Dirigible Help

Please, refer to http://help.dirigible.io for the up-to-date documentation.

Dirigible Help Portal bundles access to product documentation and also related information. Browse Dirigible Help Portal to get up-to-date information about the features, life-cycle aspects, tip-and-tricks and many more...

Project

Dirigible is an open source project that provides Integrated Development Environment as a Service (IDEaaS) as well as the runtime containers integration for the running applications. The applications created with Dirigible comply with the Dynamic Applications concepts and structure.

The main goal of the project is to provide all the required capabilities needed to develop and run an end-to-end, and yet meaningful, vertical scenario in the cloud for shortest time ever.

The environment itself runs directly in the browser, therefore does not require additional downloads and installations. It packs all the needed containers, which makes it self-contained and well integrated software bundle that can be deployed on any Java based Web Server such as Tomcat, Jetty, JBoss, etc.

The Dirigible project came out of an internal SAP initiative to address the extension and adaptation use-cases around SOA and Enterprise Services. On one hand in this project were implied the lessons learned from the standard tools and approaches so far and on the other hand, there were added features aligned with the most recent technologies and architectural patterns related to Web 2.0 and HTML5, which made it complete enough to be used as the only tool and environment needed for building and running on demand application in the cloud.

From the beginnig the project follows the principles of Simplicity, Openness, Agility, Completeness and Perfection which provides a sustainable environment where with minimal effort is achieved maximum impact.

Features section describes in detial what is included in the project. Concepts section gives you an overview about the internal and the chosen patterns. Samples shows you how to start and build your first dynamic application in seconds.

Dynamic Applications

We introduced the term **Dynamic Applications** as one that narrows the scope of the target applications, which can be created using **Dirigible**.
The overall process of building Dynamic Applications lies on well-known and proved principles:

- In-System Development - known from microcontrollers to business software systems. Major benefit is working on a live system where all the changes made by a developer take effect immediately, hence the impact and side-effects can be realized in very early stage of the development process.
- **Content Centric** - known from networking to development processes in context of Dynamic Applications it comprises all the artifacts are text-based models or executable scripts stored on a generic repository (along with the related binaries such as images). This makes the life-cycle management of the application itself as well as the transport between the landscapes (Dev/Test/Prod) straight forward.

In addition, desired effect is the ability to setup the whole system, only by pulling the content from a remote source code repository, such as git.

- Scripting Languages - programming languages written for a special run-time environment that can interpret (rather than compile) the execution of tasks. Dynamic languages existing nowadays as well as the existing smooth integration in the web servers make possible the rise of the in-system development in the cloud.
- **Shortest turn-around time** - instant access and instant value became the most important requirement for the developers, that's why it is also the driving principle for our tooling.

In general, a Dynamic Application consists of components, which can be separated to the following categories:

- **Data Structures**
  These are the artifacts representing the domain model of the application. In our case, we have choosen the well accepted JSON format for describing the normalized entity model. There is no intermediate adaptation layer in this case, hence all the entities represent directly the database artifacts - tables and views.
- **Entity Services**
  Once we have the domain model entities, next step is to expose them as web services. Following the modern web patterns we provide the scripting capabilities, so you can create your RESTful services in JavaScript, Ruby and Groovy.
- **Scripting Services**
  During the development you can use rich set of APIs which give you access to the database and HTTP layer, utilities as well as direct Java APIs underneath.
  Support for creating unit tests is important and it is integrated as atomic part of the scripting support itself - you can use the same language for the tests, which you are using for the services themselves.
- **User Interface**
  Web 2.0 paradigm as well as HTML5 specification bring the web user interfaces on another level. There are already many cool client side ajax frameworks, which you can be used depending on the nature of your application.
- **Integration Services**
  Following the principle of atomicity, one Dynamic Application should be as self-contained as possible. Unfortunately, in the real world there are always some external services that have to be integrated to your application - for data transfer, triggering external processes, lookup in external sources, etc.
  For this purpose we provide capabilities to create simple routing services and dynamic EIPs.
- **Documentation**
  The documentation is integral part of your application. The target format for describing services and overall development documentation is already well accepted - wiki.

Architecture

Dirigible architecture follows the well proved principles of simplicity and scalability in the classical service-oriented architecture.

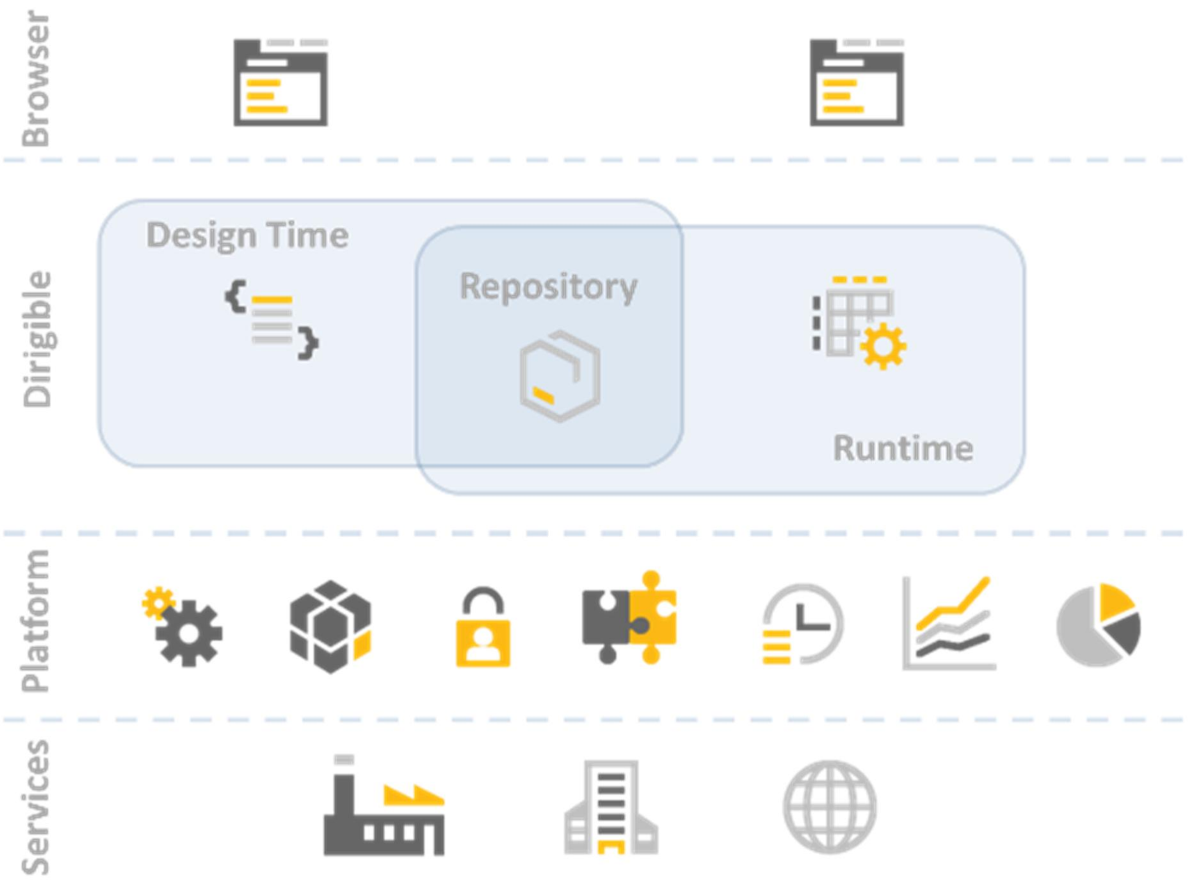
There is separation of components for design time (definition work, modeling, scripting) and the runtime (execution of services, content provisioning and monitoring).
The transition between the design time and runtime is via repository component and the only linking part is the content itself.

At design time the programmers and designers use the Web-based integrated development environment. This tooling is mainly based on the Remote Application Platform (RAP) from Eclipse. Using this robust and powerful framework the tooling itself can be easily enhanced by using well known APIs and concepts - SWT, JFaces, OSGi, extension points, etc.

The Repository is the container of the project artifacts. It is a generic file-system like content repository on relation database.

After the creation of the cloud application, it is provided by the runtime components. The main part is Java Web Profile compliant application server. On top are the Dirigible's containers for services execution depending on the scripting language and purpose - Rhino, jRuby, Groovy, Camel, CXF, etc. The runtime can scale independently by the design time part, and even can be deployed without design time at all (for productive landscapes)

Depending on the target cloud platfrom, Dirigible can be integrated to use the services provided by the underlying platfrom.



API

There are several predefined injected objects, which can be used directly from the script code:

- **context** - a standard Map that can be used as a context holder during the execution
- **out** - the standard System output, which is redirected to the trace file
- **datasource** - the default JDBC Datasource configured at server instance level
- **db** - an utillity object with methods:

- int createSequence(String sequenceName, int start)
- int getNext(String sequenceName)
- int dropSequence(String sequenceName)
- boolean existSequence(String sequenceName)
- String createLimitAndOffset(String limit, String offset) - used for paging
- String createTopAndStart(int limit, int offset) - used for paging

- **request** - the standard HttpServletRequest object
- **response** - the standard HttpServletResponse object
- **user** - the name of the current logged in user
- **repository** - the reference to Dirigible Content Repository
- **io** - Apache Commons IOUtils
- **http** - Apache Commons Http Client wrapped utility object with methods:
    - HttpGet createGet(String strURL)
    - HttpPost createPost(String strURL)
    - HttpPut createPut(String strURL)
    - HttpDelete createDelete(String strURL)
    - DefaultHttpClient createHttpClient()
    - void consume(HttpEntity entity)
- **base64** - Apache Commons Codecs Base64
- **hex** - Apache Commons Codecs Hex
- **digest** - Apache Commons Codecs Digest
- **xss** - Apache Commons StringEscapeUtils
- **upload** - Apache Commons File Upload Servlet
- **url** - URLEncode and URLDecoder wrapped in an object with methods:
    - void encode(String s, String enc) and
    - void decode(String s, String enc)
- **uuid** - Universally Unique Identifier (UUID) 128-bit generator
- **input** - the object representing a Message body in case of Route step

Features

- **Data Structures**
    - Creation of **Table Model** (JSON formatted *.table descriptor) and actual creation of the corresponding database table during activation.
    - Creation of **View Model** (JSON formatted *.view descriptor) and actual creation of the corresponding database view during activation.
    - Creation of Delimiter Separated Values **data files** (*.dsv) and populating the corresponding database table during activation.
    - **Importing of data** files (*.dsv) on the fly as direct update to corresponding table.
    - **Automatic altering** of existing tables from the models on compatible changes (adding new columns)
- **Scripting Services**
    - Support of **JavaScript** language by using Mozilla Rhino as runtime container (*.js)
    - Support of **CommonJS** based **modularization** of JavaScript services (*.jslib)
    - Support of **Ruby** language by using jRuby as runtime container along with the standard for the language modularization
    - Support of **Groovy** language by using Groovy as runtime container along with the standard for the language modularization
    - Support of predefined API as **injected global objects** such as request, response, datasource, httpclient, repository, etc. for all supported languages
- **WebContent**
    - Support of **client side web** related artifacts such as html, css, js, pictures
- **WikiContent**
    - Support of **Confluence** format of Wiki pages
    - Support of customizable **header, footer and css** for wiki pages
- **Integration Services**
    - Support of dynamic routes by using Apache **Camel**
    - Support of **JavaScript** breakouts in routes
- **Tooling**
    - **Workspace** perspective for full support of project management (new, cut, copy, paste, delete, refresh, etc.)
    - **Database** perspective for RDBMS management including SQL Console
    - Enhanced **Code Editor** with highlight support for JavaScript, Ruby, Groovy, HTML, JSON, XML, etc.
    - **Web Viewer** for easy testing of changes in web, wiki and scripting services
    - Configurable **Log Viewer** providing the server side logs and traces
    - Lots of **template based wizards** for creating new content and services
    - **Import** and **Export** of project(s) content
    - **Import of binary** files for external documents and pictures
    - **Repository** perspective for low level repository content management
- **Security**

    - **Role based** access management
    - **Security Constraints Model** (JSON formatted *.access) support
    - Few predefined roles which can be used out-of-the-box Everyone, Administrator, Manager, PowerUser, User, ReadWrite, ReadOnly
- **Registry**
    - **Activation** support - exposing the artifacts from the user's workspace publicly
    - **Auto-Activation** support for usability
    - User-Interface for **Browsing and Searching** of the activated content
    - Separate **lists of endpoints** and viewers per type of services - JavaScript, Ruby, Groovy, Routes
    - Separate browse user interface for **web and wiki** content

and more...

Data Structures

Overview

Data Structures term in the context of the cloud toolkit is used for refering the Domain Model of the application. For pragmatic reasons it is chosen that the actual entities in the domain model to correspond 1:1 to the underlying database entities - tables and views. There is no additional abstract layer between your application code and the actual model in target storage.

Tables

Table Model is a JSON formatted *.table descriptor which represents the layout of the database table which will be created during activation process.

Example descriptor:

```
{ "tableName":"TEST001", "columns": [ { "name":"ID", "type":"INTEGER", "length":"0", "notNull":"true", "primaryKey":"true", "defaultValue":"" } , { "name":"NAME", "type":"VARCHAR", "length":"20", "notNull":"false", "primaryKey":"false", "defaultValue":"" } , { "name":"DATEOFBIRTH", "type":"DATE", "length":"0", "notNull":"false", "primaryKey":"false", "defaultValue":"" } , { "name":"SALARY", "type":"DOUBLE", "length":"0", "notNull":"false", "primaryKey":"false", "defaultValue":"" } ] }
```

The supported database types are:

- **VARCHAR** - for text based fields up to 2K characters
- **CHAR** - for text based fields with fixed lenght up to 255 characters
- **INTEGER** - 32 bit
- **BIGINT** - 64 bit

- **SMALLINT** - 16 bit
- **REAL** - 7 digits of mantissa
- **DOUBLE** - 15 digits of mantissa
- **DATE** - represents a date consisting of day, month, and year
- **TIME** - represents a time consisting of hours, minutes, and seconds
- **TIMESTAMP** - represents DATE plus TIME plus a nanosecond field and time zone
- **BLOB** - binary object - images, audio, etc.

Activation of table descriptor is a process of creation of the database table in the target database. The activator constructs a "CREATE TBALE" SQL statement considering the dialect of the target database system.
If the table with the given name already exists, the activator checks whether there is a compatible change (adding of new columns) and construct "ALTER TABLE" SQL statement.
If the there is an incompatible change, then the activator returns an error which have to be solved manually via the SQL Console.

Views

View Model is also JSON formatted *.view descriptor of a database view. Usually it is a join between multiple tables used for reporting purposes.
The script should follow the SQL92 standard, or have to be aligned with the dialect of the target database.

Data Files

Delimiter Separated Values **data files** *.dsv are used for importing some test data during the development or for defining a static content for some nomenclatures for instance.
The convension is that the name of the data file should be the same as the name of the target table.
The delimiter is the char "|" and the order of the data fields should be the same as the natural order of the target table.
If you want to import some data only once, this can be done via the Import Data Wizard.

> Be careful when using the static data in tables. Entity Services (generated by the templates) are using sequence algorithm for identity column starting from 1.

The automatic reinitialization of the static content from the data file can be achieved when you create a *.dsv file under the DataStructures folder of the given project.

Scripting Services

JavaScript

Services

Primary language used to implement services in Dirigible is JavaScript. Being quite popular as a client-side scripting, it became also prefered language for server-side business logic as well.
For the underlying execution engine is used the most mature JavaScript engine written in Java - Rhino by Mozilla.
You can write your algorithms in *.js files and store them within the ScriptingServices folder. After the Activation or Publishing they can be executed by accessing the endpoint respectively at the sandbox or public registry.

An example JavaScript service looks like this:

```
var systemLib = require('system'); var count; var connection = datasource.getConnection();
try { var statement = connection.createStatement(); var rs = statement.executeQuery('SELECT
 COUNT(*) FROM BOOKS'); while (rs.next()) { count = rs.getInt(1); } systemLib.println('coun
t: ' + count); } finally { connection.close(); } response.getWriter().println(count); respo
nse.getWriter().flush(); response.getWriter().close();
```

This example shows two major benefits:

1. Modularization based on built-in CommonJS ('require' function on the first line)
2. Native usage of the Java objects as API injected in the execution context (database, response)

Libraries (Modules)

You can create your own library modules in *.jslib files. Just do not forget to add the public parts in the **exports**.

```
exports.generateGuid = function() { var guid = uuid.randomUUID(); return guid; };
```

The libraries are not directly exposed as services, hence they do not have accessible endpoints in the registry.

The reference of the library module from the service is done by using the standard function **require()** where the parameter is the location of the module constructed as follows:
<project_name>/<module_path>
Module path includes the full path to the module in the project structure without the predefined folder ScriptingServices and also without the extension *.jslib.

```
/sample_project /ScriptingServices /service1.js /library1.jslib library1.jslib is refered i
n service1.js: ... var library1 = require('sample_project/library1'); ...
```

Relative paths ('.', '..') are not supported. The project name must be explicitly defined.

Ruby

Language which also expanding its popularity in web development scenarios last years is Ruby.
You can use also the standard modularization provided by the language as well as the injected context objects in the same way as in JavaScript.
The execution engine used as runtime container is jRuby

Example service which has reference to a module can be generated from the Scripting Services wizard directly:

Service (sample.rb):

```
require "/sample_project/module1" Module1.helloworld("Jim")
```

and Module (module1.rb):

```
module Module1 def self.helloworld(name) puts "Hello, #{name}" $response.getWriter().printl
n("Hello World!") end end
```

Note that in Ruby you have to put a dollar sign ('$') in the beginning of the API objects ($response) as they are global objects

Groovy

Groovy is yet another powerful language for web development nowadays with its static types, OOP abilities and many more.

Corresponding examples in Groovy:

Service (sample.groovy):

```
import sample_project.module1; def object = new Module1(); object.hello(response);
```

Module (module1.groovy):

```
class Module1{ void hello(def response){ response.getWriter().println("Hello from Module1")
 } }
```

Integration Services

Overview

Integration Services are the connection points between the application logic and the external services - 3-thd party cloud services, On-Premise services, public services, etc. There are support for inbound as well as the outbound services - consumption and provisioning. By utilizing one of the most mature and well known framework as underlying technology - Apache Camel, there are lots of ready-to-use integration patterns.

Routes

The term **Route** is directly taken from the Apache Camel's context and refers to **definition of routing rules**. The extension of the definition file is "*.routes"

Sample route descriptor looks like this:

```
<routes xmlns="http://camel.apache.org/schema/spring"> <route id="simple_routing"> <from ur
i="servlet:///simple_routing_endpoint" /> <choice> <when> <header>name_parameter</header> <
transform> <simple>Hello ${header.name_parameter} how are you?</simple> </transform> </whe
n> <otherwise> <transform> <constant>Add a name parameter to uri, eg ?name_parameter=foo</c
onstant> </transform> </otherwise> </choice> </route> </routes>
```

The original source is here

Once you activate or publish the project the route descriptor is read by the runtime agent and the route is enabled in the container.
The endpoint of such an integration service is exposed by the Camel container at the location constructed by the following pattern:

> The pattern for the endpoint location of routes:
>
> **http //<host>:<port>/dirigible/camel/<servlet name - (from uri="servlet:///XXX")>**
>
> e.g.
>
> **http //<host>:<port>/dirigible/camel/simple_routing_endpoint**

More information about the supported integration patterns can be found at the samples portal.

Test Cases

Overview

Following the best practices unit tests are always integral part of the application code itself. For this reason in the cloud toolkit there is a predefined place for them and kind of deep integration with the scripting services.
The language supported for test cases is the same as of the target scripting service you want to tests. Technically the unit tests are not different than the services, hence the separation in this regard is only semantical. You can code in the test cases everything as you can do also in scripting services. All the API objects and context as well as usage of libraries are supported transparently.

Writing a Test Case

You start by using the action from the pop-up menu while selecting a project. Choose the New->Test Case and select from the list of the predefined templates.
You can use the following APIs which can help you to write more standardized yet comprehensible unit tests:

- **assertTrue**
  - message - the error message
  - condition - the condition usually containing the inspecting values
- **assertFalse**
  - message - the error message
  - condition - the condition usually containing the inspecting values
- **assertEquals**
  - message - the error message
  - o1 - the expecting object
  - o2 - the actual object
- **assertNull**
  - message - the error message
  - o - the inspecting object
- **assertNotNull**
  - message - the error message
  - o - the inspecting object
- **fail**
  - message - the error message

Some sample code should look like:

```
var assert = require('assert'); ... assert.assertNotNull('value is null', value); ...
```

For more comprehensive sample you can refer to sample test case.

Web Content

Overview

Web Content includes all the static client-side resources such as HTML files, CSS and related theming ingredients as well as dynamic scripts (e.g. JavaScript) and images.
In general, the web content adapter is playing a role of a tunnel, which takes the desired resource location from the request path, loads the corresponding content from the repository and send it back without any modification.

The default behavior of the adapter on a request to a collection (instead of particular resource) is to send back an error code to indicate that the listing of folders is forbidden.

> If the specific "application/json" **Accept** header is supplied with the request itself, then a JSON formatted array with the sub-folders and resources will be returned.

Templating

Common pattern in user interfaces of web based business applications is simplified templating - usually static header and footer.
To support this we introduced a special handling of HTML pages:

- **header.html** is a special page, which is recognized as a static header, so that, if exists, it is rendered in the beginning of a requested regular page
- **footer.html** is a special page, which is recognized as a static footer, so that, if exists, it is rendered in the end of a requested regular page
- **index.html** is a special page, which is recognized as a welcome page, so that no header and footer are added to it
- **nohf** is a parameter, which can be added to the request URL to disable adding of header and footer

To boost the developer productivity in the most common cases, we provide a set of templates, which can help in user interface creation.
There are set of templates, which can be used with entity services - list of entities, master-detail, input form, etc. More information can be found at samples area.
The other templates can be used as utilities e.g. for creation application shell in index.html with main menu or as samples showing most common controls on different AJAX user interface frameworks - jQuery, Bootstrap, OpenUI5.

Wiki Content

Overview

An integral part of every application is the user documentation. For this purpose we introduced a special type of artifacts which are placed in a predefined sub-folder of a project. This type of artifacts follows the de-facto standard nowadays format for documenting behaviors and algorithms of applications as well as general information about the program itself - wiki. The supported markup language as of now is confluence - well accepted by the community.

The wiki pages have to be placed under WikiContent folder of a project with *.wiki file extension. Once they are requested by GET request, underground transformation has been triggered which convert the confluence format to HTML and send the well formed web content back.

Sample of a wiki page in confluence format looks like as following:

```
h4. Confluence Markup Ideally, the markup should be _readable_ and even *clearly understand
able* when you are editing it. Inserting formatting should require few keystrokes, and litt
le thought. After all, we want people to be concentrating on the words, not on where the an
gle-brackets should go. * Kinds of Markup ** Text Effects ** Headings ** Text Breaks ** Lin
ks ** Other
```

and after the rendering you will get:

Confluence Markup

Ideally, the markup should be *readable* and even **clearly understandable** when you are editing it. Inserting formatting should require few keystrokes, and little thought.

After all, we want people to be concentrating on the words, not on where the angle-brackets should go.

