

r



# json2puml

Based on version v2.2.2

## 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.....	6
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.....	15
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	Command line .....	26

4.1.1	Help Screen.....	26
4.1.2	Command Line Parameters .....	27
4.1.3	QuickStart.....	31
4.2	Service Application .....	33
4.2.1	How to prepare.....	33
4.2.2	How to run.....	33
4.2.3	How to use it via curl .....	34
4.2.4	Service operations .....	35
4.3	Docker.....	43
4.3.1	Docker image overview.....	43
4.3.2	How to build the images .....	44
4.3.3	How to use the docker images .....	45
5	How the system is working.....	46
5.1	Main components .....	46
5.1.1	Curl pre-processor .....	46
5.1.2	Json 2 PlantUML converter .....	47
5.1.3	PlantUML post-processor.....	47
5.2	Curl pre-processor.....	48
5.2.1	Curl parametrisation.....	48
5.2.2	Curl command .....	49
5.2.3	How to use it .....	49
5.2.4	API Security / OAuth .....	56
5.2.5	OpenTelemetry Trace/Span-Id .....	60
5.2.6	How to define curl parameter .....	61
5.3	Converter .....	62
5.3.1	Basic Flow .....	62
5.3.2	Object Identification .....	65
5.3.3	Object Relationships .....	65
5.3.4	Detail Objects / Characteristic properties .....	66
5.3.5	Filtering .....	69
5.3.6	leadingObject.....	69
5.3.7	Generated Outcome .....	70
5.4	Operational Modes.....	71
5.4.1	Single File.....	71
5.4.2	List File.....	71
5.4.3	Detail / Summary generation .....	71

5.5	Generated Files .....	71
5.5.1	Variable replacement.....	72
5.5.2	Base Output path.....	73
5.5.3	Output path .....	73
5.5.4	Output suffix.....	73
5.5.5	Summary file.....	74
5.5.6	Zip File .....	74
5.5.7	PlantUML source File .....	74
5.5.8	PlantUML output files .....	74
6	Configuration.....	75
6.1	File overview .....	75
6.1.1	Global configuration file.....	75
6.1.2	Parameter file.....	75
6.1.3	Input list file.....	75
6.1.4	Converter definition file .....	75
6.1.5	Converter definition option file .....	75
6.1.6	Curl parameter file.....	75
6.1.7	Curl authentication file .....	76
6.2	Variables .....	76
6.2.1	Generator .....	76
6.2.2	Generated folder and file names.....	77
6.2.3	Configuration files.....	80
6.2.4	Curl Configuration.....	81
6.2.5	System Environment.....	81
6.3	Global configuration file .....	82
6.3.1	Data model .....	82
6.3.2	Global Configuration.....	82
6.4	Parameter file.....	87
6.4.1	Data model .....	88
6.4.2	Json2PumlParameter .....	88
6.4.3	Single File.....	91
6.4.4	Curl Parameter.....	92
6.5	Input list File.....	93
6.5.1	Data model .....	94
6.5.2	InputListFile .....	95
6.5.3	InputListDescription.....	101

6.5.4	InputListDescriptionCurlParameter .....	102
6.5.5	SingleListFile .....	103
6.5.6	Simple List File Example .....	109
6.6	Converter definition file.....	110
6.6.1	Merging of options .....	110
6.6.2	Identifying an Option .....	111
6.6.3	Data model .....	112
6.6.4	Converterdefinition.....	113
6.6.5	ConverterDefintionDescription .....	114
6.6.6	NamedOption .....	115
6.6.7	PropertyValue.....	115
6.6.8	SingleOption .....	118
6.6.9	ObjectProperty .....	125
6.6.10	CharacteristicProperty .....	126
6.6.11	ObjectFormat.....	128
6.6.12	NamedFormat.....	129
6.6.13	SingleFormat.....	130
6.7	Curl parameter file .....	134
6.7.1	Data model .....	134
6.7.2	Curl Parameter.....	134
6.8	Curl authentication file .....	135
6.8.1	Data model .....	135
6.8.2	UrlConfiguration .....	135
6.8.3	Curl Parameter.....	136
7	Appendix .....	138
7.1	Images.....	138

# 2 Introduction

## 2.1 Current Situation

### 2.1.1 Plain JSON

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

```
{"height":{"meters":70,"feet":229.6}, "diameter":{"meters":3.7,"feet":12}, "mass":{"kg":549054,"lb":1207920}, "first_stage":{"thrust_sea_level":{"kN":7607,"lbf":1710000}, "thrust_vacuum":{"kN":8227,"lbf":1849500}, "reusable":true, "engines":9, "fuel_amount_tons":385, "burn_time_sec":162}, "second_stage":{"thrust":{"kN":934,"lbf":210000}, "payloads":{}}, "composite_fairing":{"height":{"meters":13.1,"feet":43}, "diameter":{"meters":5.2,"feet":17.1}}, "option_1":"dragon"}, "reusable":false, "engines":1, "fuel_amount_tons":90, "burn_time_sec":397}, "engines":[{"isp":{"sea_level":288,"vacuum":312}, "thrust_sea_level":{"kN":845,"lbf":190000}, "thrust_vacuum":{"kN":914,"lbf":205500}, "number":9, "type":"merlin", "version":"1D+"}, {"layout":"octaweb"}, "engine_loss_max":2, "propellant_1":"liquid oxygen", "propellant_2":"RP-1kerosene"}, "thrust_to_weight":180.1}, "landing_legs":{"number":4, "material":"carbon fiber"}, "payload_weights":[{"id":"leo", "name":"Low Earth Orbit"}, {"kg":22800, "lb":50265}, {"id":"gto", "name":"Geosynchronous Transfer Orbit"}, {"kg":8300, "lb":18300}, {"id":"mars", "name":"Mars Orbit"}, {"kg":4020, "lb":8860}], "flickr_images":["https://farm1.staticflickr.com/929/28787338307_3453a11a77_b.jpg", "https://farm4.staticflickr.com/3955/32915197674_eee74d81bb_b.jpg", "https://farm1.staticflickr.com/293/32312415025_6841e30bf1_b.jpg", "https://farm1.staticflickr.com/623/23660653516_5b6cb301d1_b.jpg", "https://farm6.staticflickr.com/5518/31579784413_d853331601_b.jpg", "https://farm1.staticflickr.com/745/32394687645_a9c54a34ef_b.jpg"], "name":"Falcon 9", "type":"rocket"}, "active":true, "stages":2, "boosters":0, "cost_per_launch":50000000, "success_rate_pct":98, "first_flight":"2010-06-04", "country":"United States", "company":"SpaceX", "wikipedia":"https://en.wikipedia.org/wiki/Falcon_9", "description":"Falcon 9 is a two-stage rocket designed and manufactured by SpaceX for the reliable and safe transport of satellites and the Dragon spacecraft into orbit."}, "id":"5e9d0d95eda69973a809d1ec"}
```

