unnamedplot001.png"/>
      <file name="unnamedplot002.png"
   value="http://[server]/deployr/r/file/get/FILE-9f17eff9-eef0-4688-8d12-
   b50c8e8f5681/unnamedplot002.png"/>
      <file name="unnamedplot003.png"
   value="http://[server]/deployr/r/file/get/FILE-a1482819-0669-4556-aa59-
   f4532c209e70/unnamedplot003.png"/>
      <file name="unnamedplot004.png"
   value="http://[server]/deployr/r/file/get/FILE-d10e0bcd-7521-460a-8337-
   c8fecfa55130/unnamedplot004.png"/>
    </files>
    <console><![CDATA[df$b ~ df$a]]>
    </console>
  </response>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/session/execute/script",
            "robjects": {

            },
            "repository": {

            },
            "files": {
                "unnamedplot001.png": {
                    "value": "http://[server]/deployr/r/file/get/FILE-
   3d850e2c-864a-44f2-8188-811f31110962/unnamedplot001.png"
                },
                "unnamedplot002.png": {
                    "value": "http://[server]/deployr/r/file/get/FILE-
   9f17eff9-eef0-4688-8d12-b50c8e8f5681/unnamedplot002.png"
                },
                "unnamedplot003.png": {
                    "value": "http://[server]/deployr/r/file/get/FILE-
   a1482819-0669-4556-aa59-f4532c209e70/unnamedplot003.png"
                },
                "unnamedplot004.png": {
                    "value": "http://[server]/deployr/r/file/get/FILE-
   d10e0bcd-7521-460a-8337-c8fecfa55130/unnamedplot004.png"
                }
            },
            "console": {
                "value": "df$b ~ df$a"
            },
            "session": "LIVE-e0e67988-3403-4c4d-8200-75a0ec18e242"
        }
    }
}
```

3.10 Get Output

Retrieves the last console output from the specified R session.

Note: In order for R to produce console output, you must use the print() command.

End Point

```
/r/session/output
```

HTTP Method

```
GET
```

Request Format

```
Content-Type: application/x-www-form-urlencoded; charset=utf-8
```

Parameters

`format` – specifies the format for the output as either JSON or XML

`session` – specifies a valid session identifier

Examples

```
GET /r/session/output
format=json
session=LIVE-92d9c643-5620-40a1-8626-47ded19970cc
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/session/output" session="LIVE-92d9c643-
   5620-40a1-8626-47ded19970cc">
    <console><![CDATA[
Call:
lm(formula = x ~ y)

Residuals:
       Min         1Q     Median         3Q        Max
-2.280e-15 -8.925e-16 -2.144e-16  4.221e-16  4.051e-15

Coefficients:
             Estimate Std. Error    t value Pr(>|t|)
(Intercept) -1.000e+01  3.095e-15 -3.231e+15   <2e-16 ***
y            1.000e+00  1.963e-16  5.093e+15   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.783e-15 on 8 degrees of freedom
Multiple R-squared:     1, Adjusted R-squared:     1
F-statistic: 2.594e+31 on 1 and 8 DF,  p-value: < 2.2e-16

 ]]></console>
  </response>
</deployr>
```

Return (JSON) :


```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/session/output",
            "session": "LIVE-92d9c643-5620-40a1-8626-47ded19970cc",
            "console": {
                "value": "\nCall:\nlm(formula = x ~ y)\n\nResiduals:\n
   Min         1Q     Median         3Q        Max \n-2.280e-15 -8.925e-16
   -2.144e-16  4.221e-16  4.051e-15 \n\nCoefficients:\n
   Estimate Std. Error    t value Pr(>|t|)    \n(Intercept) -1.000e+01
   3.095e-15 -3.231e+15   <2e-16 ***\ny            1.000e+00  1.963e-16
   5.093e+15   <2e-16 ***\n---\nSignif. codes:  0 '***' 0.001 '**' 0.01
   '*' 0.05 '.' 0.1 ' ' 1 \n\nResidual standard error: 1.783e-15 on 8
   degrees of freedom\nMultiple R-squared:     1,\tAdjusted R-squared:
   1 \nF-statistic: 2.594e+31 on 1 and 8 DF,  p-value: < 2.2e-16 \n\n"
            }
        }
    }
}
```

3.11 Get History

Retrieves the complete history for the specified R session.

End Point

```
/r/session/history
```

HTTP Method

```
GET
```

Request Format

```
Content-Type: application/x-www-form-urlencoded; charset=utf-8
```

Parameters

`format` – specifies the format for the output as either JSON or XML

`session` – specifies a valid session identifier

Examples

```
GET /r/session/history
format=json
session=LIVE-92d9c643-5620-40a1-8626-47ded19970cc
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/session/history" session="LIVE-
   92d9c643-5620-40a1-8626-47ded19970cc">
    <history>
      <command><![CDATA[ x<-c(1:10);
y<-c(11:20);
lm1<-lm(x~y);
print(summary(lm1)) ]]></command>
    </history>
  </response>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/session/history",
            "history": [
                "x<-c(1:10);\ry<-c(11:20);\rlm1<-
   lm(x~y);\rprint(summary(lm1))"
            ],
            "session": "LIVE-92d9c643-5620-40a1-8626-47ded19970cc"
        }
    }
}
```

3.12 Save Workspace

Saves the R workspace of the specified R session to the repository on the server, and returns an ID to that saved object. Saving an R workspace does not close the current session.

End Point

`/r/session/workspace/save`

HTTP Method

`POST`

Request Format

`Content-Type: application/x-www-form-urlencoded; charset=utf-8`

Parameters

`format` – specifies the format for the output as either JSON or XML

`session` – specifies a valid session identifier

`descr` – provides a plain text description of the workspace

Examples