- Kinds of Markup
    - Text Effects
    - Headings
    - Text Breaks
    - Links
    - Other

Templating

Simple templating is also supported similar to web content:

- **header.html** is a special page, which is recognized as a static header, so that, if exists, it is rendered in the beginning of a requested regular page
- **footer.html** is a special page, which is recognized as a static footer, so that, if exists, it is rendered in the end of a requested regular page
- **nohf** is a parameter, which can be added to the request URL to disable adding of header and footer

Sample Pages

Sample header and footer as well as navigation page could look like:

- header.html

```
<!DOCTYPE html> <html lang="en"> <head> <meta charset="utf-8"> <meta http-equiv="X-UA-Compa
tible" content="IE=edge"> <meta name="viewport" content="width=device-width, initial-scale=
1.0"> <meta name="description" content=""> <meta name="author" content=""> <title>YOUR TITL
E HERE</title> <!-- Bootstrap core CSS --> <link href="http://netdna.bootstrapcdn.com/boots
trap/3.0.3/css/bootstrap.min.css" rel="stylesheet"> <!-- Just for debugging purposes. Don't
 actually copy this line! --> <!--[if lt IE 9]><script src="../../docs-assets/js/ie8-respon
sive-file-warning.js"></script><![endif]--> <!-- HTML5 shim and Respond.js IE8 support of H
TML5 elements and media queries --> <!--[if lt IE 9]> <script src="https://oss.maxcdn.com/l
ibs/html5shiv/3.7.0/html5shiv.js"></script> <script src="https://oss.maxcdn.com/libs/respon
d.js/1.3.0/respond.min.js"></script> <![endif]--> <link href="wiki.css" rel="stylesheet"> <
/head> <body> <div id="container"> <div id="header">YOUR HEADER TEXT HERE</div> <div id="wr
apper"> <div id="content">
```

- footer.html

```
</div> <div id="navigation"> </div> <script src="https://code.jquery.com/jquery.min.js"></s
cript> <script> $( document ).ready(function() { $( "#navigation" ).load('navigation.wiki?n
ohf'); }); </script> <div id="footer"><p>Copyright &copy; YOUR COPYRIGHT AND LICENSE HERE</
p></div> </div> </body> </html>
```

- navigation.wiki

```
* [Home|index.wiki] * [Project|project.wiki] ...
```

and of course some custom css for the wiki content

- wiki.css

```
html, body { margin:0; padding:0; height: 100%; } body { color: gray; } p { margin:0 10px 1
0px; } a { color: #555; padding:2px; } pre { margin: 20px; background: #e3ffcf; } #header
{ position: fixed; width: 100%; background: black; border-bottom: 4px solid #f0ab00; box-sh
adow: 2px 2px 5px rgba(116, 107, 61, 0.4); color: #E6AB19; font-size: 120%; text-shadow: 1p
x 1px 2px rgba(116, 107, 61, 0.74); } #extra { background: #f0ab00; position: fixed; right:
 0; top: 160px; width: 200px; height: 100px; border-radius: 5px; border-color: #f0ab00; bor
der-width: 3px; border-style: solid; padding: 10px; margin: 10px; } #extra a { font-weight:
 bold; text-decoration: none; } #wrapper { float:left; width:100%; } #content { float: lef
t; margin-left: 280px; margin-top: 30px; margin-right: 250px; margin-bottom: 30px; display:
 inline; padding: 10px; line-height: 20px; } #content a { color: #f0ab00; padding:0px; } #n
avigation { position: fixed; top: 38px; width: 250px; border-right: solid 1px #ddd; height:
 100%; padding: 10px; overflow:auto; background-color: #efefef; color: black; } #footer { p
osition: fixed; bottom: 0; width: 100%; background: black; color: lightgray; text-align: ce
nter; height: 20px; } #footer p { margin:0; padding: 0px 10px; font-size: 90%; } #footer a
{ color: white; display: initial; } .info { border-radius: 4px; padding: 20px; } .warning
{ border-radius: 4px; padding: 20px; } .error { border-radius: 4px; padding: 20px; }
```

Batch of Wiki Pages

Sometimes it is helpful to combine several already existing pages to a single page.
For this purpose you have to create a file with extension *.wikis and to list in it all the wiki pages that you want to merge.

File: **single.wikis**

```
part1.wiki part2.wiki
```

Security

Security is quite broad topic, so that here will be described mostly the authentication and autorization concept of dynamic applications.

Being a standard Java web application, the Dirigible Design-Time as well as Runtime components rely entirely on the underlying Java web container/server for the authentication process.
Usually it comes as well integrated JAAS service.

There are several predefined roles coming by default and can be used for dynamic applications:

- Administrator
- Manager
- PowerUser
- User
- ReadWrite
- ReadOnly
- Everyone

Have in mind that the above roles are shared for all the projects/dynamic applications within the account
More roles can be added only via custom build of the Dirigible's Runtime component.

As soon as the Roles definition are well standardized, User/Principals to Roles assignments are platform specific. For HANA Cloud Platfrom you can refer at https://help.hana.ondemand.com/help/SAP_HANA_Cloud.pdf section - 1.3.10.3.1

Once we have defined the Roles as well as the User-to-Roles assignments, it comes the definition of the protected resources. It is done by a simple JSON formatted *.access file under the SecurityConstraints project's sub-folder.
There is a wizard which generates the sample main.access, which looks like:

```
[ { "location":"/project1/secured", "roles": [ {"role":"User"}, {"role":"PowerUser"} ] },
{ "location":"/project1/confidential", "roles": [ {"role":"Administrator"} ] } ]
```

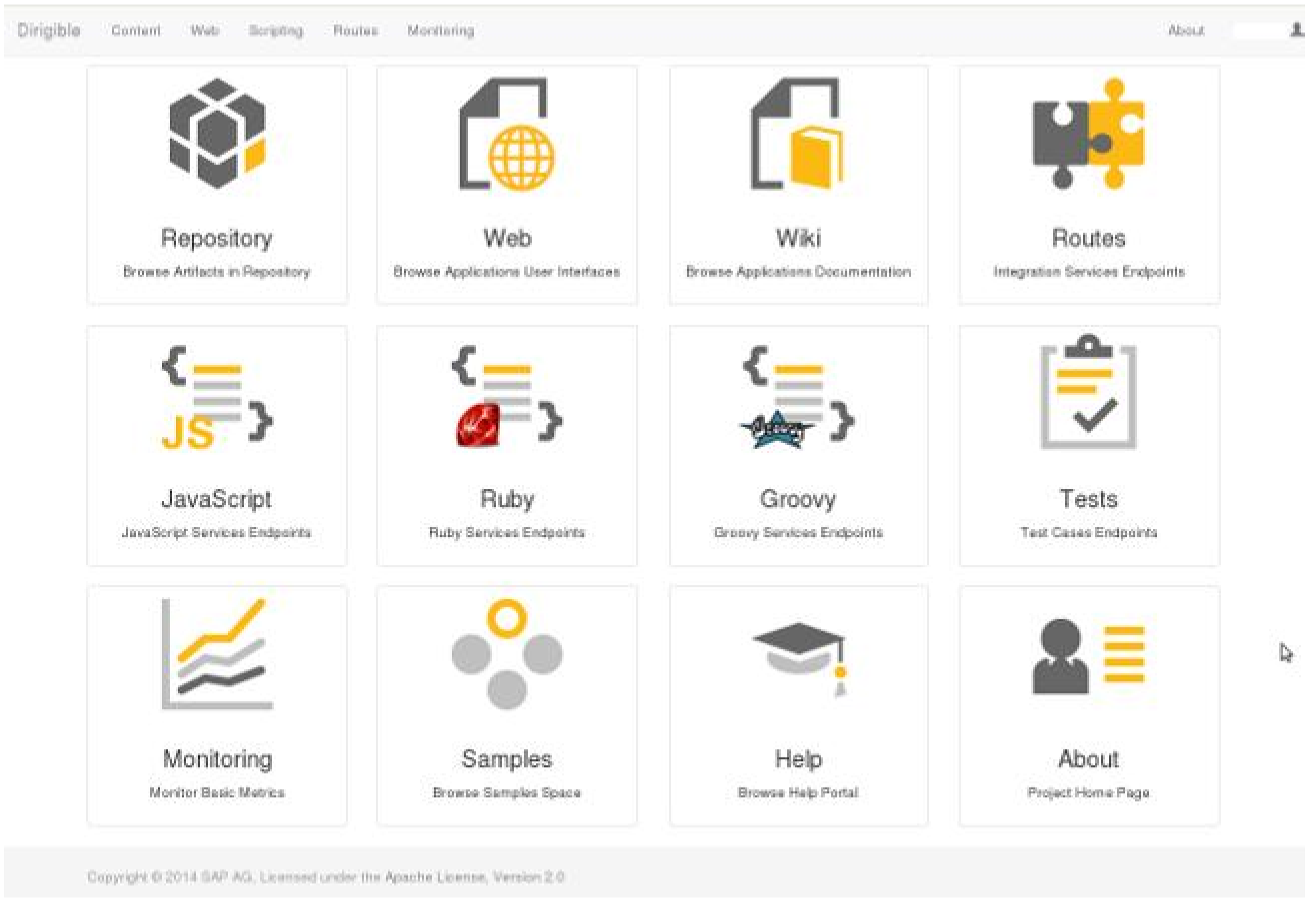
After the publishing of the project you can try to access the protected resources with users with different roles assignemnts. The impact of protection of the resources is on the web and wiki content and also on scripting services' endpoints.

The location attribute of the protected resource is transitive i.e. the most specific location wins in case of multiple definitions with equal roots

There is a Security Manager view, openned by default in Workspace perspective, where you can see the currently protected resources as well as to disable the protection.

Registry

The entry point of the searching and browsing of the service endpoints as well as monitoring and administration at runtime phase is the Registry. Technically it is a space within the Repository where all the published artifacts are placed.



Copyright © 2014 SAP AG, Licensed under the Apache License, Version 2.0

To access the user interface you can point to the runtime context - default one is "dirigible"

http //<host>:<port>/dirigible

Endpoints

From the index page of the Registry, you can navigate to the corresponding sub-pages for browsing the raw content of the Repository, published user interfaces (html, css, client-side javascript, etc.), documentation of the applications, lookup the endpoints of the scripting services as well as the endpoints of the integration services.

Monitoring Tools

The last phase of the applications lifecycle includes administration and monitoring.
Via the Registry interface you can navigate to the monitoring tools including:

- Hit Count Statistics
- Response Time Statistics
- Memory Allocations
- Log Traces

The URIs on which you want to collect information about the request parameters have to be registered in the Manage Access Locations section.

Git Integration

There is a Git connector for team development in the toolkit.
The goal is to provided the simplest way to synchronize the sources with the remote source control repository and to leave the more complex operation (e.g. merge) for external tools.

Available commands:

- **Clone** - clones remote Git repository to the toolkit as a project

Constraint: Remote Git repository must contain only one project.

- **Push** - tries to push changes to the project's remote repository. If the changes are not conflicting with what is in remote "origin/master" branch, then the push is successful. After that the project content is synchronized with the state in the remote "origin/master" branch. If there are conflicts with the newly made changes then new remote branch is created and changes are pushed in it. The remote branch's name is "changes_branch_{dirigible's username}", e.g. "changes_branch_user1234".

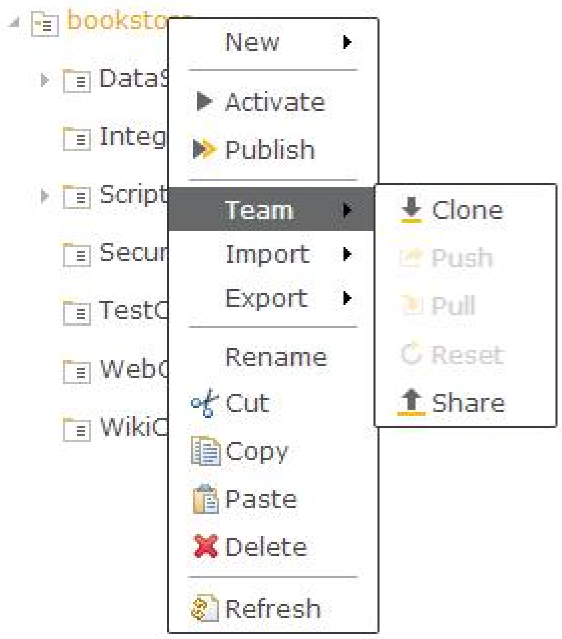
Constraint: Merging of conflicting branches should be done via an external tool e.g. "GitBash", "eGit", etc.

- **Pull** - checkouts the remote changes from the "origin/master", if there are conflicts then an error will be raised.

Constraint: If there are conflicting changes during Pull, then the recommendation is to backup the project as zip, then to Reset it and manually apply your changes again.

- **Reset** - sets the local project to be as the latest state of the remote "origin/master" branch.
- **Share** - shares the selected project to remote repository.

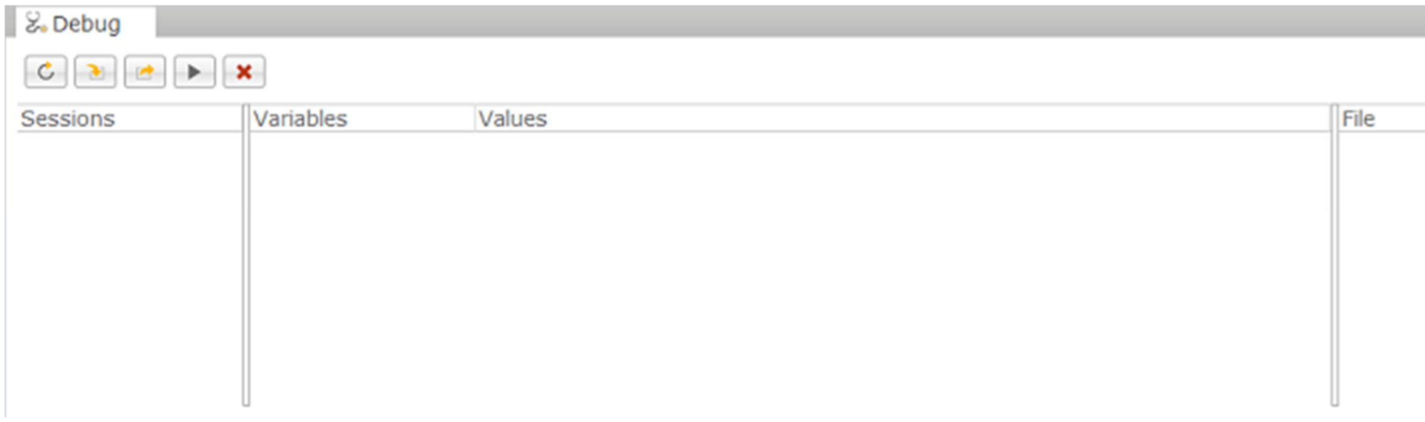
All the commands are accessible from the project's pop-up menu



Debug

The toolkit offers Debugging functionallity. The goal is to easy the developers in the hunt of server side bugs.

Debug Perspective



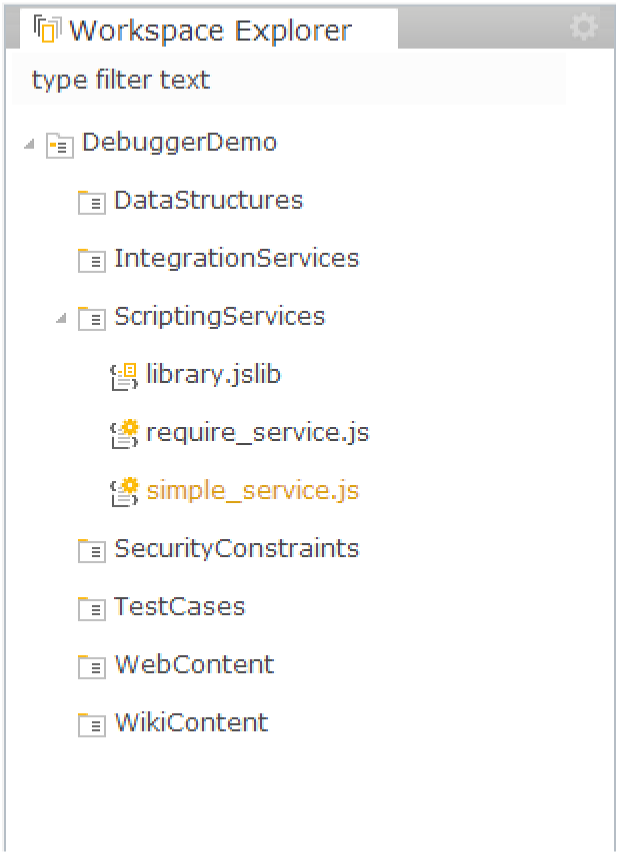
- **Sessions** - contains all debug execution sessions.
- **Variables/Values** - contains variables and their values, available in the current scope of execution.
- **File/Row/Source** - contains inforamation about in witch file and witch row, a **Breakpoint** is set.

Available Commands

- Refresh
- Step Into
- Step Over
- Continue
- Skip all breakpoints

Debugging Example

Step 0 - Project Structure



```
DebuggerDemo
    DataStructures
    IntegrationServices
    ScriptingServices
        library.jslib
        require_service.js
        simple_service.js
    SecurityConstraints
    TestCases
    WebContent
    WikiContent
```

simple_service.js

```
main(); function main(){ var message = createMessage(); var students = createStudents(); re
sponse.getWriter().println(message); response.getWriter().println(JSON.stringify(student
s)); } function createMessage(){ var initialValue = 1; var endValue = startCounter(initialV
alue); var message = 'Initial value was '+initialValue+', end value is '+endValue; return m
essage; } function startCounter(value) { for(var i = 0; i < 5; i++){ value ++; } return val
ue; } function createStudents(){ var students = []; students.push(createStudent('Desi', 1
8)); students.push(createStudent('Jordan', 21)); students.push(createStudent('Martin', 2
2)); return students; } function createStudent(name, age){ var student = {}; student.name =
 name; student.age = age; return student; }
```

library_jslib

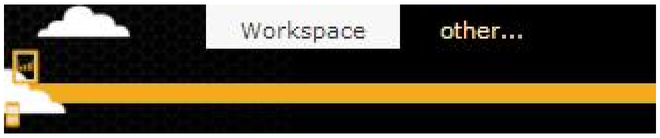
```
exports.generateGuid = function() { var guid = ''+uuid.randomUUID(); return guid; };
```

require_service.js

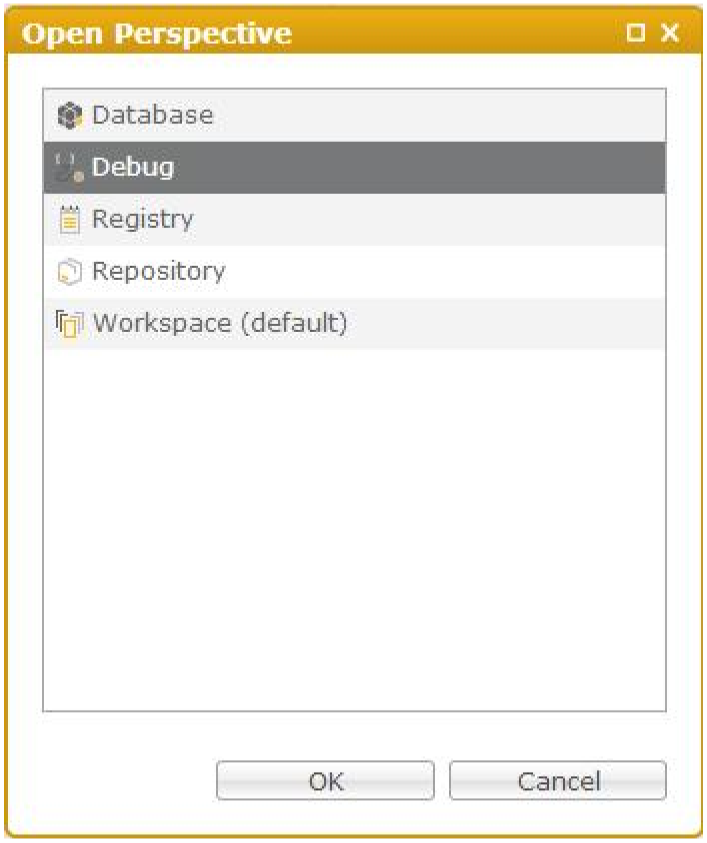
```
var guidGen = require('/DebuggerDemo/library'); var user = 'Test User'; var id = guidGen.ge
nerateGuid(); response.getWriter().println(user+", id "+id); response.getWriter().flush();
response.getWriter().close();
```

Step 1 - Open Debug Perspective

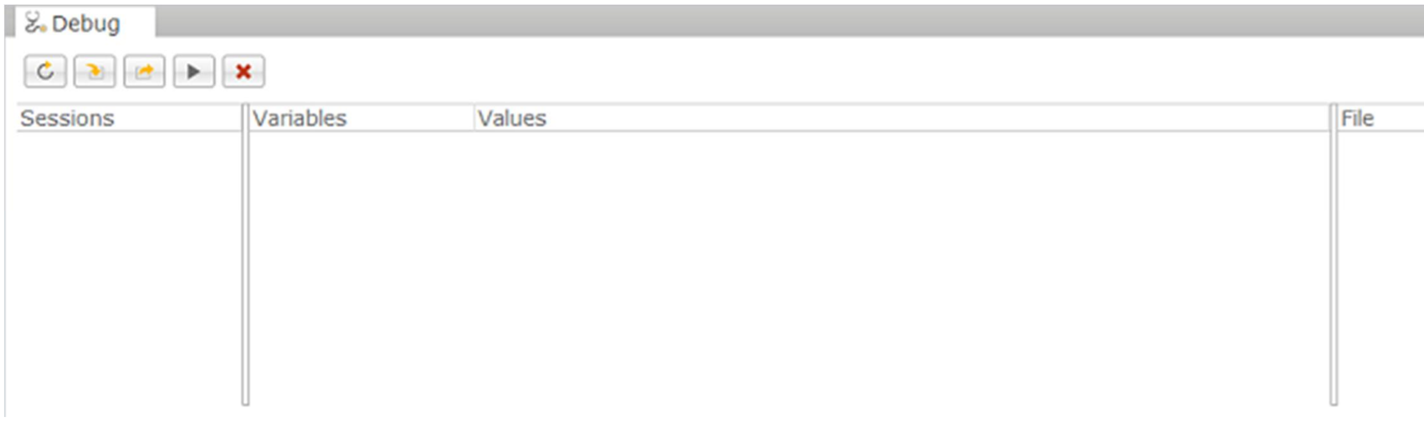
Click **other...** to list available perspectives



