json2puml

Based on version v2.0.x

Documentation

© by Jens Fudickar in 2023

1 Content

2 Introduction 6
2.1 Current Situation 6
2.1.1 Plain JSON 6
2.1.2 JSON Editor 7
2.1.3 PlantUML 7
2.2 json2puml 8
2.3 Space X API based examples 9
2.3.1 Example 1 – get /rockets 9
2.3.2 Example 2 – Data of all API's in relation to a launch 11
2.4 swapi.dev based examples 14
2.4.1 Example 1 – All data 15
2.4.2 Example 2 – Han Solo 16
2.4.3 Example 3 – Death Star 17
3 Installation 18
3.1 Steps 18
3.1.1 Welcome Screen 18
3.1.2 Definition of the installation folder 18
3.1.3 Definition of components to be installed 19
3.1.4 Definition of installation options 19
3.1.5 Summary before installation 20
3.1.6 Status screen while the installation is running 20
3.1.7 Completion screen 21
3.2 Required privileges 21
3.3 Created directories / Installed Files 21
3.4 Setup Options 22
3.4.1 Add bin directory to PATH variable 22
3.4.2 Download PlantUML Jar File 22
3.4.3 Add sample files 23
3.5 Environment Parameters 23
3.6 PlantUML Installation 24
3.6.1 Java Runtime 24
3.6.2 PlantUML software 24
3.6.3 Output generation 24
4 How to use 26
4.1 Help Screen 27

4.2 Command Line Parameters 27
4.3 QuickStart 31
5 How the system is working 32
5.1 Main components 32
5.1.1 Curl pre-processor 32
5.1.2 Json 2 PlantUML converter 33
5.1.3 PlantUML post-processor 33
5.2 Curl pre-processor 34
5.2.1 Curl parametrisation 34
5.2.2 Curl command 35
5.2.3 How to use it 35
5.2.4 API Security / OAuth 42
5.2.5 How to define curl parameter 42
5.3 Converter 44
5.3.1 Basic Flow 44
5.3.2 Object Identification 45
5.3.3 Object Relationships 46
5.3.4 Detail Objects / Characteristic properties 47
5.3.5 Filtering 48
5.3.6 Generated Outcome 49
5.4 Operational Modes 50
5.4.1 Single File 50
5.4.2 List File 50
5.4.3 Detail / Summary generation 51
5.5 Generated Files 51
5.5.1 Output path 51
5.5.2 Output suffix 52
5.5.3 Summary file 53
5.5.4 Zip File 53
5.5.5 PlantUML source File 53
5.5.6 PlantUML output files 53
6 Configuration 54
6.1 File overview 54
6.1.1 Global configuration file 54
6.1.2 Parameter file 54
6.1.3 Input list file 54

6.1.4 Converter definition file 54
6.1.5 Converter definition option file 54
6.1.6 Curl parameter file 54
6.1.7 Curl authentication file 55
6.2 Variables 55
6.2.1 Generator 55
6.2.2 Generated folder and file names 55
6.2.3 Configuration files 57
6.2.4 System Environment 58
6.3 Global configuration file 59
6.3.1 Data model 59
6.4 Parameter file 59
6.4.1 Data model 59
6.5 Input list File 60
6.5.1 Data model 61
6.5.2 InfoListFile 61
6.5.3 SingleListFile 65
6.5.4 Simple List File Example 70
6.6 Converter definition file 70
6.6.1 Merging of options 70
6.6.2 Identifying an Option 71
6.6.3 Data model 73
6.6.4 Configuration 74
6.6.5 GlobalDefinition 75
6.6.6 NamedOption 75
6.6.7 PropertyValue 76
6.6.8 SingleOption 79
6.7 (see 5.1 Main components 87
6.7.1 Curl pre-processor .... **Fehler! Textmarke nicht definiert.**
6.7.2 Json 2 PlantUML converter .... **Fehler! Textmarke nicht definiert.**
6.7.3 PlantUML post-processor .... **Fehler! Textmarke nicht definiert.**
6.8 Converter .... **Fehler! Textmarke nicht definiert.**
6.8.2 ObjectProperty 98
6.8.3 CharacteristicProperty 99
6.8.4 ObjectFormat 100
6.8.5 NamedFormat 101

6.8.6 SingleFormat 101
6.9 Curl parameter file 106
6.10 Curl authentication file 106
7 Global Definitions 107
7.1 leadingObject 107
8 Appendix 108
8.1 Images 108

2 Introduction

2.1 Current Situation

2.1.1 Plain JSON

Working with API's leads to working with JSON data.

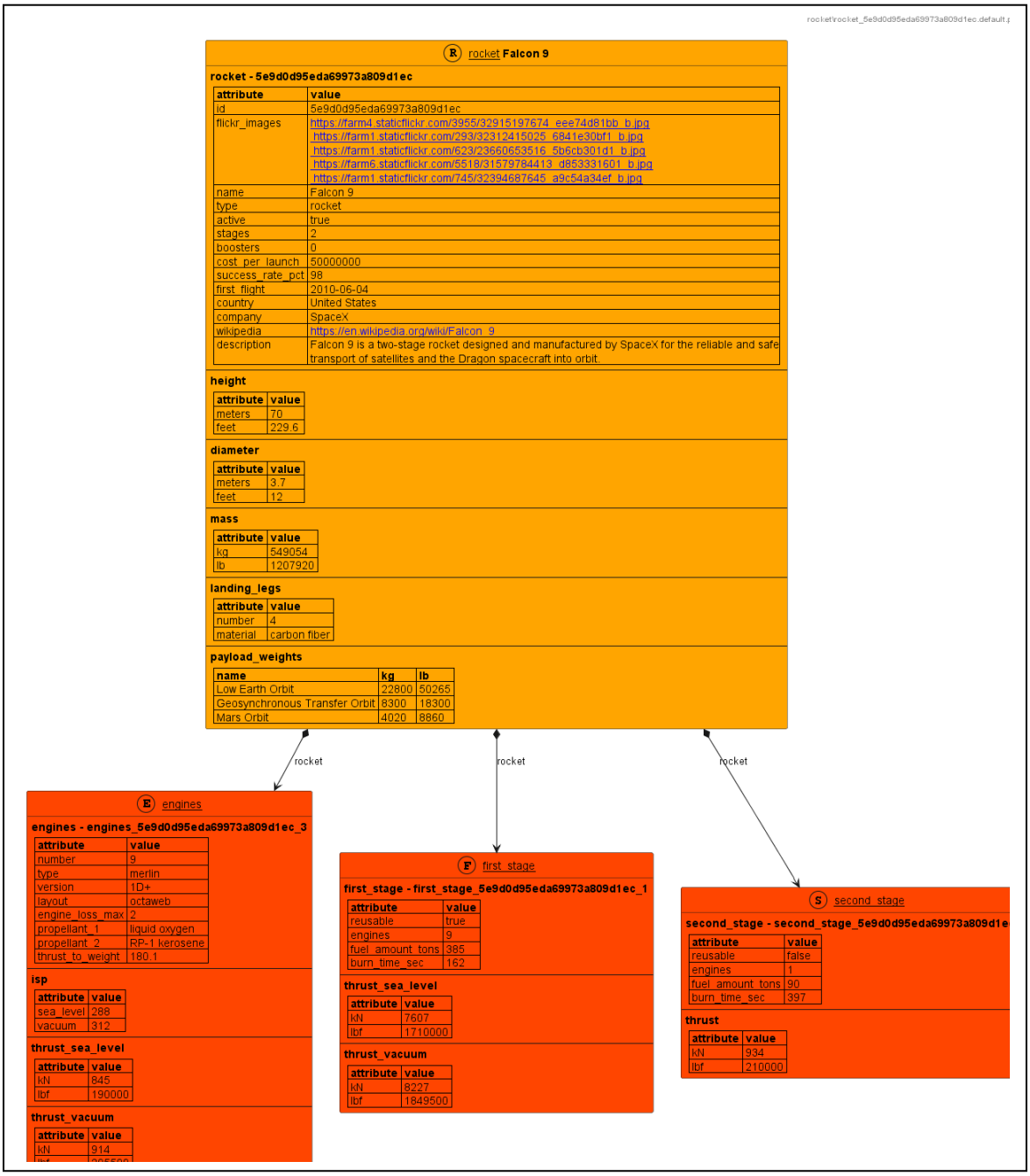


Figure 1: JSON sample

A typical answer of a TMF call can look like this:

A long line of data which is quite often not formatted or structured.

2.1.2 JSON Editor

A first starting point to understand the data is formatting and using a JSON editor which visualises the data in a better way:

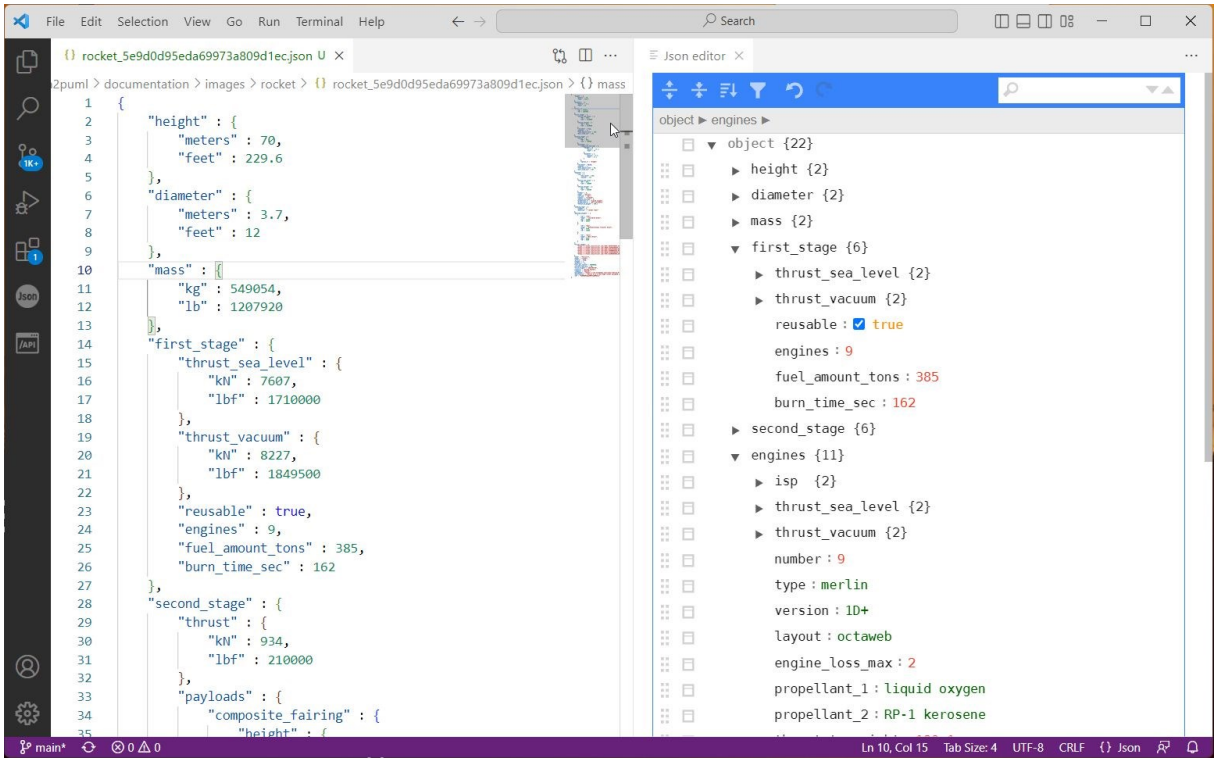


Figure 2: JSON Editor

This makes the live easier but it is still complex to understand the context of the data ☺.

2.1.3 PlantUML

Another option is to use PlantUML.

Plant is a tool/library to generate images based on a textual description. PlantUML has a JSON specific command to visualize the content of a JSON data structure.

```
@startjson
<JSON data>
@endjson
```

Using this syntax PlantUML automatically generates a picture which visualises all elements of the JSON structure as tree-based image:

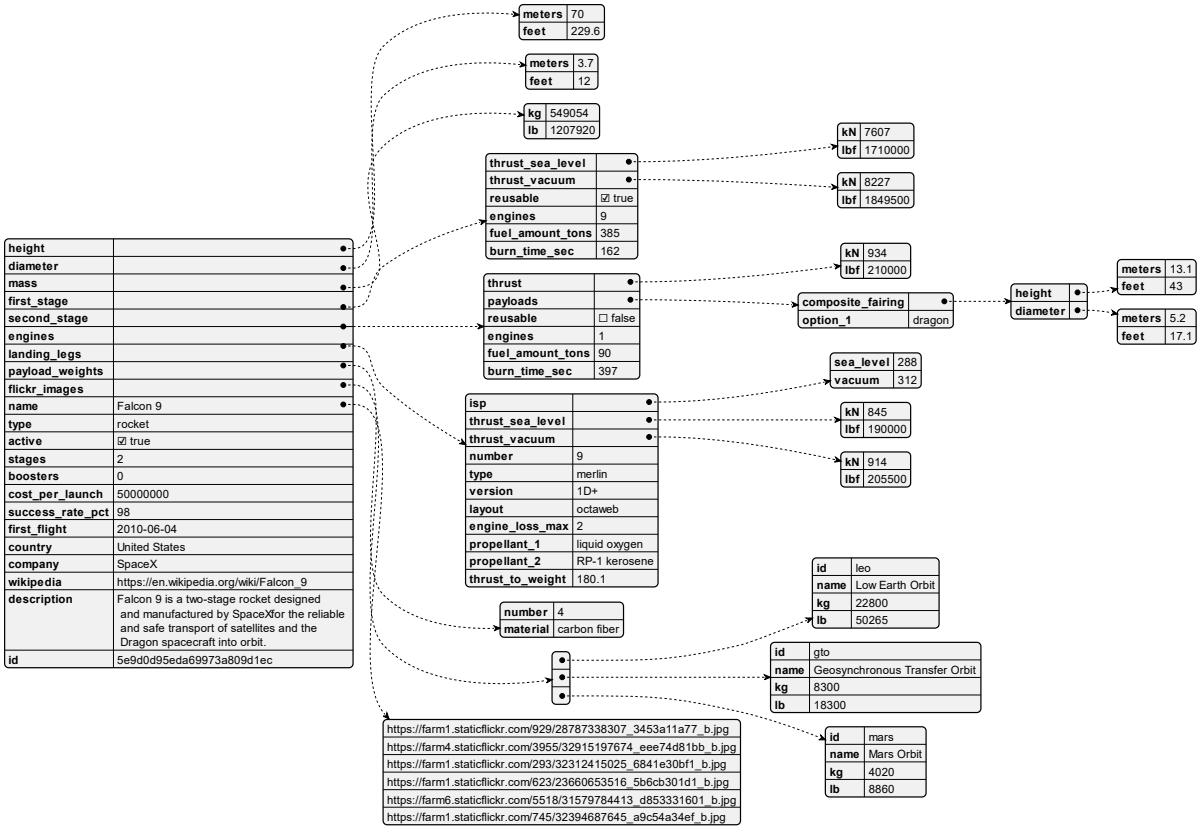


Figure 3 JSON Example PlantUML

This improves also, but there is no knowledge about the data models behind and so no logical grouping or filtering is possible.

2.2 json2puml

The solution to improve the visualisation of JSON data is

`json2puml`

`json2puml` is a command line tool developed to generate PlantUML files based JSON files (TMF based).

`json2puml` has an understanding of how data is structured in TMF and simplifies and visualises the outcome.

`json2puml` has the possibility to combine the JSON results of multiple API calls into one result set.

`json2puml` is highly configurable to generate outcomes in different detailed levels.

All images are based on the same JSON structure, the outcomes are only separated based on different configurations.

SpaceX REST API

Open-Source REST API for launch, rocket, core, capsule, starlink, launchpad, and landing pad data.

https://github.com/r-spacex/SpaceX-API

What is this?

It's an unofficial public API to receive various information about all Space X rocket launches.

2.3 Space X API based examples

2.3.1 Example 1 – get /rockets

The example data shown above is the result of a `get /rockets/{id}` api call.

Based on this data the following outputs are possible.

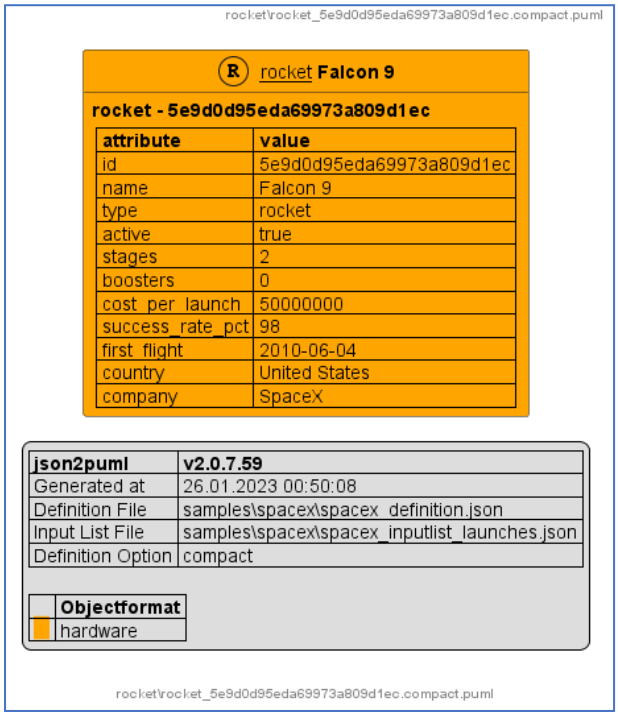


Figure 4 Space X – get /rockets – compact option

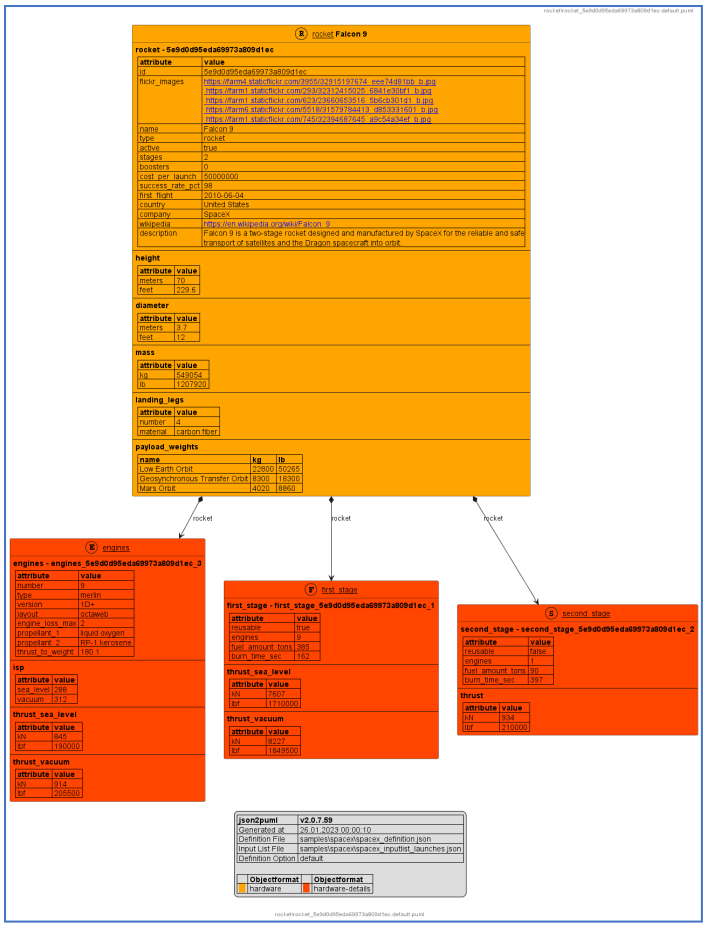


Figure 5 Space X – get /rockets – default option

Figure 6 Space X – get /rockets – full option

2.3.2 Example 2 – Data of all API's in relation to a launch

This example is based on the results of the following api calls:

- `get /launches`
- `get /capsule`
- `get /core`
- `get /crew`
- `get /dragon`
- `get /landpad`