Figure 1: JSON sample

A typical answer of a REST 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:

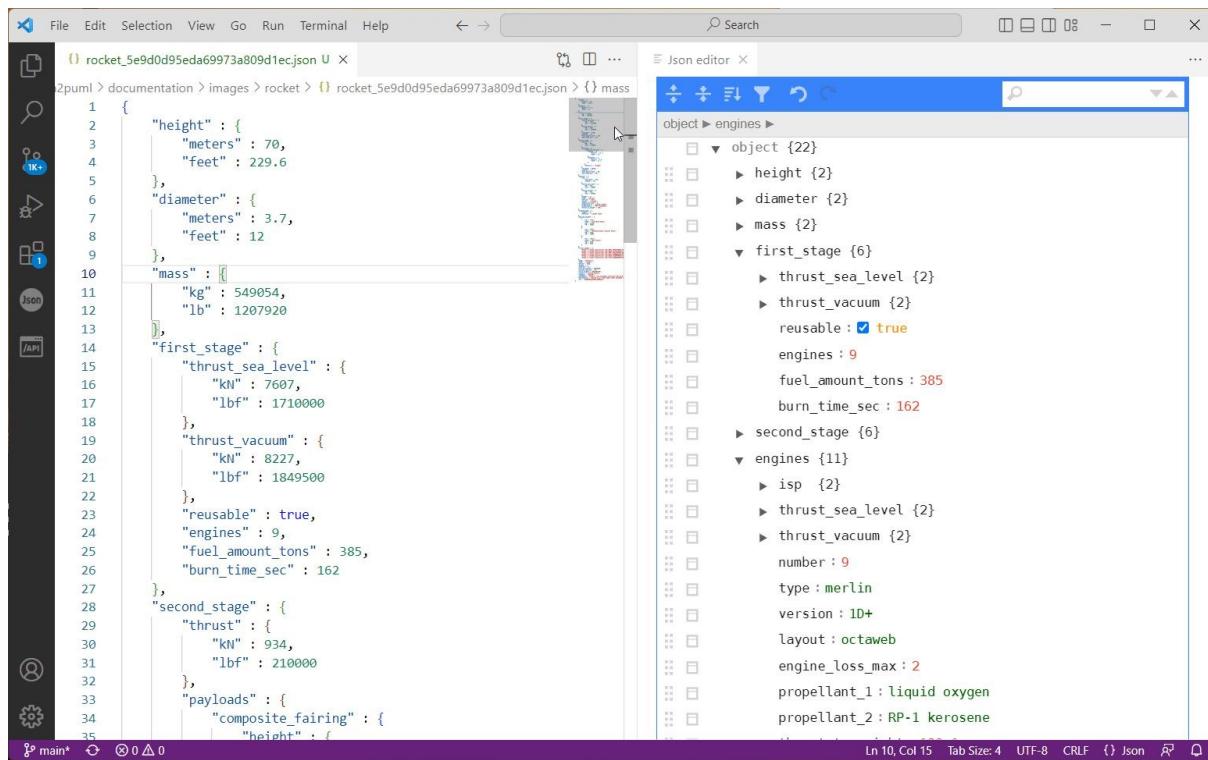


Figure 2: JSON Editor

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:

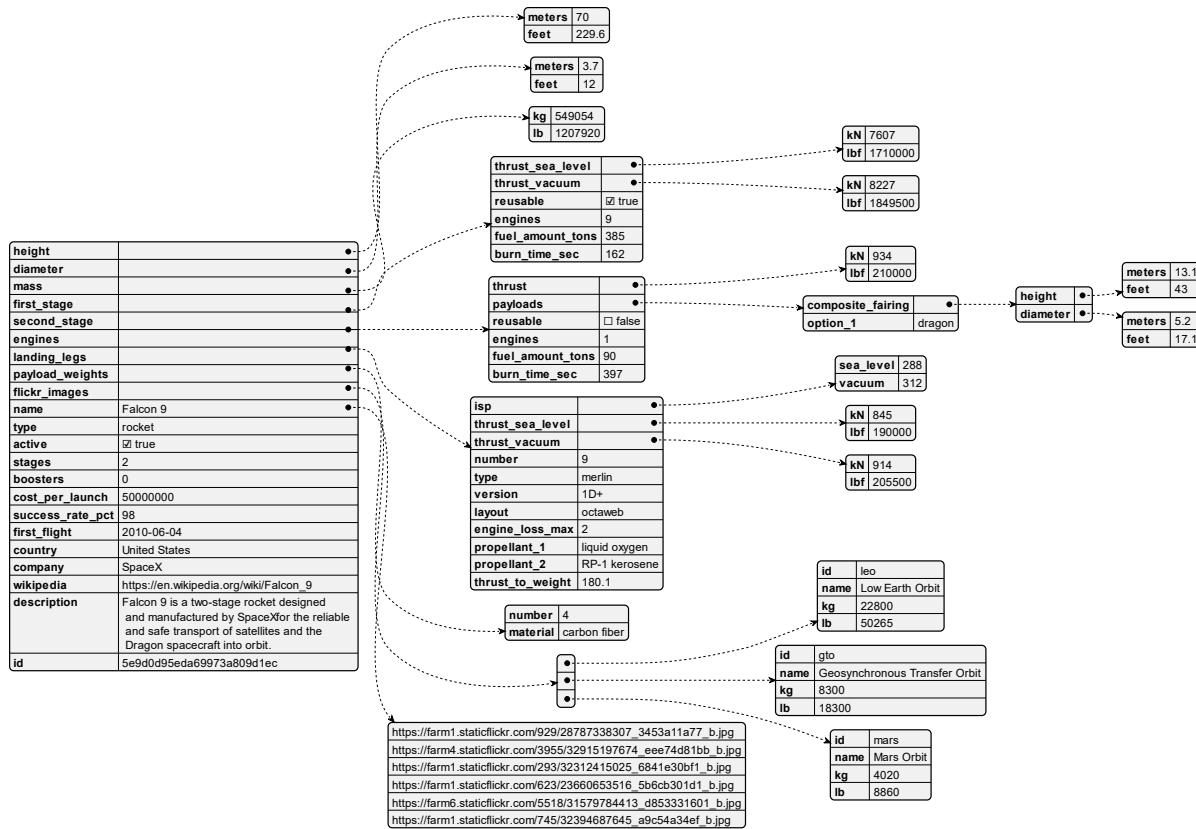


Figure 3 JSON Example PlantUML

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 allows to convert JSON data into PlantUML based diagrams.
- json2puml has an understanding of how data is structured 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.

## 2.3 Space X API based examples

### 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.

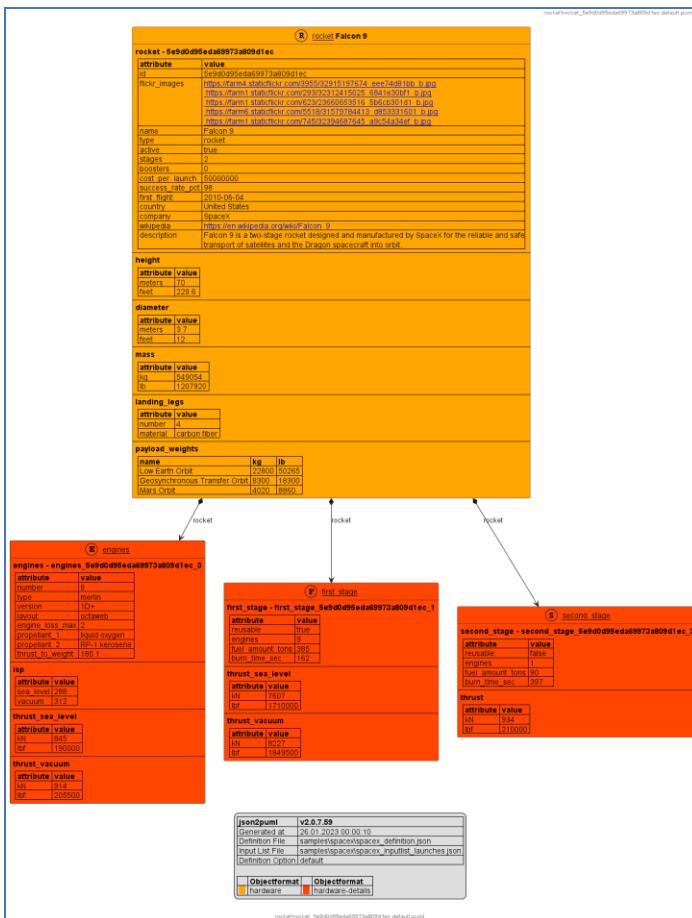
#### 2.3.1 Example 1 – get /rockets

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

Based on this data the following outputs are possible.

rocket\rocket_5e9d0d95eda69973a809d1ec.compact.puml	
<b>R</b> rocket <b>Falcon 9</b>	
<b>rocket - 5e9d0d95eda69973a809d1ec</b>	
<b>attribute</b>	<b>value</b>
id	5e9d0d95eda69973a809d1ec
name	Falcon 9
type	rocket
active	true
stages	2
boosters	0
cost_per_launch	50000000
success_rate_pct	98
first_flight	2010-06-04
country	United States
company	SpaceX
<b>json2puml</b> v2.0.7.59	
Generated at	26.01.2023 00:50:08
Definition File	samples\spacex\spacex_definition.json
Input List File	samples\spacex\spacex_inputlist_launches.json
Definition Option	compact
<b>Objectformat</b>	
<input checked="" type="checkbox"/>	hardware