From the list select **Debug** perspective

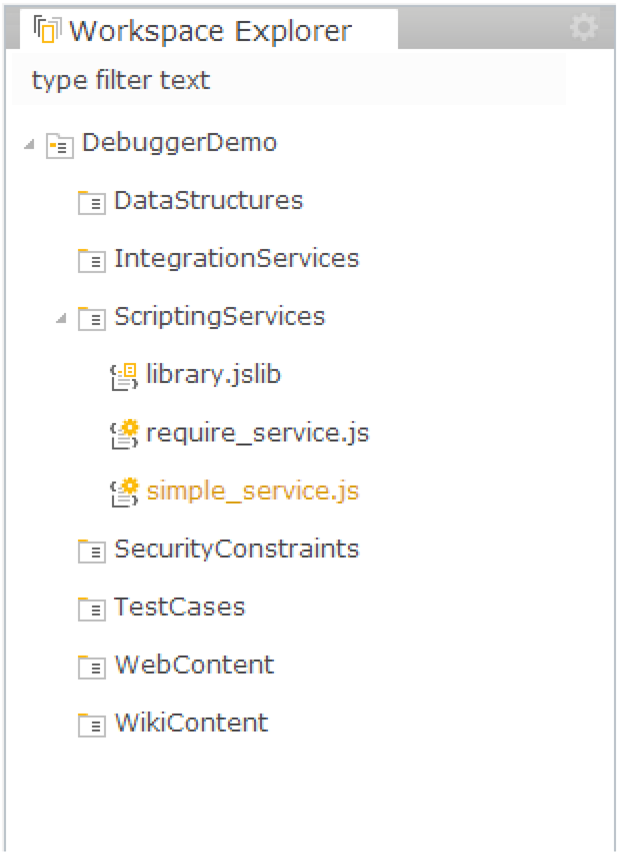


Now **Debug** perspective is opened

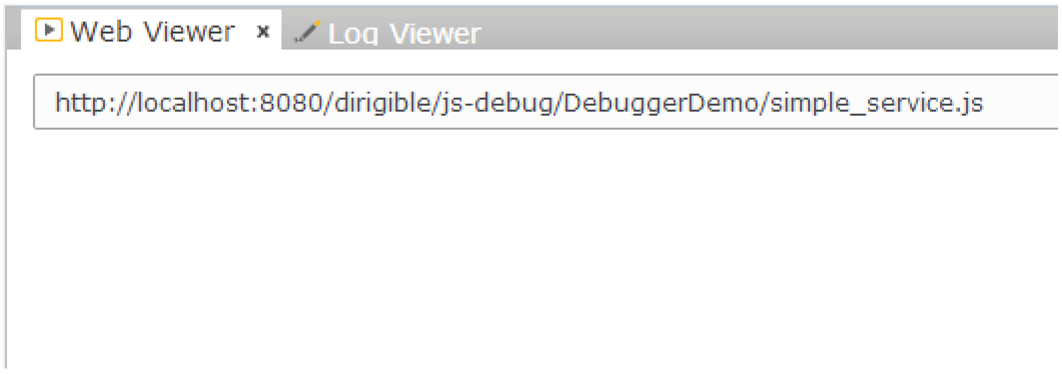


Step 2 - Start Debugging

Select **simple_service.js** from **Workspace Explorer**

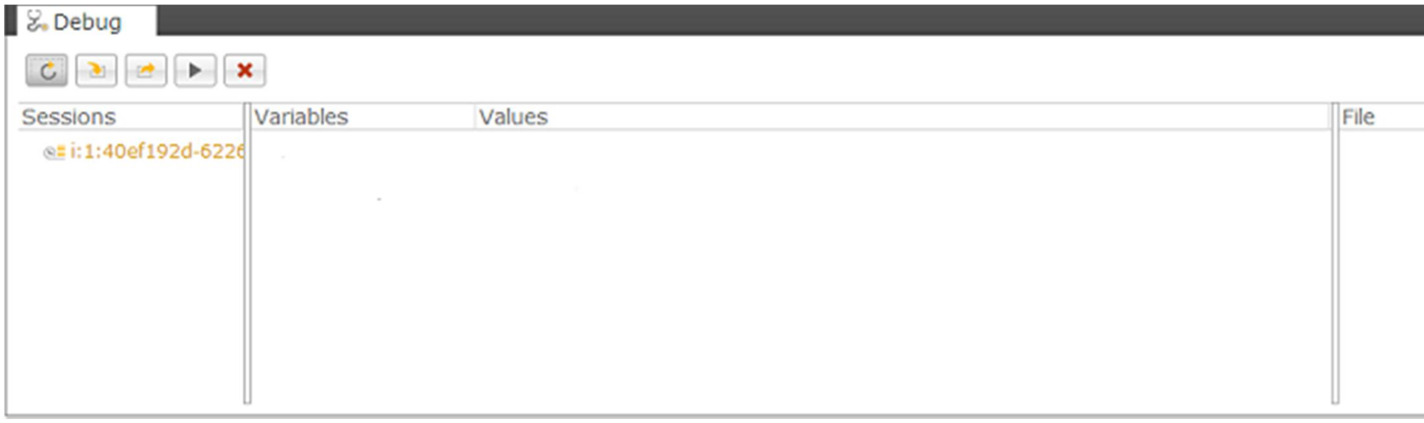


Debugger was started and waits for user interaction

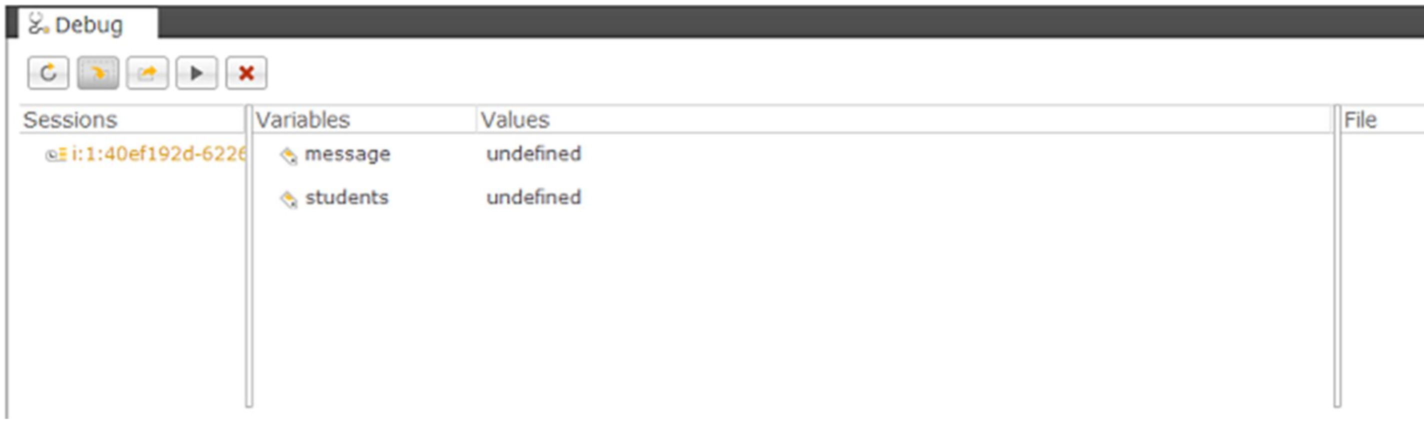


http://localhost:8080/dirigible/js-debug/DebuggerDemo/simple_service.js

In the **Debug** view press **Refresh** button to list available **Debug Sessions** and select one.

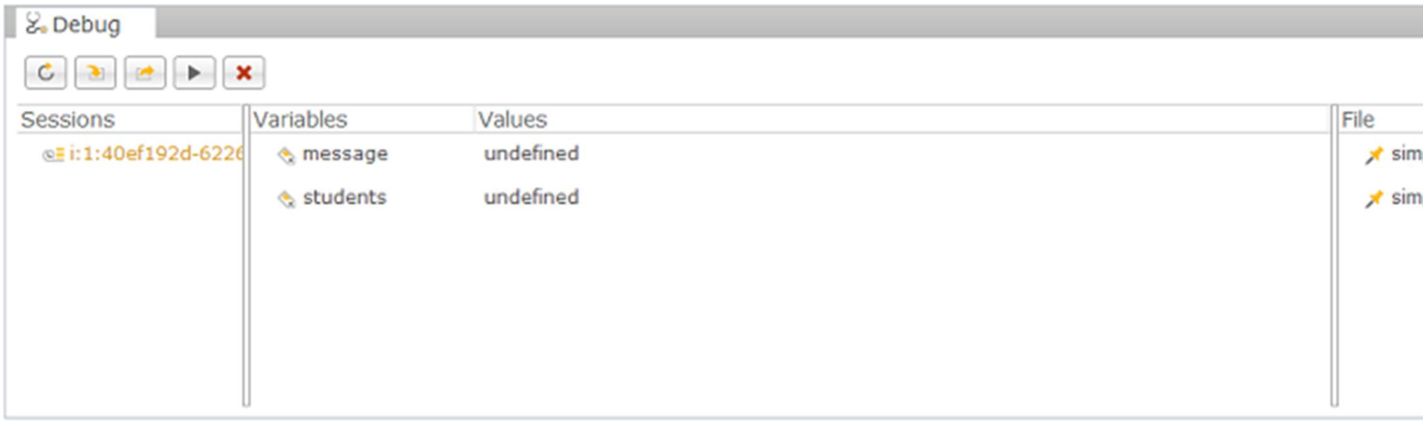


Press **Step Into** button to continue script's execution.

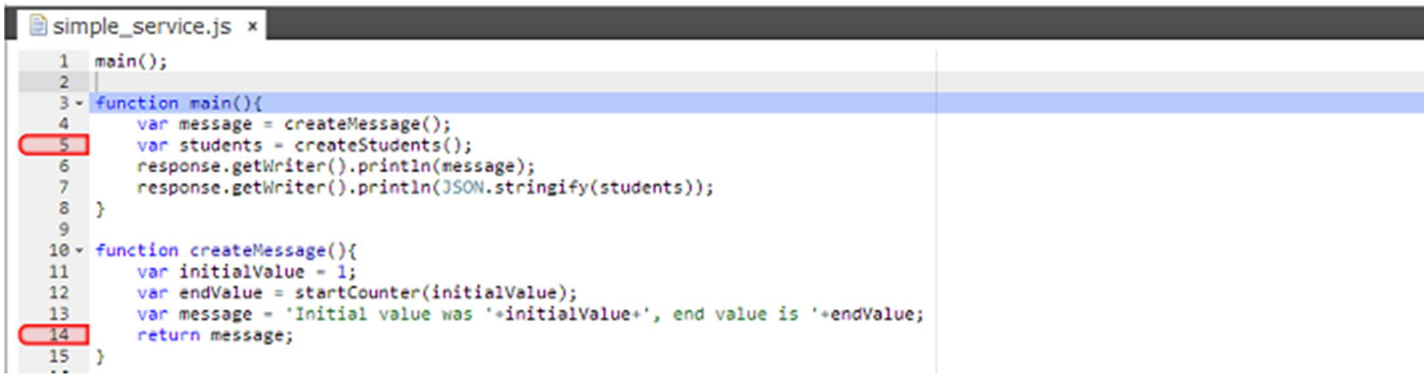


Variables	Values
message	undefined
students	undefined

Let's set some **Breakpoints**.
Click on the line numbers on the left of the opened editor.
Press **Refresh** button to see **Breakpoints** that were set.

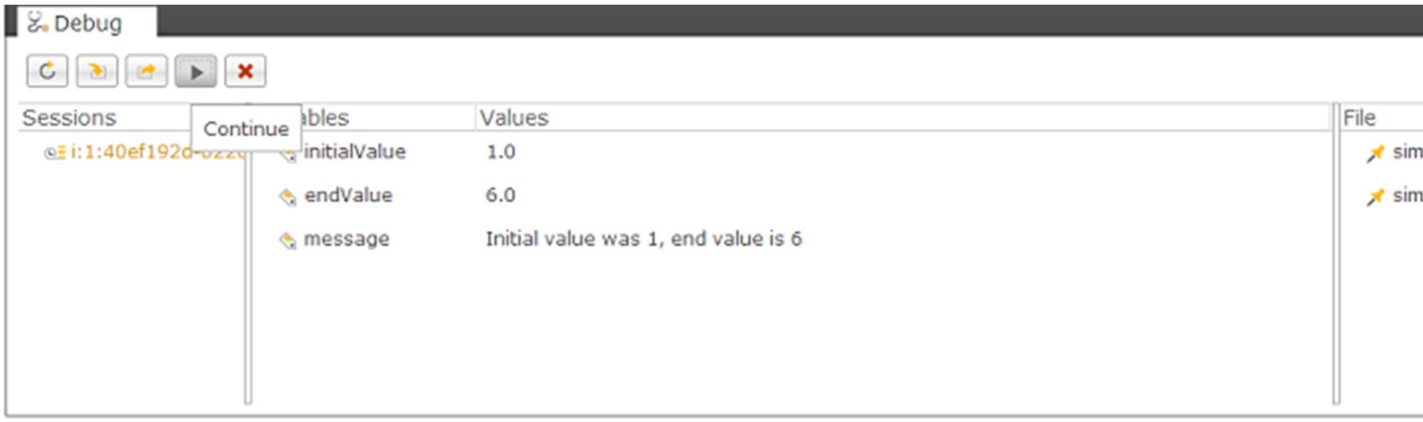


Variables	Values
message	undefined
students	undefined

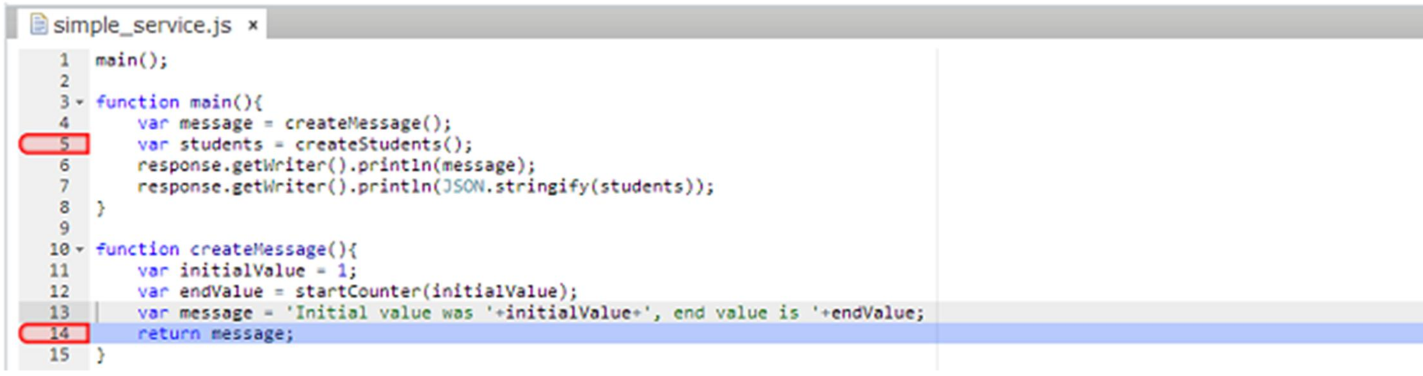


```
main();

function main(){
    var message = createMessage();
    var students = createStudents();
    response.getWriter().println(message);
    response.getWriter().println(JSON.stringify(students));
}

function createMessage(){
    var initialValue = 1;
    var endValue = startCounter(initialValue);
    var message = 'Initial value was '+initialValue+', end value is '+endValue;
    return message;
}
```

Press **Continue** ▶ button to resume script's execution to the next **Breakpoint**.

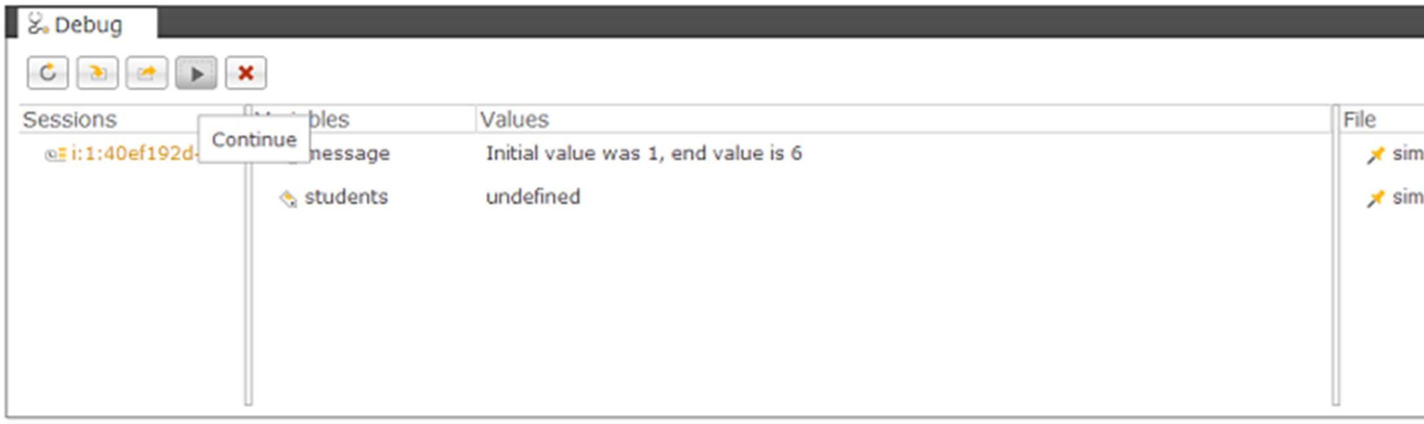


Variables	Values
initialValue	1.0
endValue	6.0
message	Initial value was 1, end value is 6



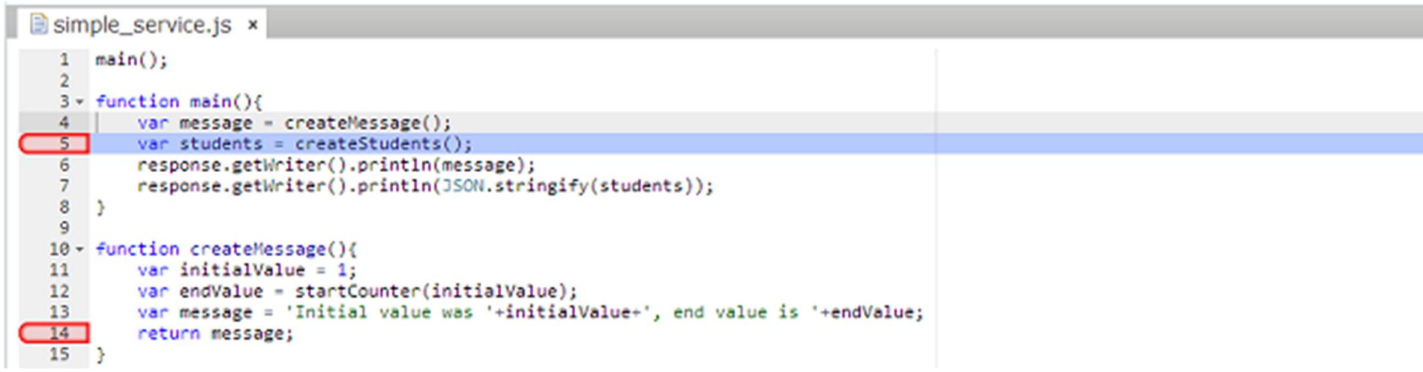
```
main();

function main(){
    var message = createMessage();
    var students = createStudents();
    response.getWriter().println(message);
    response.getWriter().println(JSON.stringify(students));
}

function createMessage(){
    var initialValue = 1;
    var endValue = startCounter(initialValue);
    var message = 'Initial value was '+initialValue+', end value is '+endValue;
    return message;
}
```

Press **Continue** ▶ button again.



i:1:40ef192d-

Variables	Values
message	Initial value was 1, end value is 6
students	undefined

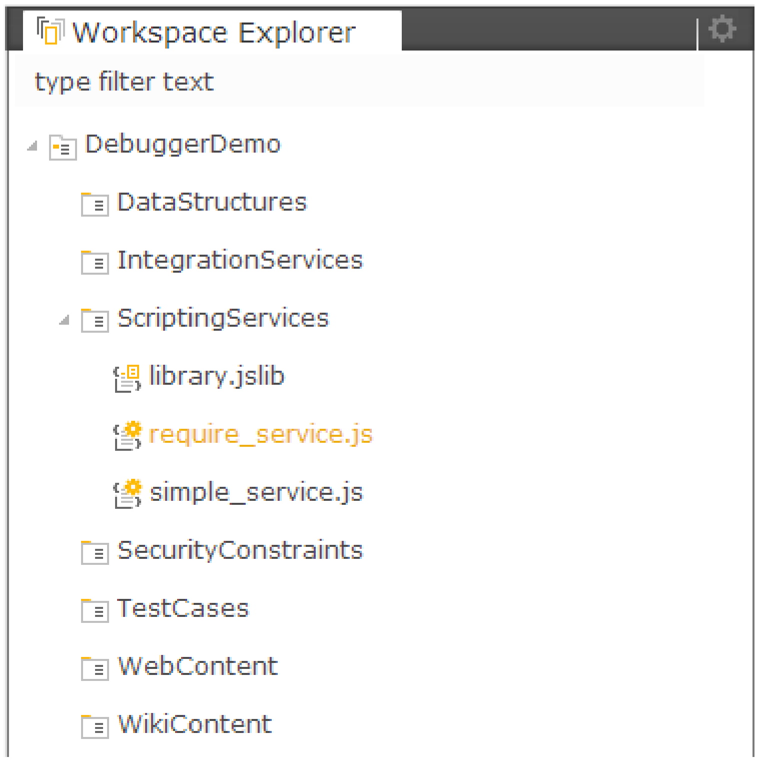


```
main();

function main(){
    var message = createMessage();
    var students = createStudents();
    response.getWriter().println(message);
    response.getWriter().println(JSON.stringify(students));
}

function createMessage(){
    var initialValue = 1;
    var endValue = startCounter(initialValue);
    var message = 'Initial value was '+initialValue+', end value is '+endValue;
    return message;
}
```

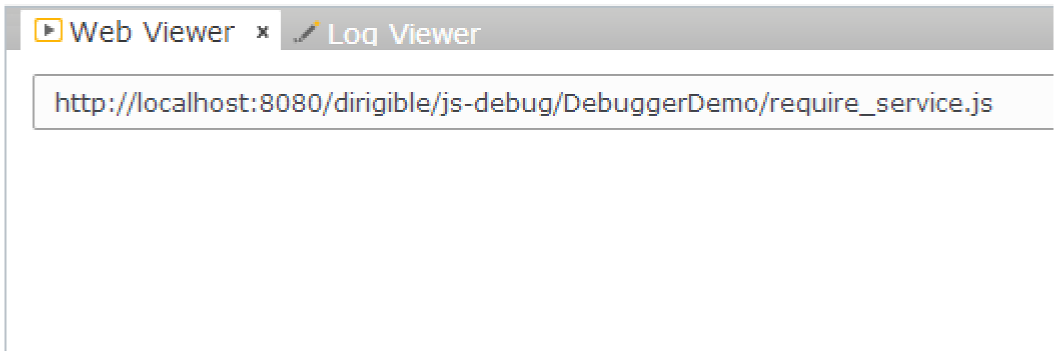
To exit **Debug Session** press **Skip all breakpoints** ✖ button
or continue pressing **Step Over** or **Step Into** buttons until script's execution finish.

