Quick start

Prerequisites: JsonSchemaDsl already installed. If you haven't yet, please read the Installation tutorial.

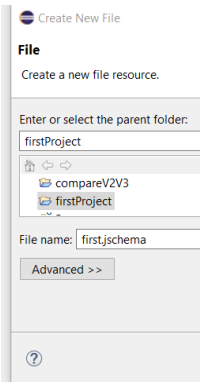
Introduction

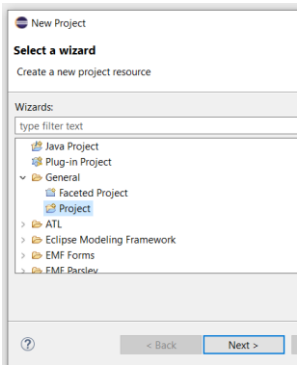
In this tutorial we show how to create and edit a Json Schema, how to generate the corresponding EMF artifacts and generate the language.
Here we also show how to use the generated language.

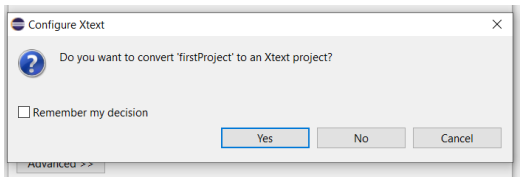
Create your first project

- Create a new general project with the New Project wizard
- Create a file with extension *.jschema*
- Click Yes when asked if you want to convert your project to an Xtext project.

.







- Open your first.jschema, and use the context assist (CTRL+space) to create your first JSON Schema.
- Open the Problems view to see the details of the validation errors and warnings.

Examples.

This section introduces a list of working examples.

Hello World examples.

Trivial examples are available. They are listed in the following folder:
https://github.com/lowcomote/jsonschemadsl.parent/tree/master/jsonschemadsl2ecore.trafo.opt/test
Such trivial examples will help you to understand the overall approach and its steps, as documented in [1]. In each subfolder you can find a jsonschema (with extension.jschema).In many of them you can find valid and invalid json for that schema.

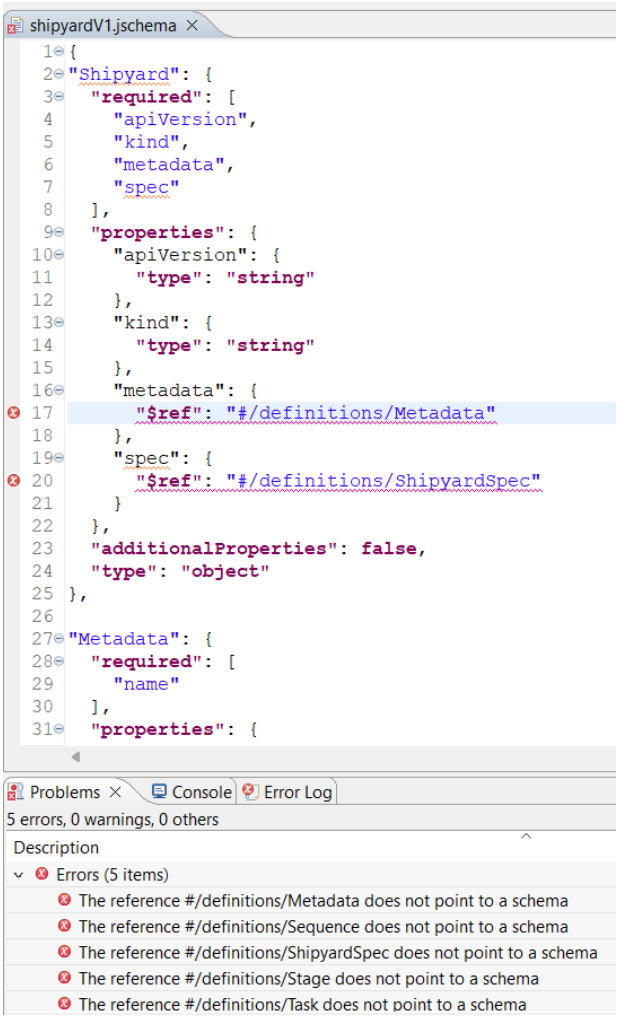
Shipyard DSL.

