**[Eclipse Dirigible](http://www.dirigible.io/" \t "_blank)** is an open source cloud development platform, part of the Eclipse Foundation and the top-level **Eclipse Cloud Development** project. The ultimate goal of the platform is to provide software developers with the right toolset for building, running, and operating business applications in the cloud. To achieve this goal, Dirigible provides both independent **Design Time** and **Runtime** components. It provides both development tools and runtime environment.

One of the strongest advantages of Dirigible is the **In-System Development** model. Right from the early days of Dirigible, it was clear that it is going to be the platform for **Business Applications Development** in the cloud and not just another general purpose IDE in the browser. The reason for that decision is pretty simple - “One size doesn’t fit all”! Making a choice between providing “In-System Development” in the cloud and adding support for a new language is really easy. The new language doesn’t really add much to the uniqueness and usability of the platform, as the In-System development model does! The goal of the In-System development model is to ultimately change the state of the system while it’s up and running, without affecting the overall performance and without service degradation.

Some of the features of Eclipse Dirigible are:

* Database

Enables you to model data structure artefacts in JSON format, and to create actual tables and views during activation/publishing.

Database perspective is full-fledged including Database Explorer view, SQL Console, meta-data inspectors, and more.

You can auto-generate mock-up data based on given table layout.

* Server-Side

Uses some of the most popular dynamic languages such as JavaScript, Groovy\* and Ruby\* for creating server-side services.

Establishes an Enterprise JavaScript API that can be used via the CommonJS modularization.

Generates fully compliant RESTful services, including meta-data support on existing database artefacts.

* Client-Side

Supports user interface generation for the RESTful services based on widely used frameworks, such as Bootstrap, jQuery, AngularJS, and OpenUI5.

It provides WYSIWYG editor for both standard HTML5 and Bootstrap-based complex controls.

It supports wiki scripting via Confluence format.

* Processes

Provides a built-in process engine based on declarative definitions of integration and orchestration of business process flows.

* Extensions

Supports declarative definition of extension points and extensions in order to achieve:

* Better adaptability

Better visibility of dependencies

Simplified life-cycle management

* Mobile

Provides an integration with Tabris.js mobile framework that allows you to develop native iOS and Android mobile applications, written entirely in JavaScript.

**‘SQL Services’ support**

It will provide the following:

1. Editor for SQL script with highlighting of the keywords
2. Icon in Workspace and Repository views showing that \*.sql files are recognized
3. Activator/Publisher, which will take care of the transfer of the SQL artifacts to Sandbox/Registry
4. Runtime dispatcher, which will provide the endpoint for access for these services
5. Runtime executor, which will take the artifact and will do the processing
6. Infrastructure - pom.xml, config.ini, feature.xml
7. User interface for endpoints in Registry
8. Sample template for SQL Service

**Editor for SQL**

Luckily we support two web editors in Dirigible - Orion and ACE. The later has good support for SQL Language, hence we can use it directly. Be sure that you enable the support of your language in the corresponding editor by adding the file extension to the editor’s ‘extensions’ parameter in the plugin.xml. In this case in plugin **org.eclipse.dirigible.ide.editor.ace**, extension point **org.eclipse.ui.editors**, class **org.eclipse.dirigible.ide.editor.ace.AceEditor**.



**Icon for \*.sql files**

Add an icon in the **resources** folder of the **org.eclipse.dirigible.ide.repository.ui** plugin, e.g. **icon-sql.png**. Add a reference of the icon and the necessary file extension in **org.eclipse.dirigible.ide.repository.ui.viewer.AbstractArtifactLabelProvider** similar like the other cases.

http://www.dirigible.io/img/posts/icon_sql.png

**Publisher adaptation**

There are a few adaptation that can enable \*.sql artifact to be considered as supported scripting services. To do that:

1. Add the corresponding constant for SQL extension in **ARTIFACT\_EXTENSION** in the class **org.eclipse.dirigible.repository.api.ICommonConstants** e.g.

 public static final String SQL = ".sql";

1. Add **SQL\_CONTAINER\_MAPPING** and **SQL\_SANDBOX\_MAPPING** in class **org.eclipse.dirigible.ide.common.CommonParameters** in similar way like the others.
2. Add the corresponding artifact extension and mappings in the static list and maps in class **org.eclipse.dirigible.ide.scripts.publish.ScriptsPublisher** in similar way like the others.

We are done at the IDE side! Now we go to the Runtime to implement the execution engine.

**Engine for SQL**

Create a new plugin which will contain all the execution engine related artifacts for the SQL support. As a template you can use already available for Java **org.eclipse.dirigible.runtime.java** - e.g. **org.eclipse.dirigible.runtime.sql**

Add corresponding **ENGINE\_TYPE**

 public static final String SQL ="sql";

in **org.eclipse.dirigible.repository.api.ICommonConstants**

Do the same for **ENGINE\_ALIAS**

Modify/check:

1. **.project** file
2. **MANIFEST.MF** file names, dependencies, exported packages
3. In **OSGi-INF** folder create a sql-executor.xml with corresponding references
4. **plugin.xml** fill the corresponding servlets and filters
5. **pom.xml**
6. Add the module definition in the parent’s pom.xml, e.g.