Step 3 - Debugging Scripts Requiring Libraries

Select **require_service.js** from **Workspace Explorer**

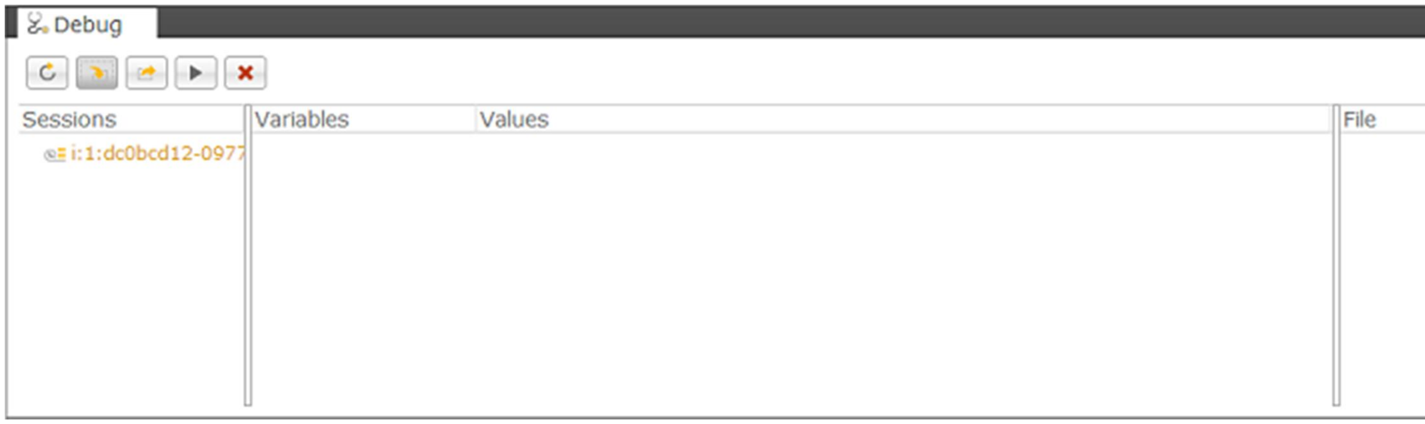


```
DebuggerDemo
    DataStructures
    IntegrationServices
    ScriptingServices
        library.jslib
        require_service.js
        simple_service.js
    SecurityConstraints
    TestCases
    WebContent
    WikiContent
```

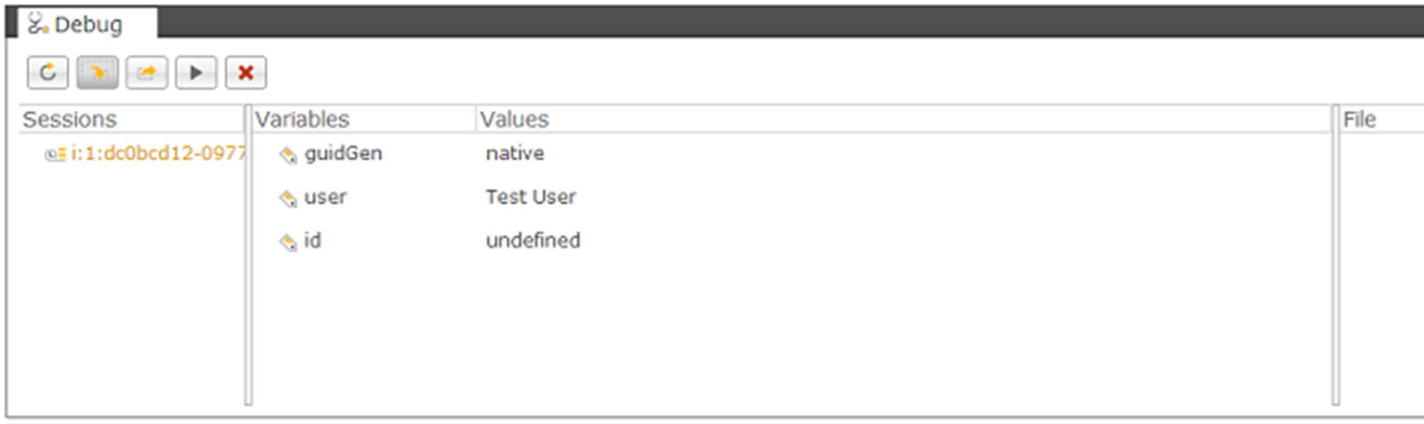


http://localhost:8080/dirigible/js-debug/DebuggerDemo/require_service.js

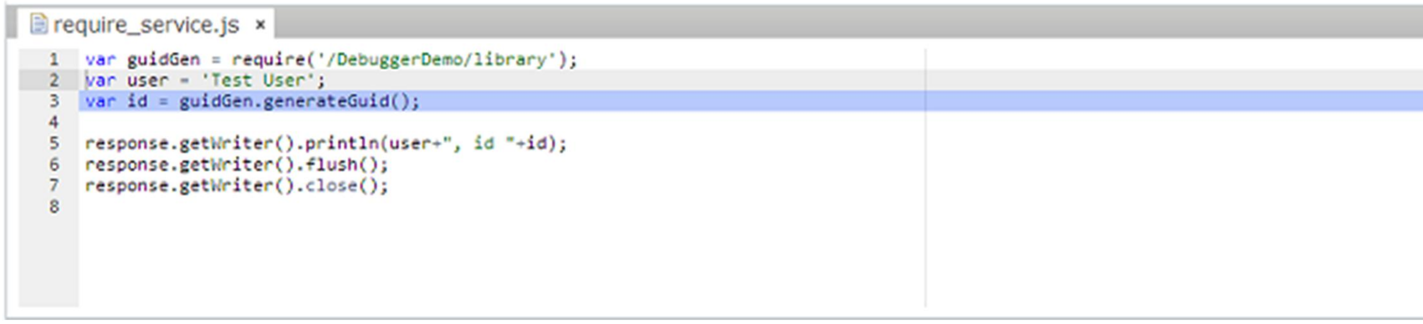
A new **Debug Session** was started.
Press **Refresh** button, select session and press **Step Into** button.



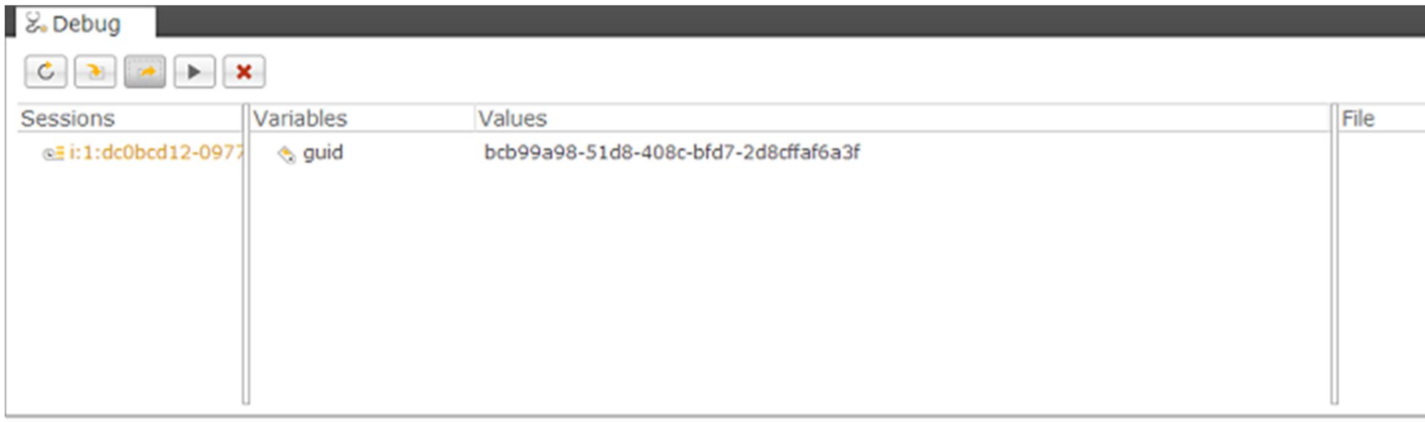
Continue debugging.



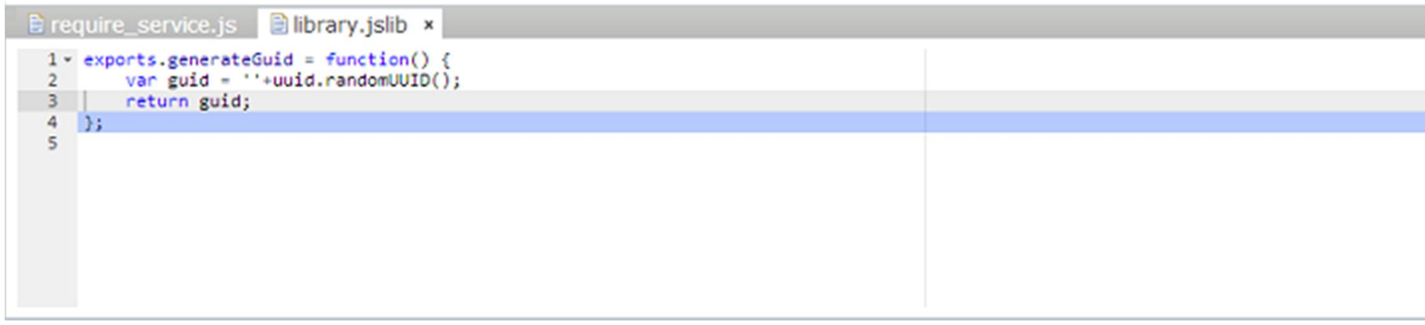
```
i:1:dc0bcd12-0977
guidGen    native
user       Test User
id         undefined
```



```
var guidGen = require('/DebuggerDemo/library');
var user = 'Test User';
var id = guidGen.generateGuid();

response.getWriter().println(user+", id "+id);
response.getWriter().flush();
response.getWriter().close();

```



```
i:1:dc0bcd12-0977
guid    bcb99a98-51d8-408c-bfd7-2d8cffaf6a3f
```

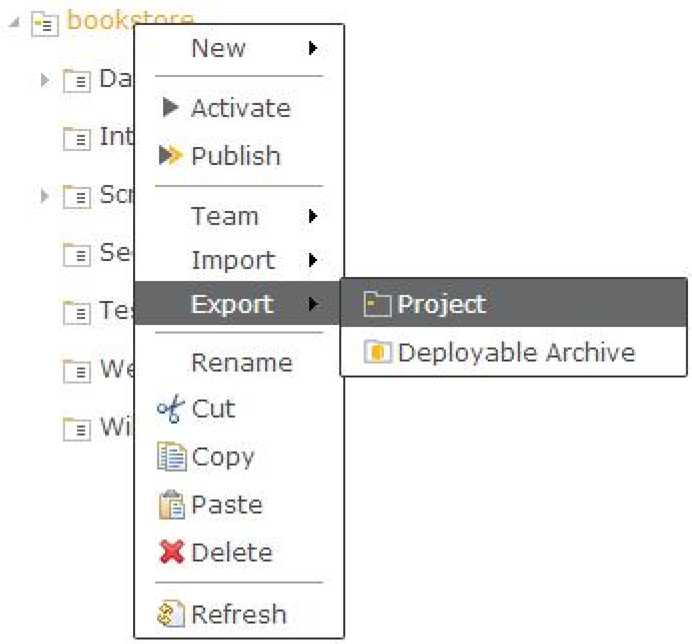


```
exports.generateGuid = function() {
    var guid = ''+uuid.randomUUID();
    return guid;
};

```

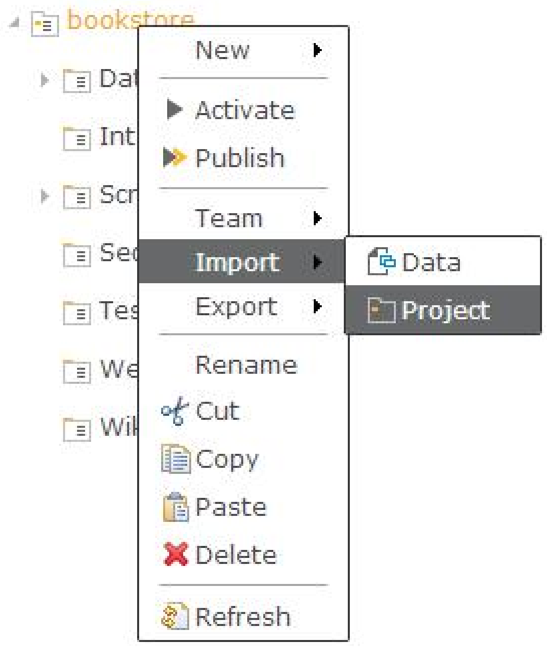
Backup - Project Import and Export

There is a possibility to export and import content of a given project or multiple projects as a zip archive for e.g. backup purposes. This is the simplest yet fastest way to transfer project content to Dirigible.
The actions are available thru the main or pop-up menu

Export



and Import



You can use this functionality also in cases of mass import of project artifacts - e.g. images in the web content

This feature can be used also for import of sample projects from public sources or sharing of projects between accounts, although for real team source management the recommended aproach is via Git.

Concepts

There are several must-to-know concepts, which are implied in the toolkit and have to be understood before starting.
Some of them are tightly related to the dynamic applications nature and behavior, others just follow the best practices from the services architecture reflected also in the cloud applications.

Isolated workspace for project management is important when we have to share a single serever instance to many users. Another related feature is sand-boxing - the way every user to have its own private runtime space where to test his/her services during the development.
Registry and the related publishing process are taken from the SOA (UDDI) and recent API Management to bring some of the strengths like discoverability, reusability, loose coupling, relevance, etc.
To boost the development productivity at the initial phase, we introduced template based generation of application artifacts via wizards.
The new Web 2.0 paradigm and the leveraged REST architectural style changed the way how the services should behave as well as how to be described. Although, there is a push for bilateral contracts only and free description of the services, we decided to introduce a bit more sophisticated kind of services for special purpose - Entity Services.

Workspace

The Workspce is the developer's place where he/she creates and manages the application artifacts. The first level entities are the projects them selves. Each project contains several system folders based on the type of the artifacts.
Default ones are:

- DataStructures - containing the tables, views, etc. database related artifacts (more info here)
- IntegrationServices - containing the definitions about the routes and related artifacts (more info here)
- ScriptingServices - containing the JavaScript, Ruby, Groovy, etc. server-side services and related artifacts (more info here)
- SecurityConstraints - containing the access control definitions artifacts (more info here)
- TestCases - containing the unit tests for the scripting services and related artifacts (more info here)

- WebContent - containing the static pages as well as the client-side scripting artifacts (more info here)
- WikiContent - containing the confluence formatted wiki pages and related artifacts (more info here)

Example layout of a project looks like this:

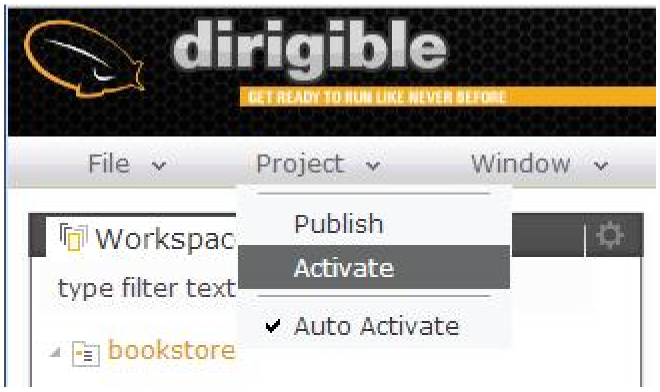
```
/db /dirigible /users /<user> (private space) /workspace /project1 /DataStructures /data1.t
able /data1.dsv /IntegrationServices /connector1.routes /ScriptingServices /service1.js /Se
curityConstraints /main.access /TestCases /service1_test.js /WebContent /index.html /defaul
t.css /WikiContent /project1.wiki /license.wiki
```

The project management can be done via the views and editors in the workspace perspective.

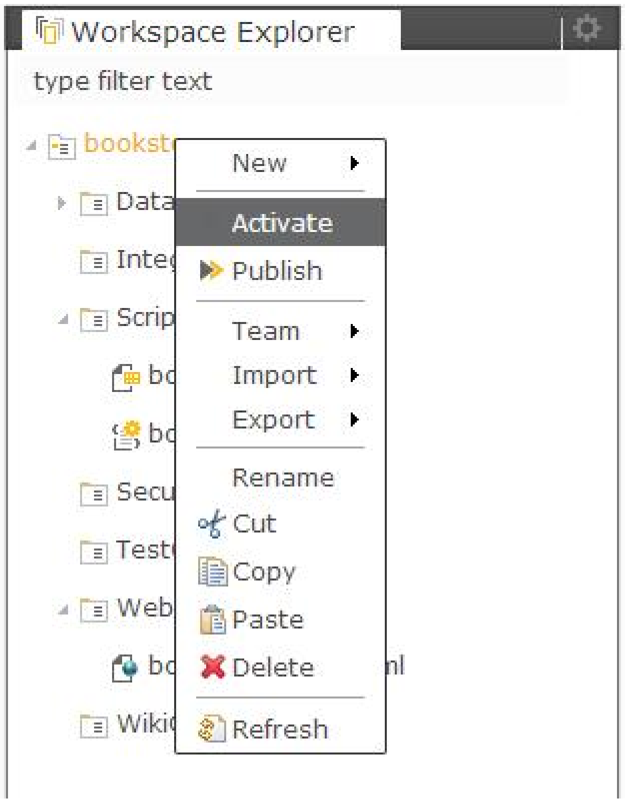
Activation

Activation is a concept related to development life-cycle of the application.
The original sources are stored in the workspace of the user. All the changes reflect direclty the source artifacts there. When the source artifact is already in the state, which can be executed (i.e. tested) the developer has to perform **activation** on the project level.
This will transfer (copy) the source artifacts from the workspace to the **sandbox**. This place is fully functional runtime container, isolated for the current user only. The difference between the **sandbox** and the **registry** space is the user isolation only.

Activation action is accessible from the main menu under the Project section or at the project's pop-up menu in the Workspace Explorer



or



```
/db /dirigible /sandbox /<user> (private space) /ScriptingServices /project1 /service1.js /
users /<user> (private space) /workspace /project1 /ScriptingServices /service1.js
```

The scripting services in the sandbox can access the services from the registry, but not vice versa

There is default **auto-activation** mechanism, which can perform the activation on save of the artifact. This can be switched on/off from the main menu -> Project (if you are in the Workspace perspective)

The auto-activation is enabled only for:

- Scripting Services
- Web Content
- Wiki Content

For:

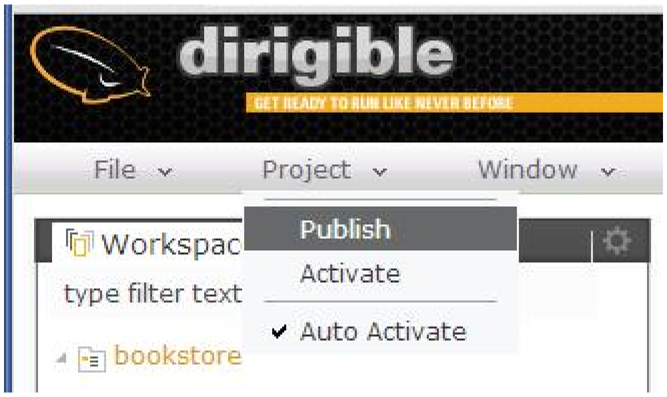
- Data Structures
- Security Constraints
- Integration Services

there is no sandboxing supported as well as auto-activation. The activation process is equal to publication in this case.

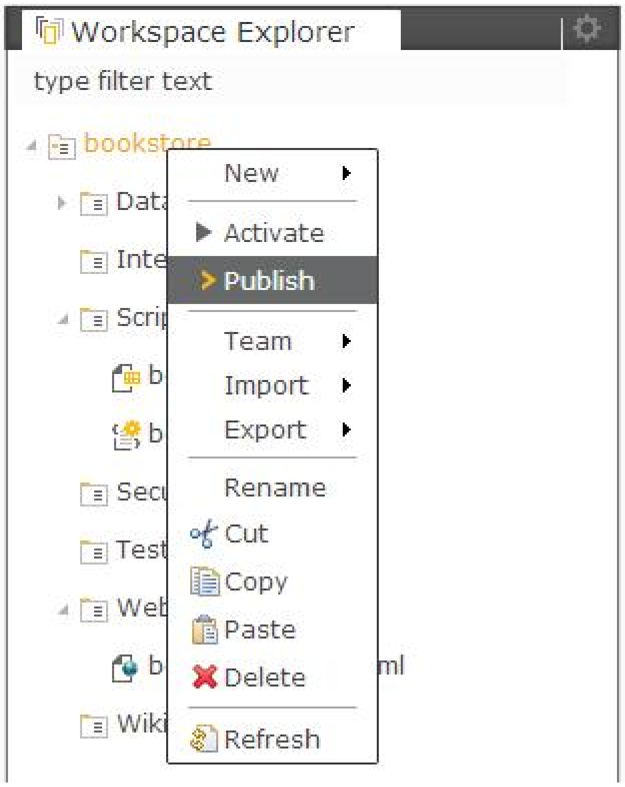
Publishing

There is a conceptual separation between design-time and runtime phases of the development life-cycle.
During the design-time phase the source artifacts are created and managed within the isolated developer's area - workspace.
When the developer is ready with a given feature, he/she have to publish the project, so that the application artifacts to become available for the users. It depends of the type of the artifact the meaning "available". For the Scripting Services for instance it is the registration of a public endpoint, for web and wiki content it is just allowed access to the artifacts them self, etc.

Publishing action is accessible from the main menu under the Project section or at the project's pop-up menu in the Workspace Explorer



or



The space within the Repository where all the public artifact are placed is called **Registry**

```
/db /dirigible /registry (public space) /public (placeholder) /ScriptingServices /project1
/service1.js /users /<user> (private space) /workspace /project1 /ScriptingServices /servic
e1.js
```

To view the currently published artifacts you can go to Registry User Interface. There are separate section by the types of the artifacts.

Generation

Template based generation of artifacts helps for developer productivity in the initial phase of building the application.
There are several application components, which have similar behavior and often very similar implementation.
A prominent example is Entity Service. It has several predefined methods based on REST concepts and HTTP - GET, POST, PUT, DELETE on an entity level as well as list of all entites. On the other hand, the most notable storage for the entity's data is the RDBMS provided by the platform. Hence to save the developer's effort to create such entity services from scratch we introduced a template, which can be used for generation from table to service.
Another example is user interface templates based on patterns - list, master-detail, input form, etc. There can be provided also templates based on different frameworks for client-side interaction.

The generation is one-time process. Once you have the generated artifact you can modify it based on your own requirements.

In contrast of the MDA, where you can expect to regenerate the PSMs every time you make changes on PIMs, we decided to choose the more pragmatic approach - single model. In general it is good to have an abstraction and technology agnostic languages and models, but in practice last years it turned out that it brings more problems than the benefits especially for support - where MDA claims to be good for.

Entity Service

In general, the Entity Service is fully capable RESTful service as it is defined by REST architecural style for performance, scalability, simplicity, etc. It exposes the CRUD operations of a given domain model object. Underneath it connects the database store as data transfer layer.
Working following RESTful paradigm on business software components, soon it turns out that there is a service pattern, which is used very often - domain object management. Having in mind also the past experience from web services, we decided to enhance a bit the standard functionality of such a services without breaking REST principles. These enhancements are useful especially for generic utilities and user interface generation.

First of all let list what we have as a standard:

- **GET** method
  - if the requested path points directly to the service endpoint (no additional parameters) - it lists all the entities of this type (in this collection)
  - if the request contains an **id** parameter - then the service returns only the requested entity
- **POST** method - creates an entity getting the fields from the request body (JSON formatted) and auto-generated id
- **PUT** method - updates the entity getting the id from the request body (JSON formatted)
- **DELETE** method - deletes the entity by the provided id parameter which is mandatory

The enhancements we added to the standard functionality:

- on **GET** as parameters
  - **count** - returns the number of the entities collection size
  - **metadata** - returns the simplified descriptor of the entity in JSON (see below)
  - **sort** - indicate the order of the entities
  - **desc** - indicates reverse order, used with the above parameter
  - **limit** - used for paging, returns limited result set
  - **offset** - used for paging, result set starts from the offset value

Example metadata for an entity

```
{"name":"books","type":"object","properties": [ {"name":"book_id","type":"integer","key":"t
rue","required":"true"}, {"name":"book_isbn","type":"string"}, {"name":"book_title","type":
"string"}, {"name":"book_author","type":"string"}, {"name":"book_editor","type":"string"},
{"name":"book_publisher","type":"string"}, {"name":"book_format","type":"string"}, {"name":
"book_publication_date","type":"date"}, {"name":"book_price","type":"double"} ] }
```

These enhancements we see as the minimal yet simplest valuable extension to the REST. Similar, but more complex specifications, which intentionally we do not included so far are OData and GData.

All these features of entity services are implied during the generation process. The template uses as input a database table and name of the entity service, which are entered in the corrsponding wizard.
Just select the *.entity artifact in the Workspace Explorer and use the pop-up menu **Generate->User Interface for Entity Service**.

There are several limitation for the table to be entity service compliant:

- there should be one and only column as primary key, which will be used for its **identity**
- only a set of database column types, which are supported by default for generation (simple types only; clob, blob - not supported)

We do not generate also generic query methods, because on one hand it will cover only very simple cases with reasonable performance, which easiliy can be written anyway as additional methods (by parameters) and on the other hand for the complex queries there is no sense to introduce additional layer, which will not give the desired performance as well in comparison to the well analysed by the developer SQL script.