```
POST /r/session/workspace/save
format=json
session=LIVE-92d9c643-5620-40a1-8626-47ded19970cc
descr=My Workspace 1
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/session/workspace/save" session="LIVE-
   92d9c643-5620-40a1-8626-47ded19970cc">
    <repository>
      <object name="My Workspace 1" value="CBNRY-b4053637-cbe8-48e4-bfb0-
   be852c81d98e"/>
    </repository>
  </response>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/session/workspace/save",
            "repository": {
                "My Workspace 1": {
                    "value": "CBNRY-7f8fa254-cc5a-4f77-a9d9-3b873be1bad3"
                }
```

```
            },
            "session": "LIVE-92d9c643-5620-40a1-8626-47ded19970cc"
        }
    }
}
```

3.13 Save Project

Save the specified R session and associated server state as a RevoDeployR project. Saving does not close the current session.

If you save a RevoDeployR project and the R session you are currently working in was not opened from a RevoDeployR project, then a new project will be created. Otherwise, you will be updating your existing project.

If you are working in a session that was loaded from a project and you want to rename your project, provide a new value for the **`descr`** parameter on this call.

End Point

`/r/session/project/save`

HTTP Method

`POST`

Parameters

`format –` specifies the format for the output as either JSON or XML

`session` – specifies a valid session identifier

`descr –` provides a plain text description of the project

Examples

```
POST /r/session/project/save
format=json
session=LIVE-5a7c6bec-7daa-455e-84b9-100396f1984c
descr=project1
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/session/project/save" session="LIVE-
   5a7c6bec-7daa-455e-84b9-100396f1984c">
    <projects>
      <project name="project1" value="PROJ-95533894-edca-489f-94c9-
   fc44cbc72151"/>
    </projects>
  </response>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/session/project/save",
            "projects": {
                "project1": {
                    "value": "PROJ-895e0152-8be3-4f9b-904a-2f4e9077811f"
                }
            },
            "session": "LIVE-5a7c6bec-7daa-455e-84b9-100396f1984c"
```

```
            }
        }
}
```

Chapter 4 R Object Management

R objects are entities in an R session that:

- Contain a value
- Can be queried

R objects can be saved into the repository for reuse and then reloaded into an R session.

For RevoDeployR, there are two R object classifications:

- *Live* R objects
- *Stored* R objects

4.1 Live R Objects

Live R objects are those objects in the current R session. For purposes of simplicity, only certain classes of R objects can have their values retrieved by the R server. To see which R objects can have their values retrieved, refer to the table below.

In R, objects have both a class and a type. RevoDeployR uses a more generic rendering of this concept in order to further abstract the client developer from having to understand the specifics of R.

RevoDeployR uses a "type" attribute in combination with the data contents to provide the mapping to R class and type. The content of the data is used to identify the R class of the data.

For example, a vector of integers is specified by the attribute "type=vector", RevoDeployR uses the content of the data, such as [1,2,4,8], to determine and return the R class attribute "rclass=numeric".

Table 2: R Objects whose values can be retrieved

RevoDeployR "type" Attribute	R Class Data types	Note
primitive	numeric, character, logical	
date	date	uses a "format" attribute to define the date
factor	factor	uses an optional "levels" and "ordered" attributes
vector	numeric, character, logical, date	
matrix	matrix	
list	list	
dataframe	data.frame	

For an example of RevoDeployR types represented in JSON, refer to Appendix B.

For more complex R objects, such as S3 and S4 objects, any attempts to retrieve a value will return a NULL.

Missing values are represented by "NA" or "NAN" in R. For inputs to RevoDeployR where NA or NAN values need to be represented, the JSON null type will be used. RevoDeployR outputs will also indicate NA or NAN using the JSON null type.

XML values are strings. An NA value is represented by an empty string, for example "" for a primitive or "1,,3" for a vector or matrix value.

4.2 Repository R Objects

Repository R objects are binary images that have been saved in the repository for reuse.

Saving Live R Objects as Repository R Objects

Any R object can be saved as a binary image during the current R session using the API call: **`/r/session/object/save`**.

The R objects saved via this call are stored in the repository where they will live for the duration of current user's HTTP session. Furthermore, they are visible only to the user who saved them.

The **`/r/session/object/save`** call is detailed later in this chapter.

Saving Reusable, Multi-user Repository R Objects

R objects can also be saved more permanently using the RevoDeployR Management Console. The R objects stored in the management console can be made accessible to one or more users and in multiple sessions.

To create reusable repository R objects, save a binary file to the server machine or send it to the administrator. Then, an administrator of the management console can upload to the repository via the management console's interface. For more information on the management console, refer to the RevoDeployR Management **Console User's Guide**.

Once an R object has been deployed into the RevoDeployR repository via the RevoDeployR Management Console, the administrator can control access to the object by associating roles with the object. Users assigned those same role(s) are granted permission to see and use the object.

Loading Repository R Objects

Repository R objects can be loaded into the current R session by the API call **`/r/session/object/load`**. This call is detailed later in this chapter.

4.3 List Live Objects

Lists the live R objects and their data types for the specified R session.

For a description of the listing repository R objects, refer to section 4.8 later in this chapter.

End Point

`/r/session/object/list`

HTTP Method

`GET`

Parameters

`format` – specifies the format for the output as either JSON or XML

`session` – specifies a valid session identifier

`filter` – optional. When true, returns a filtered list of only those R objects for which values can be retrieved. For more information, refer to section 4.1 at the beginning of this chapter. If not set or false, all R objects will be returned.

Examples