< module>org.eclipse.dirigible.runtime.sql< /module>

In the source folder (*src*), you should finally have at least:

1. **org.eclipse.dirigible.runtime.filter.SQLRegistrySecureFilter.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | package org**.**eclipse**.**dirigible**.**runtime**.**filter**;**  **public** **class** **SQLRegistrySecureFilter** **extends** AbstractRegistrySecureFilter **{**  **private** **static** **final** String SQL\_SECURED\_MAPPING **=** "/services/sql-secured"**;** *//$NON-NLS-1$*  @Override  **protected** String **getSecuredMapping()** **{**  **return** SQL\_SECURED\_MAPPING**;**  **}**  **}** |

1. **org.eclipse.dirigible.runtime.registry.SQLRegistryServlet.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package org**.**eclipse**.**dirigible**.**runtime**.**registry**;**  **public** **class** **SQLRegistryServlet** **extends** AbstractRegistryServiceServlet **{**  **private** **static** **final** **long** serialVersionUID **=** **-**7292896045277229573L**;**  @Override  **protected** String **getServletMapping()** **{**  **return** "/sql/"**;**  **}**  @Override  **protected** String **getFileExtension()** **{**  **return** ".sql"**;**  **}**  @Override  **protected** String **getRequestProcessingFailedMessage()** **{**  **return** Messages**.**getString**(**"JavascriptRegistryServlet.REQUEST\_PROCESSING\_FAILED\_S"**);**  **}**  **}** |

1. **org.eclipse.dirigible.runtime.sql.SQLExecutor.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116 | package org**.**eclipse**.**dirigible**.**runtime**.**sql**;**  import java.io.IOException**;**  import java.sql.Connection**;**  import java.sql.PreparedStatement**;**  import java.sql.ResultSet**;**  import java.sql.ResultSetMetaData**;**  import java.util.ArrayList**;**  import java.util.HashMap**;**  import java.util.List**;**  import java.util.Map**;**  import javax.servlet.http.HttpServletRequest**;**  import javax.servlet.http.HttpServletResponse**;**  import javax.sql.DataSource**;**  import org.eclipse.dirigible.repository.api.ICommonConstants**;**  import org.eclipse.dirigible.repository.api.IRepository**;**  import org.eclipse.dirigible.repository.logging.Logger**;**  import org.eclipse.dirigible.runtime.repository.RepositoryFacade**;**  import org.eclipse.dirigible.runtime.scripting.AbstractScriptExecutor**;**  import com.google.gson.Gson**;**  import com.google.gson.JsonArray**;**  import com.google.gson.JsonObject**;**  import com.google.gson.JsonPrimitive**;**  **public** **class** **SQLExecutor** **extends** AbstractScriptExecutor **{**  **private** **static** **final** String SQL\_MODULE\_NAME\_CANNOT\_BE\_NULL **=** "SQL module name cannot be null."**;**  **private** **static** **final** Logger logger **=** Logger**.**getLogger**(**SQLExecutor**.**class**);**  **private** IRepository repository**;**  **private** String**[]** rootPaths**;**  **private** Map**<**String**,** Object**>** defaultVariables**;**  **private** String classpath**;**  **public** **SQLExecutor(**IRepository repository**,** String**...** rootPaths**)** **{**  **this.**repository **=** repository**;**  **this.**rootPaths **=** rootPaths**;**  **this.**defaultVariables **=** **new** HashMap**<**String**,** Object**>();**  **this.**classpath **=** classpath**;**  **}**  @Override  **public** Object **executeServiceModule(**HttpServletRequest request**,** HttpServletResponse response**,** Object input**,** String module**,**  Map**<**Object**,** Object**>** executionContext**)** **throws** IOException **{**  String result **=** **null;**  **try** **{**  logger**.**debug**(**"entering: executeServiceModule()"**);** *//$NON-NLS-1$*  logger**.**debug**(**"module=" **+** module**);** *//$NON-NLS-1$*  **if** **(**module **==** **null)** **{**  **throw** **new** **IOException(**SQL\_MODULE\_NAME\_CANNOT\_BE\_NULL**);**  **}**  String sqlSource **=** **new** String**(**retrieveModule**(**repository**,** module**,** ""**,** rootPaths**).**getContent**());**  DataSource dataSource **=** RepositoryFacade**.**getInstance**().**getDataSource**();**  Connection connection **=** **null;**  **try** **{**  connection **=** dataSource**.**getConnection**();**  PreparedStatement pstmt **=** connection**.**prepareStatement**(**sqlSource**);**  ResultSet rs **=** pstmt**.**executeQuery**();**  *// get column names*  ResultSetMetaData rsmd **=** rs**.**getMetaData**();**  **int** columnCnt **=** rsmd**.**getColumnCount**();**  List**<**String**>** columnNames **=** **new** ArrayList**<**String**>();**  **for** **(int** i **=** 1**;** i **<=** columnCnt**;** i**++)** **{**  columnNames**.**add**(**rsmd**.**getColumnName**(**i**).**toUpperCase**());**  **}**  JsonArray array **=** **new** JsonArray**();**  **while** **(**rs**.**next**())** **{**  JsonObject obj **=** **new** JsonObject**();**  **for** **(int** i **=** 1**;** i **<=** columnCnt**;** i**++)** **{**  String key **=** columnNames**.**get**(**i **-** 1**);**  String value **=** rs**.**getString**(**i**);**  obj**.**add**(**key**,** **new** JsonPrimitive**(**value **!=** **null** **?** value **:** ""**));**  **}**  array**.**add**(**obj**);**  **}**  result **=** **new** Gson**().**toJson**(**array**);**  rs**.**close**();**  pstmt**.**close**();**  **}** **finally** **{**  **if** **(**connection **!=** **null)** **{**  connection**.**close**();**  **}**  **}**  **}** **catch** **(**Exception e**)** **{**  logger**.**error**(**e**.**getMessage**(),** e**);**  **throw** **new** **IOException(**e**);**  **}**  **return** result**;**  **}**  @Override  **protected** **void** **registerDefaultVariable(**Object scope**,** String name**,** Object value**)** **{**  defaultVariables**.**put**(**name**,** value**);**  **}**  @Override  **protected** String **getModuleType(**String path**)** **{**  **return** ICommonConstants**.**ARTIFACT\_TYPE**.**SCRIPTING\_SERVICES**;**  **}**  **}** |