Entity services are generated used JavaScript language, hence the can be accessed right after the generation and publishing on:

<protocol>://<host>:<port>/<dirigible's runtime application context>/js/<project>/<entity service path>
e.g.

https ://dirigibleide.hana.ondemand.com/dirigible/js/bookstore/books.js

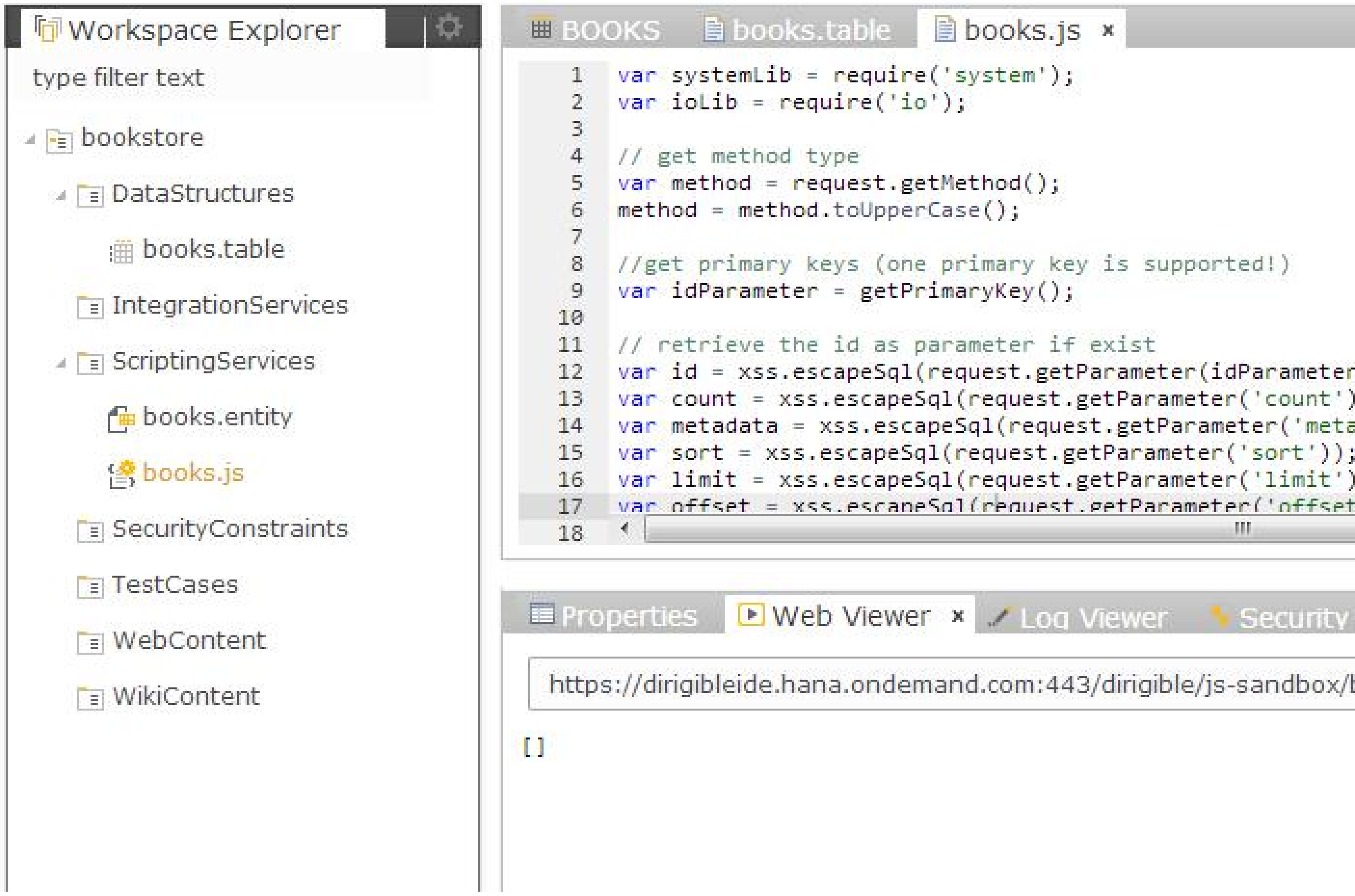
or just select them in Workspace Explorer and see the result in the Web Viewer.

Tooling

One of the biggest advantages of Dirigible project is its powerful design-time tooling. It claims to be Development Environment as a Service (DEaaS), means it is complete enough to cope with all the needs of a developer, building cloud based dynamic application only via a web browser.
It is too ambitious statement and it realy is because it is based on the greatest yet mass-used so far IDE by the on-premise developers - Eclipse and its very innovative project RAP.

The different views and editors are separated by default to a few perspective depending on the purpose:

- Workspace Perspective - responsible to overall project management.

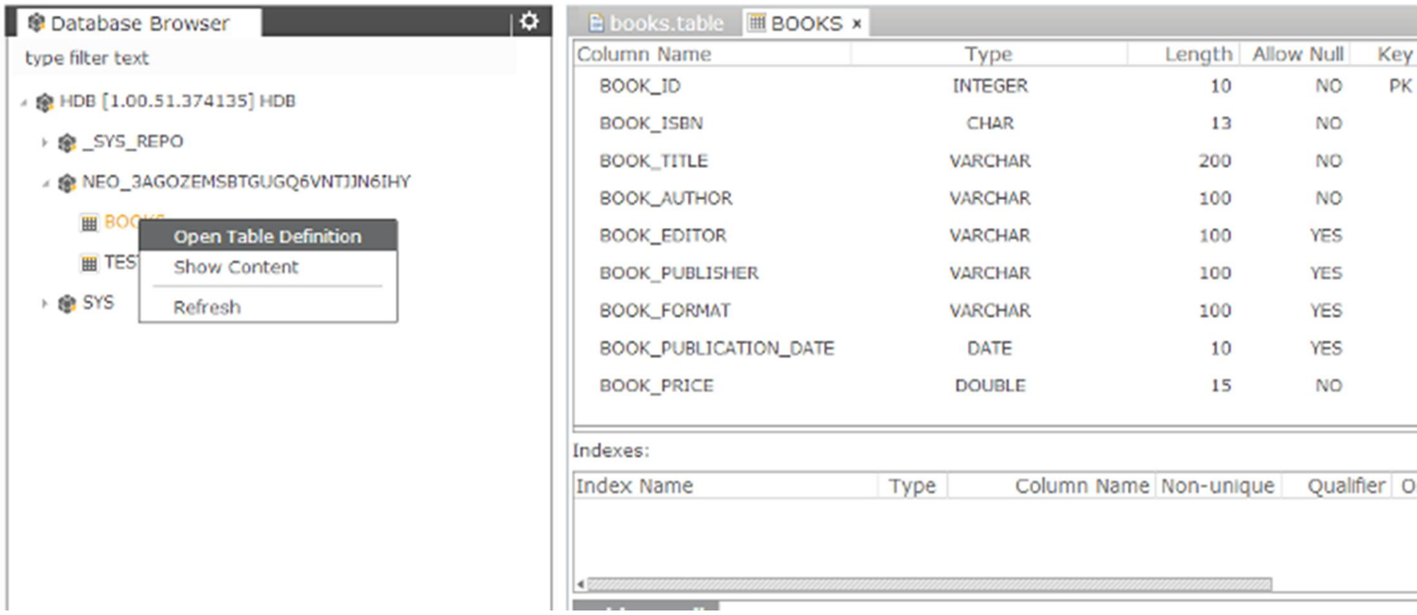


```
var systemLib = require('system');
var ioLib = require('io');

// get method type
var method = request.getMethod();
method = method.toUpperCase();

//get primary keys (one primary key is supported!)
var idParameter = getPrimaryKey();

// retrieve the id as parameter if exist
var id = xss.escapeSql(request.getParameter(idParameter
var count = xss.escapeSql(request.getParameter('count')
var metadata = xss.escapeSql(request.getParameter('meta
var sort = xss.escapeSql(request.getParameter('sort'));
var limit = xss.escapeSql(request.getParameter('limit')
var offset = xss.escapeSql(request.getParameter('offset
```

https://dirigibleide.hana.ondemand.com:443/dirigible/js-sandbox/b

```
[]
```

- Database Perspective - for inspecting low level raw content directly in the underlying database with the Database Browser and SQL Console


Database Browser

type filter text

- HDB [1.00.51.374135] HDB
  - _SYS_REPO
  - NEO_3AGOZEMSBTGUGQ6VNTJJN6IHY
    - BOOKS
    - TEST
  - SYS

Open Table Definition
Show Content
Refresh

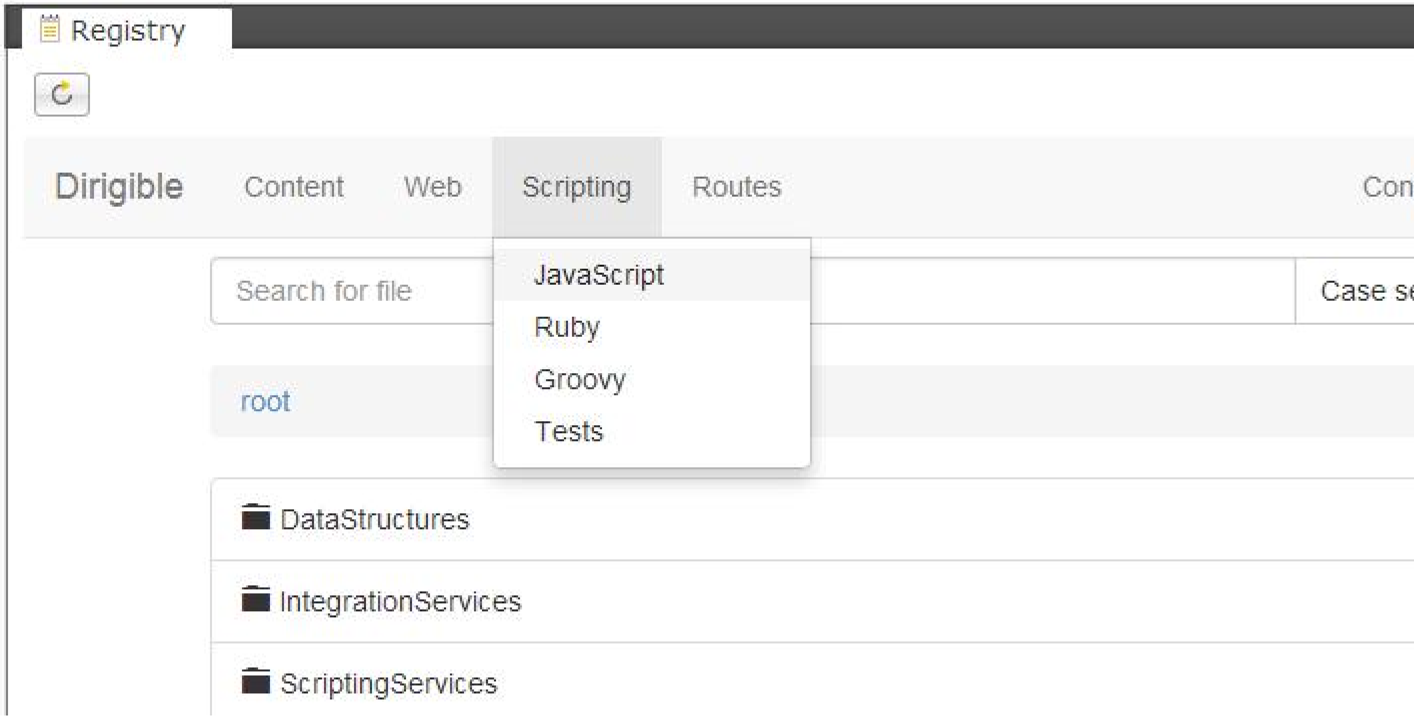
books.table | BOOKS ×

Column Name	Type	Length	Allow Null	Key
BOOK_ID	INTEGER	10	NO	PK
BOOK_ISBN	CHAR	13	NO	
BOOK_TITLE	VARCHAR	200	NO	
BOOK_AUTHOR	VARCHAR	100	NO	
BOOK_EDITOR	VARCHAR	100	YES	
BOOK_PUBLISHER	VARCHAR	100	YES	
BOOK_FORMAT	VARCHAR	100	YES	
BOOK_PUBLICATION_DATE	DATE	10	YES	
BOOK_PRICE	DOUBLE	15	NO	

Indexes:

Index Name	Type	Column Name	Non-unique	Qualifier	O

- Repository Perspective - looking into the current state as well as the history of the repository content
- Registry Perspective - an integrated view to the published resources and active service endpoints in the runtime containers



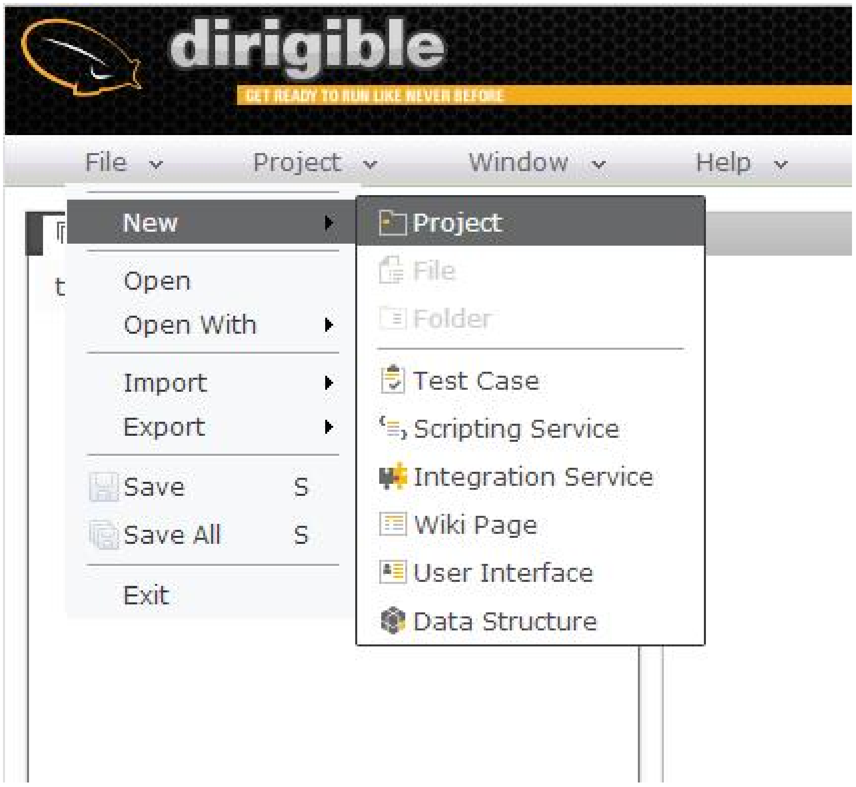
DataStructures
IntegrationServices
ScriptingServices

Workspace Perspective

This is the place where you actually develop your dynamic applications. This perspective contains all the views and editors that help you in overall implementation from domain models via services to the user interface.

Main views which are openned by default in this perspective are:

- **Workspace Explorer** - it is a standard view on the projects in your workspace. It shows the folder structure as well as the contained resources. There is a pop-up menu assigned to the project node:



via which you can create new artifacts:

- Project
- Data Structure
- Scripting Service
- User Interface
- Wiki Page
- Integration Service
- Test Case

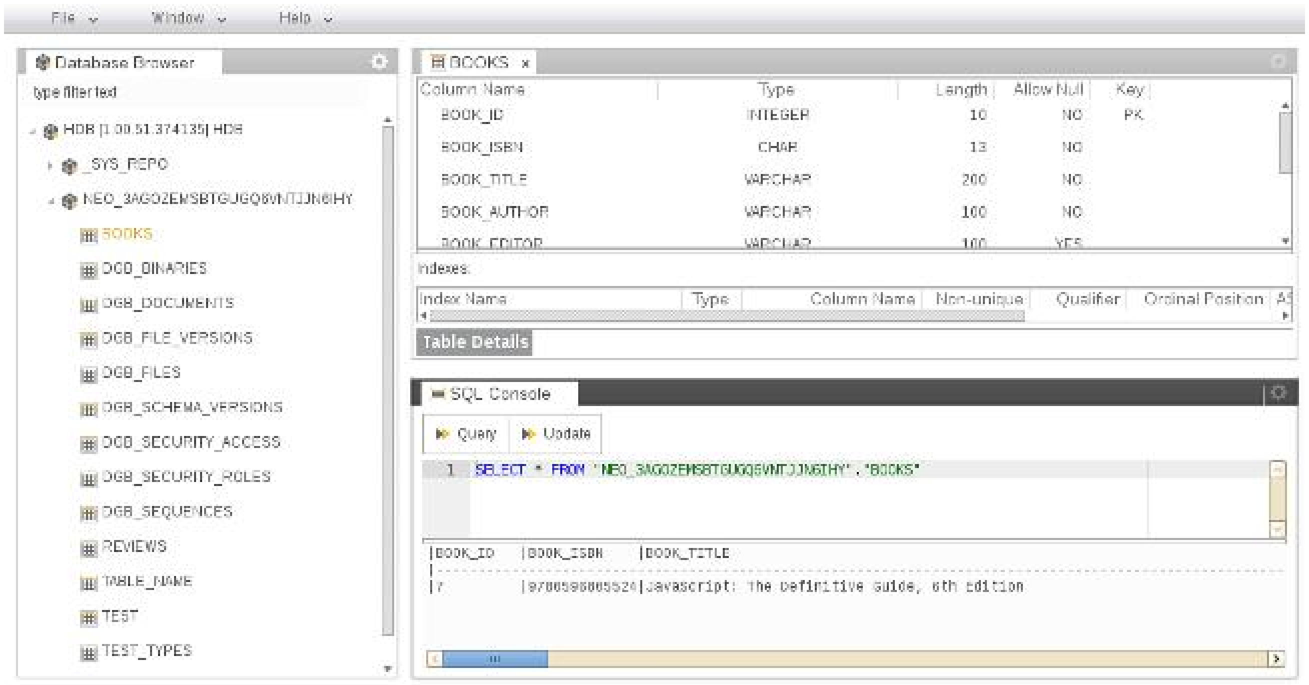
or just regular

- File and
- Folder

While selecting an artifact you can use Open or Open With actions to load its content in the corresponding editor e.g. Source Editor.

Database Perspective

Database Perspective contains the tools for inspection and manipulation of the artifacts within the underlying relational database.
You have direct access to the default target schema assigned to your account in the cloud platform. From the Database Browser you can expand the schema item and to see the list of all the tables and views created either via the models or directly via SQL script.



Column Name	Type	Length	Allow Null	Key
BOOK_ID	INTEGER	10	NO	PK
BOOK_ISBN	CHAR	13	NO	
BOOK_TITLE	VARCHAR	200	NO	
BOOK_AUTHOR	VARCHAR	100	NO	
BOOK_EDITOR	VARCHAR	100	YES	

Index Name	Type	Column Name	Non-unique	Qualifier	Ordinal Position	AS

```
SELECT * FROM "NEO_3AGOZEMSBTGUGQ6VNTJJN6IHY"."BOOKS"
```

```
|BOOK_ID    |BOOK_ISBN     |BOOK_TITLE
|--------------------------------------------------------------------------
|7          |9780596805524|Javascript: The Definitive Guide, 6th Edition
```

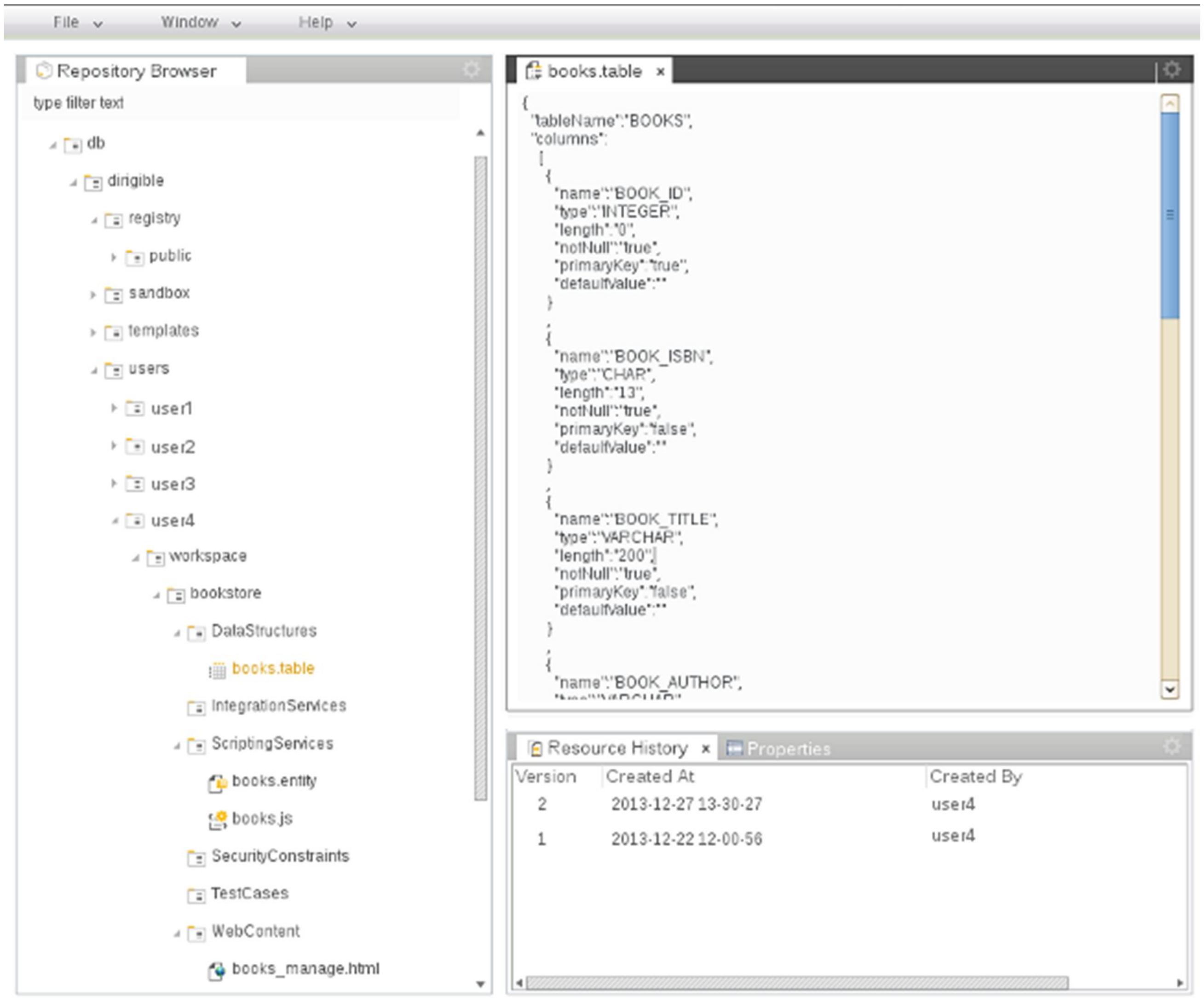
The actions which can be performed on a table or view are:

- Open Definition - presents the columns layout
- Show Content - generates a select statement in the SQL Console and executes it
- Export Data - exports the data from the table or view in *.dsv format
- Delete - removes physically the table from the database

Another tool in the Database Perspective is very generic and in the same time very powerful one - SQL Console.