Figure 4 Space X – get /rockets – compact option



*Figure 5 Space X – get /rockets – default option*

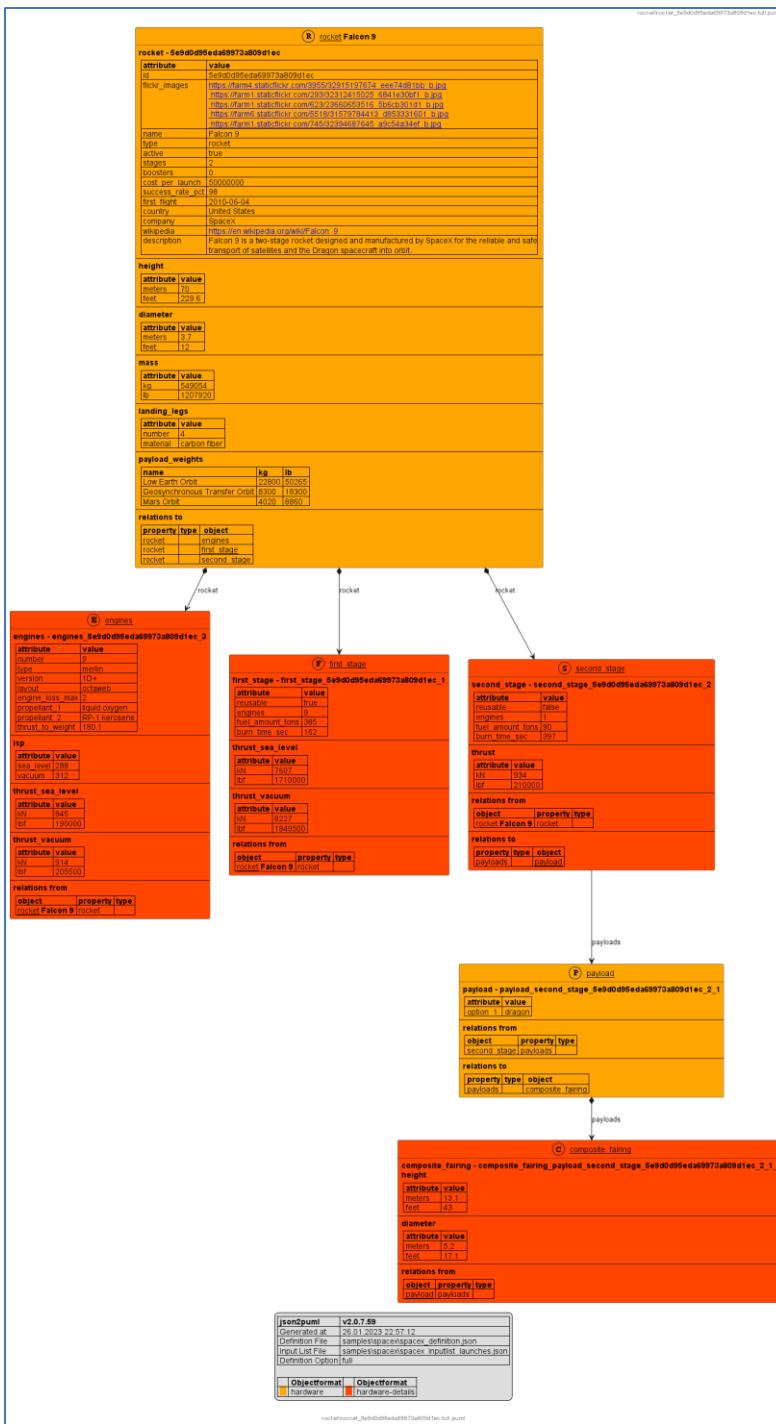


