Introduction to Rest Framework Compiler

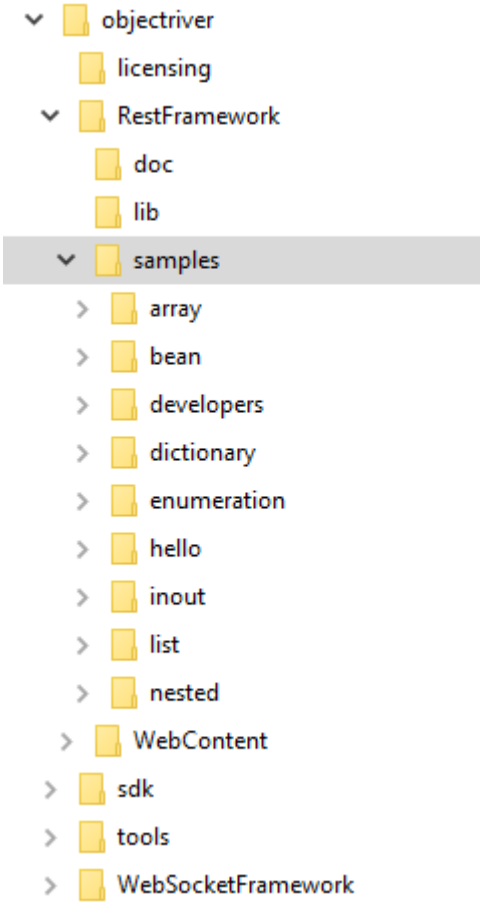
ObjectRiver has built a Cloud Compiler that is capable of parsing many metadata languages, and generating complex solutions.

This manual is describing the REST metadata needed to generate a JAXRS 2 compliant applications.

Download

http://www.objectriver.net/downloads.htm

Cloud Compiler SDK Directories



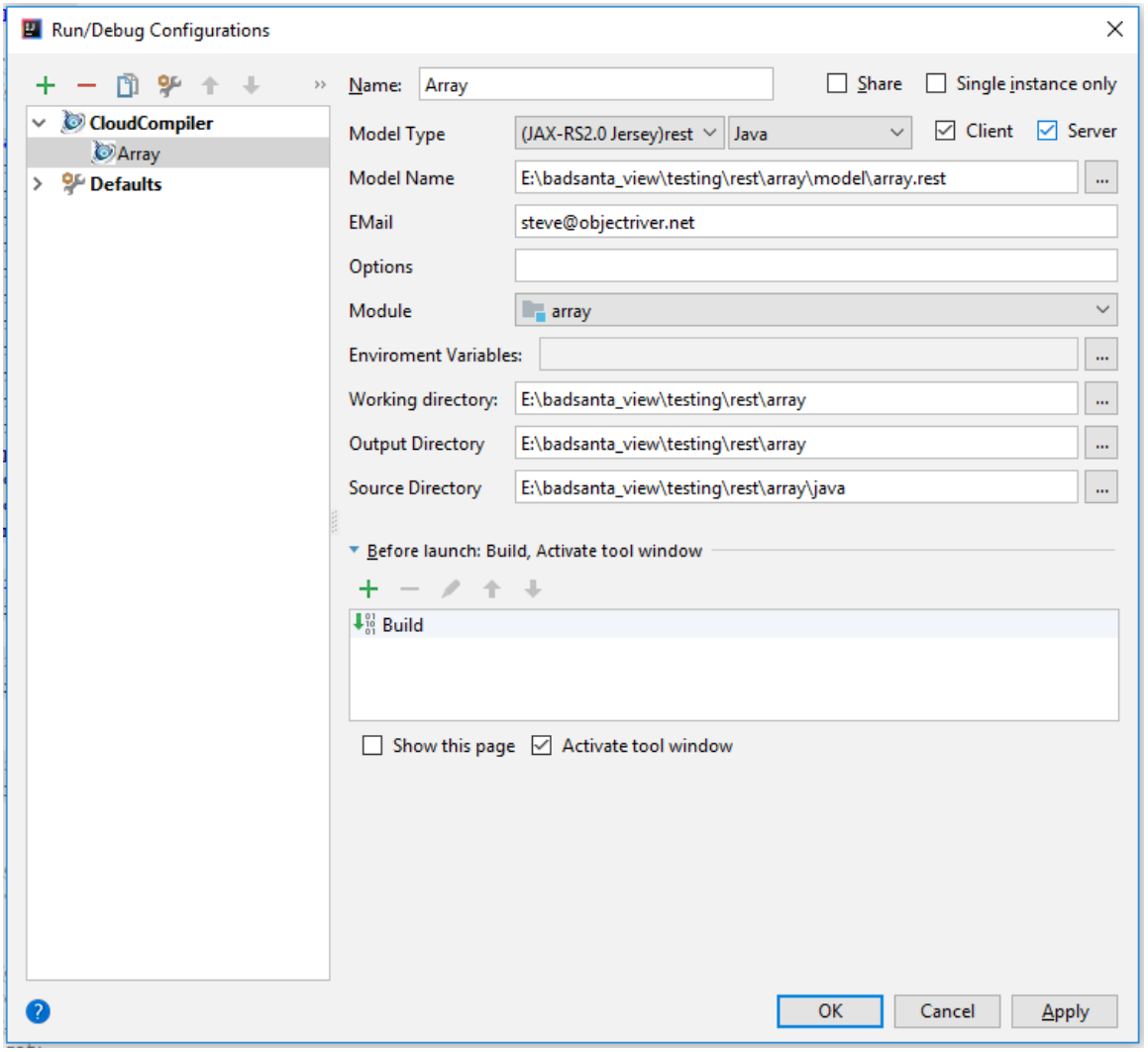
The cloud compiler tool kit is all jars files.