```
GET /r/session/object/list
format=json
session=LIVE-5a7c6bec-7daa-455e-84b9-100396f1984c
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/session/object/list" session="LIVE-
    5a7c6bec-7daa-455e-84b9-100396f1984c">
    <robjects>
      <robject name="x" type="vector" rclass="numeric"/>
      <robject name="y" type="vector" rclass="numeric"/>
    </robjects>
  </response>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/session/object/list",
            "robjects": {
                "x": {
                    "rclass": "numeric",
                    "type": "vector"
                },
                "y": {
                    "rclass": "numeric",
                    "type": "vector"
                }
```

```
            },
            "session": "LIVE-5a7c6bec-7daa-455e-84b9-100396f1984c"
        }
    }
}
```

4.4 Get Live Object

Retrieves the value of a specified R object for the current R session.

End Point

`/r/session/object/get`

HTTP Method

`GET`

Parameters

`format -` specifies the format for the output as either JSON or XML

`session`– specifies a valid session identifier

`name` – specifies the name of the object to be retrieve

`start` – optional. Specifies for those objects that support it, the start row

`length` – optional. Specifies for those objects that support it, the number of rows to retrieve

Examples

```
GET /r/session/object/get
name=x
format=json
session=LIVE-5a7c6bec-7daa-455e-84b9-100396f1984c
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/session/object/get" session="LIVE-
   5a7c6bec-7daa-455e-84b9-100396f1984c">
    <robjects>
      <robject name="x" type="vector">
        <numeric value="13,14,15"/>
      </robject>
    </robjects>
  </response>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/session/object/get",
            "robjects": {
                "x": {
                    "rclass": "numeric",
                    "type": "vector",
                    "value": [
                        13,
                        14,
                        15
```

```
                    ]
                }
            },
            "session": "LIVE-5a7c6bec-7daa-455e-84b9-100396f1984c"
        }
    }
}
```

4.5 Save Live R Object

Saves the specified R object in the RevoDeployR repository and returns an ID for the new Repository R object.

End Point

`/r/session/object/save`

HTTP Method

`POST`

Parameters

`format` – specifies the format for the output as either JSON or XML

`session`– specifies a valid session identifier

`name` – specifies the name of the R object to save

`descr` – provides a plain text description of the object that will be used as the name when the stored object name is listed using **`/r/repository/object/list`**.

Examples

```
POST /r/session/object/save
format=json
session=LIVE-5a7c6bec-7daa-455e-84b9-100396f1984c
name=x
descr = test_x
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/session/object/save" session="LIVE-
   5a7c6bec-7daa-455e-84b9-100396f1984c">
    <repository>
      <object name="test_x" value="CBNRY-47dbb4f3-eb69-4b97-b0b6-
   5f1d196ad75d"/>
    </repository>
  </response>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/session/object/save",
            "repository": {
                "test_x": {
                    "value": "CBNRY-fd8ac50d-722c-45e5-beb1-38538aecf91b"
                }
            },
            "session": "LIVE-5a7c6bec-7daa-455e-84b9-100396f1984c"
        }
    }
```

```
}
```

4.6 Delete Live R Object

Deletes the named R object from the specified R session.

End Point

`/r/session/object/delete`

HTTP Method

`POST`

Parameters

`format –` specifies the format for the output as either JSON or XML

`session`– specifies a valid session identifier

`name` – specifies the name of the R object to delete

Examples

```
POST /r/session/object/delete
format=json
session=LIVE-5a7c6bec-7daa-455e-84b9-100396f1984c
name=x
descr = test_x
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/session/object/delete" session="LIVE-
   5a7c6bec-7daa-455e-84b9-100396f1984c"/>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/session/object/delete",
            "session": "LIVE-5a7c6bec-7daa-455e-84b9-100396f1984c"
        }
    }
}
```

4.7 Load Repository R Object

Retrieves the value of a given R object from the RevoDeployR repository, and then loads it into the specified R session.

End Point

```
/r/session/object/load
```

HTTP Method

```
POST
```

Parameters

`format` – specifies the format for the output as either JSON or XML

`session`– specifies a valid session identifier

`id` – specifies the id of the object to load. You can use /r/repository/object/list to retrieve the id of an object

Examples

```
POST /r/session/object/load
format=json
session=LIVE-5a7c6bec-7daa-455e-84b9-100396f1984c
id=CBNRY-fd8ac50d-722c-45e5-beb1-38538aecf91b
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/session/object/load" session="LIVE-
    5a7c6bec-7daa-455e-84b9-100396f1984c"/>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/session/object/load",
            "session": "LIVE-5a7c6bec-7daa-455e-84b9-100396f1984c"
        }
    }
}
```

4.8 List Repository R Objects

Retrieves the list of the R objects in the RevoDeployR repository that are visible to the currently logged in user. The currently logged in user can see any objects they have saved to the repository during the HTTP session as well reusable objects for which the user has access permissions.

End Point

```
/r/repository/object/list
```

HTTP Method

```
GET
```

Parameters

`format` – specifies the format for the output as either JSON or XML

Examples