- `get /launchpad`
- `get /payload`
- `get /rocket`

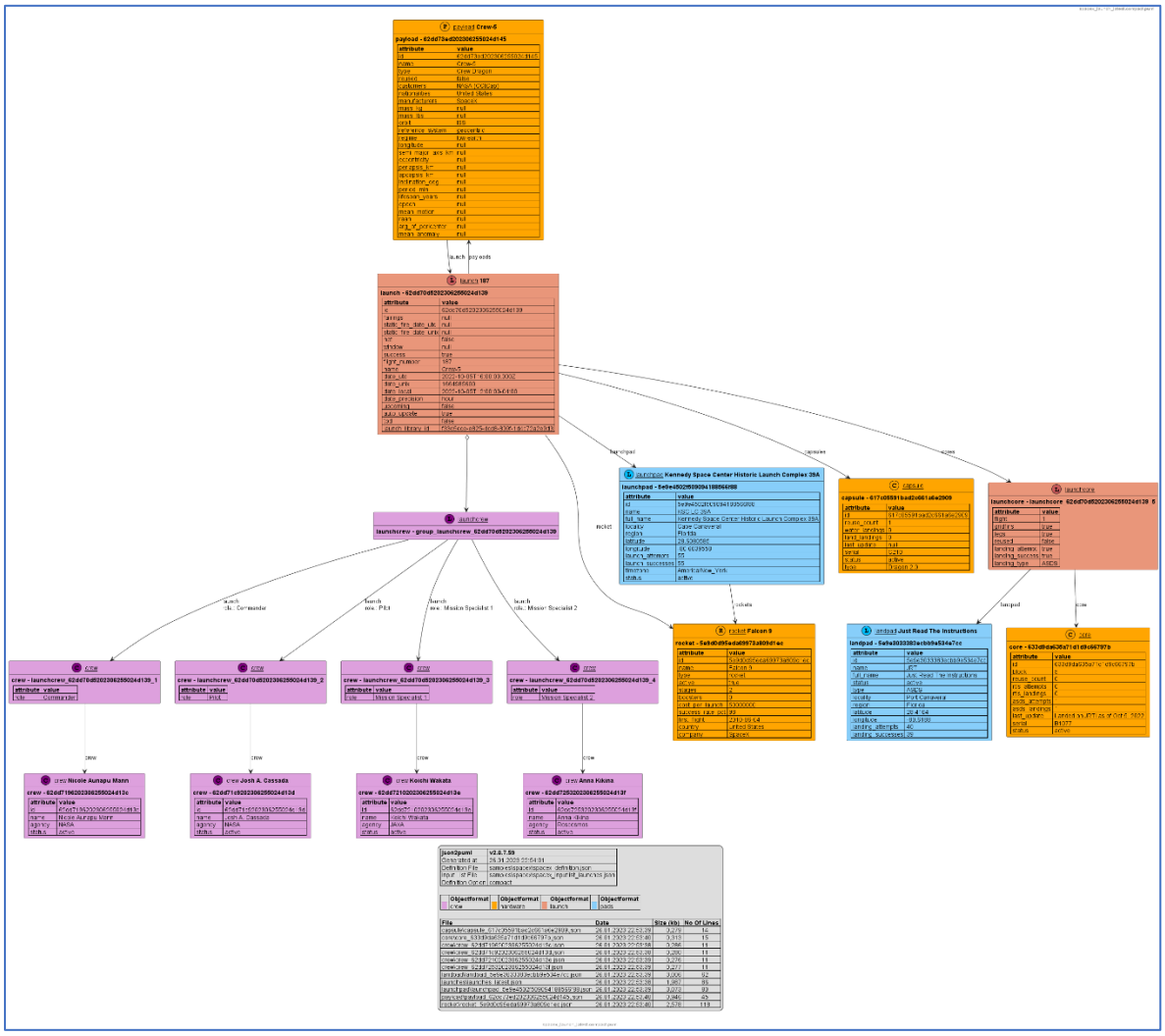


Figure 7 Space X – get all API – compact option

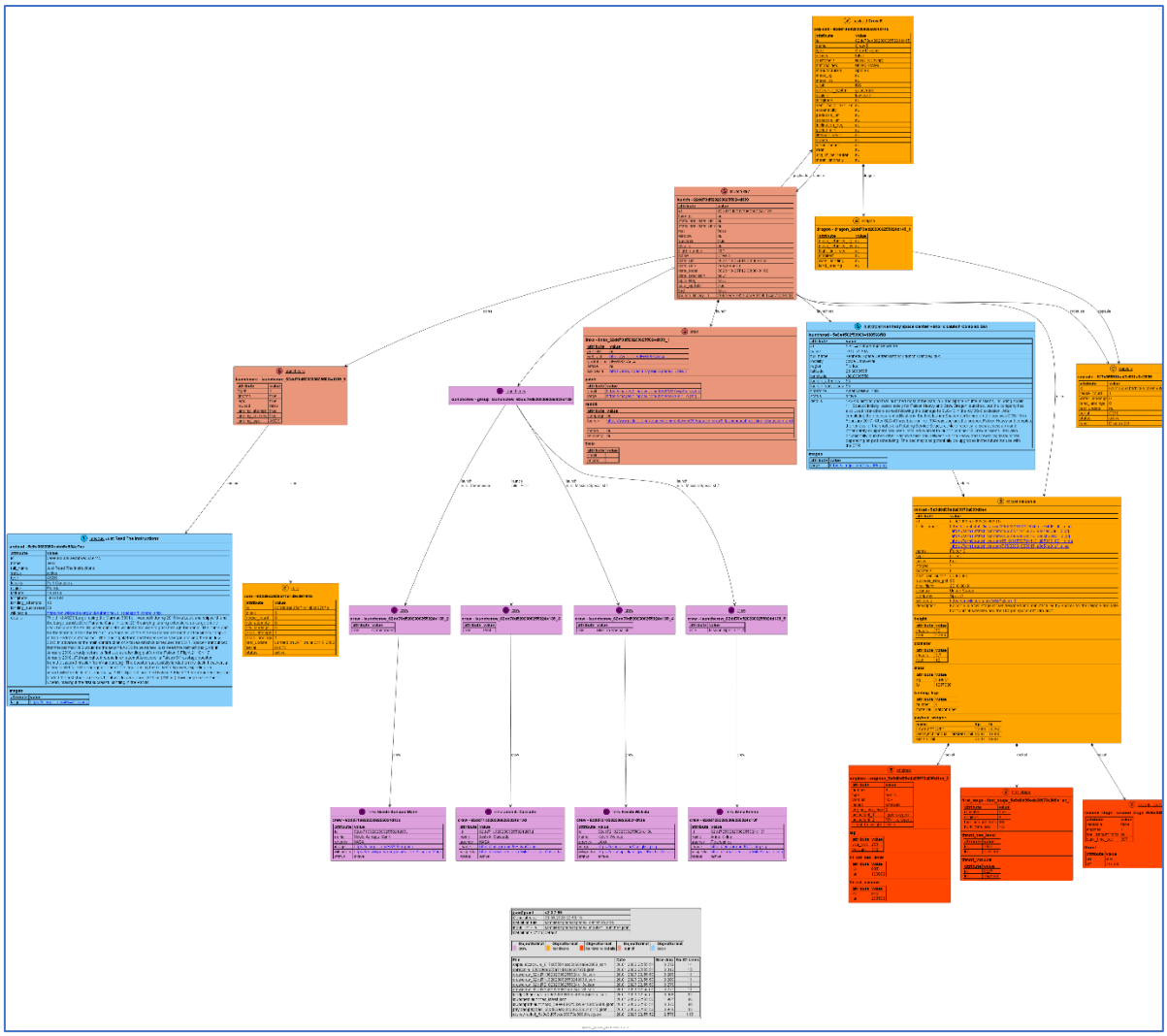


Figure 8 Space X – get all API – default option

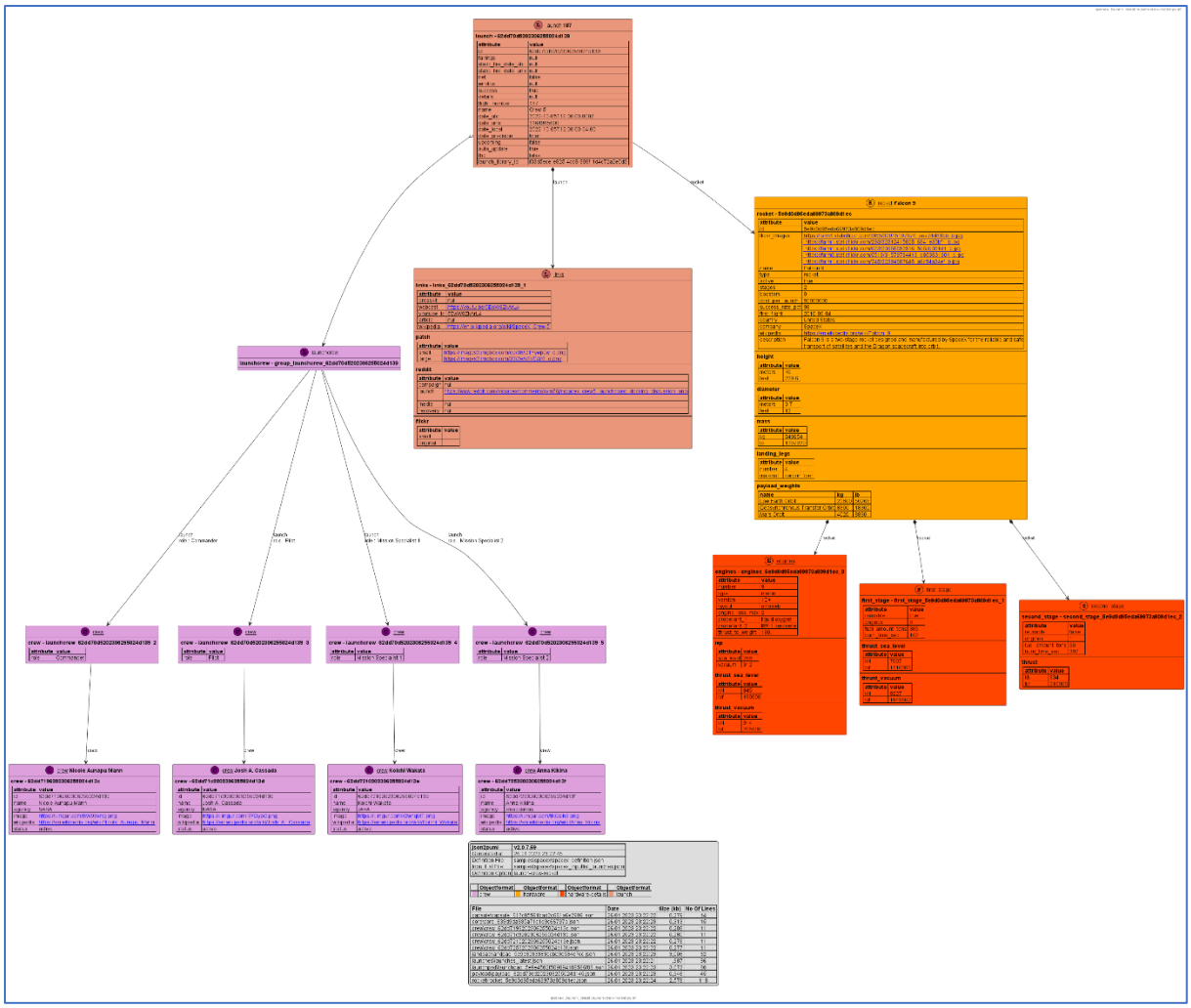


Figure 9 Space X – get all API – launch-crew-rocket option

2.4 swapi.dev based examples

SWAPI - The Star Wars API

https://swapi.dev/

What is this?

The Star Wars API, or "swapi" (Swah-pee) is the world's first quantified and programmatically-accessible data source for all the data from the Star Wars canon universe!

We've taken all the rich contextual stuff from the universe and formatted into something easier to consume with software. Then we went and stuck an API on the front so you can access it all!

2.4.1 Example 1 – All data

This image combines the data of all api results unfiltered into one image.

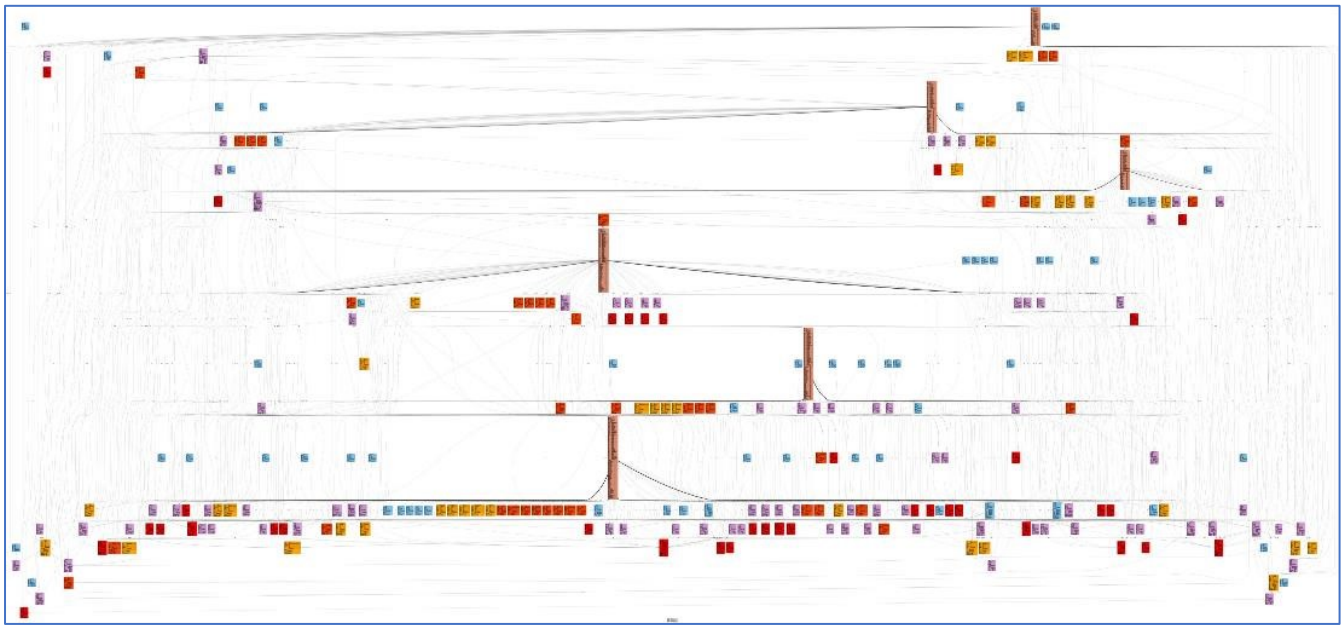


Figure 10 swapi.dev - all data - 6 Films, 82 Persons, 60 Planets, 37 Species, 36 Star ships, 39 Vehicles

2.4.2 Example 2 – Han Solo

This example is based on all data in combination with the `titlefilter` "Han Solo".

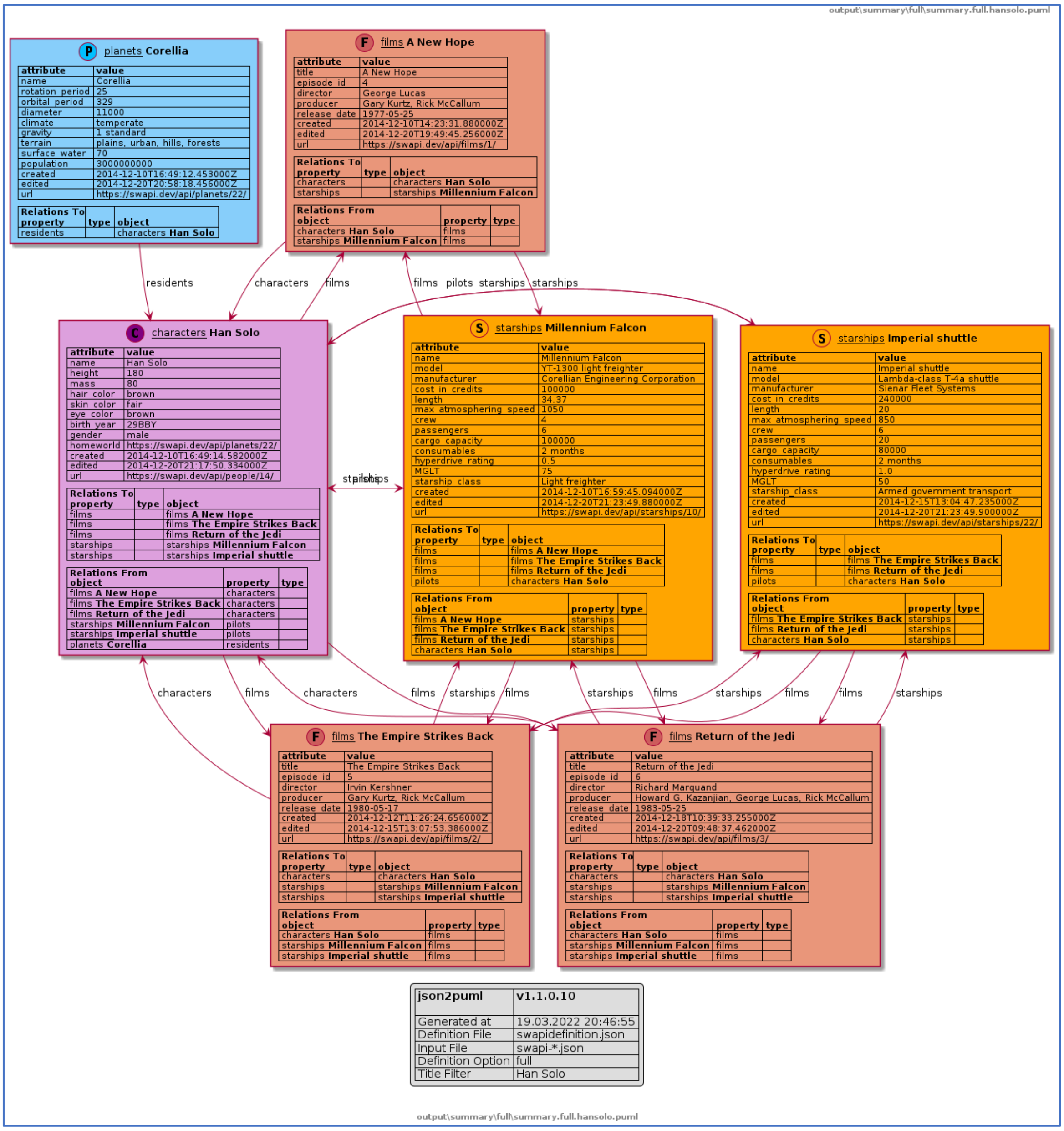


Figure 11 swapi.dev - /titlefilter = "Han Solo"

2.4.3 Example 3 – Death Star

This example is based on all data in combination with the `titlefilter` “Death Star”.

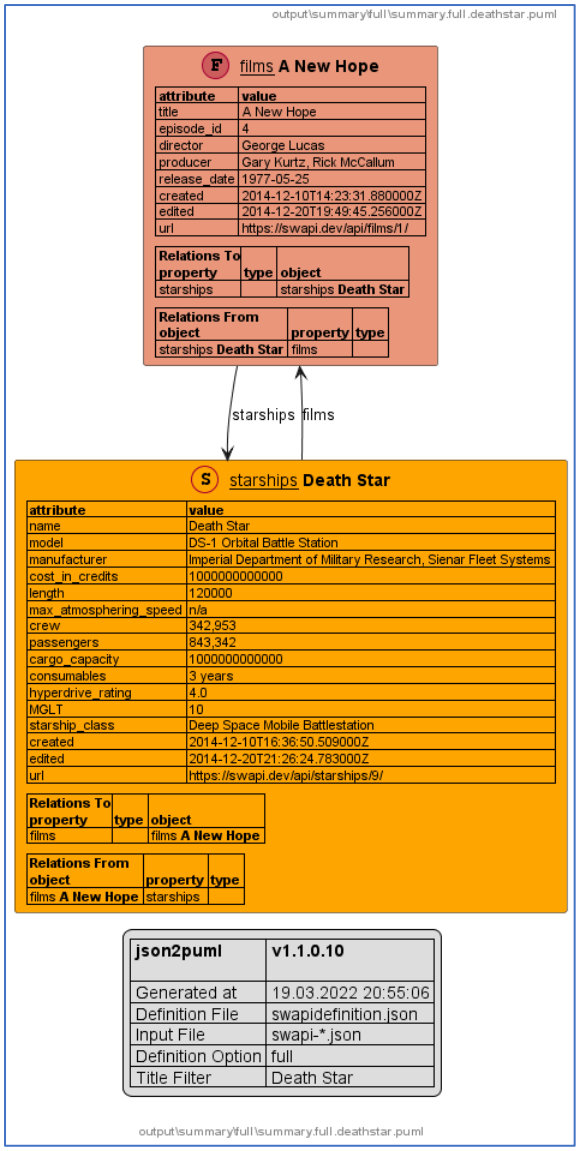


Figure 12 swapi.dev - /titlefilter = "Han Solo"

3 Installation

`json2puml` comes with an automated setup program which copies the main files to a user defined directory, optionally downloads the PlantUML jar file and configures the system to simplify the usage.

The setup file is named `json2puml.setup.<version>.exe`.