Repository Perspective

Repository Perspective gives the access to the raw structure of the underlying Repository content. There one could inspect at low level the project and folder structure as well as the artifacts content in the read-only plain editor.



```
db
  dirigible
    registry
      public
    sandbox
    templates
    users
      user1
      user2
      user3
      user4
        workspace
          bookstore
            DataStructures
              books.table
            IntegrationServices
            ScriptingServices
              books.entity
              books.js
            SecurityConstraints
            TestCases
            WebContent
              books_manage.html
```

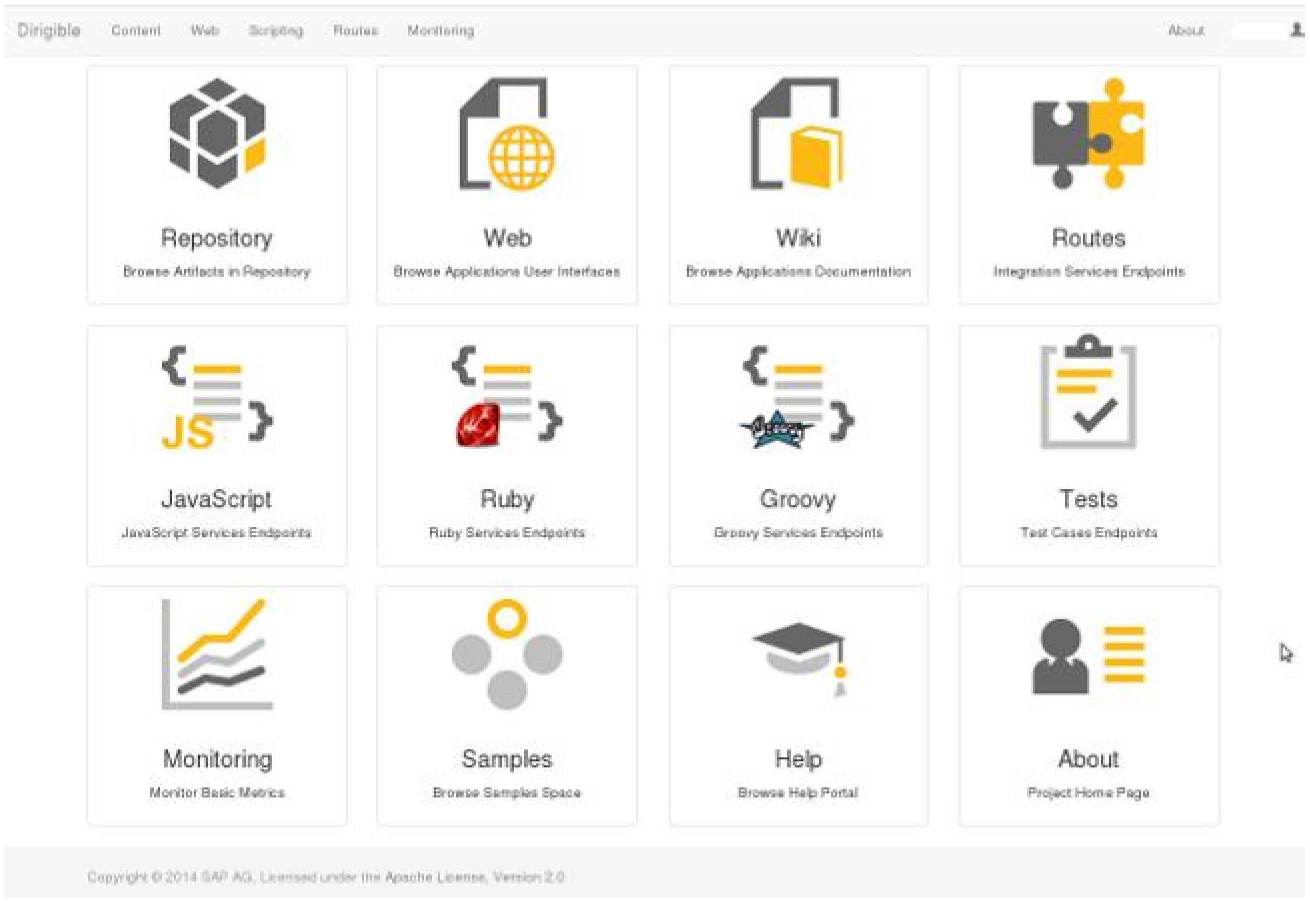
```
{
 "tableName":"BOOKS",
 "columns":
  [
   {
    "name":"BOOK_ID",
    "type":"INTEGER",
    "length":"0",
    "notNull":"true",
    "primaryKey":"true",
    "defaultValue":""
   }
   ,
   {
    "name":"BOOK_ISBN",
    "type":"CHAR",
    "length":"13",
    "notNull":"true",
    "primaryKey":"false",
    "defaultValue":""
   }
   ,
   {
    "name":"BOOK_TITLE",
    "type":"VARCHAR",
    "length":"200",
    "notNull":"true",
    "primaryKey":"false",
    "defaultValue":""
   }
   ,
   {
    "name":"BOOK_AUTHOR",
```

Version	Created At	Created By
2	2013-12-27 13-30-27	user4
1	2013-12-22 12-00-56	user4

Resource History View lists the most recent versions of the main artifact. It plays a role as local file history. As soon as the Repository component doesn't target the version control system by itself, you should rely on the Git integration for secure resource management. The history of the Repository is automatically cleared up after a given period of time.

Registry

The entry point of the searching and browsing of the service endpoints as well as monitoring and administration at runtime phase is the Registry. Technically it is a space within the Repository where all the published artifacts are placed.


Dirigible
Content
Web
Scripting
Routes
Monitoring
About
Repository
Browse Artifacts in Repository
Web
Browse Applications User Interfaces
Wiki
Browse Applications Documentation
Routes
Integration Services Endpoints
JavaScript
JavaScript Services Endpoints
Ruby
Ruby Services Endpoints
Groovy
Groovy Services Endpoints
Tests
Test Cases Endpoints
Monitoring
Monitor Basic Metrics
Samples
Browse Samples Space
Help
Browse Help Portal
About
Project Home Page
Copyright © 2014 SAP AG. Licensed under the Apache License, Version 2.0

To access the user interface you can point to the runtime context - default one is "dirigible"

http //<host>:<port>/dirigible

Endpoints

From the index page of the Registry, you can navigate to the corresponding sub-pages for browsing the raw content of the Repository, published user interfaces (html, css, client-side javascript, etc.), documentation of the applications, lookup the endpoints of the scripting services as well as the endpoints of the integration services.

Monitoring Tools

The last phase of the applications lifecycle includes administration and monitoring.
Via the Registry interface you can navigate to the monitoring tools including:

- Hit Count Statistics

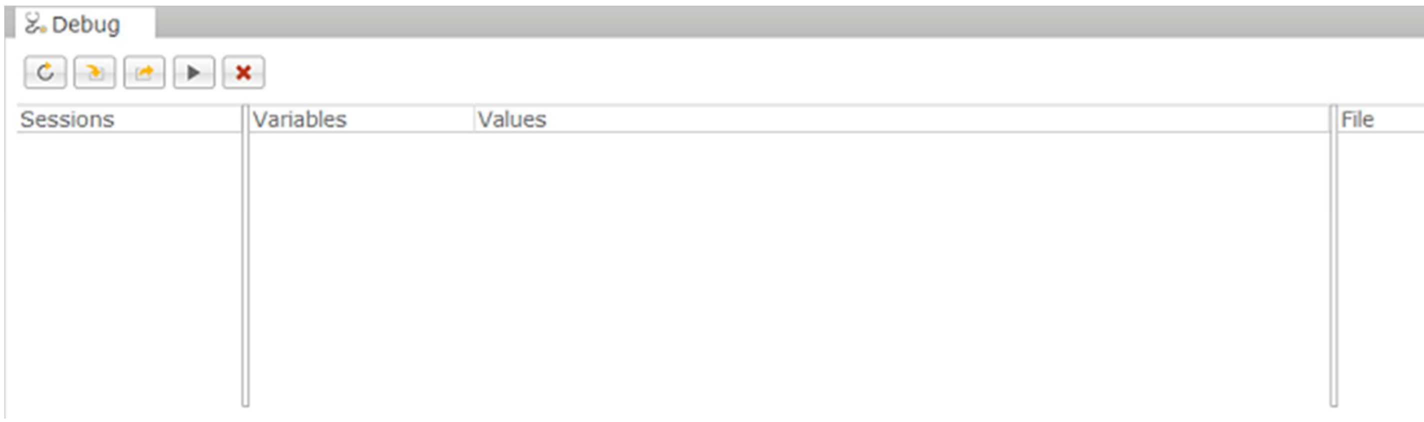
- Response Time Statistics
- Memory Allocations
- Log Traces

The URIs on which you want to collect information about the request parameters have to be registered in the Manage Access Locations section.

Debug

The toolkit offers Debugging functionallity. The goal is to easy the developers in the hunt of server side bugs.

Debug Perspective



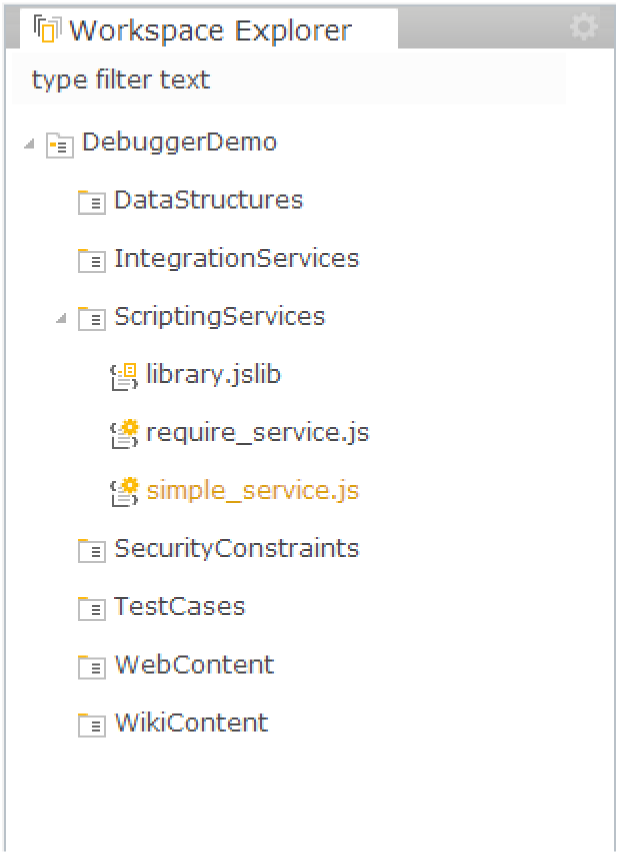
- **Sessions** - contains all debug execution sessions.
- **Variables/Values** - contains variables and their values, available in the current scope of execution.
- **File/Row/Source** - contains inforamation about in witch file and witch row, a **Breakpoint** is set.

Available Commands

- Refresh
- Step Into
- Step Over
- Continue
- Skip all breakpoints

Debugging Example

Step 0 - Project Structure



simple_service.js

```
main(); function main(){ var message = createMessage(); var students = createStudents(); response.getWriter().println(message); response.getWriter().println(JSON.stringify(students)); } function createMessage(){ var initialValue = 1; var endValue = startCounter(initialValue); var message = 'Initial value was '+initialValue+', end value is '+endValue; return message; } function startCounter(value) { for(var i = 0; i < 5; i++){ value ++; } return value; } function createStudents(){ var students = []; students.push(createStudent('Desi', 18)); students.push(createStudent('Jordan', 21)); students.push(createStudent('Martin', 22)); return students; } function createStudent(name, age){ var student = {}; student.name = name; student.age = age; return student; }
```

library_jslib

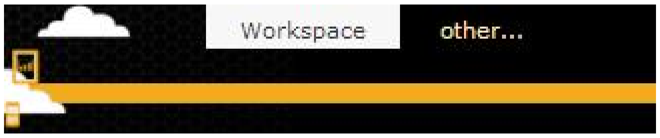
```
exports.generateGuid = function() { var guid = ''+uuid.randomUUID(); return guid; };
```

require_service.js

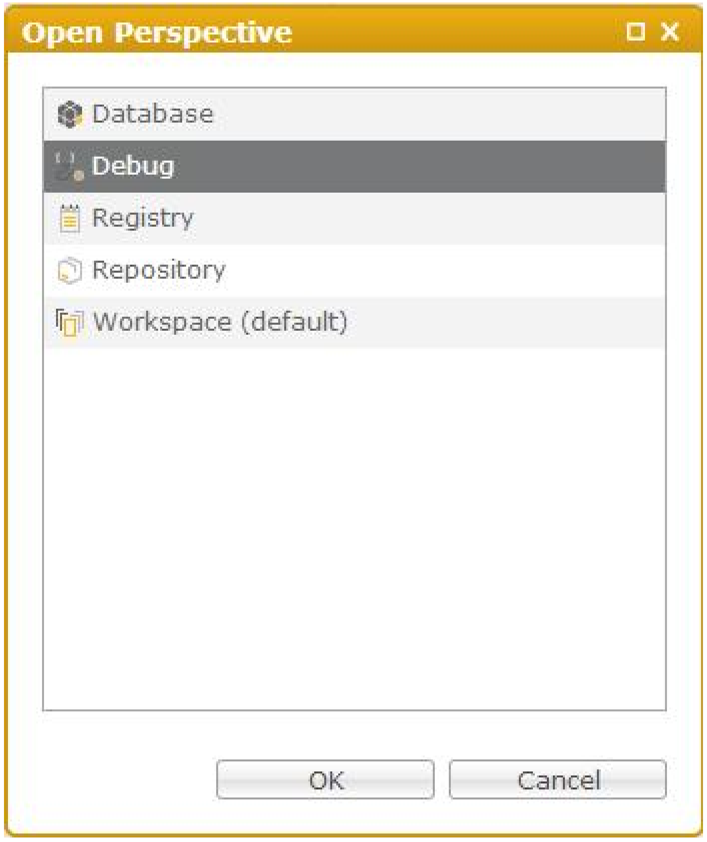
```
var guidGen = require('/DebuggerDemo/library'); var user = 'Test User'; var id = guidGen.ge
nerateGuid(); response.getWriter().println(user+", id "+id); response.getWriter().flush();
response.getWriter().close();
```

Step 1 - Open Debug Perspective

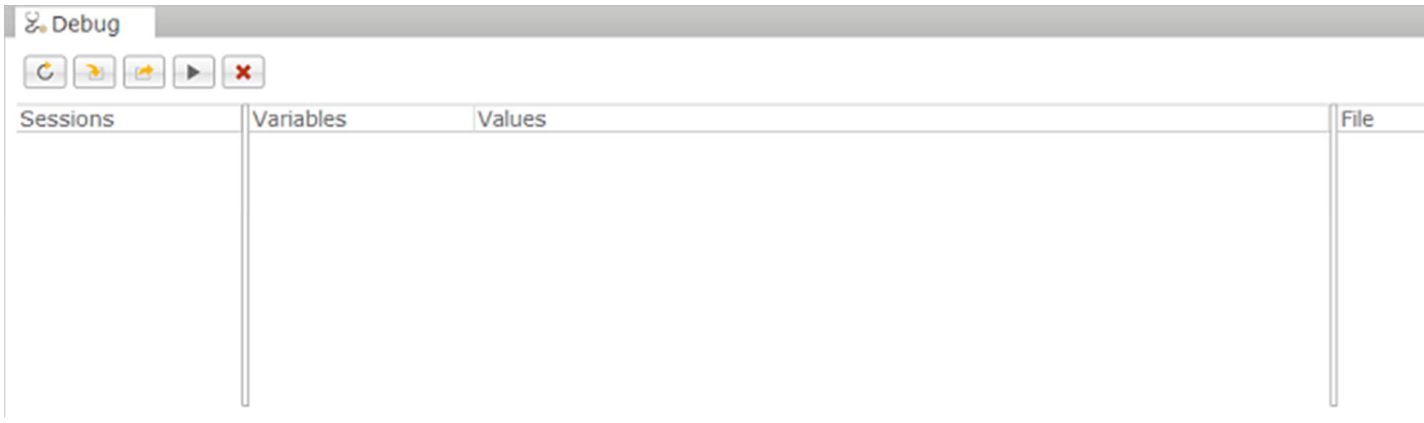
Click **other...** to list available perspectives



From the list select **Debug** perspective

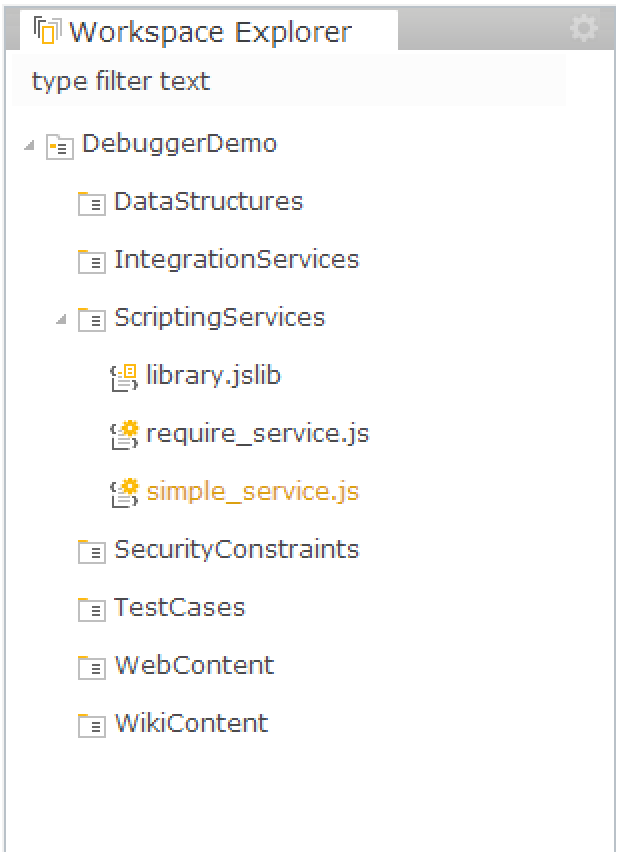


Now **Debug** perspective is opened

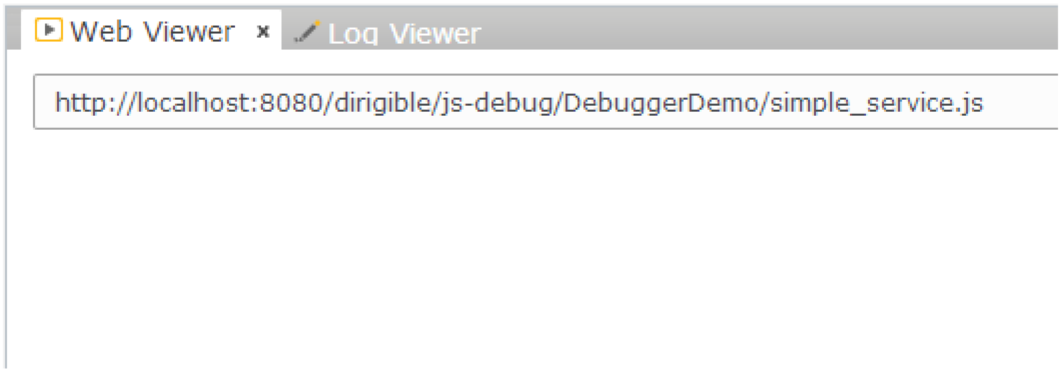


Step 2 - Start Debugging

Select **simple_service.js** from **Workspace Explorer**

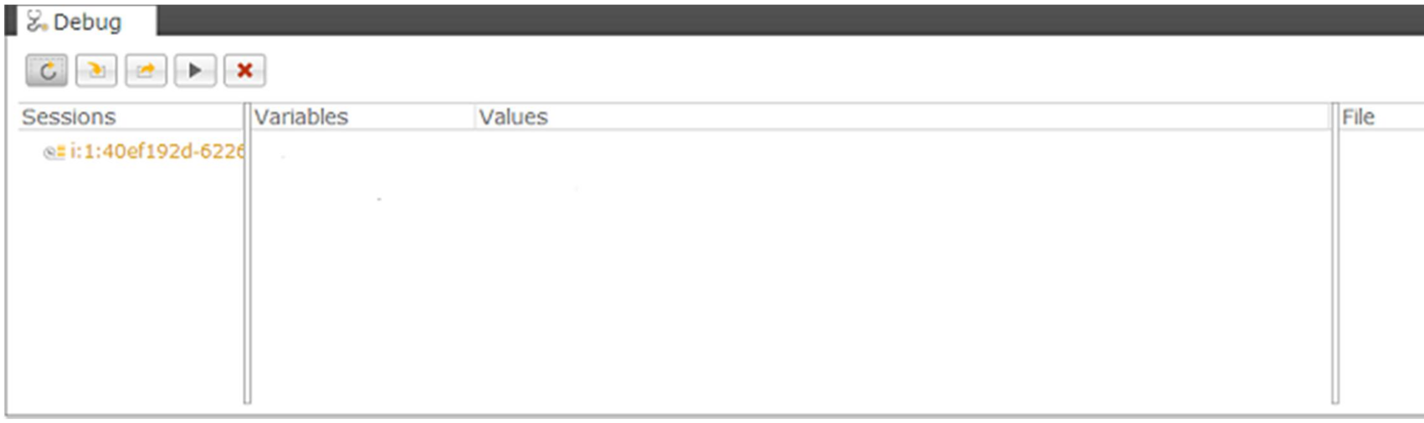


Debugger was started and waits for user interaction

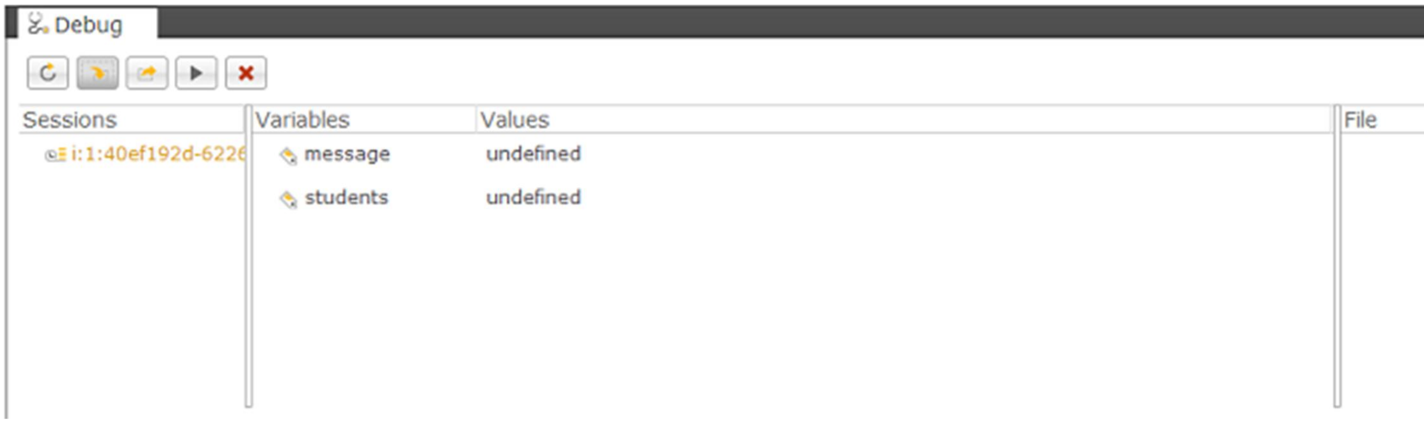


http://localhost:8080/dirigible/js-debug/DebuggerDemo/simple_service.js

In the **Debug** view press **Refresh** button to list available **Debug Sessions** and select one.

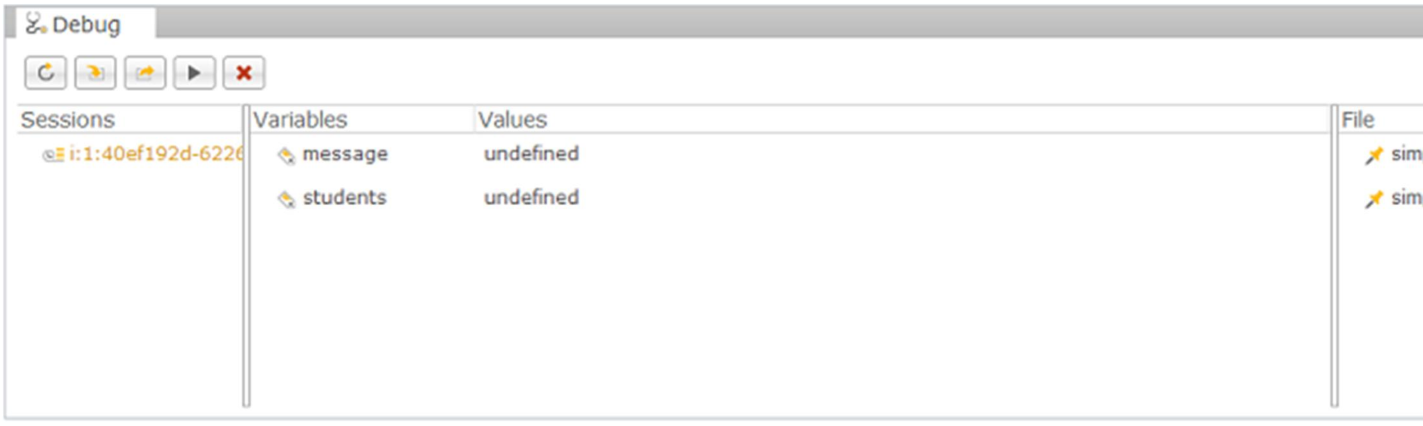


Press **Step Into** button to continue script's execution.