Figure 6 Space X – get /rockets – full option

### 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

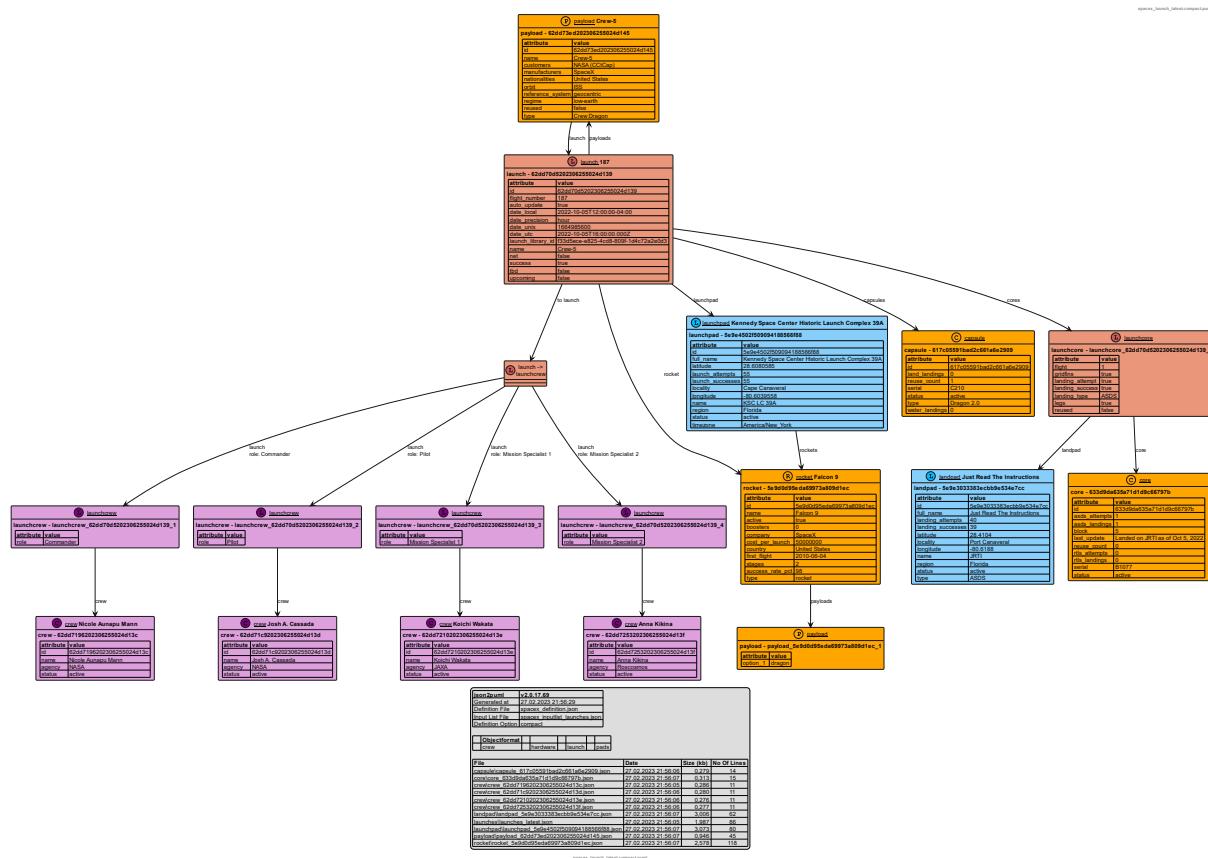
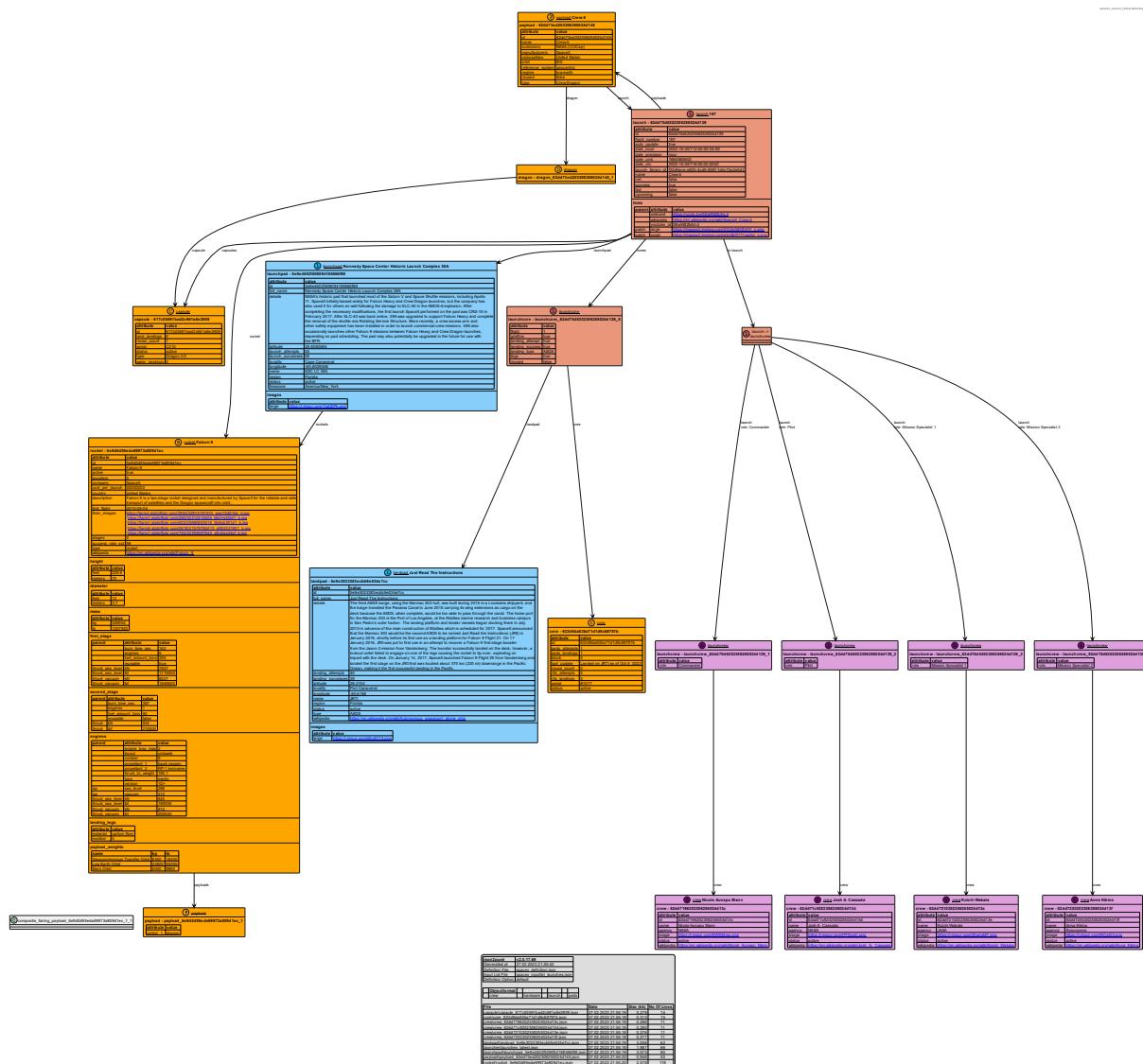