```
GET /r/repository/object/list
format=json
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/repository/object/list">
    <repository>
      <pobject name="test_x" value="CBNRY-fd8ac50d-722c-45e5-beb1-
   38538aecf91b"/>
      <pobject name="test_y" value="CBNRY-c61a9531-d1ef-4589-9a9f-
   2952512a303e"/>
    </repository>
  </response>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/repository/object/list",
            "repository": {
                "test_y": {
                    "value": "CBNRY-c61a9531-d1ef-4589-9a9f-2952512a303e"
                },
                "test_x": {
                    "value": "CBNRY-fd8ac50d-722c-45e5-beb1-38538aecf91b"
                }
            }
        }
    }
}
```

4.9 Delete Repository R Object

Deletes the designated R object from the RevoDeployR repository.

End Point

`/r/repository/object/delete`

HTTP Method

`POST`

Parameters

`format –` specifies the format for the output as either JSON or XML

`id` – specifies the identifier of the object

Examples

```
POST /r/repository/object/delete
format=json
id=CBNRY-47dbb4f3-eb69-4b97-b0b6-5f1d196ad75d
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/repository/object/delete"/>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/repository/object/delete"
        }
    }
}
```

Chapter 5 File Management

Any file uploaded by a user or generated by an R session is transformed by RevoDeployR into a URL. This URL can be referenced by the user in any R code that they subsequently execute during that HTTP session.

5.1 File Types

Users interact with two kinds of files. They are:

- Uploaded files. These are native files (.xls, .csv, etc.) that are uploaded to the server by the user using the API call **`/r/file/upload`**. Each file is made available to the user as a URL.
- Plots. There are files generated by plot commands executed in an R session. Each plot generated as a result of executing a block of R code or an R script will be made available to the user as a URL.

5.2 File Lifetime

Files that have been uploaded or created by a user only live for the duration of the current user's HTTP session. Furthermore, those files are only visible to that user.

5.3 Upload File

Uploads a file to the server that remains valid for the life of the HTTP session. It returns the URL of the file upon the successful completion of the upload.

End Point

```
/r/file/upload
```

HTTP Method

```
POST
```

Request Format

```
Content-Type: application/x-www-form-urlencoded; charset=utf-8
```

Parameters

`format –` specifies the format for the output as either JSON or XML

`name` – specifies the name of the client file to be uploaded

`file` – specifies the binary image or plain text of the file. Keep in mind that you need to include the entire contents of the file.

Examples

```
POST /r/file/upload
name=satscores.csv
format=json
file=... spreadsheet contents here ...
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/file/upload">
    <files>
      <file name="satscores.csv"
   value="http://<server>/deployr/r/repository/get/binary/CFILE-2c98d98e-
   e143-4b85-b784-630e98279e9b"/>
    </files>
  </response>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "files": {
                "satscores.csv": {
                    "value":
   "http://<server>/deployr/r/repository/get/binary/CFILE-e639b6c7-3a7a-
   4706-8c86-9757bee8fa9f"
                }
            },
            "success": true,
            "call": "/r/file/upload"
        }
```

```
        }
}
```

Chapter 6 Project Management

RevoDeployR supports the creation and management of RevoDeployR projects. Users can create as many projects as they wish.

When a project is created using **`/r/session/project/save`**, it contains a snapshot of the current session, specifically:

- The R workspace at the time the project is created
- The server state at the time the project is created, which includes the complete session command history and any outputs such as plots generated during that R session.

When a user loads a project, the associated R workspace is immediately loaded into a `new` session along with the saved server state (command history, output, and plots).

Projects have a permanent lifetime, and they will only be visible to the user that created them.

6.1 Save Project

Save the specified R session and associated server state as a RevoDeployR project. Saving does not close the current session.

As saving a project is an operation on a live R session see section 3.13 for details.

List Projects

Lists the RevoDeployR projects belonging to the currently logged in user.

End Point

/r/project/list

HTTP Method

GET

Parameters

format – specifies the format for the output as either JSON or XML

Examples

```
GET /r/project/list
format=json
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/project/list">
    <projects>
      <project name="project1" value="PROJ-95533894-edca-489f-94c9-
   fc44cbc72151"/>
    </projects>
  </response>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/project/list",
            "projects": {
                "project1": {
                    "value": "PROJ-895e0152-8be3-4f9b-904a-2f4e9077811f"
                }
            }
        }
    }
}
```

6.3 Load Project

Retrieves the specified RevoDeployR project belonging to the currently logged in user and creates a new R session from the data in the project.

End Point

`/r/project/load`

HTTP Method

`POST`

Parameters

`format` – specifies the format for the output as either `JSON` or `XML`

`id` – specifies the identifier of the project to retrieve

Examples

```
POST /r/project/load
format=json
id=PROJ-95533894-edca-489f-94c9-fc44cbc72151
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/project/load" session="LIVE-9df845b6-
   68dc-426e-a26f-afabdd4ad109"/>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/project/load",
            "session": "LIVE-2a0ca713-c5bd-45a8-ba13-aa16a1e8a178"
        }
    }
}
```

6.4 Delete Project

Deletes the specified RevoDeployR project belonging to the currently logged in user.

End Point

`/r/project/delete`

HTTP Method

`POST`

Parameters

`format` – specifies the format for the output as either `JSON` or `XML`

`id` – specifies the identifier of the object

Examples