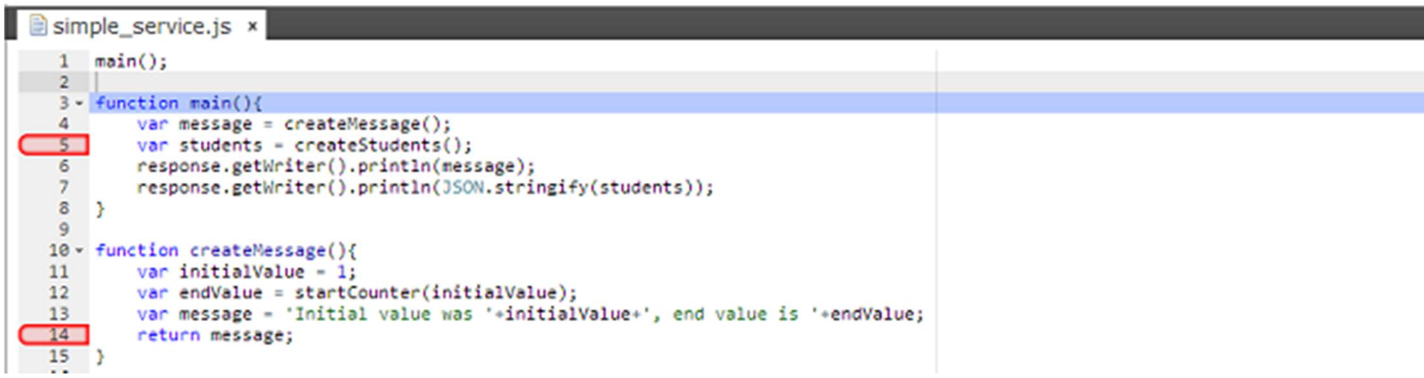


Variables	Values
message	undefined
students	undefined

Let's set some **Breakpoints**.
Click on the line numbers on the left of the opened editor.
Press **Refresh** button to see **Breakpoints** that were set.

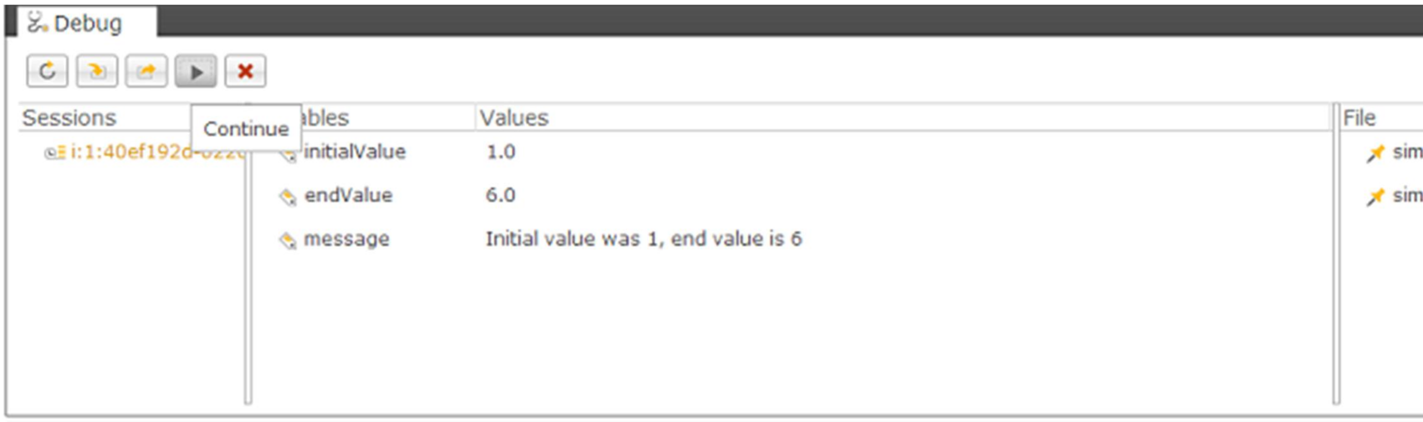


```
message     undefined
students    undefined
```

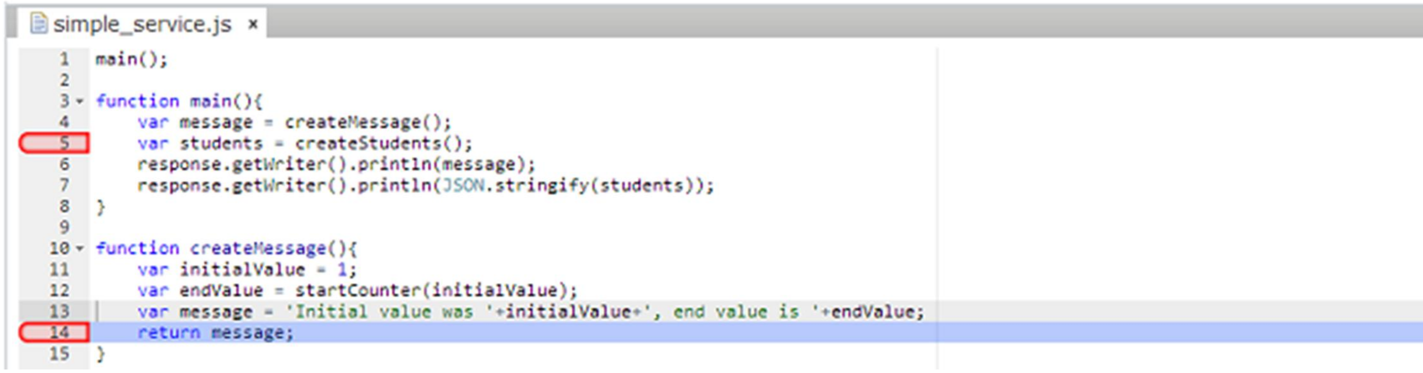


```
main();

function main(){
    var message = createMessage();
    var students = createStudents();
    response.getWriter().println(message);
    response.getWriter().println(JSON.stringify(students));
}

function createMessage(){
    var initialValue = 1;
    var endValue = startCounter(initialValue);
    var message = 'Initial value was '+initialValue+', end value is '+endValue;
    return message;
}
```

Press **Continue** ▶ button to resume script's execution to the next **Breakpoint**.

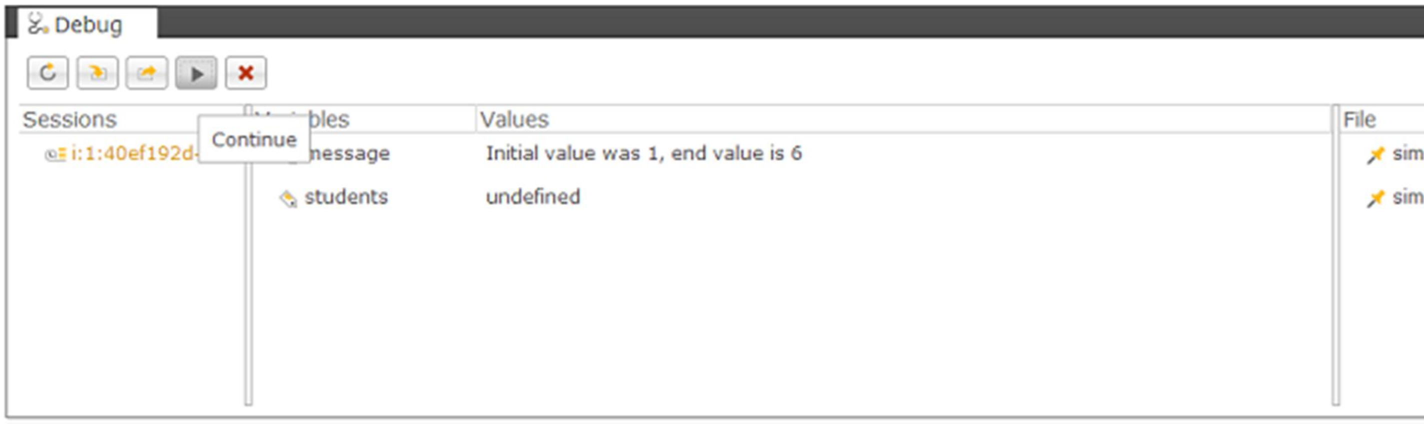


```
initialValue    1.0
endValue        6.0
message         Initial value was 1, end value is 6
```

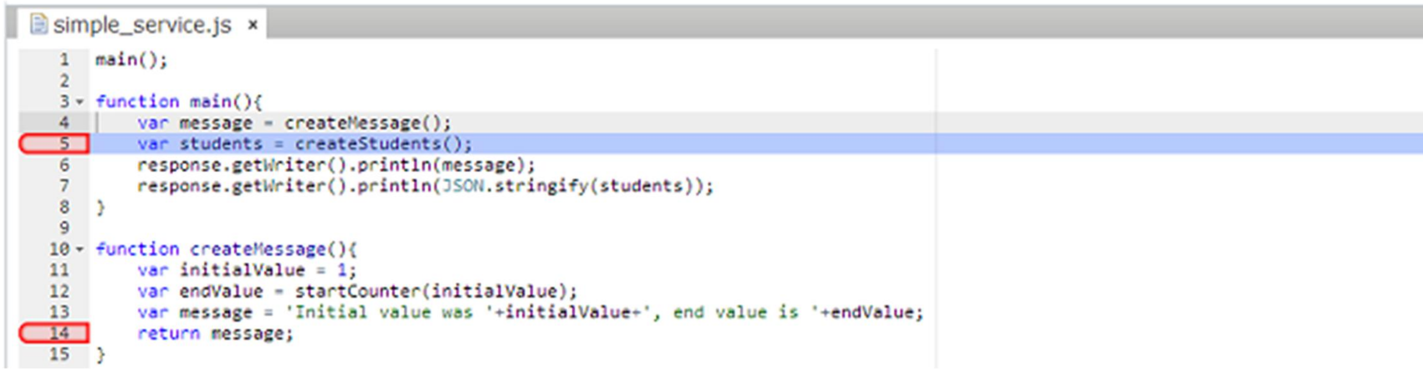


```
main();

function main(){
    var message = createMessage();
    var students = createStudents();
    response.getWriter().println(message);
    response.getWriter().println(JSON.stringify(students));
}

function createMessage(){
    var initialValue = 1;
    var endValue = startCounter(initialValue);
    var message = 'Initial value was '+initialValue+', end value is '+endValue;
    return message;
}
```

Press **Continue** ▶ button again.



Variables	Values
message	Initial value was 1, end value is 6
students	undefined

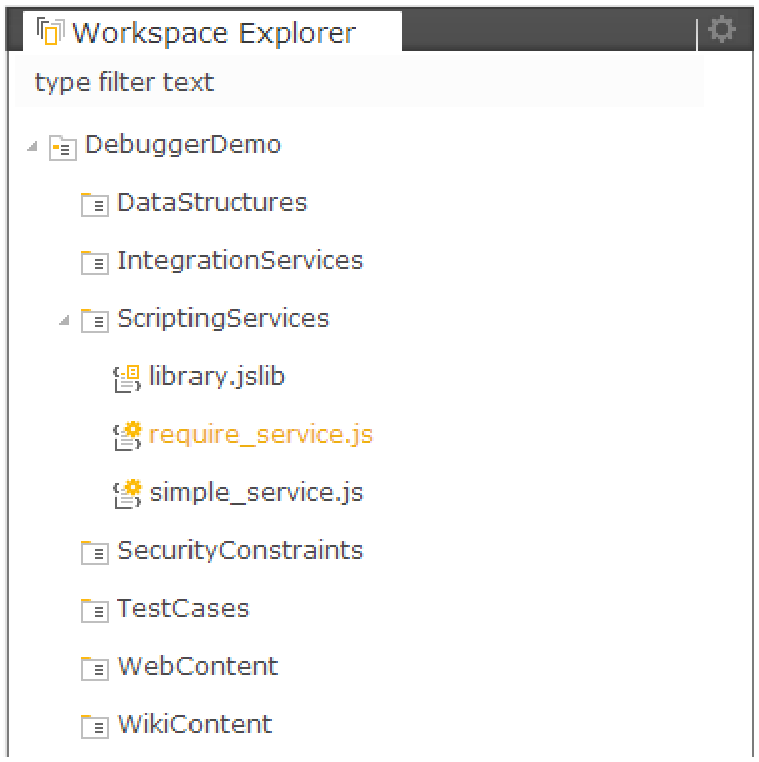


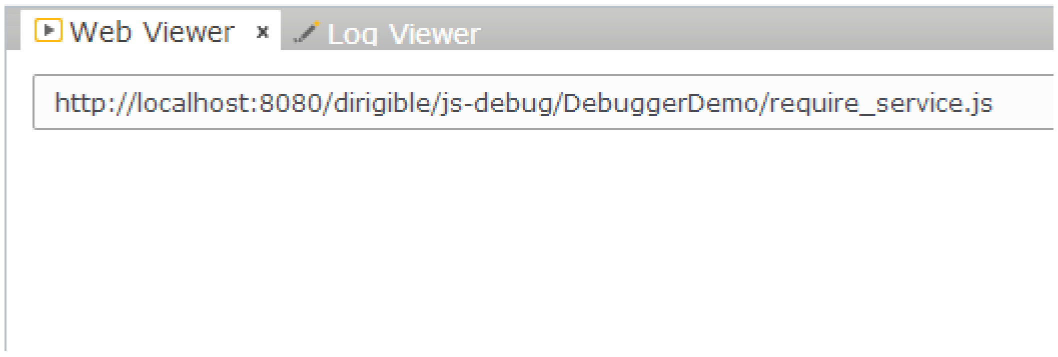
```
main();

function main(){
    var message = createMessage();
    var students = createStudents();
    response.getWriter().println(message);
    response.getWriter().println(JSON.stringify(students));
}

function createMessage(){
    var initialValue = 1;
    var endValue = startCounter(initialValue);
    var message = 'Initial value was '+initialValue+', end value is '+endValue;
    return message;
}
```

To exit **Debug Session** press **Skip all breakpoints** ✖ button
or continue pressing **Step Over** or **Step Into** buttons until script's execution finish.

Step 3 - Debugging Scripts Requiring Libraries

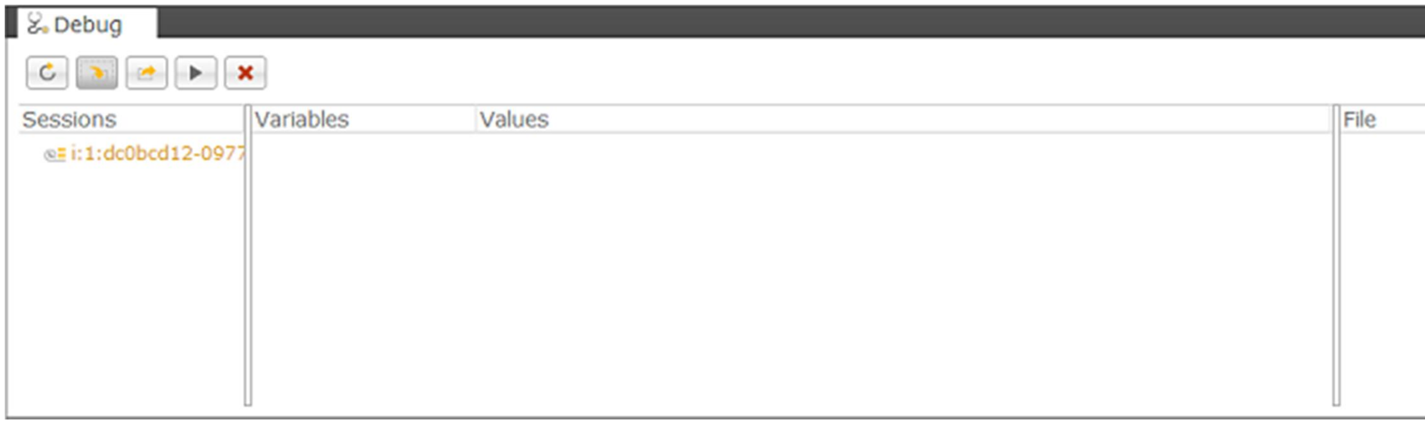
Select **require_service.js** from **Workspace Explorer**



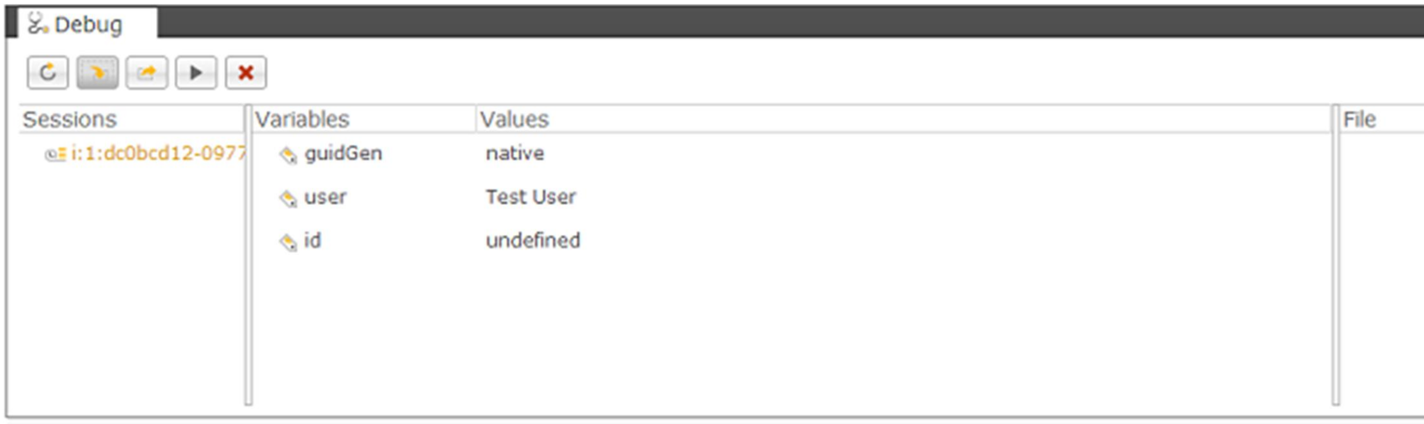


http://localhost:8080/dirigible/js-debug/DebuggerDemo/require_service.js

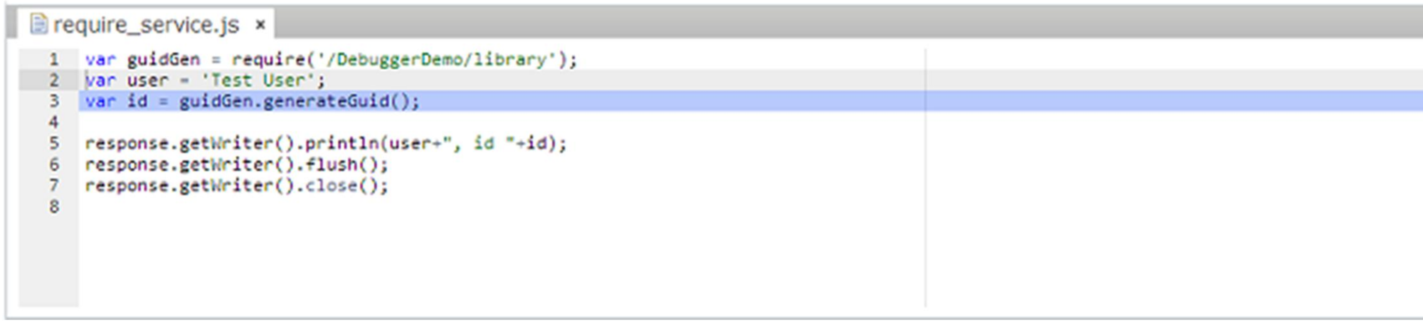
A new **Debug Session** was started.
Press **Refresh** button, select session and press **Step Into** button.



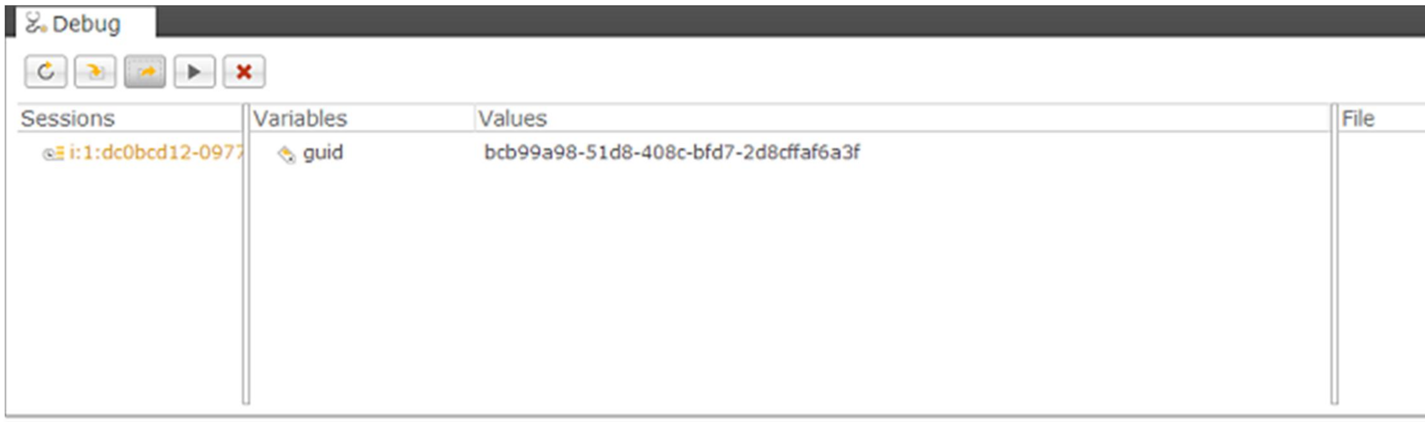
Continue debugging.