Figure 7 Space X – get all API – compact option



*Figure 8 Space X – get all API – default option*

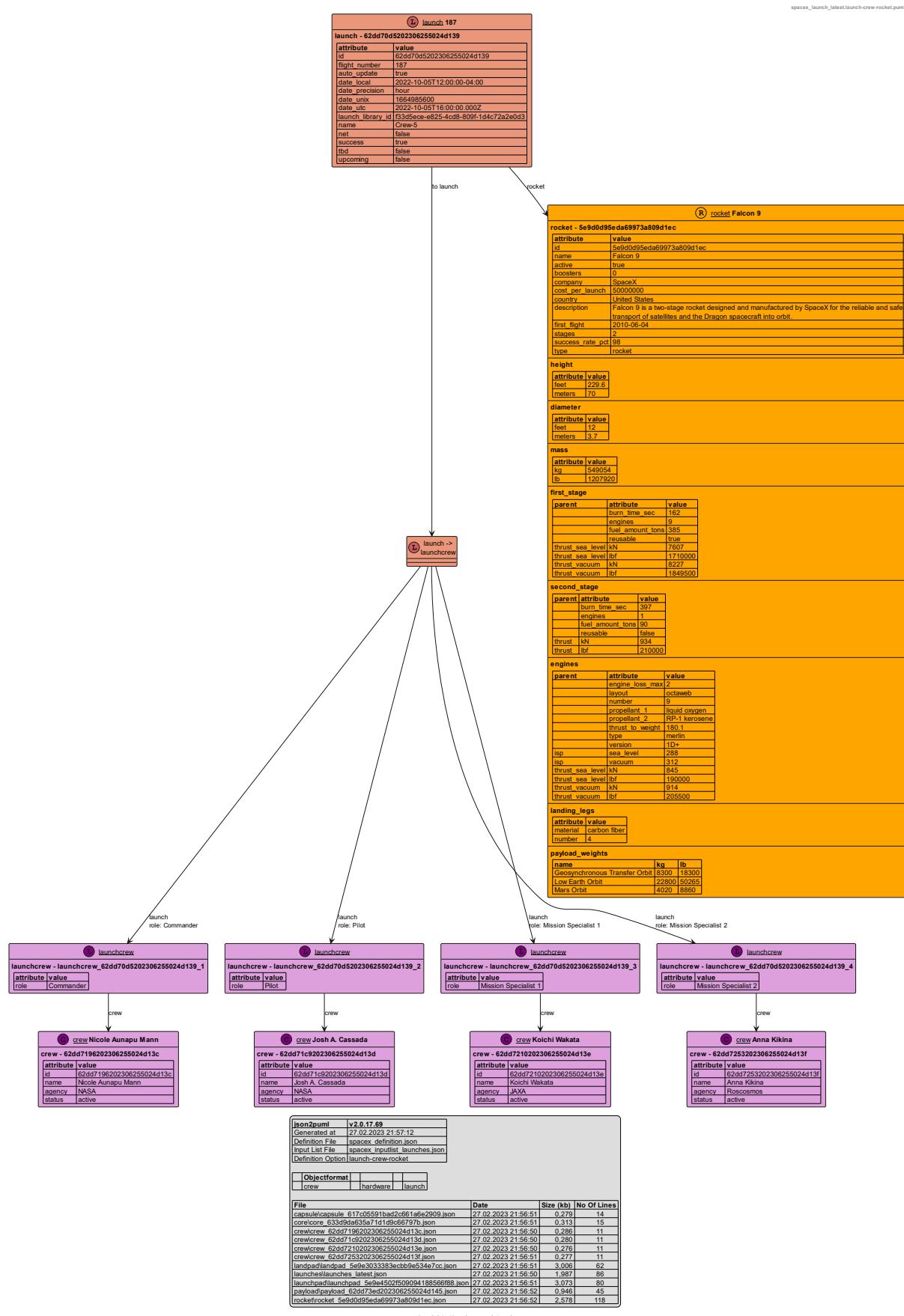


Figure 9 Space X – get all API – launch-crew-rocket option

## 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.

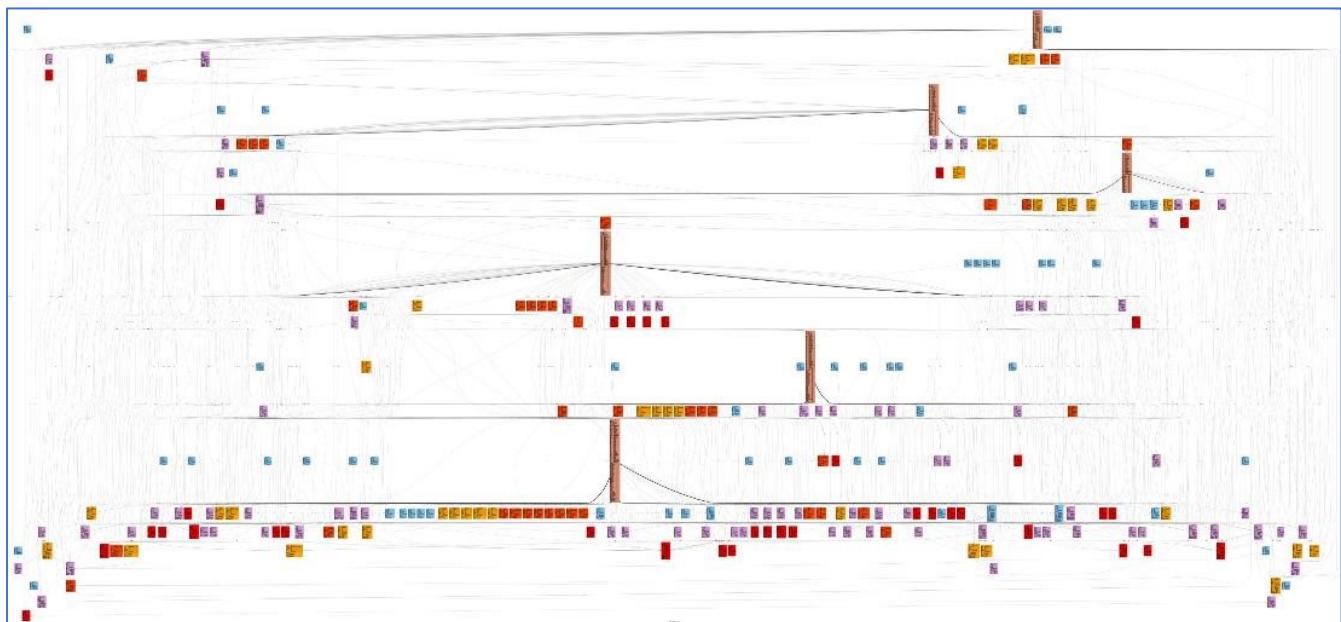


Figure 10 swapi.dev - all data - 6 Films, 82 Persons, 60 Planets, 37 Species, 36 Star ships, 39 Vehicles