3.1 Steps

The setup is following the following steps:

3.1.1 Welcome Screen

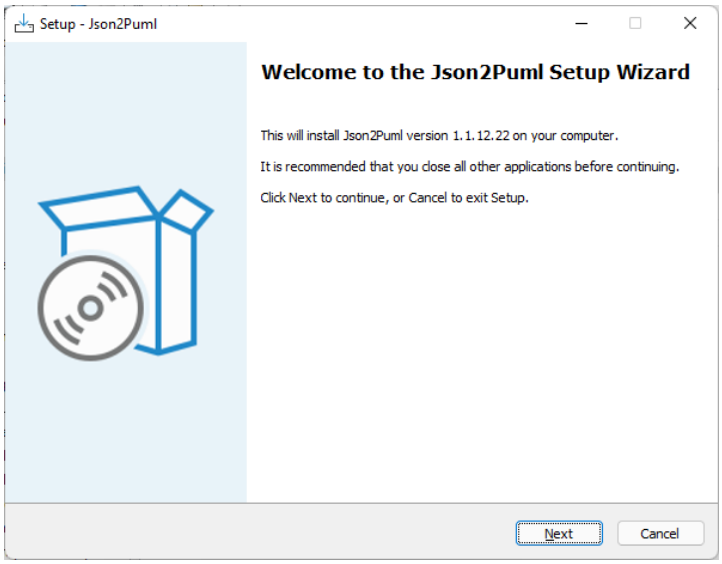


Figure 13 Setup - Welcome Screen

3.1.2 Definition of the installation folder

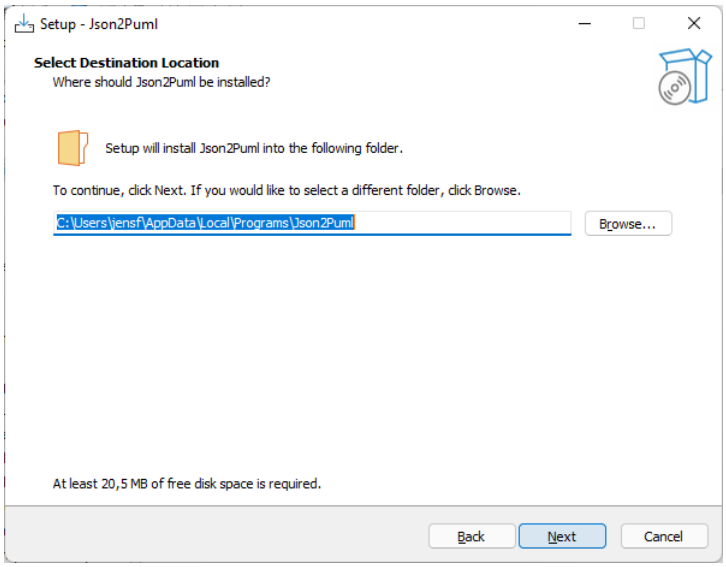
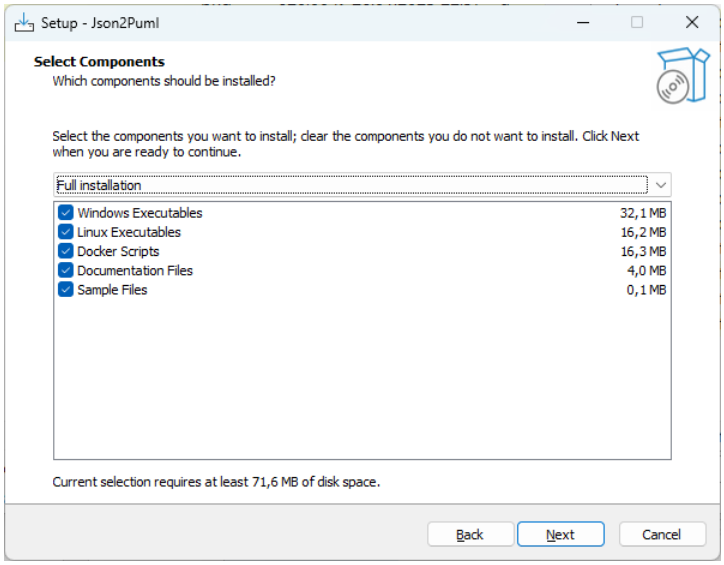


Figure 14 Setup - Definition of installation folder

3.1.3 Definition of components to be installed



3.1.4 Definition of installation options

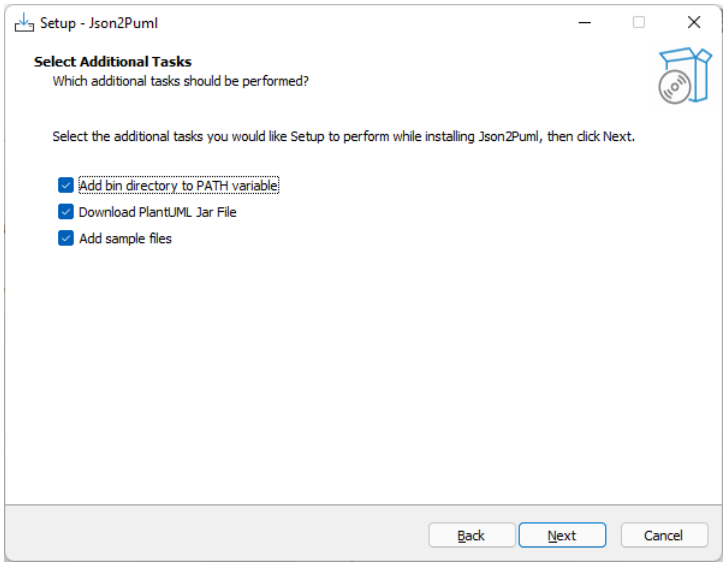


Figure 15 Setup - Definition of installation options

3.1.5 Summary before installation

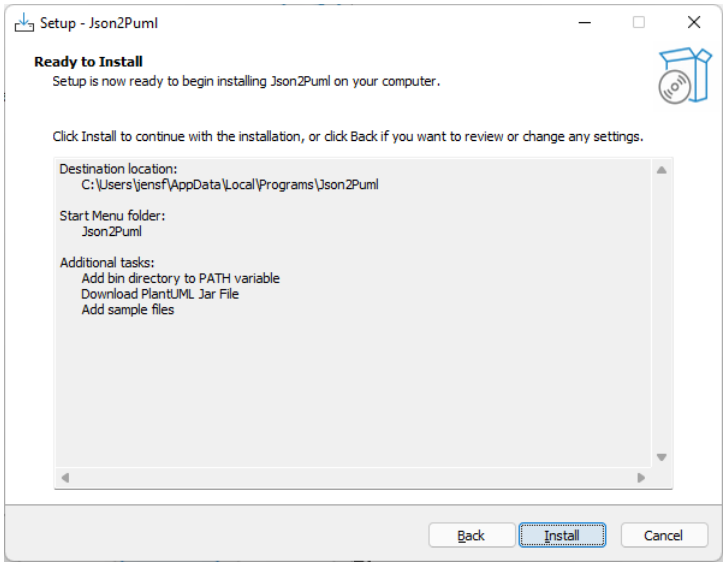


Figure 16 Setup - Summary before installation

3.1.6 Status screen while the installation is running

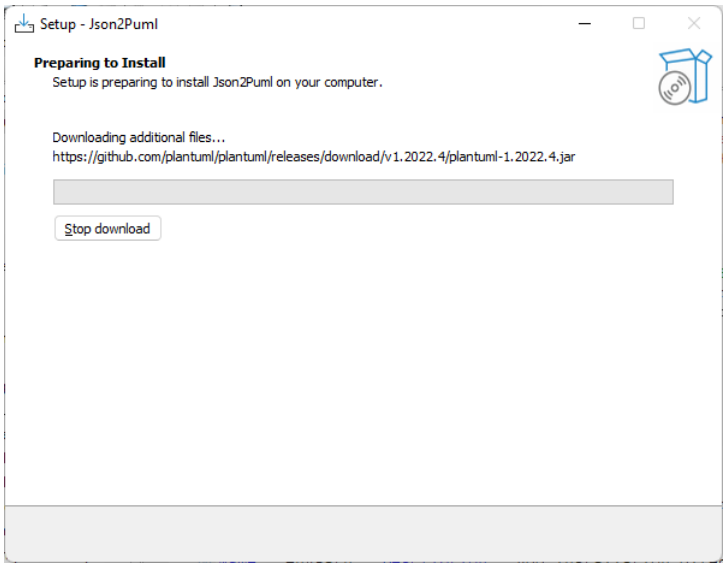


Figure 17 Setup - Status screen while installation

3.1.7 Completion screen

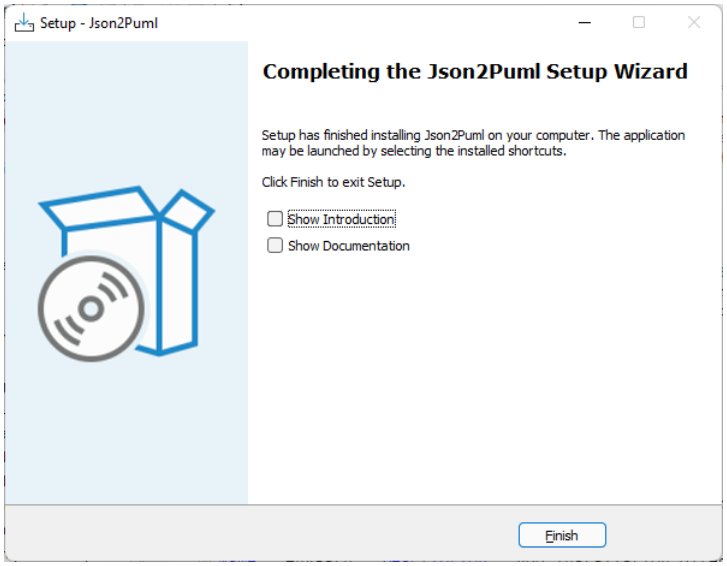


Figure 18 Setup - Completion screen

3.2 Required privileges

The installation can be executed without having admin privileges as the program is installed in the user profile of the current user.

3.3 Created directories / Installed Files

The default suggested directory is the common local application folder in the user profile of the current user.

The directory is: `%USERPROFILE%\AppData\Local\Programs\Json2Puml`

Inside this directory the following sub directories are created:

Directory	Description	Task / Option
`bin`	Windows binaries of json2puml	Windows Executables
`bin\linux`	Linux binaries of json2puml	Linux Executables
`docker`	Docker scripts to generate linux docker containers	Docker
`documentation`	Documentation files	
`PlantUML`	Destination folder for the PlantUML jar file which is downloaded from the PlantUML project	Download PlantUML jar
`samples`	Sample definition files	Add sample files

The following files are installed:

File	Folder	Description
`json2puml.exe`	`bin`	Windows command line executable

File	Folder	Description
`json2pumlui.exe`	`bin`	Windows UI version to simplify the configuration development.
`json2pumlservice.exe`	`bin`	Windows command line application which provides the microservice.
`json2pumlwindowsservice.exe`	`bin`	Windows service application which provides the microservice.
`json2puml`	`bin\linux`	Linux command line executable
`json2pumlservice`	`bin\linux`	Linux command line application which provides the microservice.
`release-notes.txt`	`documentation`	Version history
`json2puml introduction.pdf`	`documentation`	Short introduction presentation
`json2puml documentation.pdf`	`documentation`	Full documentation
`json2puml.yaml`	`Documentation`	OpenAPI based swagger definition which describes the microservice.
`PlantUML<version>.jar`	`PlantUML`	Java executable to convert the generated puml files into the graphical output.
`*`	`samples`	Examples of definition and list files for various API's.

3.4 Setup Options

The setup program has the following options:

3.4.1 Add bin directory to PATH variable

When this option is activated, the installation `bin` folder will be added to the environment parameter `%Path%`.

3.4.2 Download PlantUML Jar File

When this option is activated, the following steps will be executed:

- Download a released PlantUML jar file from Github (https://github.com/PlantUML/PlantUML/releases/download/)
- The file will be installed in the `PlantUML` folder.

- The environment parameter `%PlantUmlJarFile%` will be created/updated.

3.4.3 Add sample files

When this option is activated a set of example configuration files will be installed.

This includes examples for the following API definitions:

- jsonplaceholder
  https://jsonplaceholder.typicode.com/
- SpaceX
  https://github.com/r-spacex/SpaceX-API
- SWAPI - The Star Wars API
  https://swapi.dev/
- TMForum
  http://ww.tmforum.org
- TVMaze
  https://www.tvmaze.com/api

3.5 Environment Parameters

The following environment parameters are created/updated used:

Parameter	Description	Setup Managed
`Path`	The bin folder should be added to the path to simplify the execution of the json2puml.	Yes
`PlantUmlJarFile`	System wide definition where the PlantUML jar file is installed. This allows to include the generation of the images without the need to define the path to the jar file in the definition file or as a command line parameter.	Yes, when the option “Download PlantUML jar file” is activated
`Json2PumlDefinitionFile`	System wide definition of the default definition file to be used when no definition file is defined at command line.	No, manual action needed
`Json2PumlCurlAuthenticationFile`	System wide definition of the default definition file to define user	No, manual action needed

Parameter	Description	Setup Managed
	specific authentication parameter when using curl (See also 0 Curl)	

3.6 PlantUML Installation

Json2puml generates only PlantUML configuration files.
If configured it is possible to call the PlantUML software to also generate automatically the image files base on the configuration files.

For doing this the following prerequisites must be enabled:

3.6.1 Java Runtime

Java Runtime must be installed.
The `java` command must be available in the current search path.

3.6.2 PlantUML software

A valid PlantUML jar file must be installed and available on the computer.

This could be done manually or by the setup program activating the option "Download PlantUML jar file" (see 3.4 Setup Options).

After the generation of the PlantUML configuration file the PlantUML software is called to generate the wished output files.

3.6.3 Output generation

Json2punlThis is done calling the following command line:

```
java -jar <parameter> <plantum.jar> <configuration.puml> <format>
```

3.6.3.1 <parameter>

This is an additional runtime parameter which could be used to influence the PlantUML generation.

<u>Example:</u>

```
-DPLANTUML_LIMIT_SIZE=8192
```

This parameter can be defined at the following places (searching in the defined order, stopping when a value is defined):

1. json2puml command line parameter
   (see 4.2 Command Line Parameters)
2. json2puml configuration file

3.6.3.2 <PlantUML.jar>

This parameter must reflect the path to a PlantUML jar file.

The parameter can be defined at the following places (searching in the defined order, stopping when a value is defined):

1. json2puml command line parameter
   (see 4.2 Command Line Parameters)
2. json2puml configuration file

3. User environment parameter `%PlantUmlJarFile%`
   (see 3.5 Environment Parameters)

3.6.3.3 <configuration.puml>

This is the name of the generated PlantUML output file. This parameter will automatically determined.

3.6.3.4 <format>

This is the option to define the output format.
This parameter will automatically determined based on the command line parameter `/outputformat` (see 4.2 Command Line Parameters) or the "`outputFormats`" parameter in the configuration file (See 6.6.8.11 outputFormats).

4 How to use

```
Json2puml.exe /inputfile=data\sample.JSON /leadingobject=individual
      /option=detailed,compact /outputformat=png
```

This generates the puml file and the corresponding png image.

The output will be generated into a subfolder which is defined in the configuration file. The output folder will automatically be generated.

4.1 Help Screen

```
*****************************************************************************************************************
 json2puml v2.0.7.59 - Command line converter JSON to PUML
 ******************************************************************************************************************
 /?                                Showing this Help screen
 /plantumljarfile:<file>           PlantUML Jar file which should be used to generate the sample images. If
                                   defined this parameter overwrites the corresponding parameter in the definition
                                   file
 /javaruntimeparameter:<param>     Additional parameter which will be added to the java call when calling the
                                   PlantUML jar file to generate the output formats.

 /configurationfile:<file>         Global base configuration of json2puml. Overwrites the
                                   "Json2PumlConfigurationFile" environment parameter.
 /parameterfile:<file>             ParameterFileName which contains a set of command line parameters in one file
 /definitionfile:<file>            DefinitionFileName which contains the configuration of the mapper
 /optionfile:<file>                OptionFileName which contains only the configuration of one option which then
                                   will be used for generation
 /option:<name>                    Name of the option group of the DefinitionFileName which should be used to
                                   generate the files
 /formatdefinitionfiles            Flag to reformat the used definition files.

 /inputfile:<file>                 Filter to find the JSON files to be migrated (Wildcard supported)
 /input list file:<file>             Listfile which contains the coniguration to handle list of different files to
                                   be migrated as one big file
 /leadingobject:<name>             Name of the property which should be used as highest level of the json objects
                                   This parameter is only needed for the single file conversion
 /splitinputfile                   Flag to define if the InputFileName should be split up in single files.
                                   This option is splitting the input file in separate files when the leading
                                   structure of the input is an array. Then every record of the array will be
                                   generated as a single file.
                                   This option is only valid when generatedetails is activated
 /splitidentifier:<identifier>     Name of the property which defines the name/filename of the splitted json
                                   element

 /curlauthenticationfile:<file> CurlAuthenticationFileName which contains the central authentication
                                   configuration when using the curl automations
 /curlparameterfile:<file>         CurlParameterFileName which contains additional variables to enable a dynamic
                                   configuration when using the curl automations
 /curlparameter:<name>=<value>     Single curl command line parameter, defined as name and value.
                                   The parameter can be used multiple times to define more than one curl
                                   parameter.
                                   The command line parameter will overwrite parameter from the parameter file
                                   having the same name.

 /summaryfile:<file>               Filename of the generated summary file
 /baseoutputpath:<path>            Base path which will be combinded with the outputpath, to define where the
                                   generated files will be stored. This parameter overwrites the path configured
                                   in the definition file
 /outputpath:<path>                Output path for the generation of files. This parameter overwrites the path
                                   configured in the definition file
 /outputsuffix:<suffix>            Additional suffix added to the name of the generated files. This parameter
                                   overwrites the value configured in the definition file
 /outputformat:<format>            Format of the generated Puml converters (Allowed values: ,png,svg,pdf)
 /openoutput:[<format>]            Flag to define if the generated files should be opened after the generation.
                                   The files will be opened using the default program to handle the file format.
                                   Optional the files to be opened can be restricted by the format types (Allowed
                                   values: ,png,svg,pdf,puml,zip)

 /generatedetails:<boolean>        This allows to overwrite the generateDetails property of the Input list file
 /generatesummary:<boolean>        This allows to overwrite the generateSummary property of the Input list file

 /identfilter:<filter>             Value to filter/allow only objects where the ident matches to this filter
                                   value.
 /titlefilter:<filter>             Value to filter/allow only objects where the title matches to this filter
                                   value.

 /group:<group>                    Group name which can be used in the output path and output suffix, it's
                                   overwriting the value from the Input list file
 /detail:<detail>                  Additional detail name which can be used in the output path and output suffix
 /job:<job>                        Additional job name which can be used in the output path and output suffix

 /generateoutputdefinition         Flag to define if the merged generator definition should be stored in the
                                   output folder.
 /debug                            Flag to define that a converter log file should be generated parallel to the
                                   puml file

 /description                      Description of the generated result. This information will be put into the
                                   legend of the image.
```

4.2 Command Line Parameters

Parameter	Description
/?	Shows the help screen with all command line parameters
/plantumljarfile: <file>	Defines an alternative path to the PlantUML jar file. If defined this parameter overwrites the corresponding parameter in the definition file.

Parameter	Description
/javaruntimeparameter: <param>	Additional parameter which will be added to the java call when calling the PlantUML jar file to generate the output formats.
/configurationfile: <file>	
/parameterfile: <file>	
/definitionfile: <file>	Definition file which contains the configuration of the converter. When the parameter is not defined or the file is not known the filename is fetched from environment parameter "%Json2PumlDefinitionFile%". When this is also not defined or not value the default file "json2pumldefinition.json" is searched in the current directory.
/optionfile: <file>	Additional configuration file which contains only the option definition to be used for the conversion.
/option: <name>	Defines which configuration option should be used to generate the outcome. It is possible to define multiple options separated by "," or ";". This parameter will be ignored when the optionfile parameter is used.
/formatdefinitionfiles	Flag to reformat the used definition files. If it is defined all definition files which are defined via command line will be reformatted. The original files will be saved with the extension ".bak".
/inputfile: <file>	Filter to find the JSON files to be migrated (Wildcard supported). (See: 5.4.1 Single File)
/inputlistfile: <file>	Name of the input list file, which allows to run the generation for multiple JSON files at the same time and to combine the results into one overall image. (See: 5.4.2 List File)
/leadingobject: <name>	Name of the property which should be used as highest level of the JSON objects in the input files. (See: 7.1 leadingObject) This parameter is only needed for the single file conversion.
/splitinputfile	Flag to define if the inputfile should be split up in single files. This option is splitting the input file in separate files when the leading structure

Parameter	Description
	of the input is an array. Then every record of the array will be generated as a single file. This option is only valid when generatedetails is activated. (See 6.5.3.10 splitIdentifier) This parameter is only needed for the single file conversion.
/splitidentifier: <identifier>	Name of the property which defines the name/filename of the splitted json element. (See 6.5.3.10 splitIdentifier) This parameter is only needed for the single file conversion.
/curlauthenticationfile: <file>	CurlAuthenticationFile which contains the central authentication configuration when using the curl automations (See 0 Curl).
/curlparameterfile: <file>	CurlParameterFile which contains additional variables to enable a dynamic configuration when using the curl automations (See 0 Curl).
/curlparameter:<name>=<value>	Single curl command line parameter, defined as name and value. The parameter can be used multiple times to define more than one curl parameter. The command line parameter will overwrite parameter from the parameter file having the same name (See 0 Curl).
/outputformat: <format>	Comma separated list of output formats to be generated. This parameter overwrites the output format definition in the definition file (See 6.6.8.11 outputFormats)
/baseoutputpath: <path>	
/outputpath: <path>	Outpath for the generation of files. This parameter overwrites the path configured in the definition file and in the input list file. (See 5.5.1 Output path and 6.6.8.9 outputPath)
/outputsuffix: <suffix>	Additional name suffix for the generation of files.

Parameter	Description
	This parameter overwrites the path configured in the definition file and input list file. (See 5.5.1 Output path and 6.6.8.9 outputPath)
/openoutput: [<format>]	Flag to define if the generated files should be opened after the generation. The files will be opened using the default program to handle the file format. Optional the list of files to be opened can be restricted by the format types (Allowed values: png, svg, pdf, puml)
/generatedetails: <boolean>	Defines if an output should be generated for any included JSON input file. This parameter is only relevant in the list mode, If defined this parameter overwrites the corresponding parameter in the input list file.
/generatesummary: <boolean>	Defines if a summary output should be generated as a combination of all included JSON input file. This parameter is only relevant in the list mode. If defined this parameter overwrites the corresponding parameter in the input list file.
/identfilter: <filter>	Value to filter/allow only objects where the ident matches to this filter value. (see 5.3.5 Filtering)
/titlefilter: <filter>	Value to filter/allow only objects where the title matches to this filter value. (see 5.3.5 Filtering)
/group: <group>	Group name which can be used in the output path and output suffix. It's overwriting the value from the input list file.
/detail: <detail>	Additional detail name which can be used in the output path and output suffix. It's overwriting the value from the input list file. (see 5.5 Generated Files)
/job: <job>	Additional job name which can be used in the output path and output suffix. It's overwriting the value from the input list file. (see 5.5 Generated Files)
/generateoutputdefinition	Flag to define if the merged generator definition should be stored in the output folder.

Parameter	Description
	(see 5.5 Generated Files)
/debug	Flag to define that a converter log file should be generated parallel to the puml file
/description	The description will be added to the legend of the generated image. This allows to put additional information's about the context of the data into the generated image.

4.3 QuickStart

The minimum parameter you have to define to use json2puml are the "/definitionfile" (which contains the definition how to convert) and the "/inputfile" or the "/input list file" (which are containing the data to be converted).

Have a look into the samples folder.
Every folder has a definition file and a "information.txt" file which describes where you can find sample files or further information for the sample.

```
json2puml /definitionfile:swapidefinition.json /inputfile:"swapi-
      *.json"
```

The output will look similar to this:

```
json2puml v1.2.0.23 - Command line converter JSON to PUML

Current parameters:
    /env:plantumljarfile
      C:\Users\jensf\AppData\Local\Programs\Json2Puml\PlantUML\PlantUM
      L-1.2022.4.jar
    /definitionfile            swapidefinition.json
    /inputfile                 swapi-*.json (6 files found)


[  1/  1] Convert output\summary\default\summary.default.json
                to output\summary\default\summary.default.puml
                puml generated
                png  generated
                svg  generated
```

Then start to build your own definition file for your data files.

