Article
Yuri Marx · Jan 12, 2022 12m read

Open Exchange

JSON Schema applied to InterSystems IRIS

The JSON is a data document free of types and validation rules. However, in some scenarios it is important that the JSON document has type and business rules validation, especially in interoperability scenarios. This article demonstrates how you can leverage a market-defined JSONSchema technology that is open for everyone to use and do advanced validations.

About JSON

According to json.org, " JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language " . (Source:https://www.json.org/json-en.html). An example of a JSON document could be:

```
{
  "employees": [
    {
      "firstName": "John",
      "lastName": "Doe"
    },
    {
      "firstName": "Anna",
      "lastName": "Smith"
    },
    {
      "firstName": "Peter",
      "lastName": "Jones"
    }
  ]
}
```

About JSON Schema

According to json-schema.org, " JSON Schema is a vocabulary that allows you to annotate and validate JSON documents. " .
If JSON is easy for humans to read, write and understand, why do we need to apply a schema to validate JSON documents/content? The main reasons are:

1. To define a clear contract to interoperate JSON based data between partners and their applications.
2. To detail, document and describe your JSON files, making it easier to use.
3. To validate JSON data for automated testing, ensuring the quality of the requests and responses.
4. To generate JSON sample and/or real data from the JSON Schema.
5. To apply business/validation rules into JSON content, instead of creating language dependent validations.
6. To support the " contract first approach " on the development of REST API. More details on https://swagger.io/blog/api-design/design-first-or-code-first-api-develo....

JSON Syntax

To understand the role of JSON Schema in a JSON game, it is necessary to know more about JSON syntax. The JSON syntax is derived from JavaScript object notation syntax, so, the syntax rules are equal to JavaScript objects. The JSON rules are next (source: https://www.w3schools.com/js/jsjsonsyntax.asp):

1. Data must be specified in name/value pairs.
2. Data must be separated by commas.
3. Curly braces hold objects.
4. Square brackets hold arrays.

The JSON Data consists of a name/value pair with a field name (in double quotes), followed by a colon and followed by a value. Example:

```
"firstName": "John"
```

JSON names require double quotes and refers to JavaScript objects and to InterSystems IRIS ObjectScript Dynamic Object (%Library.DynamicObject) too. More details in https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls....

In JSON, values must be one of the following data types:

- A string, must be between “ ” . Sample: “ a string ” .
- A number, must be integer or decimal values. Sample: 10 or 20.23.
- An object, must be between {}. Sample: { “ firstName ” : “ John ” }.
- An array, must be between []. Sample: [{ “ firstName: “ John ” },{ “ firstName ” : “ Anna ” }].
- A Boolean, must be true or false. Sample: { “ isDead ” : false}.
- A null, to set no value.

When you apply a JSON document to a language object, these syntax rules are validated, but sometimes, it is necessary to establish business rules too. JSON Schema is used for this, to expand JSON basic rules with rules specified in JSON Schema documents.
The InterSystems IRIS has full support for JSON basic rules, and JSON content is very elegant to read or write.. It is easier than any other programming language, see a sample:

InterSystems IRIS Object Script writing JSON	Java writing JSON

set dynObject1 = ##class(%DynamicObject).%New()
set dynObject1.SomeNumber = 42
set dynObject1.SomeString = "a string"
set dynObject1.SomeArray =
##class(%DynamicArray).%New()
set dynObject1.SomeArray."0" = "an array element"
set dynObject1.SomeArray."1" = 123
dynObject1.%ToJSON()
	//First Employee
	JSONObject employeeDetails = new JSONObject();
	employeeDetails.put("firstName","Lokesh");
	employeeDetails.put("lastName", "Gupta");
	employeeDetails.put("website","howtodoinjava.com");
	JSONObject employeeObject = new JSONObject();
	employeeObject.put("employee", employeeDetails);
	
	//Second Employee
	JSONObject employeeDetails2 = new JSONObject();
	employeeDetails2.put("firstName", "Brian");
	employeeDetails2.put("lastName", "Schultz");
	employeeDetails2.put("website", "example.com");
	
	JSONObject employeeObject2 = new JSONObject();
	employeeObject2.put("employee", employeeDetails2);
	
	//Add employees to list
	JSONArray employeeList = new JSONArray();
	employeeList.add(employeeObject);
	employeeList.add(employeeObject2);
	employeeList.toJSONString();

While ObjectScript is a dynamic language, allowing setting JSON properties as object properties, other languages, like Java, force you to set key and values inside objects. On the other hand, Java and other languages support JSON Schema using open source and commercial packages, but the ObjectScript does not support JSON Schema at the moment. See the list from json-schema.org (source: https://json-schema.org/implementations.html):

.NET

- Json.NET Schema 2019-09, draft-07, -06, -04, -03 (AGPL-3.0-only)
- JsonSchema.Net 2020-12, 2019-09, draft-07, -06 (MIT)

C++

- f5-json-schema draft-07 (Boost Software License 1.0)
- JSON schema validator for JSON for Modern C++ draft-07 (MIT)
- Valijson draft-07 header-only library, works with many JSON parser implementations (BSD-2-Clause)
- Jsoncons draft-07 header-only library (Boost Software License 1.0)