## 2.4.2 Example 2 – Han Solo

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

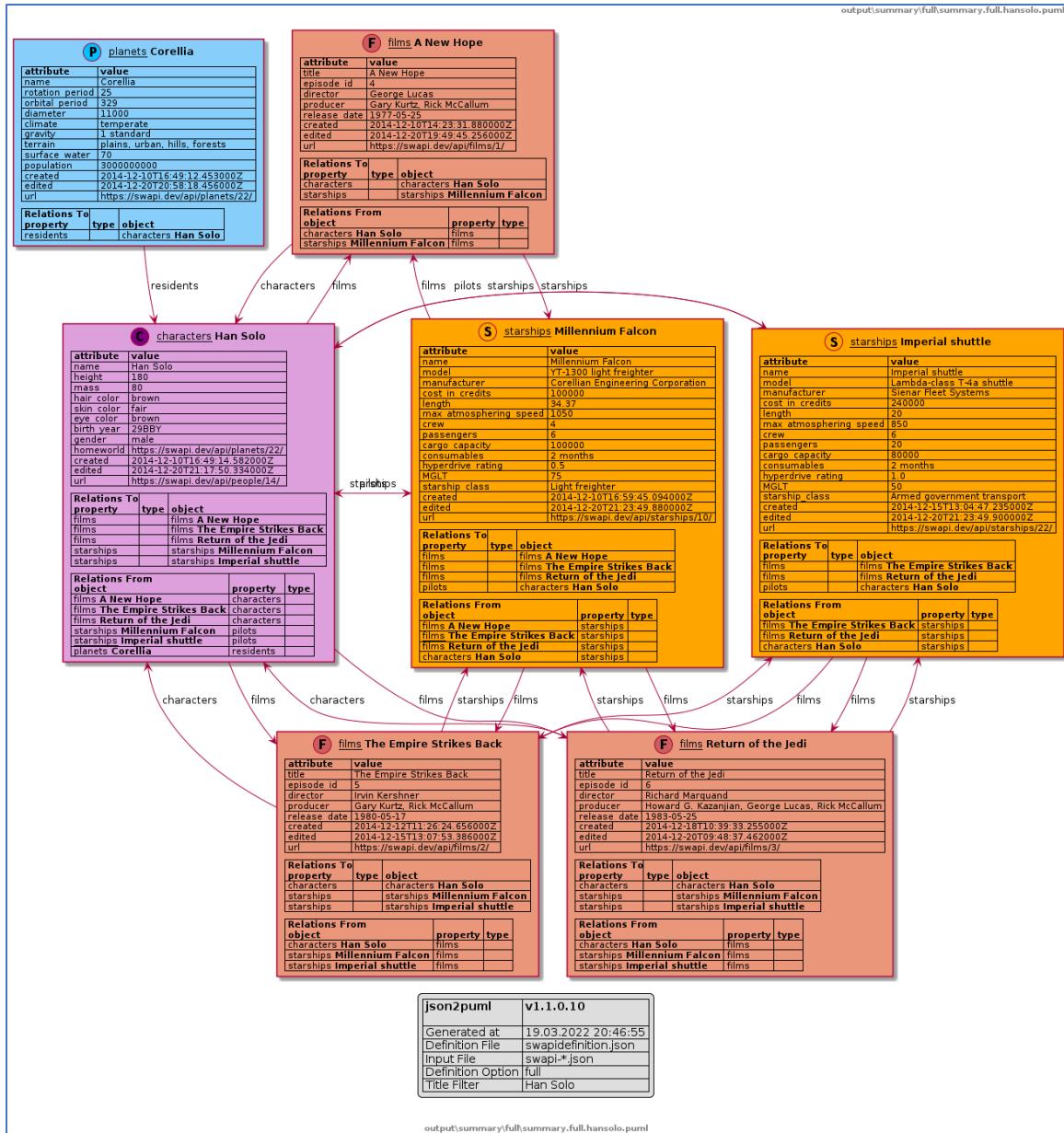


Figure 11 [swapi.dev - /titlefilter = "Han Solo"](https://swapi.dev/api/people/?titlefilter=Han%20Solo)

### 2.4.3 Example 3 – Death Star

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

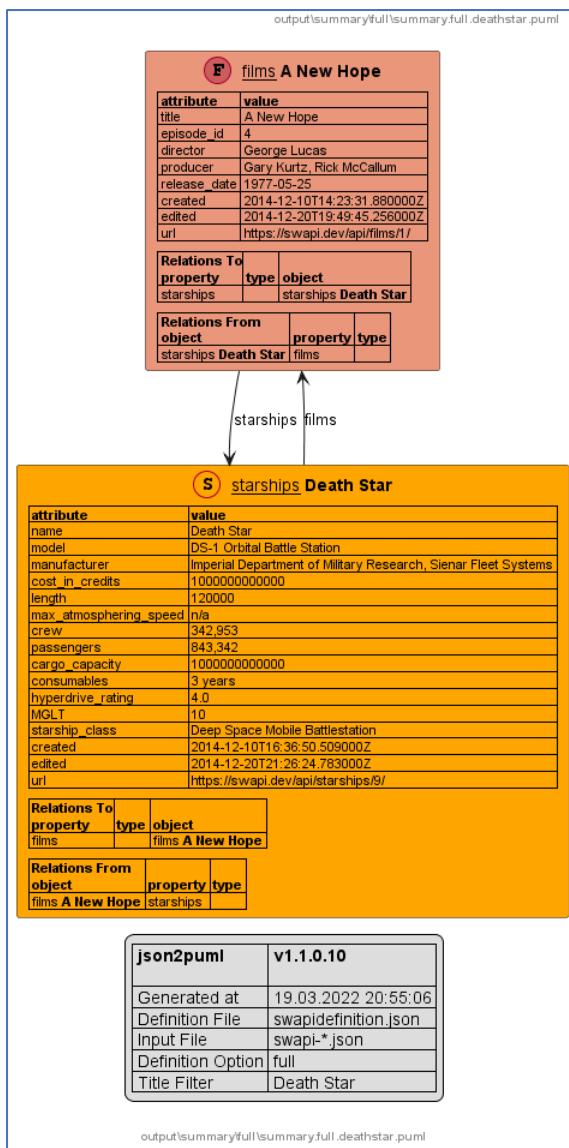


Figure 12 swapi.dev - /titlefilter = "Han Solo"