5 How the system is working

5.1 Main components

Json2puml consists of 3 main components:

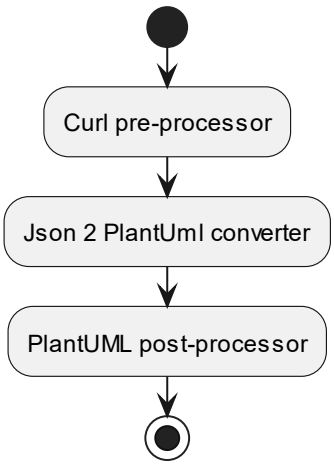


Figure 19 json2puml main components

5.1.1 Curl pre-processor

The curl pre-processor is an optional step. It allows to collect json files by calling a configures set of API calls. The "`curl`" command will be used on command line to execute the configured API calls.

When the curl pre-processor is not used the files to be converted must be already existing.

These API calls can be configured via parameters. The parameters can also be fetched from one curl result and be used for the next api calls. This allows to have dynamic chains of api calls based on one input parameter.

<u>Example:</u>

Fetching data from a shop system based on an input parameter "OrderId"

	API Call	Input Parameter	Output Parameter
1	Get /order	${orderId}:2022-1234	${custNumber}:4711 ${addressId}:8819
2	Get /customer	${custNumber}:4711	${addressId}:1219
3	Get /orderItems	${orderId}:2022-1234	${product}:a14 ${product}:a177 ${product}:b88
4	Get /address	${addressId}:8819 ${addressId}:1219	
5	Get /product	${product}:a14 ${product}:a177	

		${product}:b88	

The result would a set of json files reflecting 1 order object, 1 customer object, 3 order item object, 2 address objects and 5 product objects, which then can be combined into one overall result.

The curl pre-processor is configured in the input list file (see 6.5 Input list File), the curl parameter file(see 6.10 Curl parameter file) and the curl authentication file (see 6.11 Curl authentication file)

5.1.2 Json 2 PlantUML converter

The Json 2 PlantUML converter a set of json files into PlantUML files. These files can then be used by the PlantUML post-processor to generate the graphical representations of the json files.

The converter works on API specific definitions, which will be configured in the converter definition file (see 6.6 Converter definition file).

5.1.3 PlantUML post-processor

The PlantUML post-processor used the generated PlantUML script files and the PlantUML jar file to generate the images based on the script files.

The PlantUML jar file and a valid java runtime environment must be locally available on the operating system.

The path to the needed PlantUML jar file will be defined on the variable `plantumJarfile`. (see 6.2.4.1 PlantUML jar filename).

In addition, the `javaRuntimeParameter` variable can used to configure additional parameters to be added to the java command line (see 6.2.4.2 Java runtime parameter)

The following command line will be used :

```
java <javaRuntimeParameter> -jar <plantumJarFile> <plantumlScriptFile>
      <format>
```

5.2 Curl pre-processor

In addition to using static input files, it is also possible to get the content of the files using a curl command.

The curl support is based on the definitions of the input list file.

Here it is possible to define for every single file definition in the list file that this file should be fetched dynamically.

After all files are handled by the pre-processor the converter will start.

The curl pre-processer is based on the curl command, which will be called via command line and must be available on the operating system.

5.2.1 Curl parametrisation

To reuse a list definition file for visualisation of different dataset it is possible to use curl parameters to fetch dynamically different data.

For this the system allows to define for every converter run a set list of parameter value pairs which will be used to update the curl command before execution by replacing every occurrence of a known curl parameter name with the corresponding curl parameter value.

5.2.1.1 Defining the use of curl parameters

For this the curl parameter must be defined in the configured configuration parameters in the following pattern “`${<name>}`”. Before executing a curl command all existing strings following this pattern will be replaced by the value of the corresponding curl parameter.

The search of the name is not case sensitive.

<u>Example:</u>

- Parameter name: id
- Parameter value: 1
- Configured `curlUrl`: "`/users/${id}`"

Before executing the `curlUrl` will be replaced to “`/users/1`”.

5.2.1.2 Curl filename parametrisation

The curl parametrisation can also be used to generate the output files based on the curl parameter values.

The curl parametrisation of filenames can be used for the summary file name (6.5.2.12 summaryFileName) and for the input file name of the SingleListFile (6.5.3.1 inputFile).

As a restriction: For a single file the `curlOutParameter` of this file can only be used for the filename of the files which are defined after the current file.

5.2.2 Curl command

For every file defined in the input list file the curl command will be executed when a Url is defined.

The curl command is based build following this pattern:

```
curl <curlOptions> -o <outputFile>
      --url <curlBaseUrl>/<singleCurlUrl><curlUrlAddon>
```

The different parts of the command are build based on the following logic

- `<curlOptions>`:
  Concatenation of `inputListFile.curlOptions` and `singleInputFile.curlOptions`
- `<outputFile>`:
  Combination of `singleInputFile.inputFile` and `singleInputFile.curlFileSuffix.`
- `<curlBaseUrl>`:
  If defined the `singleInputFile.curlBaseUrl` will be used, otherwise the `inputListFile.curlBaseUrl` will be used.
- `<singleCurlUrl>`:
  `singleInputFile.curlUrl`
- `<curlUrlAddon>`:
  `inputListFile.curlUrlAddon`

To activate the curl execution the `curlUrl` parameter of the single list file definition and the `curlBaseUrl` parameter must be defined. Otherwise, the existing static file will be used.

5.2.3 How to use it

To understand how the curl pre-processor is working you can have a look into the `samples` directory. Most of the examples are based on the pre-processor to fetch data for dedicated scenarios via api.

As an example, we can have a look into the jsonplaceholder example. The jsonplaceholder api is a REST training API which could be used by developers to learn how to work with such api's.

The api has some random data and is based on the following data model:

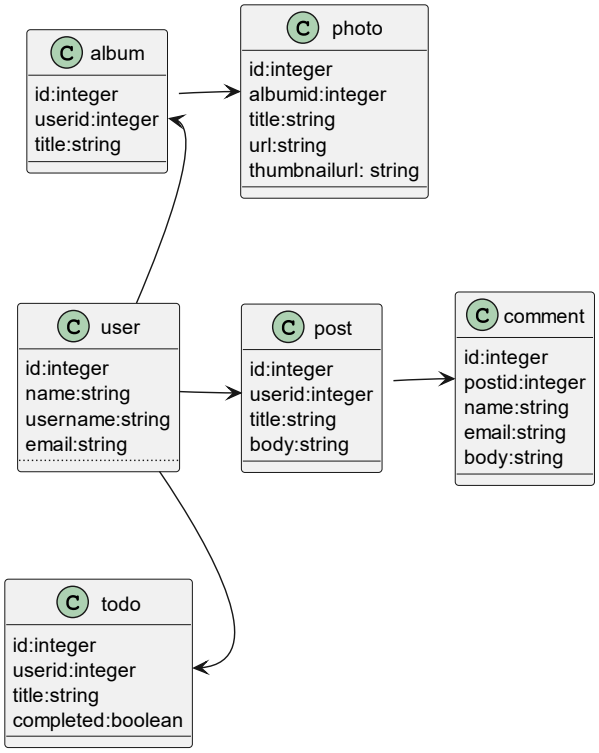


Figure 20 jsonplaceholder data model

The input list file `placeholder_inputlist_curl.json` collects and visualised data based on one user id. The id of the user can be defined via command line parameter.

```
json2puml /parameterfile:placeholder_parameter_curl.json
      /curlparameter:userid=1
json2puml /parameterfile:placeholder_parameter_curl.json
      /curlparameter:userid=3
```

In the following steps the `userid = 1` will be used as a baseline for the shown example data.

5.2.3.1 Global definition

```
  "curlBaseUrl": "https://jsonplaceholder.typicode.com/",
  "curlUrlAddon": null,
  "curlOptions": "-f",
  "summaryFile": "placeholder_summary_${name}",
```

The global definition of the input list file only defines

- `"curlBaseUrl": "https://jsonplaceholder.typicode.com/"`
  This url will be the basis for all url calls
- `"summaryFile": "placeholder_summary_${name}"`
  The generated summary file will contain the name of the defined user based

on the curl parameter `${name}`. This parameter will be filled after the data for the defined user has been fetched.

5.2.3.2 Inputfile "users.json"

The first file to be fetched will contain the data of the defined user.

5.2.3.2.1 Configuration

```
{
      "inputFile": "users.json",
      "curlFileSuffix": "_${userid}",
      "curlUrl": "/users/${userid}",
      "curlOptions": null,
      "curlOutputParameter": [
        {
          "${name}": "name"
        }
      ]
    },
```

5.2.3.2.2 Filename

The generated filename will be `users_1.json` (based on the parameters `inputFile` and `curlFileSuffix`) .

5.2.3.2.3 Command

The following command will be executed:

```
curl -f --url https://jsonplaceholder.typicode.com/users/1
      --output users_1.json
```

5.2.3.2.4 Json Result (reduced)

```
{
      "id" : 1,
      "name" : "Leanne Graham",
      "username" : "Bret",
      "email" : "Sincere@april.biz"
}
```

5.2.3.2.5 Output curl parameter

The name of the user with the id 1 will be fetched from the attribute "`name`".

As a result, the following curl parameters values are valid after the call:

Variable	Value
${userid}	1
${name}	Leanne Graham

This `${name}` parameter will then be used to generate the name of the summary file: "`placeholder_summary Leanne Graham.json`"

5.2.3.3 Inputfile “albums.json”

This file will contain the albums related to the defined user.

5.2.3.3.1 Configuration

```
{
      "inputFile": "albums.json",
      "curlFileSuffix": "_${userid}",
      "curlUrl": "albums?userId=${userid}",
      "curlOutputParameter": [
        {
          "name": "${albumid}",
          "value": "id",
          "maxValues": "2"
        }
      ]
    },
```

5.2.3.3.2 Filename

The generated filename will be `albums_1.json` (based on the parameters `inputFile` and `curlFileSuffix`).

5.2.3.3.3 Command

The following command will be executed:

```
curl -f --url https://jsonplaceholder.typicode.com/albums?userId=1
      --output albums_1.json
```

5.2.3.3.4 Json Result (reduced)


```
[
        {
            "userId" : 1,
            "id" : 1,
            "title" : "quidem molestiae enim"
        },
        {
            "userId" : 1,
            "id" : 2,
            "title" : "sunt qui excepturi placeat culpa"
        },
        {
            "userId" : 1,
            "id" : 3,
            "title" : "omnis laborum odio"
        },
        {
            "userId" : 1,
            "id" : 4,
            "title" : "non esse culpa molestiae omnis sed optio"
        },
        {
            "userId" : 1,
            "id" : 5,
            "title" : "eaque aut omnis a"
        },
        {
            "userId" : 1,
            "id" : 6,
            "title" : "natus impedit quibusdam illo est"
        },
        {
            "userId" : 1,
            "id" : 7,
            "title" : "quibusdam autem aliquid et et quia"
        },
        {
            "userId" : 1,
            "id" : 8,
            "title" : "qui fuga est a eum"
        },
        {
            "userId" : 1,
            "id" : 9,
            "title" : "saepe unde necessitatibus rem"
        },
        {
            "userId" : 1,
            "id" : 10,
            "title" : "distinctio laborum qui"
        }
]
```


5.2.3.3.5 Output curl parameter

The list of album id's is fetched from the attribute "`id`". To reduce the number of fetched parameters (and based on that make the generated image better readable) the optional parameter `"maxValues": "2"` is used to limit the number of fetched variables to 2,

The albums of the user with the id 1 will be fetched from the attribute “`id`”.

As a result, the following curl parameters values are valid after the call:

Variable	Value
${userid}	1
${name}	Leanne Graham
${albumid}	1
${albumid}	2

5.2.3.4 Inputfile “photos.json”

With this file the photos which are related to the albums fetched in the second steps will be collected.

5.2.3.4.1 Configuration

```
{
      "inputFile": "photos.json",
      "curlFileSuffix": "_${userid}_${albumid}",
      "curlUrl": "photos?albumId=${albumid}",
      "curlOutputParameter": []
    },
```

5.2.3.4.2 Filename

The `curlFileSuffix` and the `curlUrl` parameter are containing the reference to two curl parameters: `${userid}` and `${albumid}`. When a curl parameter is part of these attributes then the curl command will be execute on a permutation of all values of the used curl parameters.

The generated filenames will be `photos_1_1.json` and `photos_1_2.json` (based on the parameters `inputFile` and `curlFileSuffix`).

5.2.3.4.3 Command

In this scenario this will lead to the two following commands:

```
curl -f --url https://jsonplaceholder.typicode.com/photos?albumId=1
      --output photos_1_1.json
curl -f --url https://jsonplaceholder.typicode.com/photos?albumId=2
      --output photos_1_2.json
```

5.2.3.4.4 Json Result (reduced)


```
[
  {
    "albumId": 1,
    "id": 1,
    "title": "accusamus beatae ad facilis cum similique qui sunt",
    "url": "https://via.placeholder.com/600/92c952",
    "thumbnailUrl": "https://via.placeholder.com/150/92c952"
  },
  {
    "albumId": 1,
    "id": 2,
    "title": "reprehenderit est deserunt velit ipsam",
    "url": "https://via.placeholder.com/600/771796",
    "thumbnailUrl": "https://via.placeholder.com/150/771796"
]
```

```
[
  {
    "albumId": 2,
    "id": 51,
    "title": "non sunt voluptatem placeat consequuntur rem incidunt",
    "url": "https://via.placeholder.com/600/8e973b",
    "thumbnailUrl": "https://via.placeholder.com/150/8e973b"
  },
  {
    "albumId": 2,
    "id": 52,
    "title": "eveniet pariatur quia nobis reiciendis laboriosam ea",
    "url": "https://via.placeholder.com/600/121fa4",
    "thumbnailUrl": "https://via.placeholder.com/150/121fa4"
  }
]
```


5.2.3.4.5 Output curl parameter

For this file no curl output parameter are defined.

5.2.3.5 Further files

- The `posts.json` will be fetched based on the `${userid}` and deliver a list of `${postid}`
- The `comments.json` will be fetched based on the `${postid}`
- The `todos.json` will be fetched based on the `${userid}`.

5.2.4 API Security / OAuth

5.2.4.1 Curl Authorisation Parameter

When using API's it is quite often necessary to define user credentials for the API calls.

In principle these credentials can be configured as part of the input list file or as normal curl parameter. From a security point this is not a good solution.

- The input list file is designed to be handled via source code repository, so that multiple users can work with these definitions.
  And in a source code repository it is the best practice to not have any user credentials in the files.
- The generated curl commands are logged in execution log files and in the summary file list. This log files should also not contain any user credentials.

To solve this problem json2puml support a second type of curl parameter, the curl authentication parameter.

Curl authentication parameters are not written into the log files or into the summary file list. Curl authentication are working with

5.2.5 How to define curl parameter

Curl parameters can be defined based on the following ways:

- Curl command line parameter
- Curl parameter file
- Curl output parameter
- Curl authorisation file

5.2.5.1 Curl parameter via command line

5.2.5.1.1 Normal curl parameter

The allows to define curl parameters at the command line. A command line parameter will overwrite the value of a similar parameter from the curl parameter file.

The curl command line parameters can be defined via the `/curlparameter` command line parameter.

It is possible to define multiple command line curl parameter by repeating the `/curlparameter` call.

A command line curl parameter must be defined using the `<name>=<value>` pattern.

5.2.5.1.2 Curl authentication parameter

The curl authentication parameter can be defined in a similar way using the command line parameter `/curlauthenticationparameter`.

5.2.5.2 Curl parameter file

The curl parameter file can contain one single json record. Every element of this record will be interpreted as a name value pair for the curl parameter list.

The curl parameter file can be defined via the `curlparameterfile` command line parameter.

See 6.10 Curl parameter file

5.2.5.3 Curl output parameter

The curl output parameter definition allows to use a dedicated attribute of a curl result file as a new curl parameter for further curl calls.

As an example: As command line parameter the product name is defined as curl parameter. This product name is used as a filter for the first curl call. From the returned file the primary key of the product is fetched as a new curl parameter "`productid`". This `productId` is then used in the next call to fetch further details of the product from a different API.

A curl output parameter will overwrite the curl parameters with the same name coming from the curl parameter file or the curl command line parameter.

See 6.5.3.7 curlOutputParameter

5.2.5.4 Curl authentication file

The curl authorisation file allows to separate the user specific authentication parameter (like personalized tokens, clientId and clientSecret) from the list definition.

This allows to have for a team a common list definition to fetch the details of the api independent of the developer specific authentication parameters.

The corresponding authentication parameter will be fetched based on `baseUrl` of the current curl call and handled like the other curl parameter before executing the curl call.

The curl parameter file can be defined via the `curlauthenticationfile` command line parameter and the `Json2PumlCurlAuthenticationFile` environment variable.

Authentication parameters will not be written to any logfile and not to the standard out.

See 6.11 Curl authentication file

5.3 Converter

5.3.1 Basic Flow

The generator is working recursively on JSON structures. Based on the content of the JSON structure a list of objects and a list of relations between these objects will be identified.

The objects are created as PlantUML “class” and the relations as PlantUML links between two classes.

If multiple files are handled the objects and relations of these files are merged into one big result set.

To uniquely identify an object and to build the relations between these objects an object type and an object identifier will be defined based on the property names and values of the JSON structure.

The program follows the following high-level structure.

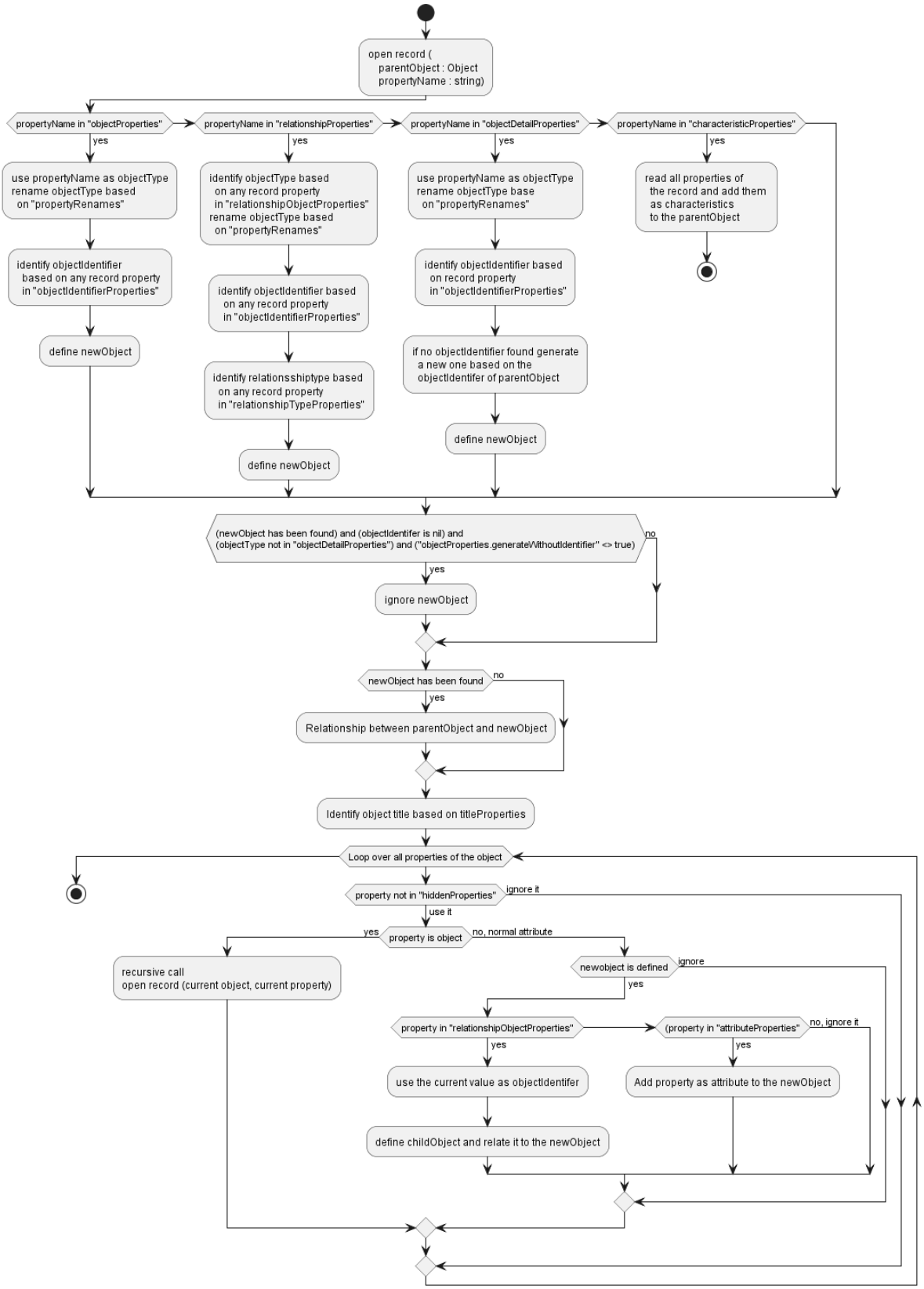


Figure 21: Basic flow how the converter is working

5.3.2 Object Identification

An object in the generated result will be identified by the following patterns:

- An object can have a type and an identifier.
- Based on the configuration parameter "`identifyObjectByTypeAndIdent`" only the ident or ident and type are used to identify an object. (see 6.6.8.2 identifyObjectsByTypeAndIdent)
- The type of an object will be identified by the following rules:
  - First the name of the record property or if it is the leading record, the name of the command line parameter "`leadingObject`".
  - If the JSON record contains a property matching one of the "`objectTypeProperties`" (see also 6.6.8.15 objectTypeProperties) the corresponding value is used.
  - The type of the object can be mapped to consolidated values using the "`objectTypeRenames`" parameter (see 6.6.8.17 objectTypeRenames). This allows to harmonize the object names.
  - If the objects are unique based on the identifier only it can happen that the object is found at different places in the JSON structure.
    In this case it is possible that object type is different for the different places.
    E.g., in TMF there is the pattern of the "`relatedParty`", which is a reference to a party object. If this is pointing to an `individual` object then this JSON object could have the object type "`individual`", but both are sharing the same "`identifier`".
    In this case the order of the elements in the configuration parameter "`objectProperties`" is used to define if the object type can be overwritten or not.
    E.g., the objectProperties list contains the value (in this order) "`individual, party, relatedParty, engagedParty`". When an object is found at two places with the types "`relatedParty`" and "`individual`" the object will have the type "`individual`".
- When for an object no identifier can be found and the `objectTypeProperties.generateWithoutIdentifier` is not set to true this object will be ignored and not created (see 6.9.2.2 generateWithoutIdentifier).

