Release Notes

Introduction

Thanks for downloading the ObjectRiver toolkit. This toolkit has been developed over the last 15 years to support our consulting jobs.

The back-end templating language is like no other, and is capable of cloning any programming pattern related to API's or databases.

A couple things to know about the implementation.

1.) Create Once templates

If you run the compiler more than once, you notice that cloud compiler will not overwrite some files. These files are implementation files, you can modify to build your application.

2.) Read-only source

If there is a need to modify some generated source that is not generated from a create once template, you can make that file read-only and the cloud complier will not over write it.

3.) New language support

If there is a language implementation that does not exist, and you are a subject material expert in that language. Look at implementing that language (client or server) with our developers example. We can partner in developing templates for your implementation.

1. Angular TypeScript Client

Steps needed to succeed:

1.) This demo was build in Angular version 6 and 11. Before you generate to setup the environment you need to run the following node commands
   a. cd <project>
   b. ng new <project name> --directory ./
   c. npm install json2typescript@latest
   d. npm install ng-pick-datetime@latest
   e. Run cloudcompiler for angular.
   f. ng build
   g. If it does not build
      i. copy **package.json.thisworks** package.json
      ii. npm install
   h. ng build
   i. ng serve
   j. Goto Browser local:4200

2. Angular WebSocket Client

   a. Angular WebSocket clients are generated from REST metadata. This is because REST was the first implementation of the angular client and we also what to support conversion of OpenAPI implementations.

b. For the WebSocket server, you need run the cloud compiler "Meta -WebSockets" to convert your REST metadata to the WebSocket definitions.

3. Angular API Only:

a. For Angular API only, the system generates a small GUI on top of the API. Unfortunately, API GUI is kind of clunky because I could not figure out how to navigate to the same page, so there is a results page.

4. Intellij/WebStorm:

a.) The only differences between Intellij and WebStorm is the lack of Ant support in WebStorm. Ant is typically used for creating War files, and docker packaging.

b.) We hope to develop a VC Code extension at some point.

5. WebSocket Java client to Node Server:

a.) If you are trying this, you need to modify the <project>Client.java to remove BinaryStream encoder because node does not support Binary encoding at this time.

6.) Start Java/Node Docker Server:

a.) Open Ant script to create docker image and start server.
This typically the same process for Java and Node.

7.) Java Grizzley Server for Rest and WebSockets:

a.) The system ships with Tyrus, and Grizzly jar files to run Grizzly based servers directly in the IDE. These jar files are not included in the war, because the webserver will supply WebSockets.

8.) Java dependencies:

a.) When you first generate a Java project, you need to go the Intellij dependencies and select pre-defined libraries that have been generated. The cloudcompiler cannot modify the .iml file where these are defined.