Shipyard, a JSON Schema-based language for workflow specification for Keptn (https://keptn.sh/), an open source tool for DevOps automation of cloud-native applications. The results of the case study show that proper editors as well as language evolution support from MDE can be reused and at the same time, the surface syntax of JSON is maintained. See [1] for further details.

Shipyard DSL versions are collected in
https://github.com/lowcomote/jsonschemadsl.parent/tree/master/samples/shipyardSchemas
The Shipyard DSL is defined by schema document, which, in turn, conforms to a given metaschema or JSON Schema Draft. We currently support JSON Schema Draft 7 (https://json-schema.org/).
In the following, we explain the approach steps applied to the Shipyard DSL.
The following steps (1-11) are expected to be performed by a Language Engineer, i.e., an user that wants to define the specification of a language.

Steps 1-4

1. Choose a Shipyard version among the ones made available (e.g., shipyardV1.jschema);
2. Repeat the steps described in **Create your first project** section for the chosen Shipyard version ()
    a. E.g., use *shipyardV1* for the project name
    b. E.g., use *shipyardV1.jschema* for the file name
    c. Click Yes when asked if you want to convert your project to an Xtext project.
3. Copy the content Shipyard version chosen at step 1 in the newly created file. You will see the keywords highlighted, the in line validation errors.
4. Open the Problems view to see the details of both errors and warnings. In the figure below, you can see the example shipyardV1.jschema with 5 errors resported due to that "$ref" points (using JSON Pointer) to the wrong place.

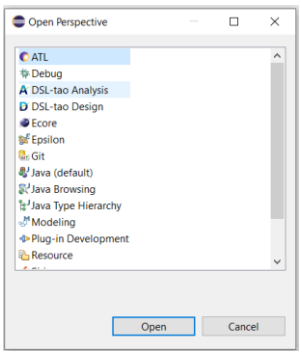


We invite the reader to repeat steps 1. to 4. choosing the *shipyardV4.jschema* at step 1. At the end of step 4, everything will be correct, with no errors or warnings.

Steps 5-11

(after choosing shipyardV4.json at step 1)

5. Save the shipyardV4.jschema.
6. A folder */model* will be created including the following artifacts:
    a. shipyardV4.jsongrammar
    b. shipyardV4.xmi
    c. shipyardV4Opt.ecore.
    d. shipyardV4.relatedSchemas
7. If you do not see the /model folder or it is empty, please refresh the /model or the whole project folder.
8. Open ATL perspective (
    a. Window->Perspective->Open Perspective->Other
    b. Select ATL and Click Open
9. Right click on the generated ecore metamodel and click on "Register Metamodel"



10. Under the root project folder, a fourth artifact (*shipyardV4Opt.ocl*) is created. Refresh the project folder if the OCL artifact does not appear.
11. If you open the ocl file it could happen that you see errors, because eclipse takes a while to synchronize with the newly registered ecore. But this is not a problem that forbids the remaining steps.

We invite the reader to practice with other shipyard versions. Not all versions of shipyard are valid json schema, as explained in the paper [1].
They can be considered a good base to experiment autonomously, as well as the tests in https://github.com/lowcomote/jsonschemadsl.parent/tree/master/jsonschemadsl2ecore.trafo.opt/test

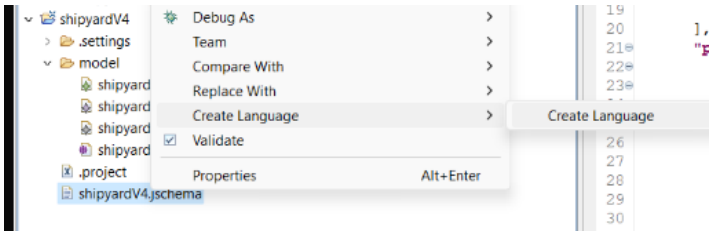
Note that so far JsonSchemaDSL is an initial prototype, and it does not support the generation of the artifacts for all the JSON Schema keywords.
We also invite the reader to create his own json schemas and repeat the entire cycle. Use the CTRL+space for the content assist.