5.3.3 Object Relationships

The systems build relationship between objects based on the hierarchy of the JSON structure.

Whenever a detailed record in a JSON object is found and the detail is identified as an object the relationship between the new object and the parent object will be established.

The relationship will be built on the unique identifier of the two objects (5.3.2 Object Identification). If an object with the identification is defined/used at many places in the JSON structures multiple relations will be shown to this object.

For every relationship a relationship type can be defined. This type and the name of the property which has defined the relationship will be shown as text near the relationship arrow and in the from / to relationship lists.

Based on the name of the property and the relationship type it is possible to defined based on the parameter “`relationshipTypeArrowFormats`” (see 6.6.8.22 relationshipTypeArrowFormats) how the arrows should be painted.

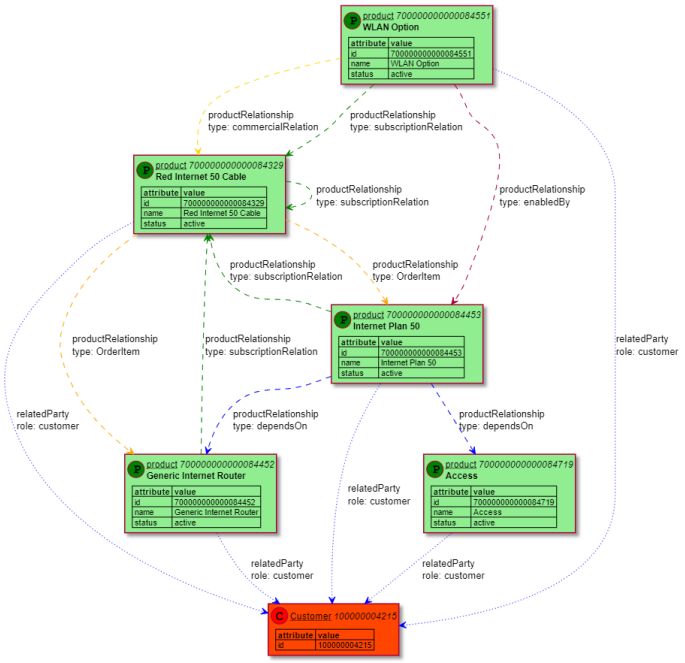


Figure 22 Object relationship example

5.3.4 Detail Objects / Characteristic properties

Based on the parameter "`generateWithoutIdentifier`" objects are normally only created when the identifier has been found for this object.

But often in JSON an object has detail records with additional information’s which are describing this object but are not own objects with own identifiers.

To show these information’s also in the diagrams there are two different ways to handle this.

5.3.4.1 Detail Objects

Detail objects will be created as separated object, but without identifier.

Detail objects will be shown in the from / to relationships, and detail objects can have own detail objects and they can have characteristics.

Detail objects will be identified based on the parameter “`objectDetailProperties`” (see 6.6.8.16 objectDetailProperties)

Detailobjects will be visualized with a diamond at the relationship:

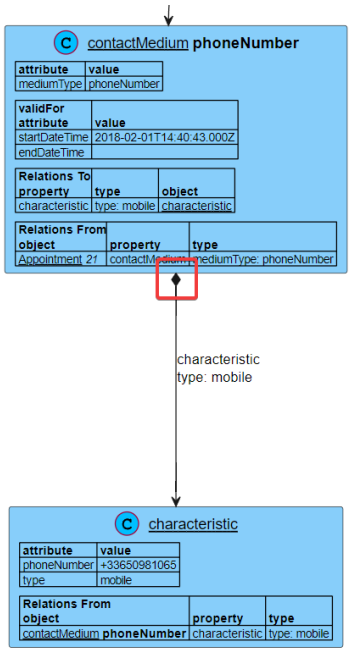


Figure 23 Visualisation of a detailed object

5.3.4.2 Characteristics properties

Characteristic properties will be included into the parent object similar to the attributes list.

There are two different type of characteristic properties:

- List
  This is valid if the property is an array.
  For the list mode it is possible to define the three different properties of the list to be included into the characteristic list, a “`name`” property, a “`value`” property and an additional “`info`” property.
- Record
  The property name will be used as “name” and the property value as “value” for the characteristic table.

For each characteristic property a new table in the generated object will created.

A characteristic property cannot have further details and the converter will not create any relationships for objects which are “under/behind” a characteristic property).

Characteristic properties will be identified based on the parameter “`characteristicProperties`”. (see 6.9.1.2 characteristicProperties)

5.3.5 Filtering

The generator has the possibility to filter out / to allow only objects based on the command line parameter /`identfilter` and /`titlefilter`. (see 4.2 Command Line Parameters)

When these parameters are defined only objects will be shown where the ident or title matches one of the filter criteria, or which are directly linked to these matching objects.

When both parameters are defined an object is shown when one of the filters is matching.

5.3.6 Generated Outcome

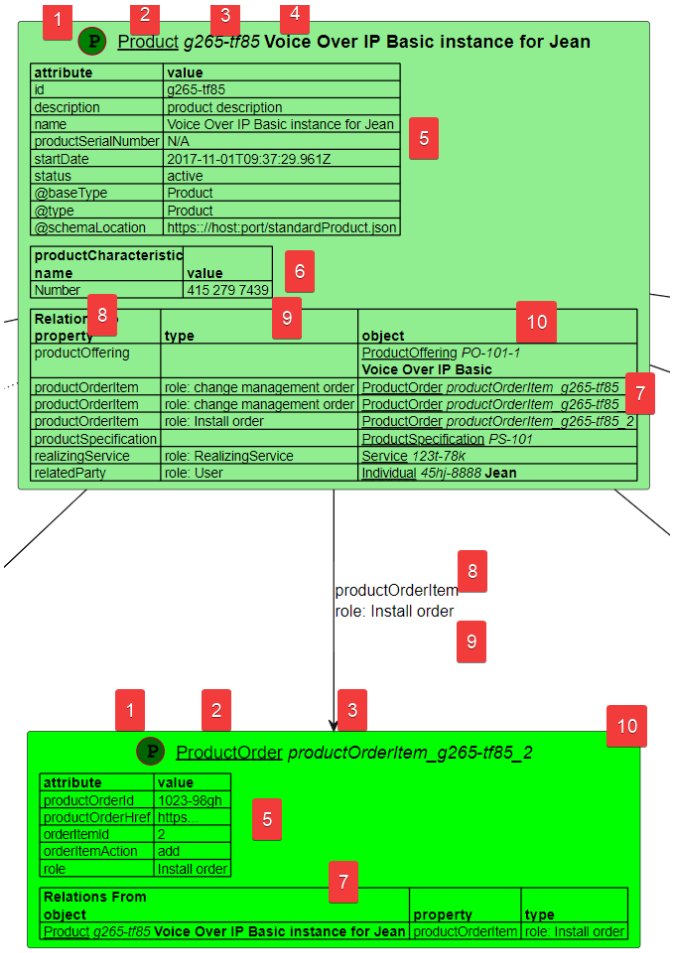


Figure 24 Generated Output Example

1. Object Icon
   Color is based on 6.9.6.2 iconColor
2. Object Type
   Based on 6.6.8.15 objectTypeProperties, 6.6.8.13 objectProperties
3. Object Ident
   Based on 6.6.8.18 objectIdentifierProperties
4. Object Title
   Based on 6.6.8.14 objectTitleProperties
5. List of Object Attributes
   Based on 6.6.8.12 attributeProperties
6. Characteristics
   Based on 6.9.1.2 characteristicProperties

7. From / To Relationship Lists
8. Relationship Property
9. Relationship Type
   Based on 6.6.8.21 relationshipTypeProperties
10. Object information of the related object

5.4 Operational Modes

5.4.1 Single File

In this mode only files based on one filter will be converted.

When the single file mode is used normally the `leadingobject` parameter needs to be defined additionally.

See 7.1 leadingObject

5.4.2 List File

The list file mode allows to generate the PlantUML and image outputs for a list of JSON files.

The list file allows to fetch the source files before converting by calling the curl command (see 0

Curl).

It allows also to generate a combined output file which combines the data of all found input files.

5.4.3 Detail / Summary generation

The modes are defining if for every input file or only for the summary files the generator is executed.

Both modes can be independently activated or deactivated.

To identify the mode the following order is executed (the first found value has the highest priority):

1. Command line parameter /generateDetails & /generateSummary
2. The corresponding parameters of the single list file (see 6.5.2.1 generateSummary and 6.5.2.2 generateDetails)

5.4.3.1 Generate Details

This mode means that for every found input file (base on the single file or list file parameter) the converter will be started and the defined images will be created.

5.4.3.2 Generate Summary

This mode merges all found source file in to one summary JSON file and converts this summary file.

5.5 Generated Files

5.5.1 Output path

Using the parameter "`outputPath`" it is possible to define a path where all generated files are stored. This allows to have the source and output files clearly separated.

The output path is a parameter of the current definition option. (see 6.6.8 Single)

The output path can be defined absolute or relative.

- An absolute path will be used without any changes
- A relative path will be created and used at the directory of the current input file.

Example relative path:

The input file is located at: `c:\data\json`
The output path is defined as: `output`
All files will be generated at: `c:\data\json`

The generator tries to create the output path if it does not exist. If this fails the path of the input file will be used.

The output path supports two string replacement which allows to structure the directory based on the name of the group coming from the input file and the name of the chosen option coming from the `/option` or `/optionfile` command line parameters

Parameters:

- `<job>`
  will be replaced with the name of the job
- `<group>`
  will be replaced with the name of the group
- `<option>`
  will be replaced with the name of the chosen option
- `<detail>`
  will be replaced with the value of the command line parameter `/detail`
- `<file>`
  will be replaced with the filename (without extension) of the handled source name

Example:

```
outputPath = output\<option>\<group>\<file>
```

will be translated to (based on the group "`crm`", the option "`default`" and the file "`TMF632_sample.json`")

```
output\default\crm\TMF632_sample
```

The output path can be defined in the configuration file (see 6.6.8 SingleOption), the input list file (see 6.5.2 InfoListFile) or as a command line parameter (4.2 Command Line Parameters). The value from the input list overwrites the configuration file value and the command line parameter overwrites both definition file values.

5.5.2 Output suffix

The output suffix will be added to then end of the original filename.

The output suffix is a parameter of the current definition option. (see 6.6.8 Single)

The output suffix allows the similar replacement like the output path using `<group>`, `<option>` and `<detail>`. This allows to easily identify the generated files when multiple options have been used.

The output suffix can be defined in the configuration file (see 6.6.8 SingleOption), the input list file (see 6.5.2 InfoListFile) or as a command line parameter (4.2 Command Line Parameters). The value from the input list overwrites the configuration file value and the command line parameter overwrites both definition file values.

5.5.3 Summary file

A summary file will be generated when more then one json files will be handled and the `generateSummary` option (see 4.2 Command Line Parameters and 6.6.8 Single) is active.

The default base name of the summary file is `summary.json`. This name will be enhanced with the output suffix of the current definition. The name can be configured when using the operational mode “List File” (see 5.4.2 List File) by using the `summaryFileName` option (see 6.5.2.12 summaryFileName) of the list file.

The summary file will be generated in the defined output path directory.

5.5.4 Zip File

The zip file will be generated when the `outputFormat` contains the format `zip` or when the `post /json2pumlRequestZip` microservice is used.

The zip file contains all files which where generated with the current `json2puml` execution.

5.5.5 PlantUML source File

The PlantUML file will be generated in the defined output path directory.

The filename will be the same then the input file which is converted. The file extension of the filename will be replaced with “`puml`”.

5.5.6 PlantUML output files

The PlantUML output file will be generated in the defined output path directory.

The filename will be the same then the input file which is converted. The file extension of the filename will be replaced with the format specific extensions “`png`”, “`svg`” and “`pdf`”.

6 Configuration

All configuration files are JSON based.

If a configuration file has not a valid JSON structure an error will be shown and the program will stop.

6.1 File overview

The application can be configured based on a set of json based configuration files.

6.1.1 Global configuration file

This file contains the system wide configuration of

- global filenames
- global folders
- Definitions of the service application (e.g. Port number)

6.1.2 Parameter file

This file allows to define a set of command line parameters into one file to have a simplified option to call the converter. The same file can be used as an input to the microservice based converter.

See 6.4 Parameter file

6.1.3 Input list file

This file allows to define a list of files to converted.

See 6.1.3 Input list file

6.1.4 Converter definition file

This file contains the different configurations how the JSON structures should be interpreted and converted.

See 6.6 Converter definition file

6.1.5 Converter definition option file

This file contains a single format option which allows to use a separate configuration option without modifying the main converter definition file.

6.1.6 Curl parameter file

This file can be used to have the curl parameters for a conversion defined in a file independent of the parameter file. This allows to call the same curl preprocessor without modifying the parameter file and having for every call separate curl parameter (file).

See 6.10 Curl parameter file

6.1.7 Curl authentication file

This file contains authentication secrets to be used by the curl pre-processor.

See 6.11 Curl authentication file

6.2 Variables

The system is based on some variables which could be configured / defined at various places.

The parameters will be searched at various places in ascending order. When a parameter is found at a defined place it will be used otherwise the next place will be searched.

6.2.1 Generator

6.2.1.1 Description

The `description` variable can be used to generate an additional description into the legend of the generated image.

Rank	Place	Detail
1	Command line	/description
2	Parameter file	{description}

6.2.1.2 Option

The `option` variable defines which format definition of the converter definition will be used to convert the json data. The variable can be used in the `output path` and `output suffix` variables as a replace variable `<option>`.

Rank	Place	Detail
1	Format option file	{option name}
2	Format option file	Constant “single”
3	Command line	/option
4	Parameter file	{option}
5	Input list file	{option}
6	Converter definition file	{defaultOption}

6.2.2 Generated folder and file names

6.2.2.1 Job

The variable can be used in the `output path` and `output suffix` variables as a replace variable `<job>`.

Rank	Place	Detail
1	Command line	/job

Rank	Place	Detail
2	Parameter file	{job}
3	Input list file	{job}
4	Input list file	<Name of input list file>

6.2.2.2 Group

The variable can be used in the `output path` and `output suffix` variables as a replace variable `<group>`.

Rank	Place	Detail
1	Command line	/group
2	Parameter file	{group}
3	Inputlist file	{group}

6.2.2.3 Detail

The variable can be used in the `output path` and `output suffix` variables as a replace variable `<detail>`.

Rank	Place	Detail
1	Command line	/detail
2	Parameter file	{detail}
3	Input list file	{detail}

6.2.2.4 Output suffix

The variable is added at the end of every generated file. In combination with the variables `<job>`, `<group>`, `<detail>` and `<option>` one definition set could be used to generate better readable output file names.

Rank	Place	Detail
1	Command line	/detail
2	Parameter file	{outputSuffix}
3	Input list file	{outputSuffix}

6.2.2.5 Base output path

The variable defines where the base directory for generating all files.

Rank	Place	Detail
1	Command line	/baseoutputpath
2	Global configuration file	{baseOutputPath}
3	Input list file	Path part of the filename of the input list file.
4	Parameter file	Path part of the filename of the parameter file

Rank	Place	Detail
5		Current directory where the executable has been started.

In a service application the ThreadId of the current OS thread will be added to the base output path to prevent file collisions when the application is handling multiple requests at the same time.

6.2.2.6 Output path

The variable allows to define an additional path which will be added to the `base output path`. In combination with the variables `<job>`, `<group>`, `<detail>` and `<option>` this can be used to structure the generated files when multiple executions are executed.

6.2.2.7 Output formats

The variable defines which files should be generated and converted.

Rank	Place	Detail
1	Command line	/outputformats
2	Parameter file	{outputFormats}
3	Input list file	{outputFormats}

6.2.3 Configuration files

6.2.3.1 Configuration filename

The variable defines the name of the global configuration file (see **Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.**).

Rank	Place	Detail
1	Command line	/configurationfile
2	Environment	%Json2PumlConfigurationFile%

6.2.3.2 Curl Authentication filename

The variable defines the name of the curl authentication file (see **Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.**).

Rank	Place	Detail
1	Command line	/curlauthenticationfile
2	Global configuration file	{curlAuthenticationFile}
3	Environment	%Json2PumlCurlAuthenticationFile%

6.2.3.3 Input list filename

The variable defines the name of the input list file (see **Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.**).

Rank	Place	Detail
1	Command line	/inputlistfile
2	Parameter file	{inputlistfile}

6.2.3.4 Summary filename

The variable defines the name of the generated summary file.

Rank	Place	Detail
1	Parameter file	{summaryFile}
2	Input list file	{summaryFile}
3	Default	“summary”

6.2.4 System Environment

6.2.4.1 PlantUML jar filename

The variable defines where the PlantUML jar file is installed in the current system environment. This path is used when calling java from the PlantUML post processor.

Rank	Place	Detail
1	Command line	/plantumljarfile
2	Global configuration file	{plantumlJarFile}
3	Environment	%PlantUmlJarFile%

6.2.4.2 Java runtime parameter

The variable is added as additional parameter to the java command line from the PlantUML post processor.

Rank	Place	Detail
1	Command line	/javaruntimeparameter
2	Global configuration file	{javaRuntimeParameter}

6.3 Global configuration file

6.3.1 Data model

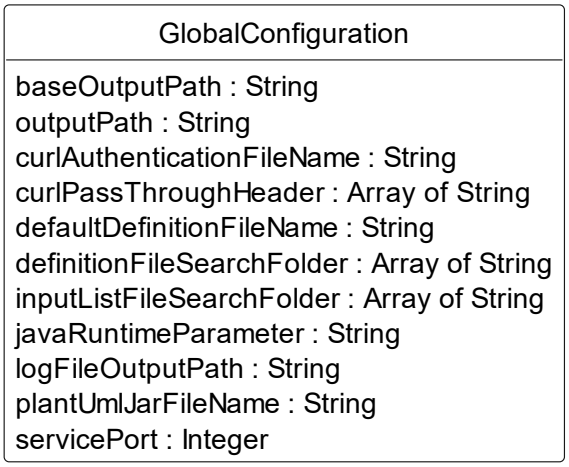


Figure 25 Data model global configuration file

6.4 Parameter file

6.4.1 Data model

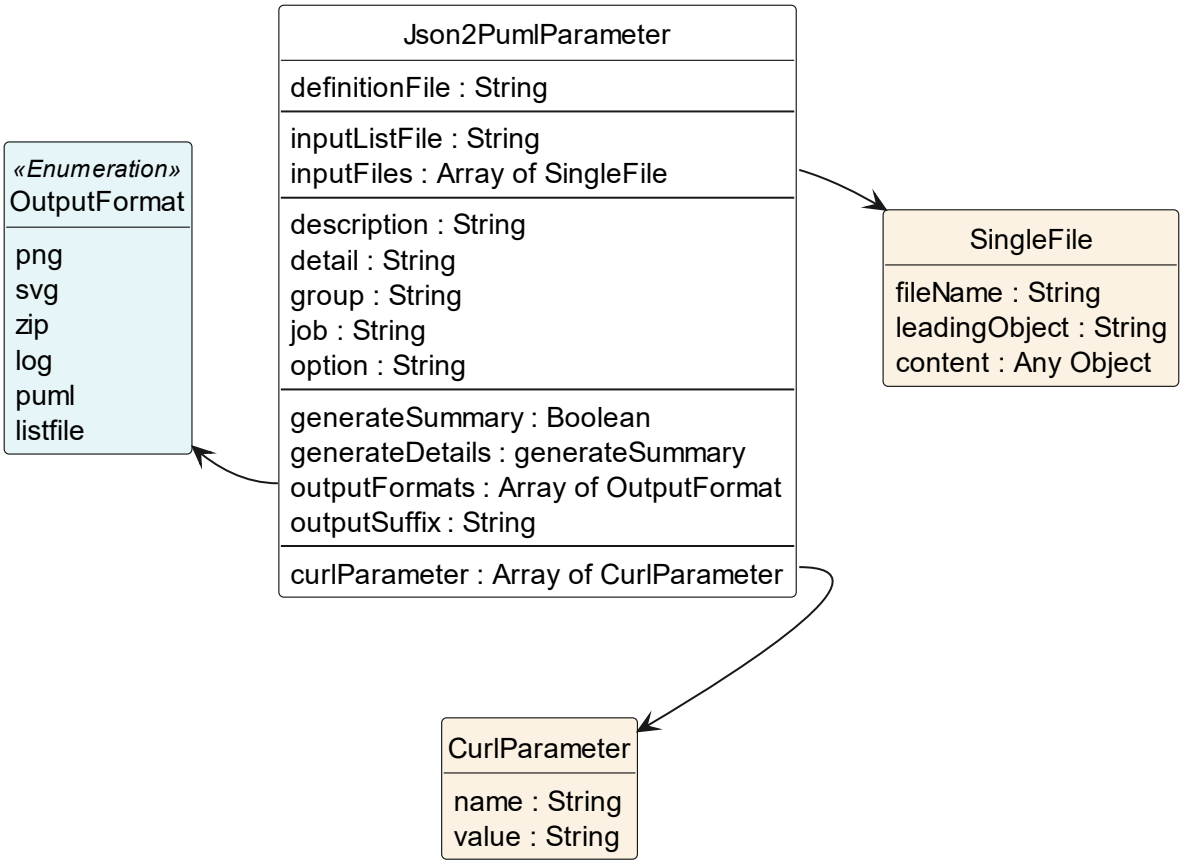


Figure 26 Data model parameter file

6.5 Input list File

This file allows to define a list of files to converted.

It is also possible to configure `curl` calls to fetch the content of the json files directly from the source before converting the contents. For further details see 0

Curl .

For every file it is possible to define the name of the leading object of the JSON structures included.