```
POST /r/project/delete
format=json
id=PROJ-895e0152-8be3-4f9b-904a-2f4e9077811f
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/project/delete"/>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/project/delete"
        }
    }
}
```

Chapter 7 User Services

To start any R session on RevoDeployR, the user must first login to RevoDeployR using a username and password.

User accounts are created and managed in the RevoDeployR management console. You must have administrator access to manage these user accounts. For more information, refer to the RevoDeployR Management Console User's Guide.

7.1 Basic Authentication

HTTP basic access authentication and SSL are used to encrypt the user identity between client and server. The username and password are authenticated on the server against the database of user accounts that were defined in the RevoDeployR Management Console.

A user account is often assigned one or more role(s). A role grants the user certain access privileges. Keep in mind that all stateful service calls require authenticated access. Using the RevoDeployR Management Console R scripts can be configured for role-based authenticated access, anonymous-yet-trusted access, or fully-anonymous access.

7.2 Enterprise Authentication

Interfaces that support enterprise-level services for authentication of users, such as OpenID-LDAP and SAML, are available; however, implementation of connection to and managing authorization will be available in future releases.

7.3 Login

Allows a user to log in using basic authentication, which requires a username and password.

End Point

`/r/user/login`

HTTP Method

`POST`

Request Format

`Content-Type: application/x-www-form-urlencoded; charset=utf-8`

Parameters

`format –` specifies the format for the output as either JSON or XML

`username` – specifies the username to log into the system

`password` – specifies the password for username (SHA1 Encoded)

`autosave` – optional. When true, indicates that all live sessions created by the user within an HTTP session should be automatically saved. For details on how autosave works, refer to section 7.6, AutoSave.

Examples

```
POST /r/user/login
format=json
username=johndoe
password=c51b5139556f939768f770dab8e5277a
autosave=false
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/user/login"
   cookie="4D0D8B6A921B4E70D0E52226693EAC53"/>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "cookie": "7B55A60B55093E7B5F320285490DBA75",
            "success": true,
            "call": "/r/user/login"
        }
    }
}
```

7.4 Logout

Logs the user out of the active HTTP session.

End Point

`/r/user/logout`

HTTP Method

`POST`

Parameters

`format` – specifies the format for the output as either JSON or XML

Examples

```
POST /r/user/logout
format=json
```

Return (XML):

```
<deployr>
    <response success="true" call="/r/user/logout"/>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": "true",
            "call": "/r/user/logout"
        }
    }
}
```

7.5 Whoami

Returns username and display name details of the currently logged in user.

End Point

`/r/user/whoami`

HTTP Method

`GET`

Request Format

`Content-Type: application/x-www-form-urlencoded; charset=utf-8`

Parameters

`format` – specifies the format for the output as either JSON or XML

Examples

```
GET /r/user/whoami
format=json
```

Return (XML)

```
<deployr>
 <response success="true" call="/r/user/whoami">
   <identity username="admin" displayname="RevoDeployr System Admin"/>
 </response>
</deployr>
```

Return (JSON) :

```
{
   "deployr": {
       "response": {
           "success": true,
           "call": "/r/user/whoami",
           "identity": {
               "username": "admin",
               "displayname": "RevoDeployR System Admin"
           }
       }
   }
}
```

7.6 Autosave

Enables or disables auto-saving of R sessions active within the user's HTTP session. When this parameter is set to 'true', all live R sessions in the user's HTTP session will be automatically saved as a project when:

- The user logs out
- The HTTP session times out
- The user closes an R session.

For more information about projects, refer to the chapter 6.

Autosave will either update existing projects or create new ones. When a user logs out or if a session times out, the **`autosave`** feature, if enabled, either update existing projects or create new ones where there are live R sessions that are not yet backed by an underlying project.

New projects will have a name following the format of **`Autosaved$DATE`**. Existing projects are those that were loaded by the user with **`/r/project/load`** during the HTTP session.

Automatic saving is disabled by default. It can be enabled using **`/r/user/login`** or **`/r/user/autosave`**. It stays turned on from the point or activation for the duration of the user's HTTP session or until it is explicitly disabled on a call to **`/r/user/autosave`**.

For a given user, the server remembers what sessions it persists at log out time or when a session times out. This means if **`autosave`** is enabled at login:

- All R sessions that were "live" when the user last logged out are reloaded.
- These reloaded live R sessions will respond to the same set of LIVE session IDs as used on the previous R session.

End Point

```
/r/user/autosave
```

HTTP Method

```
POST
```

Request Format

```
Content-Type: application/x-www-form-urlencoded; charset=utf-8
```

Parameters

`format –` specifies the format for the output as either JSON or XML

`save` – specifies whether AutoSave should be enabled (true) or disabled for the user's current HTTP session

Examples