1. **org.eclipse.dirigible.runtime.sql.SQLServlet.java**
2. **org.eclipse.dirigible.runtime.sql.SQLSandboxServlet.java**
3. **org.eclipse.dirigible.runtime.sql.SQLSecuredServlet.java**
4. **org.eclipse.dirigible.runtime.sql.SQLScriptExecutorProvider.java**

**Include the Plugin as a Feature**

There is a feature for the runtime plugins in the project **p2.runtime.feature** Add the SQL plugin to the feature.xml accordingly

**Include the Plugin for Packaging**

You have to include just created plugin into the configuration files for Equinox OSGi:

1. In the project **releng/dirigible-all-tomcat** > sub-folder **src/main/webapp/WEB-INF/configuration** > file **config.ini**
2. In the project **releng/dirigible-runtime-tomcat** > sub-folder **src/main/webapp/WEB-INF/configuration** > file **config.ini**

be very careful with the white spaces in the beginning of the line

**Security Constrains in web.xml**

1. In the project **releng/dirigible-all-tomcat** > sub-folder **src/web/** > all files **web.xml** excluding **trial**
2. In the project **releng/dirigible-runtime-tomcat** > sub-folder **src/web/** > all files **web.xml** excluding **trial**

**Flows and Jobs Integration**

Luckily we have already implemented the extensibility in a way that **SQLScriptExecutorProvider** from above is automatically registered and can be used in Flows.

**Registry Section for SQL Services**

The plugin containing the registry user interface is **org.eclipse.dirigible.runtime.ui**

1. Create a file **sql.html** in the sub-folder **resources/ui/templates/scripting/sql**

|  |  |
| --- | --- |
| 1  2  3 | <div id="content" ng-include="'templates/default.html'"></div> |

1. Adapt **$routeProvider** in the **app.js** file by adding routing for **sql** pages
2. In the same file add a corresponding section in **$scope.homeData**
3. Add the controller itself in **default.js** defaultControllers.controller(‘SQLCtrl’, function($scope, $resource) { $scope.restService = $resource(‘../scripting/sql’); });
4. Do not forget to add some image file also
5. Add a manu item in the main menu - **menu.json**

**Template for SQL Scripting Service**

To complete the SQL support we can add at least one template to be available in the **New->ScriptinService** wizard. To do that, in the plugin **org.eclipse.dirigible.ide.template.ui.js**

1. Create file **sql-service.sql** under the folder **src/org/eclipse/dirigible/ide/template/ui/js/templates**

|  |  |
| --- | --- |
| 1  2  3 | **SELECT** **\*** **FROM** DGB\_FILES |

1. Add the definition in the **plugin.xml** accordingly

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <template  category="ScriptingServices"  image="/icons/sql-service.png"  location="/org/eclipse/dirigible/ide/template/ui/js/templates/sql-service.sql"  text="SQL Sample Query Service">  </template> |

1. Do not forget the icon as well
2. Add the default extension file recognition in **org.eclipse.dirigible.ide.template.ui.js.wizard.JavascriptServiceTemplateTargetLocationPage**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **...**  **}** **else** **if** **(**"/org/eclipse/dirigible/ide/template/ui/js/templates/sql-service.sql" *//$NON-NLS-1$*  **.**equals**(**model**.**getTemplate**().**getLocation**()))** **{**  jsOrLibExt **=** "sql"**;** *//$NON-NLS-1$*  **}**  **...** |