6.5.1 Data model

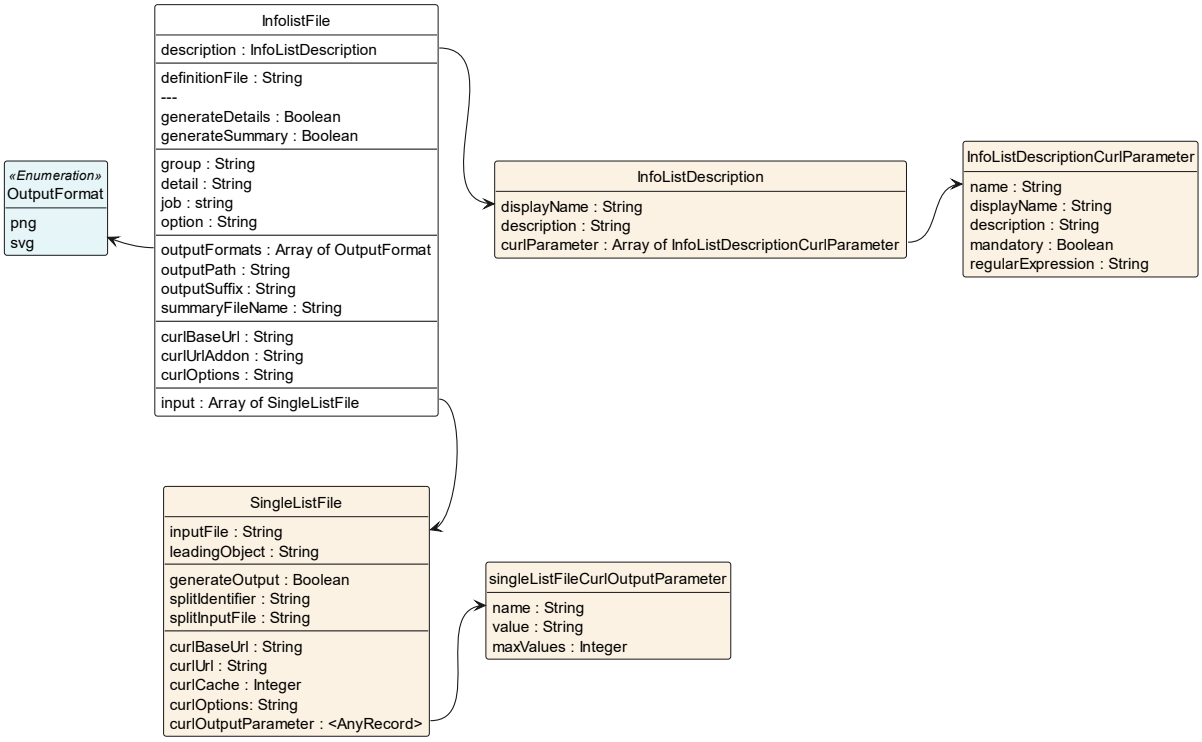


Figure 27: Data model input list file

6.5.2 InfoListFile

6.5.2.1 generateSummary

Name	Cardinality	Datatype	Default
generateSummary	0..1	Boolean	
Description			
Flag to define if a combined summary file of all input files should be generated.			

6.5.2.2 generateDetails

Name	Cardinality	Datatype	Default
generateDetails	0..1	Boolean	
Description			
Flag to define if the generator should be executed for every single input file			

6.5.2.3 outputPath

Name	Cardinality	Datatype	Default
outputPath	0..1	String	

Description

The output path could be used to separate the generated files from the original source files.
The output path could be defined absolute or relative to the source file of the generation.
To structure the directories, it is possible to include the current option, group and detail into the folder name.
The following replacements are supported:

- <option>
  Replaced by the name of the use configuration option
- <group>
  Replaced by the corresponding value from the list definition file (see 6.5.2 InfoListFile) or the corresponding command line parameter `/group` (see 4.2 Command Line Parameters)
- <detail>
  Replaced by the corresponding value from the list definition file (see 6.5.2 InfoListFile) or the corresponding command line parameter `/detail` (see 4.2 Command Line Parameters)

Example:

```
"outputPath": "output\\<group>\\<option>"
```

6.5.2.4 outputSuffix

Name	Cardinality	Datatype	Default
outputSuffix	0..1	String	

Description

The output suffix will be added to filename of any generated file.
This could be used to separate the generated files from the original source files and to separate them if multiple options or filters have been used to generate them.
To structure the files, it is possible to include the current option, group and detail into the folder name.
The following replacements are supported:

- <option>
  Replaced by the name of the use configuration option
- <group>
  Replaced by the corresponding value from the list definition file (see 6.5.2 InfoListFile) or the corresponding command line parameter `/group` (see 4.2 Command Line Parameters)
- <detail>
  Replaced by the corresponding command line parameter `/detail` (see 4.2 Command Line Parameters)

Example:

Name	Cardinality	Datatype	Default
"outputSuffix": ".<group>.<option>" Input file : "tmf666_based.json" will lead to a filename "tmf666_based.crm.full.json" based on the group "crm" and the option "full".			

6.5.2.5 outputFormats

Name	Cardinality	Datatype	Default
outputFormats	0..*	OutputFormat	
Description			
List of image formats the generator should create based on the created puml file. The value can be overwritten by the command line parameter /outputFormats. (see 4.2 Command Line Parameters) For further details see 5.5.6 PlantUML output files.			

6.5.2.6 curlBaseUrl

Name	Cardinality	Datatype	Default
curlBaseUrl	0..1	String	
Description			
Base URL which will be added before all curlUrl parameters (6.5.3.3 curlBaseUrl) of the inputfile to define the full URL. This allows to simplify the URL definition when multiple files from the same URL have to be fetched. This parameter can be overwritten by the single file curlBaseUrl parameter (6.5.3.3 curlBaseUrl) See 0 Curl .			

6.5.2.7 curlUrlAddon

Name	Cardinality	Datatype	Default
curlUrlAddon	0..1	String	
Description			
Additional URL part which will be added after all curlUrl parameters (6.5.3.3 curlBaseUrl) of the inputfile to define the full URL. This allows to simplify the URL definition when common definitions like authentications needs to be added to all URL's of the list file. See 0 Curl .			

6.5.2.8 curlOptions

Name	Cardinality	Datatype	Default
curlOptions	0..1	String	
Description			
Additional parameter which will be added to all curl commands of the input list. This could be used to add common additional parameters like authentication information to the curl command. See 0 Curl .			

6.5.2.9 Option

Name	Cardinality	Datatype	Default
Option	0..1	String	
Description			
Defines which configuration option should be used to generate the outcome. It is possible to define multiple options separated by “,” or “;”. This parameter will be ignored when the `optionfile` parameter is used. This parameter will be overwritten by the corresponding command line parameter `option`. (see 4.2 Command Line Parameters)			

6.5.2.10 Group

Name	Cardinality	Datatype	Default
Group	0..1	String	
Description			
Optional name to group the generated files by enhancing the generated file names and folders with the group (see 6.6.8.9 outputPath and 6.6.8.10 outputSuffix). This could be useful if multiple list files are used in one folder. This parameter will be overwritten by the corresponding command line parameter `group`. (see 4.2 Command Line Parameters)			

6.5.2.11 Detail

Name	Cardinality	Datatype	Default
Group	0..1	String	
Description			
Optional name to group the generated files by enhancing the generated file names and folders with the detail (see 6.6.8.9 outputPath and 6.6.8.10 outputSuffix). This could be useful if multiple list files are used in one folder.			

Name	Cardinality	Datatype	Default
This parameter will be overwritten by the corresponding command line parameter `detail`. (see 4.2 Command Line Parameters)			

6.5.2.12 summaryFileName

Name	Cardinality	Datatype	Default
summaryFileName	0..1	String	summary.json
Description			
Name of the generated summary file. This name will be enhanced with the output suffix of the current definition (see also 5.5.3 Summary file). When using the curl option, the filename can be enhanced with curl parameter values. See 0 Curl .			

6.5.2.13 Input

Name	Cardinality	Datatype	Default
Input	0..*	SingleListFile	
Description			
List of file definitions to be imported. See 6.5.3 SingleListFile			

6.5.3 SingleListFile

6.5.3.1 inputFile

Name	Cardinality	Datatype	Default
inputFile	0..1	String	
Description			
Filter to find the JSON input files. The path to the files can be relative or absolute defined. Relative paths will be expanded based on the source path of the list input file. When using the curl option, the filename is not allowed to have filters. When using the curl option, the filename can be enhanced with curl parameter values. See 0			

Name	Cardinality	Datatype	Default
Curl			

6.5.3.2 leadingObject

Name	Cardinality	Datatype	Default
leadingObject	0..1	string	
Description			
Name of the leadingObject to be used for all input files found based on the "`inputFile`" parameter. See 7.1 leadingObject			

6.5.3.3 curlBaseUrl

Name	Cardinality	Datatype	Default
curlBaseUrl	0..1	string	
Description			
Base URL which will be added before the curlUrl parameters (6.5.3.3 curlBaseUrl) of the inputfile to define the full URL. This parameter overwrites the curlBaseUrl parameter of the list filed (6.5.2.6 curlBaseUrl) See 0 Curl			

6.5.3.4 curlUrl

Name	Cardinality	Datatype	Default
curlUrl	0..1	string	
Description			
Url to fetch the json file from. See 0 Curl			

6.5.3.5 curlOptions

Name	Cardinality	Datatype	Default
curlOptions	0..1	string	
Description			
Additional parameters added to the curl command when fetching the json data.			

Name	Cardinality	Datatype	Default
See 0 Curl .			

6.5.3.6 curlCache

Name	Cardinality	Datatype	Default
curlCache	0..1	Integer	0
Description			
Number of seconds while the generated curl result file will be cached and not refreshed. See 0 Curl			

6.5.3.7 curlOutputParameter

Name	Cardinality	Datatype	Default
curlOutputParameter	0..1	<anyRecord>	0
Description			

The curlOutputParameter can be any simple json record structure (no subobjects). This parameter defines which additional curl parameter should be read from the current output file and been added to the list of curl parameter.
For each element of this record the name will be used as curl parameter name and the value will be used to fetch the curl parameter value from the current output file. For reading the value the path can be defined using a recursive `<name>.<subname>.<subsubname>` pattern.

Example output file:

```
{
  "id": 1,
  "name": "Leanne Graham",
  "email": "Sincere@april.biz",
  "location": {
    "zipcode": "92998-3874",
    "geo": {
      "lat": "-37.3159",
      "lng": "81.1496"
    }
  }
}
```

Example curlOutputParameter:

Name	Cardinality	Datatype	Default

```
{
  "name": "name",
  "geo_lat": "location.geo.lat",
  "geo_lng": "location.geo.lng"
}
```

This would lead to the following curl parameters:

Parameter	Value
${name}	Leanne Graham
${geo_lat}	-37.3159
${geo_lng}	81.1496

These curl parameter values can now be used as curl input parameter for the next curl executions of further files.

See 0

Curl

6.5.3.8 generateOutput

Name	Cardinality	Datatype	Default
generateOutput	0..1	Boolean	True
Description			
Flag to define if an output should be generated for this file or that it should be included into the summary file. This parameter could when an oAuth call is used to fetch an access token with the first curl command and then use this token as parameter for all further curl commands. In this case the result of the oAuth token generation should not be included into the generated output. See 0 Curl			

6.5.3.9 splitInputFile

Name	Cardinality	Datatype	Default
splitInputFile	0..1	Boolean	
Description			
This option can be used to split datafiles into more detailed datafiles. This option is only possible/valid when the datafile has a JSON array on the highest level. If the top-level JSON structure is not an array the parameter will be ignored. This option is only valid when generatedetails (see 0 generateDetails) is activated.			

Name	Cardinality	Datatype	Default
A common scenario for this use case is search/get results returning multiple objects of the same type. This option allows to generate detail data files and based on that generate separate images for every instance of the result set. The generated filename of the splited files will be enhanced with an identifier and sequence number (to make the filename unique). The identifier will be based on the “`splitIdentifier`” parameter (see 6.5.3.10 splitIdentifier).			

6.5.3.10 splitIdentifier

Name	Cardinality	Datatype	Default
splitIdentifier	0..1	String	
Description			
This parameter allows to have “readable” filenames for the splitted data files. The identifier must be a property name of the splitted JSON structure. When the parameter is defined and a corresponding property has been found the value of this property will be used as the identifier which will be added to the filename. Otherwise, the fixed value “`split`” will be used as an identifier. The property name can be defined using sub object names (e.g. `"splitIdentifier":"ticketType.name"`). When the property is an JSON array the different values of this array will be concatenated to one identifier. <u>Example:</u> `{ "id": "875153820701008909", "ticketType": [ { "name": "Incident" }, { "name": "Admin" } ],` The parameter `"splitIdentifier":"ticketType.name"`) would lead to an identifier “`Incident.Admin`”.			

6.5.4 Simple List File Example

```
{
  "generateSummary":"true",
  "generateDetails":"false",
  "group":"customer",
  "input": [
    {
      "inputFile": "TMF629*.json",
      "leadingObject": "customer"
    },
    {
      "inputFile": "TMF632*.json",
      "leadingObject": "individual"
    },
    {
      "inputFile": "TMF666*.json",
      "leadingObject": "account"
    },
    {
      "inputFile": "TMF670.json",
      "leadingObject": "paymentMethod"
    }
  ]
}
```

6.6 Converter definition file

The converter definition file contains the different configurations how the JSON structures should be interpreted and converted.

For every type of structures, it can be necessary to have independent configuration files.

E.g., having one definition file for some TMF based API's and a different configuration file for an SAP based API.

6.6.1 Merging of options

The configuration file allows to have multiple options configured how the system is working. For example, there could be one option which only shows the attributes of the objects and in another option also the characteristics and relations will be shown for an object.

To simplify the configuration a mechanism of having a base configuration which then is merged with the chosen configuration option.

Normally in the base configuration all things are configured which are needed to classify and structure the outcome. Also, the object specific format configuration are defined here as a default.

The target configuration is built in that way, that first the base configuration is read and this configuration is merged with the chosen specific configuration.

Merging means for simple attributes that the value is overwritten by the value of the specific option when it's defined, otherwise the default value stays.

For list properties there are two ways to merge the data.
The default way is to merge the content of the specific list to the values of base configuration. It is also possible to define that the specific list replaces the values of the base configuration.

The merge mode is the default mode. In the merge mode only the values to be merged are needed, the replace mode must be identified with "`operation`" and contains the new values in the "`list`" property.

Example Merge Mode:

```
{
  "option": "detailed",
  "definition": {
    "allowedProperties": [
      "*"
     ]
    }
  }
},
```

In this example for an option "`detailed`" the value "*" is added to the list of the base configuration.

Example Replace Mode:

```
{
  "option": "short",
  "definition": {
    "allowedProperties": {
      "operation": "replace",
      "list": [
        "id",
        "*name",
        "status"
      ]
    }
  }
},
```

In this example for an option "`short`" the list values of the base configuration are replaced by three new value "id", "*name" and "status".

The merging mechanism works similar also for the "`objectFormat`" property. (see 6.9.4 ObjectFormat)

6.6.2 Identifying an Option

To identify the option which should be used the following order is executed:

The command line parameter `/option` (see 4.2 Command Line Parameters) is used first. If this parameter is not defined the parameter "`defaultOption`" from the configuration file is used (see 6.6.4.3 defaultOption).

This value is then used to search in the "`options`" list of the configuration file (based on the "`optionName`" property of the options list (see 6.6.4.1 options).

The found specific option will then be merged with the `baseOption`, if no matching specific option is found, the `baseOption` will be used.

6.6.3 Data model

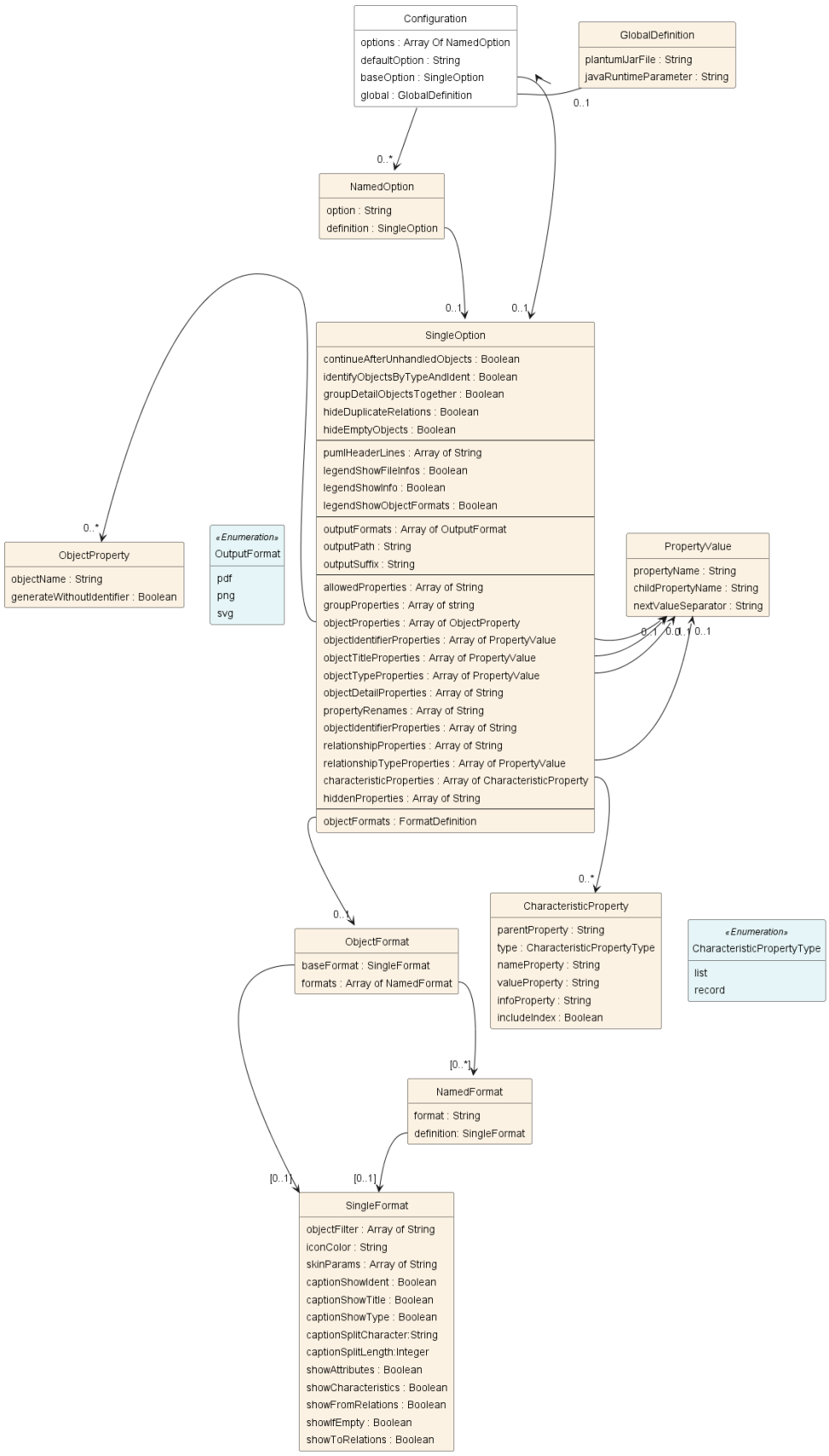


Figure 28 Main Configuration Model

6.6.4 Configuration

6.6.4.1 options

Name	Cardinality	Datatype	Default
Options	0..*	NamedOption	
Description			
This array defines the list of possible options which could be used for analysing and formatting the output. Which option is used can be configured via command line parameter `/option` (see 4.2 Command Line Parameters). If the command line parameter is not used the parameter `defaultOption` is used instead (see 6.6.4.3 defaultOption).			

6.6.4.2 baseOption

Name	Cardinality	Datatype	Default
baseOption	0..1	SingleOption	
Description			
This array defines the list of possible options which could be used for analysing and formatting the output. Which option is used can be configured via command line parameter `/option` (see 4.2 Command Line Parameters). If the command line parameter is not used the parameter `defaultOption` is used instead (see 6.6.4.3 defaultOption).			

6.6.4.3 defaultOption

Name	Cardinality	Datatype	Default
defaultOption	0..1	String	
Description			
Name of the option which should be used when the command line parameter `/option` is not defined.			

6.6.4.4 global

Name	Cardinality	Datatype	Default
Global	0..1	GlobalDefinition	
Description			
Global environment specific options which are not depending on a chosen option.			

6.6.5 GlobalDefinition

6.6.5.1 plantUmlJarFile

Name	Cardinality	Datatype	Default
plantUmlJarFile	0..1	String	
Description			
Name and path of the PlantUML jar file which will be used to generate the PlantUML output files. For further details see 5.5.6 PlantUML output files.			

6.6.5.2 javaRuntimeParameter

Name	Cardinality	Datatype	Default
javaRuntimeParameter	0..1	String	
Description			
Additional runtime parameter which is added to the java commandline when calling the PlantUML jar file to generate the output files. <u>Example:</u> `"javaRuntimeParameter": "-DPLANTUML_LIMIT_SIZE=8192"` This parameter is used to allow PlantUML to consume more memory, which allows to generate bigger PNG image files.			

6.6.6 NamedOption

Record to define an entry in the options list. Each entry is defined by a `optionName` and the corresponding `definition`.
(See 6.6.1 Merging of option)

6.6.6.1 optionName

Name	Cardinality	Datatype	Default
optionName	0..1	String	
Description			
Name of the option			

6.6.6.2 baseOption

Name	Cardinality	Datatype	Default
definition	0..1	SingleOption	
Description			

Name	Cardinality	Datatype	Default
The definition contains the option specific values which should be merged into the base configuration.			

6.6.7 PropertyValue

Record to define how the related information should be calculated.

This record could be used to define the following objects details:

- 6.6.8.14 objectTitleProperties
- 6.6.8.15 objectTypeProperties
- 6.6.8.18 objectIdentifierProperties
- 6.6.8.21 relationshipTypeProperties

When only the "`propertyName`" is defined the attribute names are not needed and only the value could be stored. This allows a simplified storage structure of the definition file.

