Quick start

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

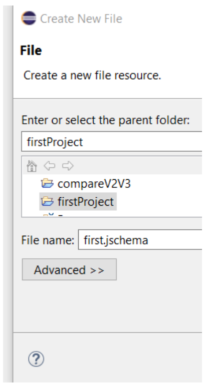
Introduction

In this tutorial we show how to create and edit a Json Schema, how to generate the corresponding ecore, ocl.
The generation of the language from the Json Schema created is delegated to the Language Editor Generation Tutorial.
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.

.


Create New File

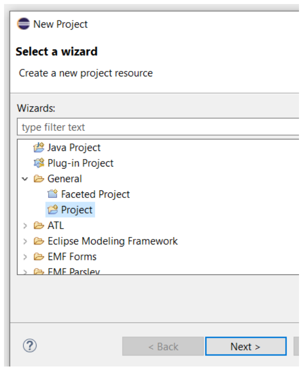
File

Create a new file resource.

Enter or select the parent folder: firstProject

compareV2V3
firstProject

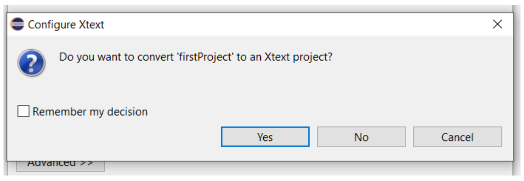
File name: first.jschema


New Project

Select a wizard

Create a new project resource

Wizards:


Configure Xtext

Do you want to convert 'firstProject' to an Xtext project?

☐ Remember my decision

- 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) and the artifacts that will be generated from it by JsonSchemaDSL.

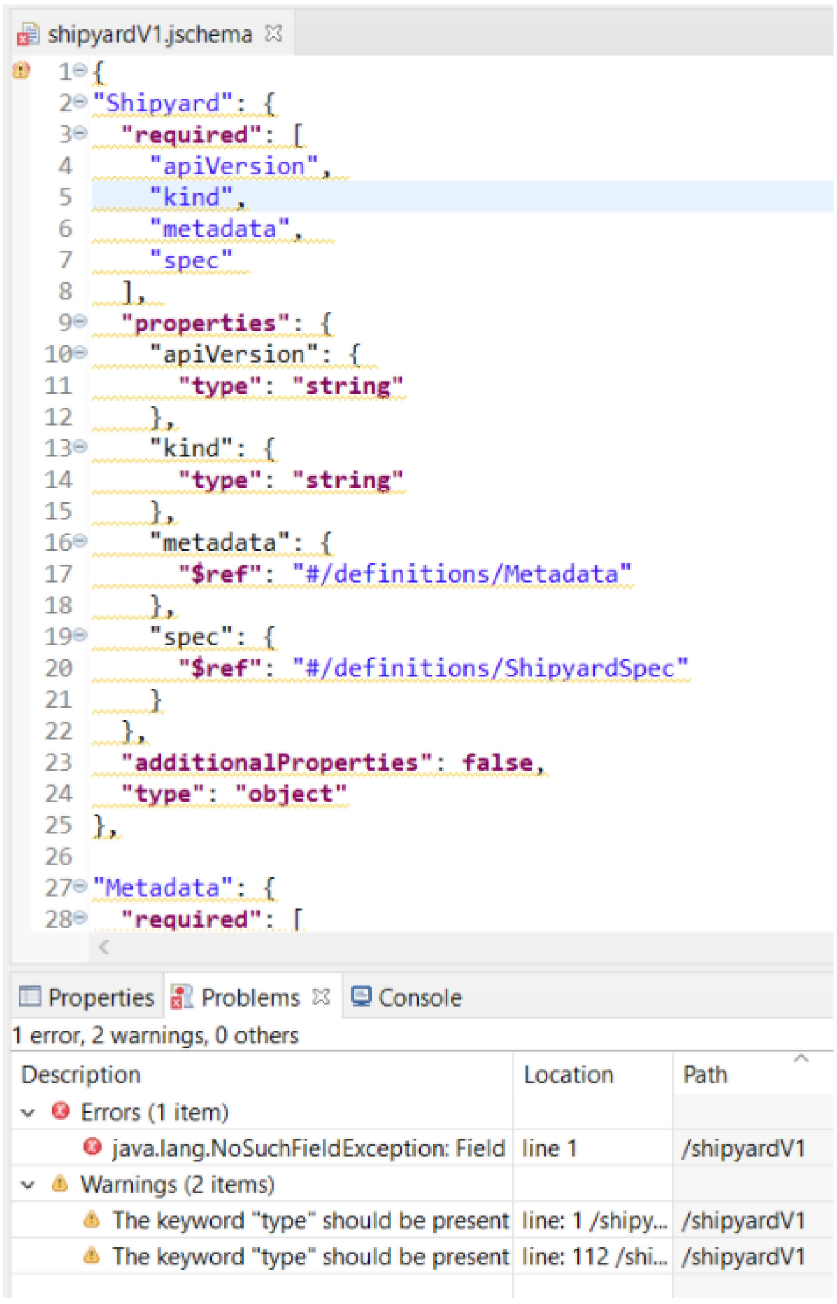
Shipyard DSL.