The list of supported keywords/feature so far is available in

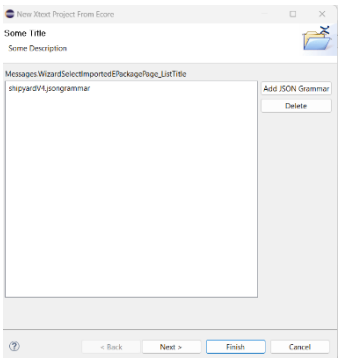
https://github.com/lowcomote/jsonschemadsl.parent/blob/master/FEATURES_LIST.md

Usage of the language defined by your Json Schema

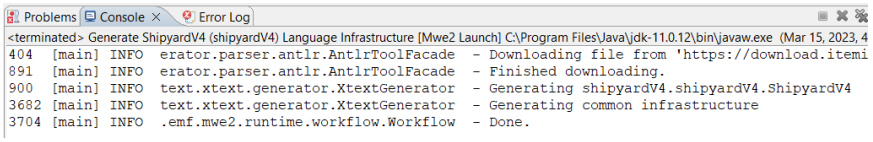
Once completed steps 1-8, the EMF/Xtext-based editor for the DSL defined by the .jschema artifact can be generated. Right click on the .jschema file and and the on the 'Create Language' option.



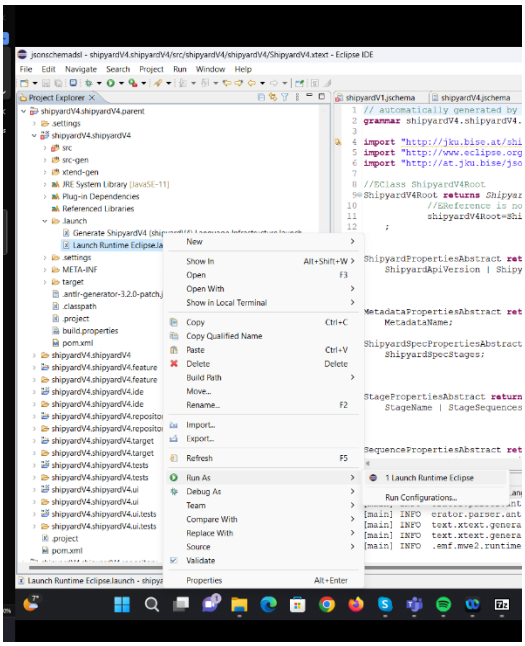
After few seconds the following popup appears.



The user has to click the ‘Finish’ button and wait that in the Console it appears the message that the work is done, like in the following picture.

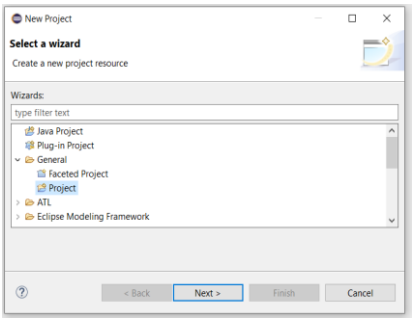


Once the language has been generated, the Runtime Eclipse can be launched as shown in the following picture.



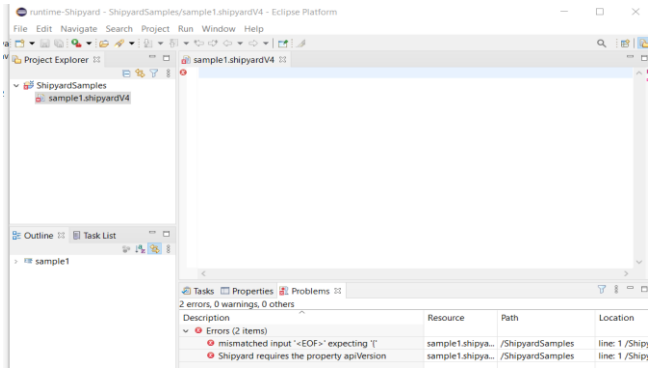
In the Runtime Eclipse a new simple project and create inside it a new file with the extension of the language (e.g., .shipyardV4).
In the launched Runtime Eclipse

- create a new general project with the New Project wizard (File->New->Project to open the wizard);