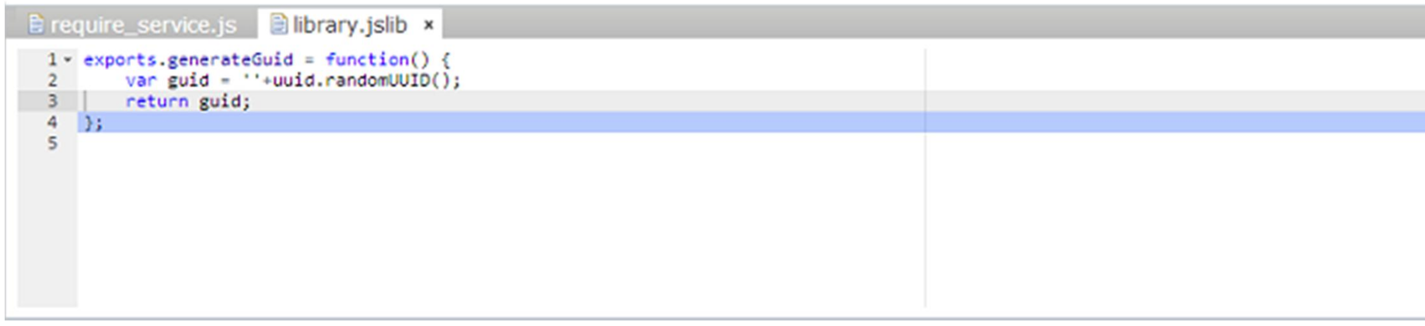
Variables	Values
guidGen	native
user	Test User
id	undefined



```
var guidGen = require('/DebuggerDemo/library');
var user = 'Test User';
var id = guidGen.generateGuid();

response.getWriter().println(user+", id "+id);
response.getWriter().flush();
response.getWriter().close();

```



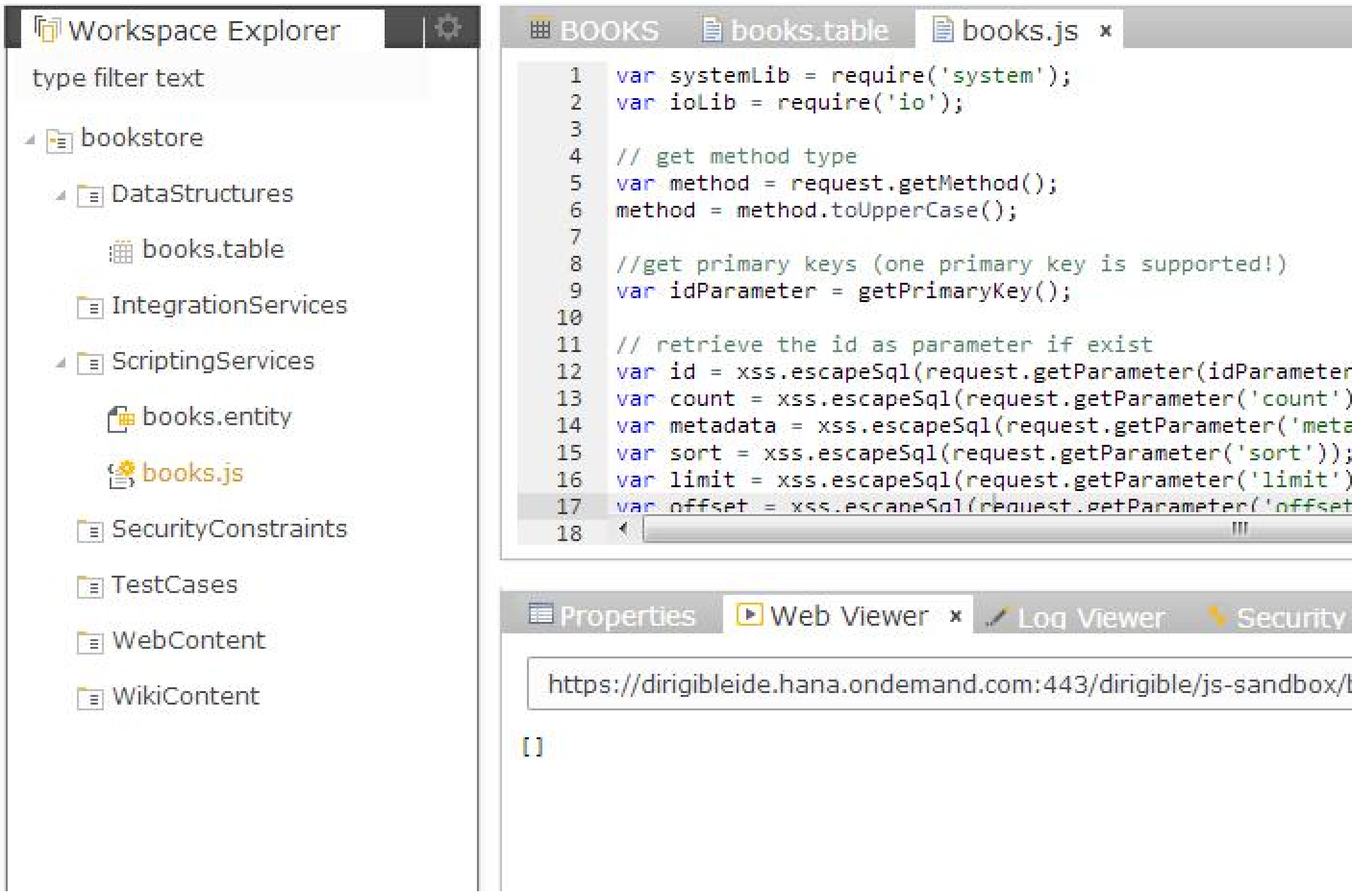
Variables	Values
guid	bcb99a98-51d8-408c-bfd7-2d8cffaf6a3f



```
exports.generateGuid = function() {
    var guid = ''+uuid.randomUUID();
    return guid;
};

```

Source Editor

Technically Source Editor provided by the toolkit is embedded version of well known ACE Editor in RAP environment.
It supports syntax highlighting of mostly used development languages and data formats such as JavaScript, Ruby, Groovy, JSON, XML, HTML and many more.



```
var systemLib = require('system');
var ioLib = require('io');

// get method type
var method = request.getMethod();
method = method.toUpperCase();

//get primary keys (one primary key is supported!)
var idParameter = getPrimaryKey();

// retrieve the id as parameter if exist
var id = xss.escapeSql(request.getParameter(idParameter
var count = xss.escapeSql(request.getParameter('count')
var metadata = xss.escapeSql(request.getParameter('meta
var sort = xss.escapeSql(request.getParameter('sort'));
var limit = xss.escapeSql(request.getParameter('limit')
var offset = xss.escapeSql(request.getParameter('offset
```

```
https://dirigibleide.hana.ondemand.com:443/dirigible/js-sandbox/
```

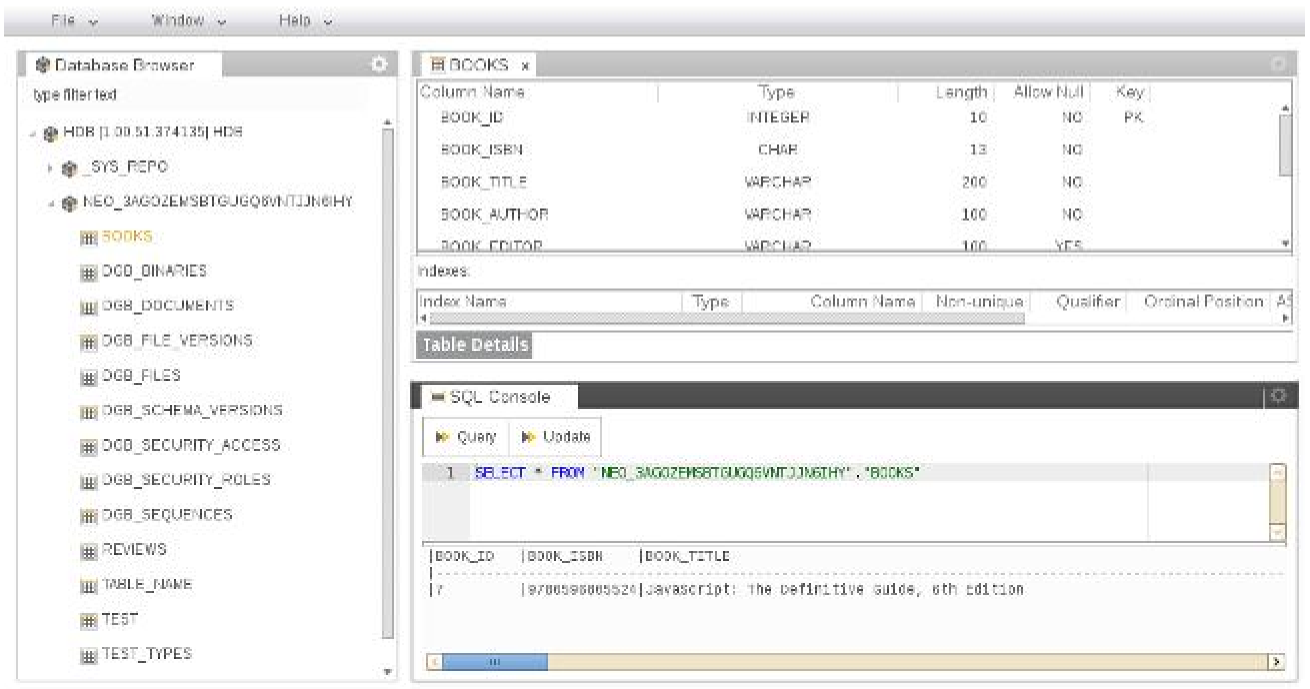
```
[]
```

Cut, Copy, Paste as well as Undo/Redo actions are built-in.
Key bindings aligned to the standard text editors also contribute to the increasing of development productivity.

SQL Console

Generic and in the same time most powerful and most dangerous tool for database management - SQL Console.

There are two separated areas - at the top you can enter the SQL script compliant to the underlying database system.
At the bottom - you get the result of the execution in fixed text format.



Column Name	Type	Length	Allow Null	Key
BOOK_ID	INTEGER	10	NO	PK
BOOK_ISBN	CHAR	13	NO	
BOOK_TITLE	VARCHAR	200	NO	
BOOK_AUTHOR	VARCHAR	100	NO	
BOOK_EDITOR	VARCHAR	100	YES	

Index Name	Type	Column Name	Non-unique	Qualifier	Ordinal Position	AS

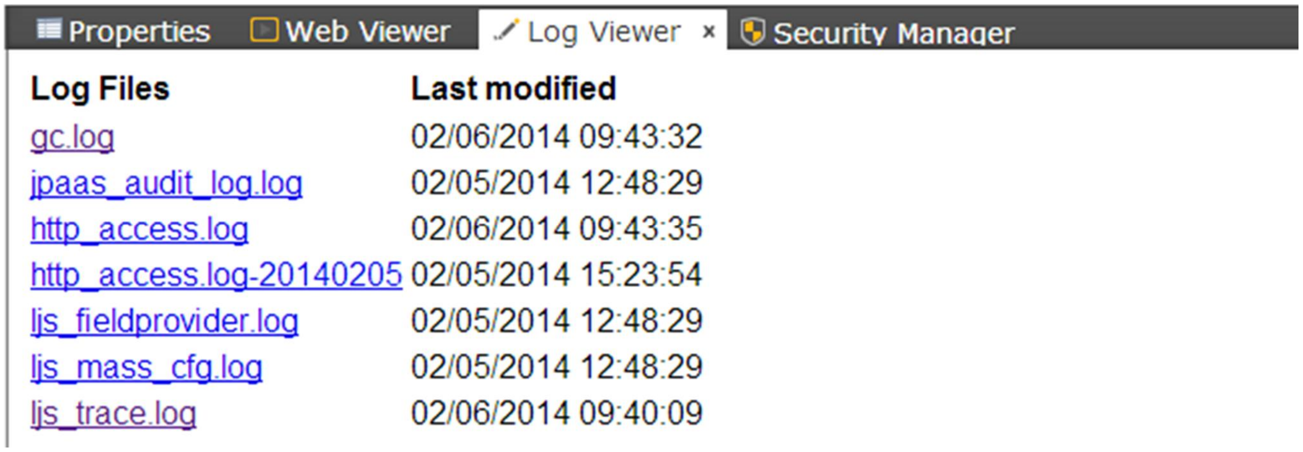
```
SELECT * FROM "NEO_3AGOZEMSBTGUGQ6VNTJJN6IHY"."BOOKS"
```

```
|BOOK_ID    |BOOK_ISBN    |BOOK_TITLE
|------------------------------------------------------------
|7          |9780596805524|Javascript: The Definitive Guide, 6th Edition
```

There is also separation between the **Query** and **Update** statements, which the user can execute depending on his/her roles.

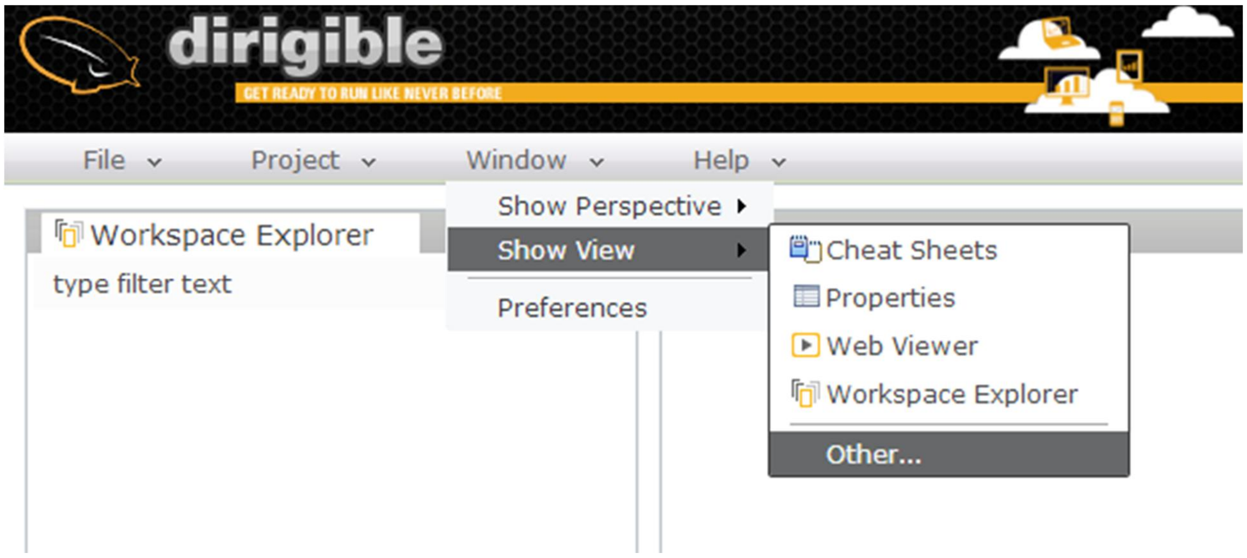
Log Viewer

List all available log files

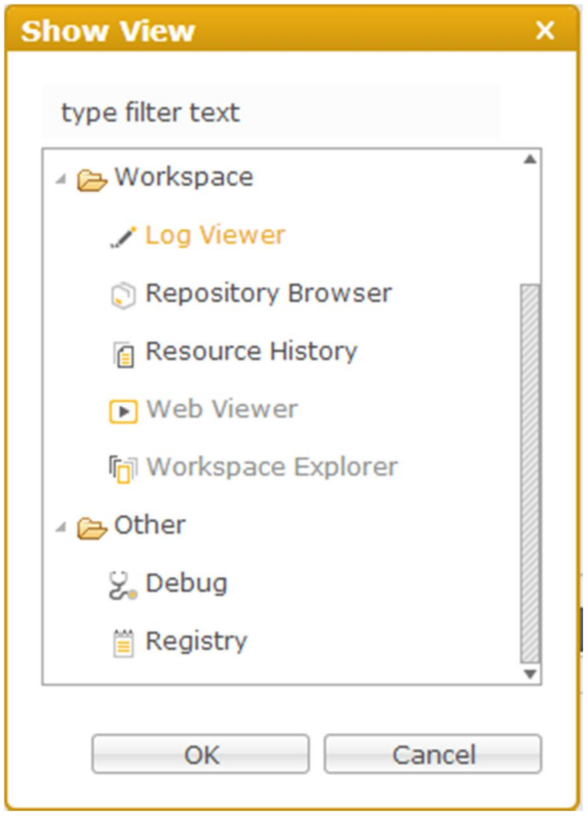


Log Files	Last modified
gc.log	02/06/2014 09:43:32
jpaas_audit_log.log	02/05/2014 12:48:29
http_access.log	02/06/2014 09:43:35
http_access.log-20140205	02/05/2014 15:23:54
ljs_fieldprovider.log	02/05/2014 12:48:29
ljs_mass_cfg.log	02/05/2014 12:48:29
ljs_trace.log	02/06/2014 09:40:09

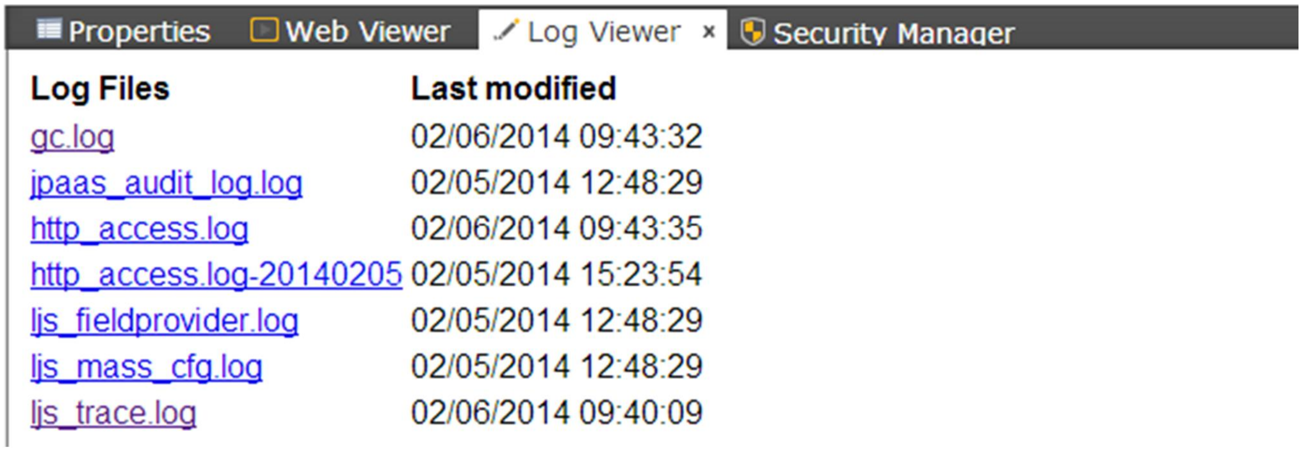
If the **Log Viewer** is not available, navigate to **Window->Show View->Other...**



Then select from **Workspace** folder **Log Viewer**

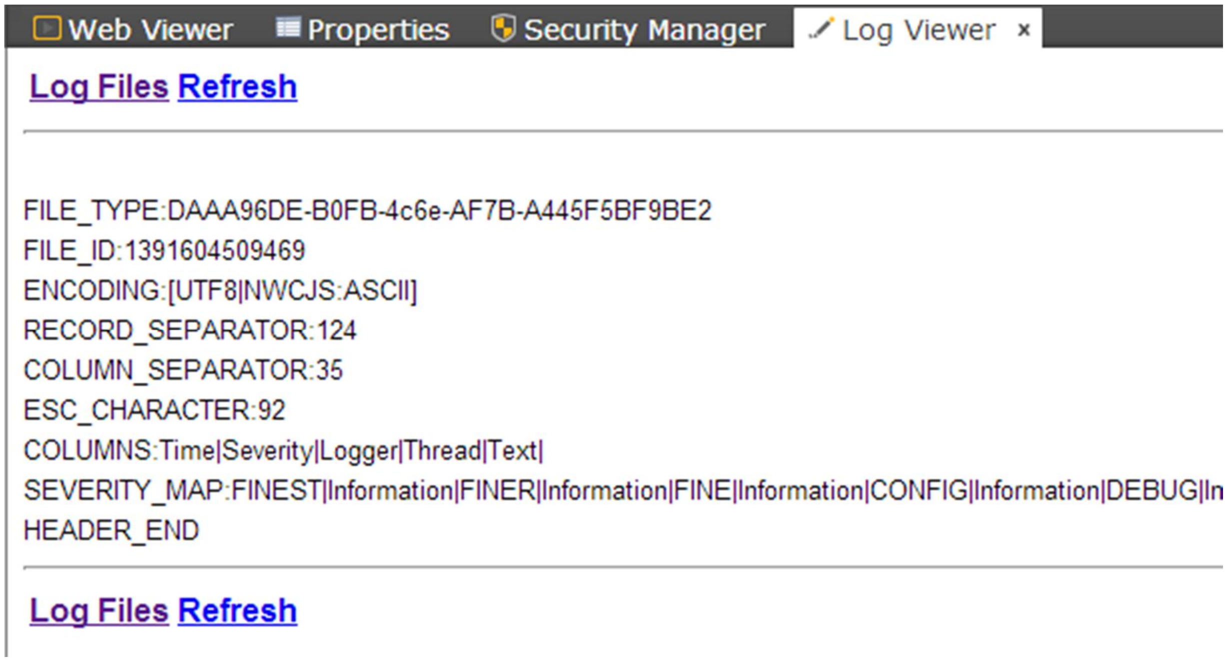


Now the **Log Viewer** is available



Log Files	Last modified
gc.log	02/06/2014 09:43:32
jpaas_audit_log.log	02/05/2014 12:48:29
http_access.log	02/06/2014 09:43:35
http_access.log-20140205	02/05/2014 15:23:54
ljs_fieldprovider.log	02/05/2014 12:48:29
ljs_mass_cfg.log	02/05/2014 12:48:29
ljs_trace.log	02/06/2014 09:40:09

Open any log file, for example **ljs_mass_cfg.log**



```
FILE_TYPE:DAAA96DE-B0FB-4c6e-AF7B-A445F5BF9BE2
FILE_ID:1391604509469
ENCODING:[UTF8|NWCJS:ASCII]
RECORD_SEPARATOR:124
COLUMN_SEPARATOR:35
ESC_CHARACTER:92
COLUMNS:Time|Severity|Logger|Thread|Text|
SEVERITY_MAP:FINEST|Information|FINER|Information|FINE|Information|CONFIG|Information|DEBUG|In
HEADER_END
```

Log Files - go back to the list of available log files
Refresh - refresh the content of the opened log file (needs to be pressed if new logging event occurred)

More about the server side logs and configurations can be found at: logging service section.

Runtime Services

There are several REST services available at runtime, which can give you another communication channel with Dirigible containers.

- Export
- Operational

- Memory
- Search
- Logging
- Access Log
- Repository

Operational Service

Operational Service exposes some utility functions.

The endpoint is: **/op**

- To get the current logged-in user name:

Parameter: **user**

http //<host>:<port>/dirigible/ **op?user**

- To log-out from the the current user session:

Parameter: **logout**

http //<host>:<port>/dirigible/ **op?logout**

Memory Service

Memory Service dumps the current information from the Runtime.

The endpoint is: **/memory**

The result is in JSON format, e.g.:

```
{ "totalMemory":126353408, "availableProcessors":4, "maxMemory":1877475328, "freeMemory":10
9053320 }
```

To retrieve the chart compliant data:

Parameter: **log**
The endpoint is: **/memory?log**

Search Service

Description of the topic here...

Logging Service

Logging Service exposes the list of log files as well as theirs content.

The endpoint is: **/logging**

- To list all the available log files use:

http //<host>:<port>/dirigible/ **logging**

- To retrieve the content of a log file (e.g. ljs_trace.log) use:

Parameter: **log**

http //<host>:<port>/dirigible/ **logging?log=ljs_trace.log**

The user interface in the IDE for accessing the logs is Log Viewer.

Access Log Service

Via the Access Logs Service one can manage the locations to be filtered and registered as well as to receive the comprehensive information about the accessed ones for the latest time period.

The endpoint is: **/acclog**

For Management of Locations:

GET: http //<host>:<port>/dirigible/ **acclog**

POST: http //<host>:<port>/dirigible/ **acclog** /<project_name>/<location>

DELETE: http //<host>:<port>/dirigible/ **acclog** /<project_name>/<location>

DELETE: http //<host>:<port>/dirigible/ **acclog** /all

GET: http //<host>:<port>/dirigible/ **acclog** /locations

For chart compliant data:

Parameter **hitsPerPattern** - hits count calculated grouped by the locations above
GET: http //<host>:<port>/dirigible/ **acclog** ? **hitsPerPattern**

Parameter **hitsPerProject** - hits count calculated grouped by the project names
GET: http //<host>:<port>/dirigible/ **acclog** ? **hitsPerProject**

Parameter **hitsPerURI** - hits count calculated grouped by the actual requested URI
GET: http //<host>:<port>/dirigible/ **acclog** ? **hitsPerURI**

Parameter **hitsByURI** - hits count calculated grouped hierarchically
GET: http //<host>:<port>/dirigible/ **acclog** ? **hitsByURI**

Repository Service

Repository Service gives full access to the Dirigible Repository API.

The endpoint is: **/repository**

To be able to use the service:

User must be assigned to Role: **Repository**
Basic Authentication headers must be provided
Header **Accept** must be provided with value **application/json**

- To get the catalog of the full content:

http //<host>:<port>/dirigible/ **repository**

- To get the index of a given collection:

http //<host>:<port>/dirigible/ **repository/db/dirigible**

```
{ "name" : "root", "path" : "/", "files" : [ { "name" : "registry", "path" : "/dirigible/re
pository/db/dirigible/registry/", "folder" : true }, { "name" : "sandbox", "path" : "/dirig
ible/repository/db/dirigible/sandbox/", "folder" : true }, { "name" : "templates", "path" :
 "/dirigible/repository/db/dirigible/templates/", "folder" : true }, { "name" : "users", "p
ath" : "/dirigible/repository/db/dirigible/users/", "folder" : true } ] }
```

- To get the content of a given artifact:

http //<host>:<port>/dirigible/
repository/db/dirigible/registry/WebContent/<my_web_project>/index.html