```
POST /r/user/autosave
format=json
save=true
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/user/autosave" />
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/user/autosave"
        }
    }
}
```

Chapter 8 R Script Services

For many applications, having access to a collection of predefined R scripts makes it easy to expose data analytics to any Web client. The key to making this work is to provide a standardized data protocol for inputs as well as a well-defined interface to invoke these scripts.

In most cases, R scripts are used in the context of stateless R executions. In other words, the client asks for some analysis. Next, an R session is created. Then, the script is executed. Finally, the results are returned and the R session is destroyed.

In other cases it can be useful to execute an R script in a stateful context within a pre-existing live R session. Each object and output generated by the R script become part of the live R session context and continue to be available within that R session context after the R script execution completes.

RevoDeployR R script support provides layers of abstraction that simplify the development of these types of analytics programs for two audiences:

- R programmers: they do not need to have any knowledge of which Web-based client application(s) will ultimately use their script. They code solely in R. When an administrator deploys an R script in the management console, a Developer R Script Summary is generated that can be shared via a URL with application developer.
- R Web application developers: they can interface with the R script without having any knowledge of script internals. The script simply appears as a black box with well-defined inputs and outputs only. The client application passes parameters via the data protocol that that R script interprets for them. If the script was created in the management console, then a developer summary page is available and presents annotations, descriptions of inputs and outputs, and other basic information to help with implementation.

8.1 Execute Script in a Stateless Context

Executes the specified R script. The following is returned, if generated:

- Specified R objects
- Any errors or warnings
- References to any generated plots. Plots can be either named or unnamed. Named plots must be specified by their short file name, such as myplot.png, using the files parameter. Unnamed plots are any other plots generated during execution.
- Any console output

This call is executed using the access privileges of the caller. If the caller has not authenticated with RevoDeployR, then the call is considered to have anonymous privileges, which means only R scripts deployed for anonymous access can be executed. If the caller has authenticated with RevoDeployR then this call is

executed with the caller's access privileges, which are used to verify access to any secured R script before execution.

Note: In order for R to produce console output, use the "**`print()`**" command.

End Point

`/r/script/execute`

HTTP Method

`POST`

Request Format

`Content-Type: application/x-www-form-urlencoded; charset=utf-8`

Parameters

`format` – specifies the format for the output as either **`JSON`** or **`XML`**

`rscript` – specifies the name of a valid R Script

`preload` – optional. Specifies the id of the object to preload in R session

`inputs` – optional. Specifies the XML/JSON encoded inputs for R Script

`robjects` – optional. Specifies the comma-separated list of R objects to be returned

`files` – optional. Specifies the comma-separated list of files (e.g. plots, text files) to be returned

`saveworkspace` – optional. When true, the R workspace is saved after executing R Script

Examples

```
POST /r/script/execute
format=json
rscript=Sample Script
inputs={
    "df" : {
        "type": "dataframe",
        "value": {
            "a": {
                "type": "vector",
                "value": [
                    1,
                    2,
                    3
                ]
            },
            "b": {
                "type": "vector",
                "value": [
                    4,
                    5,
                    6
                ]
            }
```

```
        }
    },
    "x": {
        "value": "df$b"
    },
    "y": {
        "value": "df$a"
    }
}
```

RScript

This is the server-deployed R script that will be executed.

```
##The inputs to this script are "df", "x" and "y".

myformula<-formula(paste(x,"~",y))
print(myformula)
lm1<-lm(myformula)
plot(lm1)
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/script/execute">
    <robjects>
    </robjects>
    <repository>
    </repository>
    <files>
      <file name="unnamedplot001.png"
   value="http://[server]/deployr/r/file/get/FILE-3d850e2c-864a-44f2-8188-
   811f31110962/unnamedplot001.png"/>
      <file name="unnamedplot002.png"
   value="http://[server]/deployr/r/file/get/FILE-9f17eff9-eef0-4688-8d12-
   b50c8e8f5681/unnamedplot002.png"/>
      <file name="unnamedplot003.png"
   value="http://[server]/deployr/r/file/get/FILE-a1482819-0669-4556-aa59-
   f4532c209e70/unnamedplot003.png"/>
      <file name="unnamedplot004.png"
   value="http://[server]/deployr/r/file/get/FILE-d10e0bcd-7521-460a-8337-
   c8fecfa55130/unnamedplot004.png"/>
    </files>
    <console><![CDATA[df$b ~ df$a]]>
    </console>
  </response>
</deployr>
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/script/execute",
            "robjects": {

            },
```

```
            "repository": {

            },
            "files": {
                "unnamedplot001.png": {
                    "value": "http://[server]/deployr/r/file/get/FILE-
    3d850e2c-864a-44f2-8188-811f31110962/unnamedplot001.png"
                },
                "unnamedplot002.png": {
                    "value": "http://[server]/deployr/r/file/get/FILE-
    9f17eff9-eef0-4688-8d12-b50c8e8f5681/unnamedplot002.png"
                },
                "unnamedplot003.png": {
                    "value": "http://[server]/deployr/r/file/get/FILE-
    a1482819-0669-4556-aa59-f4532c209e70/unnamedplot003.png"
                },
                "unnamedplot004.png": {
                    "value": "http://[server]/deployr/r/file/get/FILE-
    d10e0bcd-7521-460a-8337-c8fecfa55130/unnamedplot004.png"
                }
            },
            "console": {
                "value": "df$b ~ df$a"
            }
        }
     }
}
```

8.2 Execute Script in a Stateful Context

Executes the specified R script in a specified R session.

As executing a script in a stateful context is an operation on a live R session see section 3.9 for details.

8.3 List Scripts

Lists all of the R scripts available to the current user.

End Point

`/r/script/list`

HTTP Method

`GET`

Parameters

`format` – specifies the format for the output as either **`JSON`** or **`XML`**

Examples

```
GET /r/script/list
format=json
```

Return (XML):

```
<deployr>
  <response success="true" call="/r/script/list">
    <scripts>
      <script name="Simple Assignment Script" value="SCRPT-9d9943c6-6b03-
   435d-8bca-1cf884baa566"/>
      <script name="stockplot-png" value="SCRPT-d9fd3853-5d40-4ecd-8987-
   9b3e96bb4a9d"/>
    </scripts>
  </response>
</deployr>
```

Return (json):