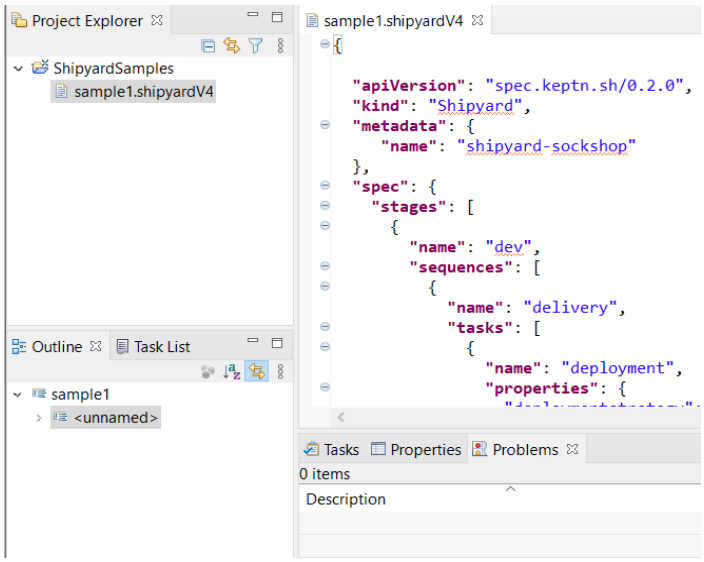
- create a file with the extension you chose in the language editor creation(e.g., shipyardV4).
- You will be asked if you want to convert the project to an XText project. Answer YES.

- Open your file
- Open the *Problems* and *Properties* view to see more details.
- Use CTRL+space for the content assist and code completion. Remember that all the keywords are between double quotes (“)



Use our examples.

For the shipyardV4.jschema example used in this tutorial you can try json instance examples in the folder *samples/shipyardSchemas/shipyardV4/instanceExamples* where you will find sample1.shipyardV4 and samples2.shipyardV4.
In the screenshot below you can see an excerpt of sample1.shipyardV4.
Open the Problems and Properties clicking on the Eclipse top menu Window->Show View->Other... and searching for them.

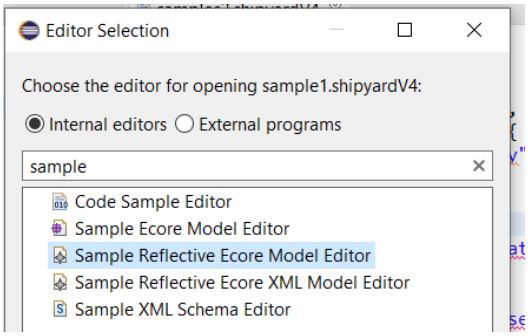


The reader is encouraged also to generate the languages for the json schemas provided in https://github.com/lowcomote/jsonschemadsl.parent/tree/master/jsonschemadsl2ecore.trafo.opt/test and independently create some instances that conform to them.

Sample Reflective Ecore Model Editor

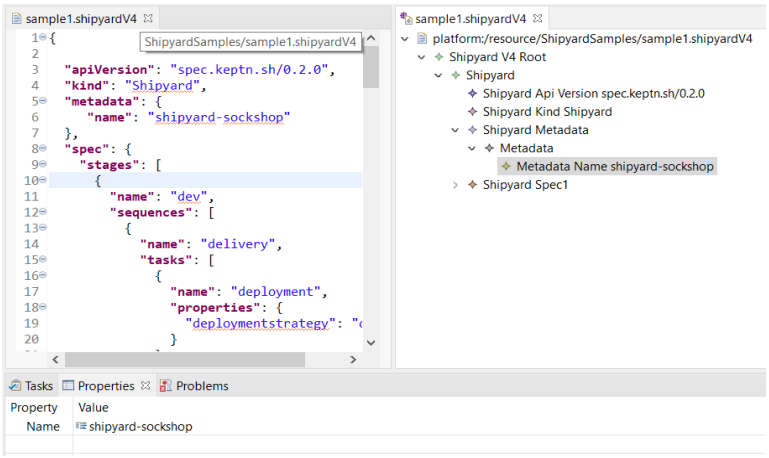
- Right click on the file with the extension that you have created (e.g., sample1.shipyardV4)

- Select *open with -> other*.
- and you will see the tree editor for your file, that is a model.



- Open the view properties, to see more details when you select an element of the tree.

As shown in the screenshot below, you can see sample1.shipyardV4 as a json conforming to the shipyardV4.jschema or as a model conforming to the previously generated shipyardV4Opt.ecore.
Changes applied to the tree are reflected in the file json style and vice versa.



[1] Leveraging Model-Driven Technologies for JSON Artefacts: The Shipyard Case Study
Alessandro Colantoni, Antonio Garmendia, Luca Berardinelli, Manuel Wimmer, Johannes Bräuer