Java

- Snow 2019-09, draft-07, -06 Uses Maven for the project and Gson under the hood. (GNU Affero General Public License v3.0)
- Vert.x Json Schema 2019-09, draft-07 includes custom keywords support, custom dialect support, asynchronous validation (Apache License, Version 2.0)
- Everit-org/json-schema draft-07, -06, -04 (Apache License 2.0)
- Justify draft-07, -06, -04 (Apache License 2.0)
- Networknt/json-schema-validator draft-07, -06, -04 Support OpenAPI 3.0 with Jackson parser (Apache License 2.0)
- Jsonschemafriend 2019-09, draft-07, -06, -04, -03 (Apache License 2.0)

JavaScript

- Ajv 2019-09, 2020-12, draft-07, -06, -04 for Node.js and browsers - supports user-defined keywords and $data reference (MIT)
- Djv draft-06, -04 for Node.js and browsers (MIT)
- Hyperjump JSV 2019-09, 2020-12, draft-07, -06, -04 Built for Node.js and browsers. Includes support for custom vocabularies. (MIT)
- Vue-vuelidate-jsonschema draft-06 (MIT)
- @cfworker/json-schema 2019-09, draft-07, -06, -04 Built for Cloudflare workers, browsers, and Node.js (MIT)

Python

- jschon 2019-09, 2020-12 (MIT)
- jsonschema 2019-09, 2020-12, draft-07, -06, -04, -03 (MIT)
- fastjsonschema draft-07, -06, -04 Great performance thanks to code generation. (BSD-3-Clause)
- jsonschema-rs draft-07, -06, -04 Python bindings to Rust ' s jsonschema crate (MIT)

This is a sample how to use JSON schema to validate JSON content (using Python, source: https://jschon.readthedocs.io/en/latest/):

Create a JSON schema:

```
from jschon import create_catalog, JSON, JSONSchema

create_catalog('2020-12')

demo_schema = JSONSchema({
    "$id": "https://example.com/demo",
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "type": "array",
    "items": {
        "anyOf": [
            {
                "type": "string",
                "description": "Cool! We got a string here!"
            },
            {
                "type": "integer",
                "description": "Hey! We got an integer here!"
            }
        ]
    }
})
```

Validate JSON data:

```
result = demo_schema.evaluate(
    JSON([12, "Monkeys"])
)
```

Generate JSON Schema-conformant output:

```
>>> result.output('basic')
{
    "valid": True,
    "annotations": [
        {
            "instanceLocation": "",
            "keywordLocation": "/items",
            "absoluteKeywordLocation": "https://example.com/demo#/items",
            "annotation": True
        },
        {
            "instanceLocation": "/0",
            "keywordLocation": "/items/anyOf/1/description",
            "absoluteKeywordLocation": "https://example.com/demo#/items/anyOf/1/description",
            "annotation": "Hey! We got an integer here!"
        },
        {
            "instanceLocation": "/1",
            "keywordLocation": "/items/anyOf/0/description",
            "absoluteKeywordLocation": "https://example.com/demo#/items/anyOf/0/description",
            "annotation": "Cool! We got a string here!"
        }
    ]
}
```

Fortunately, IRIS allows you to create packages or frameworks using the programming language of your choice (.NET or Java using PEX or Language Server). So, it is possible to create an IRIS extension package to handle JSON Schema in IRIS. Another possibility is to use Embedded Python and create a JSON validation method class (in the version 2021.2+).

Extending the InterSystems IRIS to support JSON Schema using Java Language Server (Java Gateway)

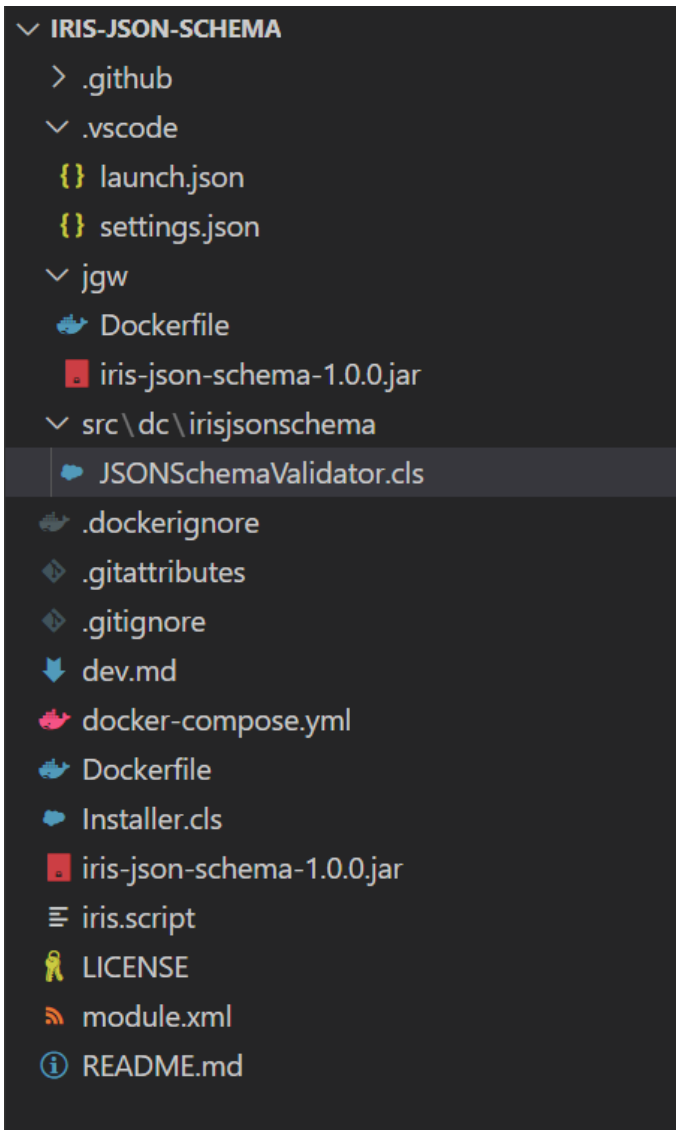
Among Java frameworks, the networknt/json-schema-validator is used the most to validate JSON using JSON Schema.
To use this Java framework, you can get the application https://openexchange.intersystems.com/package/IRIS-JSON-Schema-Validator. This application has the following files and folders:

1. The folder jgw has the necessary files to create a Java Gateway (bridge to allow communication between Java and IRIS classes);

2. The iris-json-schema-1.0.0.jar has the Java classes and libraries (including json-schema-validator) to service JSON Schema validations;

3. The JSONSchemaValidator.cls has the ObjectScript code to use the Java class and do JSON validations using JSON schema by the validation rules;

4. The Dockerfile and docker-compose.yml run the Java Gateway and the IRIS as docker instances.



The Java Class has a validation method, which uses the framework json-schema-validator to validate the JSON Schema and the JSON, and to return the validation results. See the Java Code:

Java Class for JSON Schema Validation

The InterSystems IRIS uses this Java validation method to validate JSON content. To do that it is necessary to create a JAR file with the validate class inside (iris-json-schema-1.0.0.jar) and to configure a Java Gateway (bridge between Java and IRIS communication), allowing ObjectScript to call the Java methods.
The ObjectScript code which uses the Java Class and the JSONSchemaValidator class is presented here:

Final Java Class read to Validate JSON inside IRIS

With this ObjectScript class and the Validate class method, it is possible to use any JSON content and any JSON Schema definition to validate basic and advanced validation rules.
To see this, execute these steps:

1. Go to https://openexchange.intersystems.com/package/IRIS-JSON-Schema-Validator

2. Git-clone the repository into any local directory

```
$ git clone https://github.com/yurimarx/iris-json-schema.git
```

3. Open the terminal in this directory and run it:

```
$ docker-compose build
```

4. Run the IRIS container with your project:

```
$ docker-compose up
```

5. Go to the IRIS terminal (open a new VSCode Terminal)

```
docker exec -it iris-json-schema_iris_1  bash iris session iris
```

6. Change to the IRISAPP namespace

```
set $namespace = "IRISAPP"
```

7. Get a sample JSON Schema

```
set jsonSchema = ##class(dc.irisjsonschema.JSONSchemaValidator).GetJSONSchema()
```

8. Get a sample valid JSON

```
set jsonContent = ##class(dc.irisjsonschema.JSONSchemaValidator).GetValidSampleJSON()
```

9. Get a validation equals to valid

```
set st = ##class(dc.irisjsonschema.JSONSchemaValidator).Validate(jsonSchema,jsonConte
nt,.result)
write result
```

10. Now, get a sample INVALID JSON

```
set jsonContent =   ##class(dc.irisjsonschema.JSONSchemaValidator).GetInvalidSampleJS
ON()
```

11. After that, get validation equals to INVALID

```
set st = ##class(dc.irisjsonschema.JSONSchemaValidator).Validate(jsonSchema,jsonConte
nt,.result)
write result
```

12. The JSON Schema used was:

JSON Schema used to define the validation rules

This JSON Schema configured the fields and the tags array with limited values (enum) as required.

The types of the JSON fields are defined too. So, for this schema the following JSON Object is valid:

```
obj
obj name    "Agent 007"
obj artist    "Pierce Brosman"
obj description    "007 actor"
tags    "license" "kill"
obj tags    tags
```

All properties use the right data type and the tags use values inside values allowed in the Schema for this array.
Now, see a JSON Object invalid:

```
obj
obj name    "Agent 007"
obj artist    "Pierce Brosman"
obj description    1
tags
tags "0"    "license"
tags "1"    "love"
obj tags    tags
```

This object sets an integer property to a description (the valid type is string) and sets the value “ love ” , out of valid values allowed to array tags.
The site https://json-schema.org/ has tools, samples and learning resources to learn on how to validate and write advanced validation rules using JSON Schema.

#JSON #InterSystems IRIS
Check the related application on InterSystems Open Exchange