```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/script/list",
            "scripts": {
                "Simple Assignment Script": {
                    "value": "SCRPT-9d9943c6-6b03-435d-8bca-1cf884baa566"
                },
                "stockplot-png": {
                    "value": "SCRPT-d9fd3853-5d40-4ecd-8987-9b3e96bb4a9d"
                }
            }
        }
    }
}
```

8.4 Execution Flow

The basic process by which an R script call begins when the client application requests an R script execution and passes the appropriate parameters to the one of the API calls, using either:

- stateless - **`/r/execute/script`**
- stateful - **`/r/session/execute/script`**

Then, it depends on whether a new R session is needed or already exists, as follows:

1. If a new R session being created for stateless execution (**`/r/execute/script`**)
   a. The R session is created.
   b. If needed, a predefined workspace image is loaded from the repository based on a previously saved id. The id can either be `local` to the web session, where a temporary workspace for the specific client/web session is referenced, or `global` to the application, where every R session for this application gets the same workspace.
   c. The R script for this application is loaded, the specified input objects are created in the R session, and the script is finally executed.
   d. The defined output objects as well as any warning or errors are retrieved from the session and formatted for return.
   e. The R session is destroyed.
   f. The output is returned to the client application.
2. If an R session already exists for the user, it can be used to invoke the R Script in a stateful context (**`/r/session/execute/script`**)
   a. The R script for this application is loaded
   b. The specified input objects are created in the R session.
   c. The R script is then executed.
   d. The defined output objects as well as any warning or errors are retrieved from the session and formatted for return.
   e. The output is returned to the client.

8.5 Example

Scenario

The Web client application allows a user to upload a file, run some analyses, and finally produce a plot.

Step 1: Logging In

In order to upload a file to the server, the user must be logged in. Therefore, the client calls **`/r/user/login`** on server and logs in as *testuser*.

```
POST /r/user/login
format=json
username=testuser
password=c51b5139556f939768f770dab8e5277a
autosave=false
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "cookie": "7B55A60B55093E7B5F320285490DBA75",
            "success": true,
            "call": "/r/user/login"
        }
    }
}
```

Step 2: Uploading

The client calls **`/r/file/upload`** on server. A URL to the uploaded file is returned.

```
POST /r/file/upload
name=satscores.csv
format=json
file=... spreadsheet contents here ...
```

Return (JSON) :

```
{
    "deployr": {
        "response": {
            "files": {
                "satscores.csv": {
                    "value":
   "http://[server]/deployr/r/repositoryget/binary/CFILE-e639b6c7-3a7a-
   4706-8c86-9757bee8fa9f"
                }
            },
            "success": true,
            "call": "/r/file/upload"
        }
    }
```

Step 3: Executing

The client calls **`/r/script/execute`** and passes the following data:

```
POST /r/script/execute
format=json
rscript=Sample Analysis Script
inputs={
    "myFile": {
        "value": "http://[server]/deployr/r/repository/get/binary/CFILE-
   e639b6c7-3a7a-4706-8c86-9757bee8fa9f"
    },
    "dependent": {
        "value": "VerbalSAT"
    },
    "independent": {
        "value": "MathSAT"
    }
}
```

Rscript

This is the server-deployed R script that will be executed.

```
##The inputs to this R script are "myFile=URL to csv file
##dependent=name of column in csv file that will be the dependent variable
   in the regression
##independent=name of column in csv file that will the independent variable
   in the regression

df<-read.csv(myFile)
myformula<-formula(paste("df$",dependent,"~","df$",independent))
print(myformula)
lm1<-lm(myformula)
plot(lm1)
```

Return (JSON) :

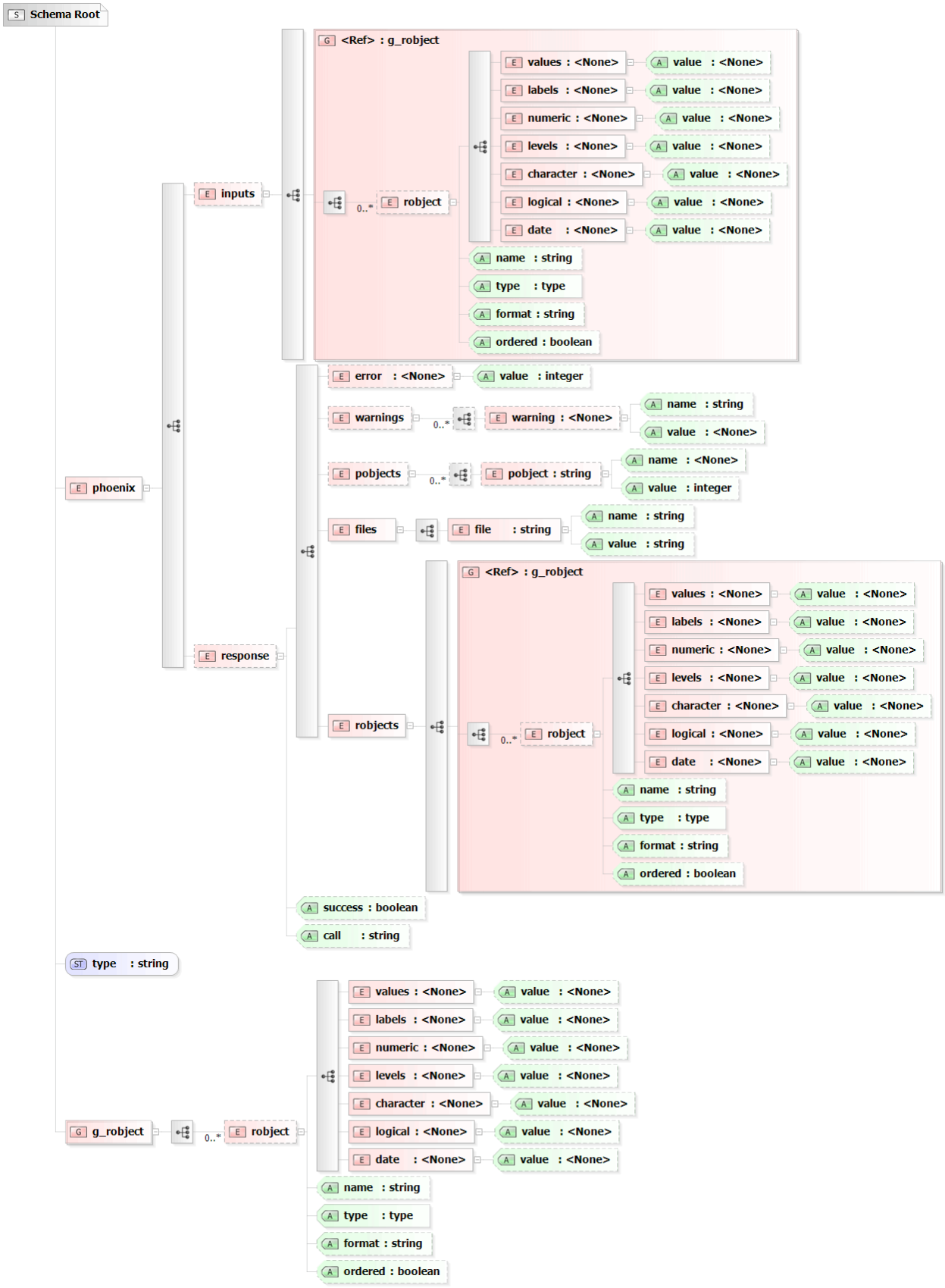
```
{
    "deployr": {
        "response": {
            "success": true,
            "call": "/r/session/execute/script",
            "robjects": {