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 and warnings.
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 an error due to mistyped "$ref" value (it has to be a correct JSON Pointer) string, and two warnings.

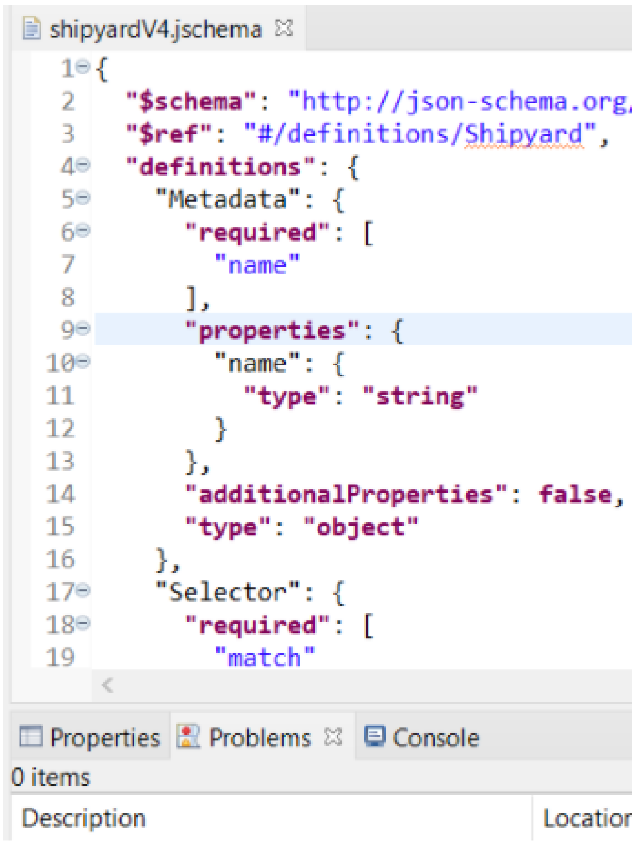


```
{
"Shipyard": {
  "required": [
    "apiVersion",
    "kind",
    "metadata",
    "spec"
  ],
  "properties": {
    "apiVersion": {
      "type": "string"
    },
    "kind": {
      "type": "string"
    },
    "metadata": {
      "$ref": "#/definitions/Metadata"
    },
    "spec": {
      "$ref": "#/definitions/ShipyardSpec"
    }
  },
  "additionalProperties": false,
  "type": "object"
},

"Metadata": {
  "required": [
```

```
1 error, 2 warnings, 0 others
Description                                  Location            Path
v Errors (1 item)
    java.lang.NoSuchFieldException: Field    line 1              /shipyardV1
v Warnings (2 items)
    The keyword "type" should be present     line: 1 /shipy...   /shipyardV1
    The keyword "type" should be present     line: 112 /shi...   /shipyardV1
```

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.

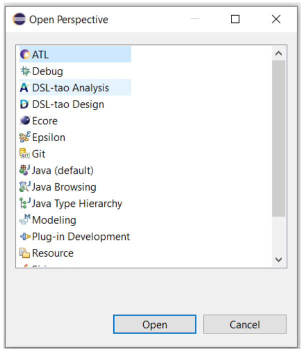


```
{
  "$schema": "http://json-schema.org,
  "$ref": "#/definitions/Shipyard",
  "definitions": {
    "Metadata": {
      "required": [
        "name"
      ],
      "properties": {
        "name": {
          "type": "string"
        }
      },
      "additionalProperties": false,
      "type": "object"
    },
    "Selector": {
      "required": [
        "match"
```

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.
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.
The steps to generate it are described in the dedicated tutorial Language Editor Generation.
Once the language has been generated, and the Runtime Eclipse as been launched as explained in the Language Editor Generation tutorial, the reader can create in the Runtime Eclipse a new simple project and create inside it a new file with the extension of the language (e.g., .shipyardV4).
These steps are described in detail in the Language Editor Generation tutorial.
In the same tutorial is also described how to use some examples of json that conform to shipyardV4.jschema.
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.

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