- objectriver/RestFramework/lib
  - RestFramework libraries with source
- objectriver/licensing
  - MIT license
- objectriver/doc
  - documentation
- objectriver/sdk
  - cloudcclient.jar
    ObjectRiver Cloud Compiler client, needed for Ant task if needed.
  - ObjectRiverCloudCompilerPlugin
    Intellij plugin for the Cloud Compiler
- objectriver/WebContent
  - libraries required for build server-side web servers.
- objectriver/tools
  - Addition libraries
- objectriver/RestFramework/samples/bean
  Test example for bean definitions.
- objectriver/WebSockerFramework
  - See documentation in objectriver/WebSockerFramework/doc

Executing Cloud Compiler

The compiler currently ships with a IDE plugin for Intellij. Installing this plugin is the easiest way to execute the compiler. Navigate Intelli plugin page and instll CompilerCompilerPlugin from disk.

Intellij Plugin



Hello World

Here is a “Hello World” example showing how the Rest Cloud Compiler generate a synchronous message.

Interface Definition

```
Application Hello = com.companyname {
    Endpoint HelloEndpoint = 510 {
        Exception HelloException {};
        Server Stateless { // Stateless, Session, Singleton
         @Path("hello_query")
            @Produces(JSON)
            @GET List<String> hello_query(
                 @QueryParam("hello") String hello) throws HelloException;

         @Path("hello_path/{hello}/{world}")
            @Produces(JSON)
            @GET List<String> hello_path(
                 @PathParam("hello") String hello,
                 @PathParam("world") String world) throws HelloException;

         @Path("hello_body")
            @Produces(JSON)
            @Consumes(JSON)
            @POST List<String> hello_body(
                 @In String hello) throws HelloException;
      };
    };
};
```

Application

Syntax: **Application Simple = com.companyname { … }**
Application describes the namespace for the REST project. Application name will be used as a prefix on much of the generated source code. From the example above the **com.companyname** will be used for the Java package location.

Endpoint

Syntax: **Endpoint Simple= 1 { … }**
Endpoint describes the interface name for the procedures being distributed. This statement also has the version number for the interface. Version is not currently implemented because the system supports transient variables.

Data Types

- String
  String/string both define ASCII string type. **String** type can be null, where **string** is defined as not null. Empty strings (“”) are considered null.

- Short/Integer/Long
  Short/short, Integer/int, Long/long integer types. Short/Integer/Long can be null, where short/int/long can not be null.

- **Float/Double**

Float/float, Double/double are decimal types. Float/Double can be null, where float/double can not be null.

- Date
  Date/date represent a calendar date. Date can be null, where date can not be null.

- Timestamp
  Timestamp/timestamp represent date and time. Timestamp can be null, where timestamp can not be null.

- Boolean
  Boolean/boolean represent Boolean value. Boolean can be null, where Boolean can not be null.

- Character
  Character/char represents a single character. Character can be null, where char can not be null.

- Bean
  Bean is an aggregate type for defining classes..

- Dictionary
  Dictionary is map of data types, which are mapped with strings.

Client & Server Method Definitions

```
Server Stateless {
        <<method definitions >>
};
```

Currently implementation only supports Stateless.

Method Defintions

- Arguments
  Methods have arguments and a return value. Arguments can be annotated with **@In, @Inout, @Out**, and **@NotNull** modifiers that indicate the direction in which the argument must be passed. **@NotNull** just indicate that the argument can not be null, and a marshalling error will be thrown.
- Exceptions
  Methods can throw application based exceptions. Exceptions can be defined like the following example.
  Exception SimpleException {};
  Exceptions can also contain members that are defined with the curly braces as follows. **Exception SimpleException { Integer code; };**

Data Structures

- Enumeration

```
Enumeration MyMonth { Jan=1, Feb=2, Mar=3, Apr=4, May=5, Jun=6,
        Jul=7, Aug=8, Sept=9, Oct=10, Nov=11, Dec=12 };
Enumeration MyDay { Sun=1, Mon=2, Tues=3, Wed=4, Thu=5, Fri=6, Sat=7 };
```

- Bean

```
Bean MyTime {
   int hours;
   int minutes;
   int seconds;
```

```
};
Bean MyDate {
        int year;
        MyMonth month;
        MyDay day;
};
Bean MyDateTime {
    MyDate date;
     MyTime time;
};
```

Beans are aggregated data types used for constructing records or just collections of other data types.

- Dictionary

```
Dictionary<Integer> IntegerDictionary;
Bean FooBean {
    IntegerDictionary intDict;
};
Dictionary<FooBean> FooBeanDictionary;
```

- List

```
List<String> args
```

List are collections of data types.

- Exceptions

```
Exception MyException { int code;};
```

Exceptions are beans which by default contain a String message, and a Throwable cause elements. Exception may contain additional members. Note: Exceptions are not versioned.

Three Server choices

- **IOT (Internet Of Things) Server**.
  The toolkit ships Oracle/Grizzly libraries for running a service within your IDE. This allows direct debugging of server implementation.
- **Docker Tomcat Container**
  Generation of Alpine/Tomcat container. Server runs in native Tomcat/JAXRS-2 environment.
- **War file for web server deployment**.
  Generated War file for direct deployment to web server.