            },
            "repository": {

            },
            "files": {
                "unnamedplot001.png": {
                    "value": "http://[server]/deployr/r/file/get/FILE-
   3d850e2c-864a-44f2-8188-811f31110962/unnamedplot001.png"
                },
                "unnamedplot002.png": {
```

```
                    "value": "http://[server]/deployr/r/file/get/FILE-
    9f17eff9-eef0-4688-8d12-b50c8e8f5681/unnamedplot002.png"
                },
                "unnamedplot003.png": {
                    "value": "http://[server]/deployr/r/file/get/FILE-
    a1482819-0669-4556-aa59-f4532c209e70/unnamedplot003.png"
                },
                "unnamedplot004.png": {
                    "value": "http://[server]/deployr/r/file/get/FILE-
    d10e0bcd-7521-460a-8337-c8fecfa55130/unnamedplot004.png"
                }
            },
            "console": {
                "value": "df$VerbalSAT ~ df$MathSAT"
            },
            "session": "LIVE-e0e67988-3403-4c4d-8200-75a0ec18e242"
        }
    }
}
```

Appendix A: XML Schema



Appendix B: RevoDeployR Type Examples for JSON


```
{
    "deployr": {
        "response": {
            "success": "true",
            "robjects": {
                "v1": {
                    "type": "vector",
                    "value": [
                        1.0,
                        2.0
                    ]
                },
                "v2": {
                    "type": "vector",
                    "value": [
                        "age",
                        "gender",
                        "IQ"
                    ]
                },
                "x": {
                    "type": "primitive",
                    "value": "3.0"
                },
                "bflag": {
                    "type": "primitive",
                    "value": true
                },
                "name": {
                    "type": "character",
                    "value": "Here is my name"
                },
                "myDate": {
                    "type": "date",
                    "format": "%Y-%m-%d",
                    "value": "2010-04-20"
                },
                "myGender": {
                    "type": "factor",
                    "value": [
                        "male",
                        "female",
                        "male",
                        "female"
                    ]
                },
                "myScore": {
                    "type": "factor",
```

```
                    "ordered": true,
                    "levels": [
                        "low",
                        "medium",
                        "high"
                    ],
                    "value": [
                        "low",
                        "high",
                        "medium",
                        "low",
                        "high",
                        "high"
                    ]
                },
                "myMatrix": {
                    "type": "matrix",
                    "value": [
                        [
                            1,
                            2,
                            3
                        ],
                        [
                            11,
                            12,
                            13
                        ]
                    ]
                },
                "myDataFrame" : {
                    "type": "dataframe",
                    "value": {
                        "age": {
                            "type": "vector",
                            "value": [
                                13,
                                15,
                                16
                            ]
                        },
                        "gender": {
                            "type": "vector",
                            "value": [
                                "female",
                                "male",
                                "female"
                            ]
                        },
                        "IQ": {
                            "type": "vector",
                            "value": [
                                94,
                                114,
                                127
```

```
                            ]
                        }
                    }
                },
                "myList": {
                    "type": "list",
                    "value": {
                        "abc": {
                            "type": "primitive",
                            "value": "1.0"
                        },
                        "anotherDataFrame": {
                            "type": "dataframe",
                            "value": {
                                "x": {
                                    "type": "vector",
                                    "value": [
                                        0,
                                        10,
                                        20
                                    ]
                                },
                                "y": {
                                    "type": "vector",
                                    "value": [
                                        "zero",
                                        "ten",
                                        "twenty"
                                    ]
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```