To calculate the information for every record in the corresponding list the "`propertyName`" is searched in the current JSON object.

If a corresponding property is found the value of this property is used as the expected information. When the property "`nextValueSeparator`" is not defined for the current list record, the search is stopped. When the "`nextValueSeparator`" is defined the search will be continued. When an additional value is found the values will be concatenated with the "`nextValueSeparator`" of the previous search record.
This is also the case when the found property is an array. Then if the "`nextValueSeparator`" is defined the values of the array will be combined to calculate the information.

The "`childPropertyName`" supports that the found property is again an object record. Then the search is recursively iterating through the sub objects to find the information.

<u>Example:</u>

```
"objectTitleProperties": [
      "characteristicValue.value",
      "name",
      "fullName",
      {"propertyName": "ticketType",
          "childPropertyName": "name",
          "nextValueSeparator": "."},
      {"propertyName": "transition.sourceStatus",
          "childPropertyName": "name",
          "nextValueSeparator": "->"},
      {"propertyName": "transition.targetStatus",
          "childPropertyName": "name",
          "nextValueSeparator": ""}
],
```

Explanation:

The object title will be calculated when

1. the object is of type “`characteristicValue`” and has a simple property “`value`”.
2. the object has a simple property “`name`” or “`fullName`”
3. the object has a complex property “`ticketType`” and this object has a simple child property “`name`”. If the property is an array the values will be concatenated using “.” as a separator.
4. the object is of type “`transition`”
   The next two items are defined as a combination.
   The first part of the name is coming from the complex property “`sourceStatus`” and the child property “`name`” combined with the separator “=>” and the second part of the name coming from the complex property “`targetStatus`” and the child property “`name`” .

Samples:

No	JSON	Outcome
1a	`"characteristic": { "name": "Product", "value": "Voice Over IP Basic",`	"Voice Over IP Basic"
1b	`"product": { "name": "Prodcuct", "value": "Voice Over IP Basic",`	N/A (because of the not matching object type product)
2	`"product": { "id": "g265-tf85", "name": "VOIP", "fullName": "Voice Over IP Basic" "productSerialNumber": "N/A",`	“VOIP” “`fullName`” is ignored because the “`name`” is coming first in the definition list.

No	JSON	Outcome
3	`"ticketType": [ { "name": "Incident" }, { "name": "Admin" }, { "name": "Escalation" } ]`	“Incident.Admin.Escalation”
4	`"transition": { "sourceStatus": { "name": "Draft" }, "targetStatus": { "name": "New" }, }`	“Draft->New”

6.6.7.1 propertyName

Name	Cardinality	Datatype	Default
propertyName	1	String	
Description			
Name of the property where the information is searched. The name can be defined as `<objectType>.<propertyName>` or as `<propertyName>`. If no object type is defined the propertyName is checked against all object types. Wildcards are supported.			

6.6.7.2 childPropertyName

Name	Cardinality	Datatype	Default
childPropertyName	0..1	String	
Description			
When the defined property is a complex property the `childPropertyName` must be defined to identify the detail property which contains the data. This search is working recursively through the structure of found property. <u>Example:</u>			

Name	Cardinality	Datatype	Default

```
"escalation": {
      "id": "escalation_9913",
      "ticket": {
          "id": "ticket_4711",
          "ticketType": [
                {
                      "name": "Incident"
                },
                {
                      "name": "Admin"
                },
                {
                      "name": "Escalation"
                }
      }
}
```

To get the name of the escalation the following definition could be done.

propertyName	**childPropertyName**	**nextValueSeparator**
escalation.ticket	ticketType.name	.

The result will be “`Incident.Admin.Escalation`”.

6.6.7.3 nextValueSeparator

Name	Cardinality	Datatype	Default
nextValueSeparator	0..1	String	
Description			
When this property is filled the search will be continued and if an additional matching property is found they will be concatenated using the defined separator.			

6.6.8 SingleOption

6.6.8.1 continueAfterUnhandledObjects

Name	Cardinality	Datatype	Default
continueAfterUnhandledObjects	0..1	Boolean	False
Description			
Defines if the generator should continue the search for the children of a property when the property has been identified as an object but no identifying attribute has been found.			

6.6.8.2 identifyObjectsByTypeAndIdent

Name	Cardinality	Datatype	Default
identifyObjectsByTypeAndIdent	0..1	Boolean	False
Description			
Defines if the objects should be identified and linked based on the object type and the object ident of the object or only based on the object ident. To use the object ident only is only possible/valid if the identifiers are unique over all object types. When including the type also the type must be included in the data of an included property. (See 5.3.2 Object Identification)			

6.6.8.3 hideDuplicateRelations

Name	Cardinality	Datatype	Default
hideDuplicateRelations	0..1	Boolean	True
Description			
Defines if relations between two objects which have the same attributes ("`from object`", "`to object`", "`relationshipTypeProperty`" and "`replationshipType`") should be hidden or generated.			

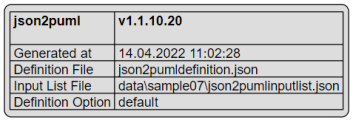
6.6.8.4 hideEmptyObjects

Name	Cardinality	Datatype	Default
hideEmptyObjects	0..1	Boolean	False
Description			
Defines if empty objects (without any attributes, characteristics and relationships) should be hidden or not.			

6.6.8.5 groupDetailObjectsTogether

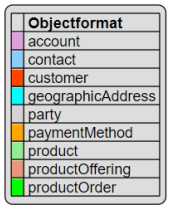
Name	Cardinality	Datatype	Default
groupDetailObjectsTogether	0..1	Boolean	False
Description			
This parameter allows to generate "together" objects in PlantUML to group the detail objects to their parent objects. PlantUML then tries to draw the grouped objects nearer together and not to distribute them. Sometimes based on the grouping PlantUML generates duplicate lines which made the diagrams hard to read.			

6.6.8.6 legendShowInfo

Name	Cardinality	Datatype	Default
legendShowInfo	0..1	Boolean	True
Description			
Defines if the common information block should be generated into the legend of the drawing. Example: 			

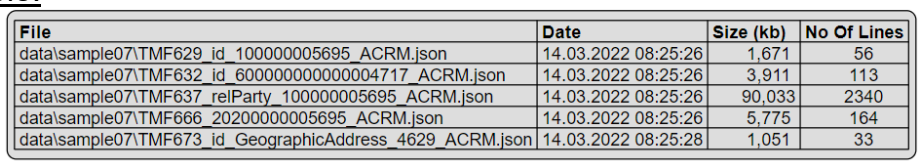
json2puml	**v1.1.10.20**
Generated at	14.04.2022 11:02:28
Definition File	json2pumldefinition.json
Input List File	data\sample07\json2pumlinputlist.json
Definition Option	default

6.6.8.7 legendShowObjectFormats

Name	Cardinality	Datatype	Default
legendShowObjectFormats	0..1	Boolean	True
Description			
Defines if the object format color list should be generated into the legend of the drawing. Example: 			

Objectformat
account
contact
customer
geographicAddress
party
paymentMethod
product
productOffering
productOrder

6.6.8.8 legendShowFileInfos

Name	Cardinality	Datatype	Default
legendShowFileInfos	0..1	Boolean	True
Description			
Defines if the input file information list should be generated into the legend of the drawing. Example: 			

File	Date	Size (kb)	No Of Lines
data\sample07\TMF629_id_100000005695_ACRM.json	14.03.2022 08:25:26	1,671	56
data\sample07\TMF632_id_60000000000004717_ACRM.json	14.03.2022 08:25:26	3,911	113
data\sample07\TMF637_relParty_100000005695_ACRM.json	14.03.2022 08:25:26	90,033	2340
data\sample07\TMF666_20200000005695_ACRM.json	14.03.2022 08:25:26	5,775	164
data\sample07\TMF673_id_GeographicAddress_4629_ACRM.json	14.03.2022 08:25:28	1,051	33

6.6.8.9 *outputPath*

Name	Cardinality	Datatype	Default
outputPath	0..1	String	
Description			

The output path could be used to separate the generated files from the original source files.
The output path could be defined absolute or relative to the source file of the generation.
To structure the directories, it is possible to include the current option, group and detail into the folder name.
The following replacements are supported:

- <option>
  Replaced by the name of the use configuration option
- <group>
  Replaced by the corresponding value from the list definition file (see 6.5.2 InfoListFile) or the corresponding command line parameter `/group` (see 4.2 Command Line Parameters)
- <detail>
  Replaced by the corresponding value from the list definition file (see 6.5.2 InfoListFile) or the corresponding command line parameter `/detail` (see 4.2 Command Line Parameters)

Example:

```
"outputPath": "output\\<group>\\<option>"
```

6.6.8.10 *outputSuffix*

Name	Cardinality	Datatype	Default
outputSuffix	0..1	String	
Description			

The output suffix will be added to filename of any generated file.
This could be used to separate the generated files from the original source files and to separate them if multiple options or filters have been used to generate them.
To structure the files, it is possible to include the current option, group and detail into the folder name.
The following replacements are supported:

- <option>
  Replaced by the name of the use configuration option
- <group>
  Replaced by the corresponding value from the list definition file (see 6.5.2 InfoListFile) or the corresponding command line parameter `/group` (see 4.2 Command Line Parameters)
- <detail>
  Replaced by the corresponding command line parameter `/detail` (see 4.2 Command Line Parameters)

Example:

Name	Cardinality	Datatype	Default
"outputSuffix": ".<group>.<option>" Input file : "tmf666_based.json" will lead to a filename "tmf666_based.crm.full.json" based on the group "crm" and the option "full".			

6.6.8.11 outputFormats

Name	Cardinality	Datatype	Default
outputFormats	0..*	OutputFormat	
Description			
List of image formats the generator should create based on the created puml file. The pre-configured value can be overwritten by the corresponding input list file parameter outputFormats (see 6.5.2.5 outputFormats) and the command line parameter /outputFormats. (see 4.2 Command Line Parameters) For further details see 5.5.6 PlantUML output files.			

6.6.8.12 attributeProperties

Name	Cardinality	Datatype	Default
attributeProperties	0..*	String	
Description			
List of properties for which the values should be handled as an attribute of the object. Only when the name of the properties matches a value in the list (wildcards are supported) the value will be added. To include all properties, add the value "*" to the list.			

6.6.8.13 objectProperties

Name	Cardinality	Datatype	Default
objectProperties	0..*	ObjectProperty	
Description			
List of properties which should be handled as an object. ObjectProperty is a record which contains two attributes: "objectName" and "generateWithoutIdentifier". When "generateWithoutIdentifier" is not defined only a value can be used to define the objectname without the propertyname. <u>Example:</u>			

Name	Cardinality	Datatype	Default

```
"objectProperties": [
      "account",
      "party",
      {"objectName": "productTerm",
      "generateWithoutIdentifier": "true"},
      {"objectName": "contactMedium",
      "generateWithoutIdentifier": "true"}
],
```

6.6.8.14 *objectTitleProperties*

Name	Cardinality	Datatype	Default
objectTitleProperties	0..*	PropertyValue	
Description			
List of properties which should be handled as the title of an object. The title is an optional attribute which could be shown in the header of the objects and as part of the from / to relationship description.			

6.6.8.15 *objectTypeProperties*

Name	Cardinality	Datatype	Default
objectTypeProperties	0..*	PropertyValue	
Description			
List of properties which should be handled as the type of an object. The value of the property will be used as object type of the object. Using the "`objectTypeRenames`" parameter (see 6.6.8.17 objectTypeRenames) the object type could be renamed. (See 5.3.2 Object Identification)			

6.6.8.16 *objectDetailProperties*

Name	Cardinality	Datatype	Default
objectDetailProperties	0..*	String	
Description			
List of properties which will be handled as detail object. (see 5.3.4 Detail Objects / Characteristic properties)			

6.6.8.17 *objectTypeRenames*

Name	Cardinality	Datatype	Default
objectTypeRenames	0..*	String	

Description

List of name replacements to be used for object type. (See 5.3.2 Object Identification)
The format of the rename property is:
“`<oldname>=<newname>`”.

Example:

```
"objectTypeRenames": [
      "compositeProduct=product",
      "defaultPaymentMethod=paymentMethod",
      "place=geographicAddress",
      "email=contactMedium",
]
```

6.6.8.18 *objectIdentifierProperties*

Name	Cardinality	Datatype	Default
objectIdentifierProperties	0..*	PropertyValue	

Description

List of properties which should be handled as the identifier of an object.
The value of the property will be used as identifier of the object.
(See 5.3.2 Object Identification)

6.6.8.19 *groupProperties*

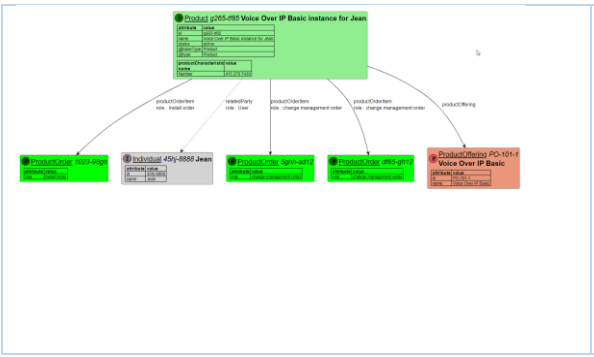
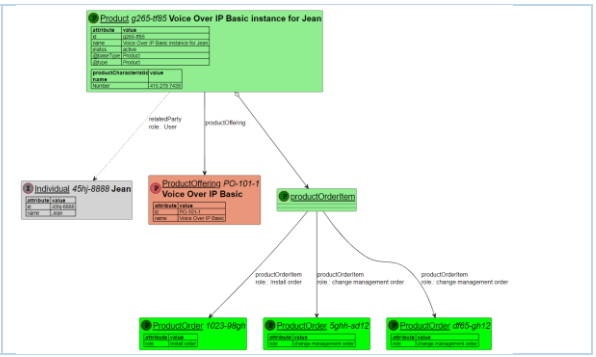
Name	Cardinality	Datatype	Default
groupProperties	0..*	String	

Description

A group property allows to structure the generated output in that way that an additional blank object is created which collects the relations of the related detail objects of the same name.

Example:

Without GroupProperty	**With GroupProperty**

Name	Cardinality	Datatype	Default
 			

6.6.8.20 *relationshipProperties*

Name	Cardinality	Datatype	Default
relationshipProperties	0..*	String	
Description			
List of properties which should be handled to identify typed relations to another object. A relationship property can, but most not be defined as a separate object. Based on the parameter "`relationshipTypeProperties`" a type of the relationship could be identified. (See 5.3.3 Object Relationships)			

6.6.8.21 *relationshipTypeProperties*

Name	Cardinality	Datatype	Default
relationshipTypeProperties	0..*	String	
Description			
List of properties which should be handled as the type of a relationship. (See 5.3.3 Object Relationships)			

6.6.8.22 *relationshipTypeArrowFormats*

Name	Cardinality	Datatype	Default
relationshipTypeArrowFormats	0..*	PropertyValue	
Description			
Mapping definition how an arrow should be formatted based on the property name which has defined the relationship and the optional type of the relationship. The format has to be defined using the following pattern: `<relationship pattern>[.<relationship type>]=<PlantUML arrow format>` <u>Example:</u>			

Name	Cardinality	Datatype	Default

```
"relationshipTypeArrowFormats": [
      "relatedparty=[dotted]",
      "relatedparty.customer=[#blue,dotted]",
      "relatedparty.individual=[#lightgrey,dotted]",
      "relatedparty.organisation=[#grey,dotted]",
      "engagedparty=[dotted]",
      "engagedparty.individual=[#lightgrey,dotted]",
      "engagedparty.organisation=[#grey,dotted]"
]
```

(See 5.3.3 Object Relationships)

6.6.8.23 hiddenProperties

Name	Cardinality	Datatype	Default
hiddenProperties	0..*	String	
Description			

List of properties which will stop the recursive handling of the converter.

6.7 (see 5.1 Main components

Json2puml consists of 3 main components:

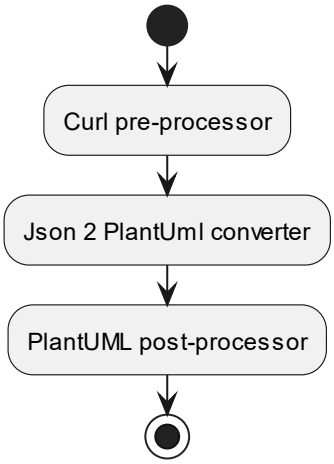


Figure 19 json2puml main components

6.7.1 Curl pre-processor

The curl pre-processor is an optional step. It allows to collect json files by calling a configures set of API calls. The “`curl`” command will be used on command line to execute the configured API calls.
When the curl pre-processor is not used the files to be converted must be already existing.
These API calls can be configured via parameters. The parameters can also be fetched from one curl result and be used for the next api calls. This allows to have dynamic chains of api calls based on one input parameter.

Name	Cardinality	Datatype	Default

Example:
Fetching data from a shop system based on an input parameter “OrderId”

	API Call	Input Parameter	Output Parameter
1	Get /order	${orderId}:2022-1234	${custNumber}:4711 ${addressId}:8819
2	Get /customer	${custNumber}:4711	${addressId}:1219
3	Get /orderItems	${orderId}:2022-1234	${product}:a14 ${product}:a177 ${product}:b88
4	Get /address	${addressId}:8819 ${addressId}:1219	
5	Get /product	${product}:a14 ${product}:a177 ${product}:b88	

The result would a set of json files reflecting 1 order object, 1 customer object, 3 order item object, 2 address objects and 5 product objects, which then can be combined into one overall result.
The curl pre-processor is configured in the input list file (see 6.5 Input list File), the curl parameter file(see 6.10 Curl parameter file) and the curl authentication file (see 6.11 Curl authentication file)

6.7.2 Json 2 PlantUML converter

The Json 2 PlantUML converter a set of json files into PlantUML files. These files can then be used by the PlantUML post-processor to generate the graphical representations of the json files.
The converter works on API specific definitions, which will be configured in the converter definition file (see 6.6 Converter definition file).

6.7.3 PlantUML post-processor

The PlantUML post-processor used the generated PlantUML script files and the PlantUML jar file to generate the images based on the script files.
The PlantUML jar file and a valid java runtime environment must be locally available on the operating system.
The path to the needed PlantUML jar file will be defined on the variable `plantumJarfile`. (see 6.2.4.1 PlantUML jar filename).
In addition, the `javaRuntimeParameter` variable can used to configure additional parameters to be added to the java command line (see 6.2.4.2 Java runtime parameter)
The following command line will be used :

```
java <javaRuntimeParameter> -jar <plantumJarFile>
      <plantumlScriptFile> <format>
```

6.8 Curl pre-processor

In addition to using static input files, it is also possible to get the content of the files using a curl command.
The curl support is based on the definitions of the input list file.
Here it is possible to define for every single file definition in the list file that this file should be fetched dynamically.
After all files are handled by the pre-processor the converter will start.
The curl pre-processer is based on the curl command, which will be called via command line and must be available on the operating system.

6.8.1 Curl parametrisation

To reuse a list definition file for visualisation of different dataset it is possible to use curl parameters to fetch dynamically different data.
For this the system allows to define for every converter run a set list of parameter value pairs which will be used to update the curl command before execution by replacing every occurrence of a known curl parameter name with the corresponding curl parameter value.

6.8.1.1 Defining the use of curl parameters

For this the curl parameter must be defined in the configured configuration parameters in the following pattern "`${<name>}`". Before executing a curl command all existing strings following this pattern will be replaced by the value of the corresponding curl parameter.
The search of the name is not case sensitive.
<u>Example:</u>

- Parameter name: id
- Parameter value: 1
- Configured `curlUrl`: "`/users/${id}`"

Before executing the `curlUrl` will be replaced to "`/users/1`".

6.8.1.2 Curl filename parametrisation

The curl parametrisation can also be used to generate the output files based on the curl parameter values.
The curl parametrisation of filenames can be used for the summary file name (6.5.2.12 summaryFileName) and for the input file name of the SingleListFile (6.5.3.1 inputFile).
As a restriction: For a single file the `curlOutParameter` of this file can only be used for the filename of the files which are defined after the current file.

6.8.2 Curl command

For every file defined in the input list file the curl command will be executed when a Url is defined.
The curl command is based build following this pattern:

```
curl <curlOptions> -o <outputFile>
      --url <curlBaseUrl>/<singleCurlUrl><curlUrlAddon>
```

Name	Cardinality	Datatype	Default

The different parts of the command are build based on the following logic

- `<curlOptions>`:
  Concatenation of `inputListFile.curlOptions` and `singleInputFile.curlOptions`
- `<outputFile>`:
  Combination of `singleInputFile.inputFile` and `singleInputFile.curlFileSuffix.`
- `<curlBaseUrl>`:
  If defined the `singleInputFile.curlBaseUrl` will be used, otherwise the `inputListFile.curlBaseUrl` will be used.
- `<singleCurlUrl>`:
  `singleInputFile.curlUrl`
- `<curlUrlAddon>`:
  `inputListFile.curlUrlAddon`

To activate the curl execution the `curlUrl` parameter of the single list file definition and the `curlBaseUrl` parameter must be defined. Otherwise, the existing static file will be used.

6.8.3 How to use it

To understand how the curl pre-processor is working you can have a look into the `samples` directory. Most of the examples are based on the pre-processor to fetch data for dedicated scenarios via api.
As an example, we can have a look into the jsonplaceholder example. The jsonplaceholder api is a REST training API which could be used by developers to learn how to work with such api's.
The api has some random data and is based on the following data model:

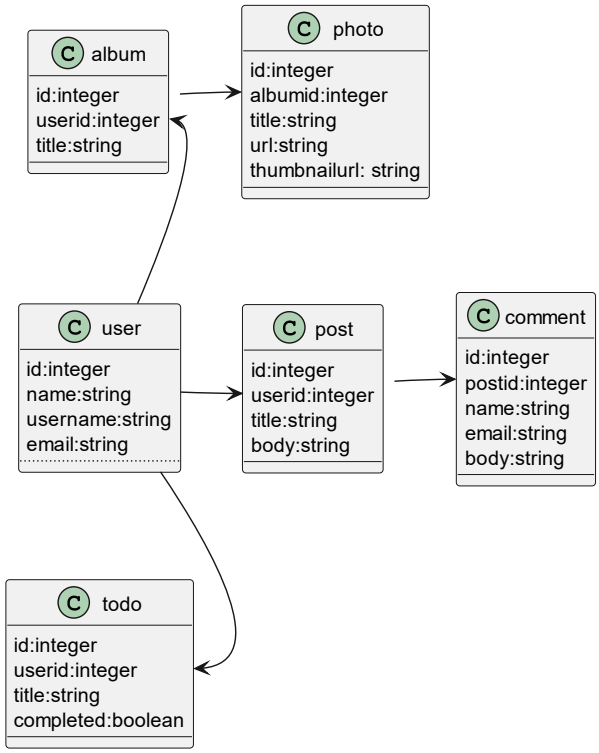


Figure 20 jsonplaceholder data model

Name	Cardinality	Datatype	Default

The input list file `placeholder_inputlist_curl.json` collects and visualised data based on one user id. The id of the user can be defined via command line parameter.

```
json2puml /parameterfile:placeholder_parameter_curl.json
      /curlparameter:userid=1
json2puml /parameterfile:placeholder_parameter_curl.json
      /curlparameter:userid=3
```

In the following steps the `userid = 1` will be used as a baseline for the shown example data.

6.8.3.1 Global definition

```
  "curlBaseUrl": "https://jsonplaceholder.typicode.com/",
  "curlUrlAddon": null,
  "curlOptions": "-f",
  "summaryFile": "placeholder_summary_${name}",
```

The global definition of the input list file only defines

- `"curlBaseUrl": "https://jsonplaceholder.typicode.com/"`
  This url will be the basis for all url calls
- `"summaryFile": "placeholder_summary_${name}"`
  The generated summary file will contain the name of the defined user based on the curl parameter `${name}`. This parameter will be filled after the data for the defined user has been fetched.

6.8.3.2 Inputfile "users.json"

The first file to be fetched will contain the data of the defined user.

6.8.3.2.1 Configuration

```
{
      "inputFile": "users.json",
      "curlFileSuffix": "_${userid}",
      "curlUrl": "/users/${userid}",
      "curlOptions": null,
      "curlOutputParameter": [
        {
          "${name}": "name"
        }
      ]
    },
```

6.8.3.2.2 Filename

The generated filename will be `users_1.json` (based on the parameters `inputFile` and `curlFileSuffix`) .

6.8.3.2.3 Command

The following command will be executed:

```
curl -f --url https://jsonplaceholder.typicode.com/users/1
      --output users_1.json
```

6.8.3.2.4 Json Result (reduced)

```
{
      "id" : 1,
      "name" : "Leanne Graham",
      "username" : "Bret",
      "email" : "Sincere@april.biz"
}
```

6.8.3.2.5 Output curl parameter

The name of the user with the id 1 will be fetched from the attribute "`name`".
As a result, the following curl parameters values are valid after the call:

Variable	Value
${userid}	1
${name}	Leanne Graham

This `${name}` parameter will then be used to generate the name of the summary file: "`placeholder_summary Leanne Graham.json`"

6.8.3.3 Inputfile "albums.json"

This file will contain the albums related to the defined user.

6.8.3.3.1 Configuration

```
{
      "inputFile": "albums.json",
      "curlFileSuffix": "_${userid}",
      "curlUrl": "albums?userId=${userid}",
      "curlOutputParameter": [
        {
          "name": "${albumid}",
          "value": "id",
          "maxValues": "2"
        }
      ]
    },
```

6.8.3.3.2 Filename

The generated filename will be `albums_1.json` (based on the parameters `inputFile` and `curlFileSuffix`).

6.8.3.3.3 Command

The following command will be executed:

```
curl -f --url https://jsonplaceholder.typicode.com/albums?userId=1
      --output albums_1.json
```

6.8.3.3.4 Json Result (reduced)

```
[
      {
          "userId" : 1,
          "id" : 1,
          "title" : "quidem molestiae enim"
      },
      {
          "userId" : 1,
          "id" : 2,
          "title" : "sunt qui excepturi placeat culpa"
      },
      {
          "userId" : 1,
          "id" : 3,
          "title" : "omnis laborum odio"
      },
      {
          "userId" : 1,
          "id" : 4,
          "title" : "non esse culpa molestiae omnis sed optio"
      },
      {
          "userId" : 1,
          "id" : 5,
          "title" : "eaque aut omnis a"
      },
      {
          "userId" : 1,
          "id" : 6,
          "title" : "natus impedit quibusdam illo est"
      },
      {
          "userId" : 1,
          "id" : 7,
          "title" : "quibusdam autem aliquid et et quia"
      },
      {
          "userId" : 1,
          "id" : 8,
          "title" : "qui fuga est a eum"
      },
      {
          "userId" : 1,
          "id" : 9,
          "title" : "saepe unde necessitatibus rem"
      },
      {
          "userId" : 1,
          "id" : 10,
          "title" : "distinctio laborum qui"
      }
]
```

6.8.3.3.5 Output curl parameter

The list of album id's is fetched from the attribute "`id`". To reduce the number of fetched parameters (and based on that make the generated image better readable) the optional parameter `"maxValues": "2"` is used to limit the number of fetched variables to 2,
The albums of the user with the id 1 will be fetched from the attribute "`id`".
As a result, the following curl parameters values are valid after the call:

Variable	Value
${userid}	1
${name}	Leanne Graham
${albumid}	1
${albumid}	2

6.8.3.4 Inputfile "photos.json"

With this file the photos which are related to the albums fetched in the second steps will be collected.

6.8.3.4.1 Configuration

```
{
      "inputFile": "photos.json",
      "curlFileSuffix": "_${userid}_${albumid}",
      "curlUrl": "photos?albumId=${albumid}",
      "curlOutputParameter": []
    },
```

6.8.3.4.2 Filename

The `curlFileSuffix` and the `curlUrl` parameter are containing the reference to two curl parameters: `${userid}` and `${albumid}`. When a curl parameter is part of these attributes then the curl command will be execute on a permutation of all values of the used curl parameters.
The generated filenames will be `photos_1_1.json` and `photos_1_2.json` (based on the parameters `inputFile` and `curlFileSuffix`).

6.8.3.4.3 Command

In this scenario this will lead to the two following commands:

```
curl -f --url
      https://jsonplaceholder.typicode.com/photos?albumId=1
      --output photos_1_1.json
curl -f --url
      https://jsonplaceholder.typicode.com/photos?albumId=2
      --output photos_1_2.json
```

6.8.3.4.4 Json Result (reduced)

```
[
  {
    "albumId": 1,
```

```
    "id": 1,
    "title": "accusamus beatae ad facilis cum similique qui sunt",
    "url": "https://via.placeholder.com/600/92c952",
    "thumbnailUrl": "https://via.placeholder.com/150/92c952"
  },
  {
    "albumId": 1,
    "id": 2,
    "title": "reprehenderit est deserunt velit ipsam",
    "url": "https://via.placeholder.com/600/771796",
    "thumbnailUrl": "https://via.placeholder.com/150/771796"
]
```

```
[
  {
    "albumId": 2,
    "id": 51,
    "title": "non sunt voluptatem placeat consequuntur rem
      incidunt",
    "url": "https://via.placeholder.com/600/8e973b",
    "thumbnailUrl": "https://via.placeholder.com/150/8e973b"
  },
  {
    "albumId": 2,
    "id": 52,
    "title": "eveniet pariatur quia nobis reiciendis laboriosam ea",
    "url": "https://via.placeholder.com/600/121fa4",
    "thumbnailUrl": "https://via.placeholder.com/150/121fa4"
  }
]
```

6.8.3.4.5 Output curl parameter

For this file no curl output parameter are defined.

6.8.3.5 Further files

- The `posts.json` will be fetched based on the `${userid}` and deliver a list of `${postid}`
- The `comments.json` will be fetched based on the `${postid}`
- The `todos.json` will be fetched based on the `${userid}`.

6.8.4 API Security / OAuth

6.8.4.1 Curl Authorisation Parameter

When using API's it is quite often necessary to define user credentials for the API calls.
In principle these credentials can be configured as part of the input list file or as normal curl parameter. From a security point this is not a good solution.

Name	Cardinality	Datatype	Default

- The input list file is designed to be handled via source code repository, so that multiple users can work with these definitions.
  And in a source code repository it is the best practice to not have any user credentials in the files.
- The generated curl commands are logged in execution log files and in the summary file list. This log files should also not contain any user credentials.

To solve this problem json2puml support a second type of curl parameter, the curl authentication parameter.
Curl authentication parameters are not written into the log files or into the summary file list. Curl authentication are working with

6.8.5 How to define curl parameter

Curl parameters can be defined based on the following ways:

- Curl command line parameter
- Curl parameter file
- Curl output parameter
- Curl authorisation file

6.8.5.1 Curl parameter via command line

6.8.5.1.1 Normal curl parameter

The allows to define curl parameters at the command line. A command line parameter will overwrite the value of a similar parameter from the curl parameter file.
The curl command line parameters can be defined via the `/curlparameter` command line parameter.
It is possible to define multiple command line curl parameter by repeating the `/curlparameter` call.
A command line curl parameter must be defined using the `<name>=<value>` pattern.

6.8.5.1.2 Curl authentication parameter

The curl authentication parameter can be defined in a similar way using the command line parameter `/curlauthenticationparameter`.

6.8.5.2 Curl parameter file

The curl parameter file can contain one single json record. Every element of this record will be interpreted as a name value pair for the curl parameter list.
The curl parameter file can be defined via the `curlparameterfile` command line parameter.
See 6.10 Curl parameter file

6.8.5.3 Curl output parameter

The curl output parameter definition allows to use a dedicated attribute of a curl result file as a new curl parameter for further curl calls.

Name	Cardinality	Datatype	Default

As an example: As command line parameter the product name is defined as curl parameter. This product name is used as a filter for the first curl call. From the returned file the primary key of the product is fetched as a new curl parameter "`productid`". This `productId` is then used in the next call to fetch further details of the product from a different API.
A curl output parameter will overwrite the curl parameters with the same name coming from the curl parameter file or the curl command line parameter.
See 6.5.3.7 curlOutputParameter

6.8.5.4 Curl authentication file

The curl authorisation file allows to separate the user specific authentication parameter (like personalized tokens, clientId and clientSecret) from the list definition.
This allows to have for a team a common list definition to fetch the details of the api independent of the developer specific authentication parameters.
The corresponding authentication parameter will be fetched based on `baseUrl` of the current curl call and handled like the other curl parameter before executing the curl call.
The curl parameter file can be defined via the `curlauthenticationfile` command line parameter and the `Json2PumlCurlAuthenticationFile` environment variable.
Authentication parameters will not be written to any logfile and not to the standard out.
See 6.11 Curl authentication file

6.9 Converter

Basic Flow)

6.9.1.1 pumlHeaderLines

Name	Cardinality	Datatype	Default
pumlHeaderLines	0..*	String	
Description			

The "`pumlHeaderlines`" will be written at the beginning of the generated PlantUML file.
This allows to have generic formatting definitions.
<u>Example:</u>

```
"pumlHeaderLines": [
      "hide stereotype",
      "skinparam HeaderFontSize 18"
],
```

6.9.1.2 characteristicProperties

Name	Cardinality	Datatype	Default
characteristicProperties	0..*	CharacteristicProperty	
Description			
List of properties which will be handled as characteristics. (see 5.3.4 Detail Objects / Characteristic properties)			

6.9.1.3 objectFormats

Name	Cardinality	Datatype	Default
objectFormats	0..1	ObjectFormat	
Description			
Format definition for the generated output.			

6.9.2 ObjectProperty

6.9.2.1 objectName

Name	Cardinality	Datatype	Default
objectName	0..1	String	
Description			
List of properties which should be handled as an object. The name of the property will be used as object type of the object. Using the "`objectTypeRenames`" parameter (see **Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.**) the object type could be renamed. (See 5.3.2 Object Identification)			

6.9.2.2 generateWithoutIdentifier

Name	Cardinality	Datatype	Default
generateWithoutIdentifier	0..1	String	false
Description			
Flag to define if the objects of this type should be generated without having an identifier. In this the identifier will automatically generated based on the identifier of the parent object (enhanced with a unique number) (See 5.3.2 Object Identification)			

6.9.3 CharacteristicProperty

The characteristic property defines how a property which is identified as a characteristic should be handled. (see 5.3.4.2 Characteristics properties)

6.9.3.1 parentProperty

Name	Cardinality	Datatype	Default
parentProperty	1	String	
Description			
Name of the property which should be handled as a characteristic			

6.9.3.2 type

Name	Cardinality	Datatype	Default
Type	1	CharacteristicPropertyType	
Description			
Type of the characteristic Possible values: - list - record			

6.9.3.3 nameProperty

Name	Cardinality	Datatype	Default
nameProperty	0..1	String	
Description			
Name of the detailed characteristic which should be transferred in the name column of the characteristic list. This parameter is only needed for the characteristic type “`list`”.			

6.9.3.4 valueProperty

Name	Cardinality	Datatype	Default
valueProperty	0..1	String	
Description			
Name of the detailed characteristic which should be transferred in the value column of the characteristic list. This parameter is only needed for the characteristic type “`list`”.			

6.9.3.5 infoProperty

Name	Cardinality	Datatype	Default
nameProperty	0..1	String	
Description			
Name of the detailed characteristic which should be transferred in the additional info column of the characteristic list. This parameter is only needed for the characteristic type “`list`”.			

6.9.3.6 includeIndex

Name	Cardinality	Datatype	Default
includeIndex	0..1	Boolean	false
Description			
Flag to define if for each characteristic record the position/index of the record should be added to the generated list. This property is only needed for the characteristic type “`list`”. <u>Example:</u> idx: 0 ticketType name: Incident idx: 1 ticketType name: Admin idx: 2 ticketType name: Incident			

6.9.4 ObjectFormat

The objectFormat defines how the generated objects should be formatted.

To identify the object specific format first the `baseFormat` will be used. Then the `formats` list will be search based on the type of the object. If a named format is found this format will be merged with the base format (only the values which are defined in the named format will be overwritten).

(see 6.6.1 Merging of options)

6.9.4.1 baseFormat

Name	Cardinality	Datatype	Default
baseFormat	0..1	SingleFormat	
Description			
Default format definition which can be merged with an object specific format definition.			

6.9.4.2 formats

Name	Cardinality	Datatype	Default
formats	0..*	NamedFormat	
Description			
List of all named format definitions.			

6.9.5 NamedFormat

6.9.5.1 formatName

Name	Cardinality	Datatype	Default
formatName	1	6.9.6string	
Description			
Name of the format definition. This name will be used as PlantUML creole to identify the format.			

6.9.5.2 definition

Name	Cardinality	Datatype	Default
Definition	0..1	SingleFormat	
Description			
Named format definition which will be merged with the base format definition to define the format for an object.			

6.9.6 SingleFormat

Example:

```
"baseFormat": {
      "objectFilter":[],
      "iconColor": "blue",
      "skinParams":["BackgroundColor=STRATEGY"],
      "captionShowIdent": "true",
      "captionShowTitle": "true",
      "captionShowType": "true",
      "captionSplitCharacter": "_",
      "captionSplitLength": "20",
      "showAttributes": "true",
      "showCharacteristics": "true",
      "showFromRelations": "false",
      "showIfEmpty": "false",
      "showToRelations": "false"
},
```

6.9.6.1 objectFilter

Name	Cardinality	Datatype	Default
objectFilter	0..*	string	
Description			
List of object names for which the format should be used. This allows to have multiple objects formatted with one format definition. The property will be ignored in the `baseFormat` definition.			

6.9.6.2 iconColor

Name	Cardinality	Datatype	Default
iconColor	0..1	string	
Description			
Name of the color which will be used to draw the icon in the object header. You can use either PlantUML standard color name or RGB code.			

6.9.6.3 skinParams

Name	Cardinality	Datatype	Default
skinParams	0..*	string	
Description			
List of skinparams which will be added to the beginning of the PlantUML file to define the format.			

Name	Cardinality	Datatype	Default

The format of the definition is:

```
<paramname>=<paramvalue>
```

The result will be:

```
<paramname> <formatname> <paramvalue>
```

Example:

```
"formatName": "troubleTicket",
"definition": {
      "objectFilter":[
          "troubleTicket",
          "workorder"
      ],
      "iconColor": "Cornflowerblue",
      "skinParams":["BackgroundColor=deepskyBlue"]
}
```

This would lead to

```
skinparam class {
BackgroundColor<<troubleticket>> deepskyBlue
}
```

6.9.6.4 captionShowIdent

Name	Cardinality	Datatype	Default
captionShowIdent	0..1	Boolean	True
Description			
Flag to define if the object identifier should be shown in the header of an object and in the object column of the from / to relations. The ident will be shown in *italic*.			

6.9.6.5 captionShowTitle

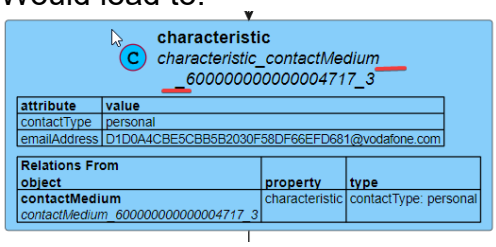
Name	Cardinality	Datatype	Default
captionShowTitle	0..1	Boolean	True
Description			
Flag to define if the object title should be shown in the header of an object and in the object column of the from / to relations. The ident will be shown in **bold**.			

6.9.6.6 captionShowType

Name	Cardinality	Datatype	Default
captionShowType	0..1	Boolean	True
Description			
Flag to define if the object type should be shown in the header of an object and in the object column of the from / to relations. The ident will be shown underlined.			

6.9.6.7 captionSplitCharacter

Name	Cardinality	Datatype	Default
captionSplitCharacter	0..1	String	
Description			

This parameter can be used to split object identifiers in multiple lines when they are too long. This reduces the size of the generated object boxes and improves the readability of the diagrams.
The identifier will be splitted in multiple lines whenever a "`captionSplitCharacter`" is found at a position greater then the "`captionSplitLength`".
Example:
`"captionSplitCharacter": "_",`
`"captionSplitLength": "20",`
 	
Would lead to:
 
Figure 29 Example of an object identifier split			

6.9.6.8 captionSplitLength

Name	Cardinality	Datatype	Default
captionSplitLength	0..1	Integer	0
Description			
Parameter to define if and when the generated caption (combination of type, ident and title) of an object should be split into multiple lines. When the parameter is defined the split will be executed when a part of the caption is getting longer then the defined value.			

6.9.6.9 showAttributes

Name	Cardinality	Datatype	Default
showAttributes	0..1	Boolean	True
Description			
Flag to define if the attribute list should be generated for the object. (see 5.3.6 Generated Outcome)			

6.9.6.10 showCharacteristics

Name	Cardinality	Datatype	Default
showCharacteristics	0..1	Boolean	True
Description			
Flag to define if the characteristics lists should be generated for the object. (see 5.3.6 Generated Outcome)			

6.9.6.11 showFromRelations

Name	Cardinality	Datatype	Default
showFromRelations	0..1	Boolean	True
Description			
Flag to define if the from relation list should be generated inside of the object. (see 5.3.6 Generated Outcome)			

6.9.6.12 showIfEmpty

Name	Cardinality	Datatype	Default
showIfEmpty	0..1	Boolean	False
Description			
Flag to define if the object should be listed if no attributes, characteristics or relations are found. This could happen if the object is only identified by it's identifier as a single sub object of another object.			

6.9.6.13 showToRelations

Name	Cardinality	Datatype	Default
showAttributes	0..1	Boolean	True
Description			
Flag to define if the to relation list should be generated inside of the object. (see 5.3.6 Generated Outcome)			

6.10 Curl parameter file

6.11 Curl authentication file

7 Global Definitions

7.1 leadingObject

The converter is based on names of properties in a JSON file. These names are mapped to objects and other attributes based on the configuration. Only for known properties/objects a class structure in the PlantUML output will be generated.

Most of the JSON data structures are without a leading property name, because this information can be determined from the API call executed.

As the converter did not have this context the name must be defined as commandline parameter or as part of the listfile definition.

Example

The JSON file has been generated based on a TMF632 `GET /individual` API call, then the objectname should be defined as `/leadingobject=individual`.

For a TMF 629 based API call `GET /customer` it should be `/leadingobject=customer`.

8 Appendix

8.1 Images

Figure 1: JSON sample 6
Figure 2: JSON Editor..... 7
Figure 3 JSON Example PlantUML 8
Figure 4 Space X – get /rockets – compact option 9
Figure 5 Space X – get /rockets – default option 10
Figure 6 Space X – get /rockets – full option 11
Figure 7 Space X – get all API – compact option 12
Figure 8 Space X – get all API – default option 13
Figure 9 Space X – get all API – launch-crew-rocket option 14
Figure 10 swapi.dev - all data - 6 Films, 82 Persons, 60 Planets, 37 Species, 36 Star ships, 39 Vehicles..... 15
Figure 11 swapi.dev - /titlefilter = "Han Solo"..... 16
Figure 12 swapi.dev - /titlefilter = "Han Solo"..... 17
Figure 13 Setup - Welcome Screen..... 18
Figure 14 Setup - Definition of installation folder 18
Figure 15 Setup - Definition of installation options..... 19
Figure 16 Setup - Summary before installation..... 20
Figure 17 Setup - Status screen while installation 20
Figure 18 Setup - Completion screen 21
Figure 19 json2puml main components 32
Figure 20 jsonplaceholder data model..... 36
Figure 21: Basic flow how the converter is working 45
Figure 22 Object relationship example 47
Figure 23 Visualisation of a detailed object 48
Figure 24 Generated Output Example..... 49
Figure 25 Data model global configuration file..... 59
Figure 26 Data model parameter file 59
Figure 27: Data model input list file 61
Figure 28 Main Configuration Model..... 73
Figure 29 Example of an object identifier split 104