# 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

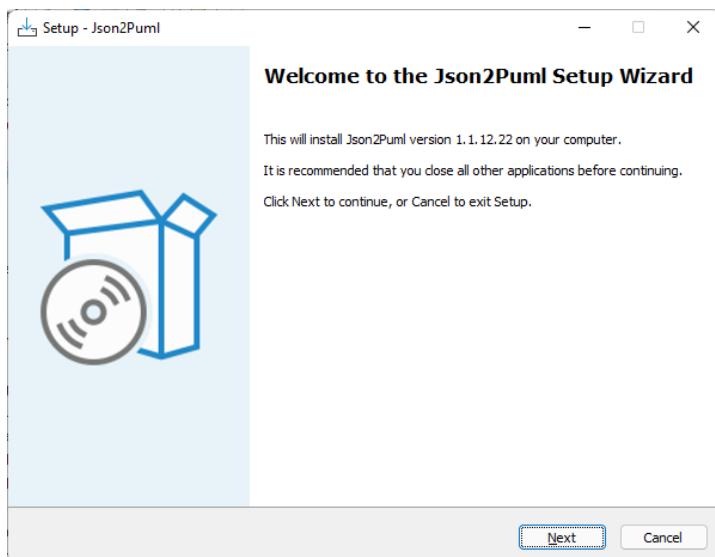


Figure 13 Setup - Welcome Screen

### 3.1.2 Definition of the installation folder

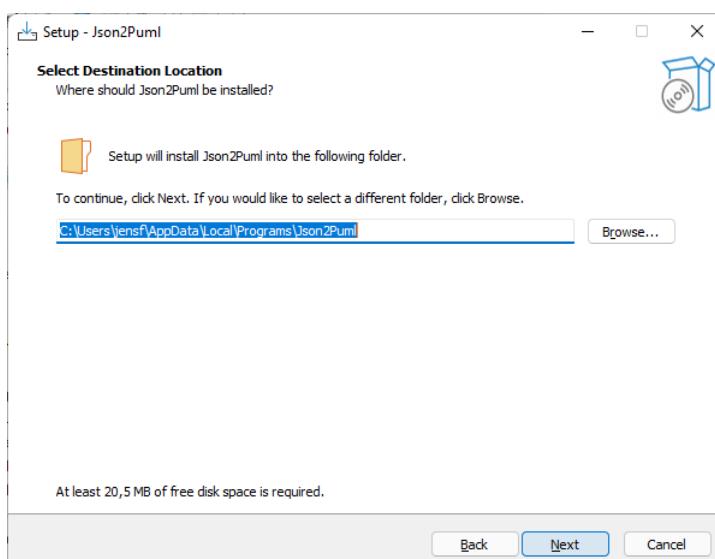
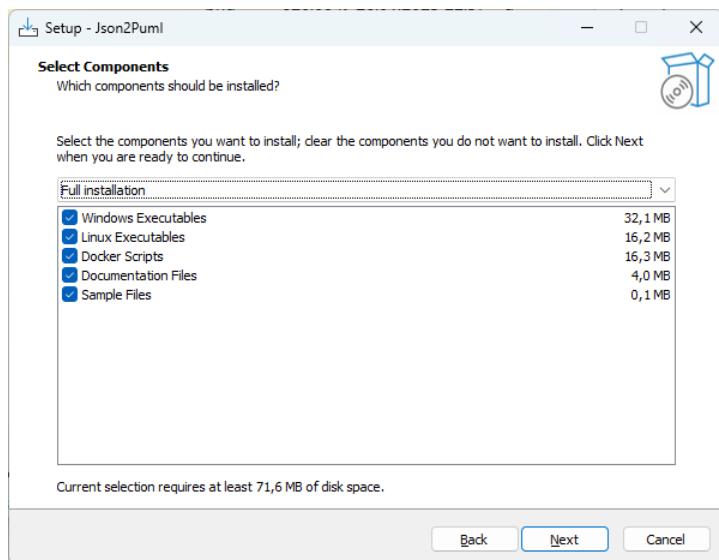


Figure 14 Setup - Definition of installation folder

### 3.1.3 Definition of components to be installed



### 3.1.4 Definition of installation options

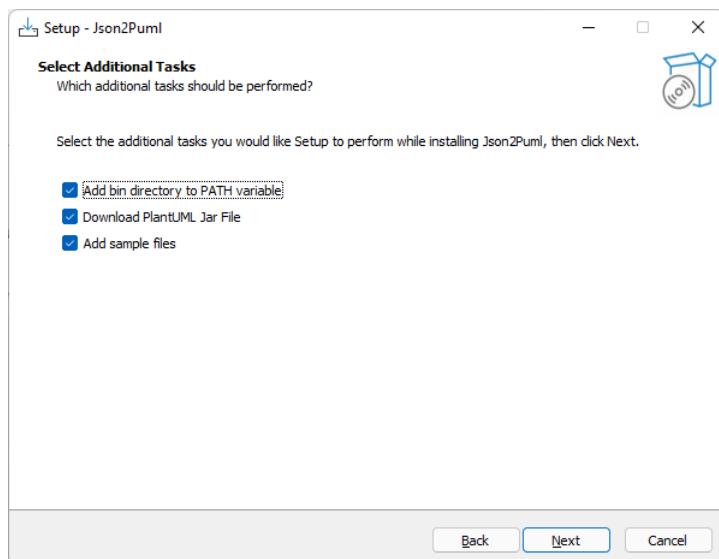


Figure 15 Setup - Definition of installation options

### 3.1.5 Summary before installation

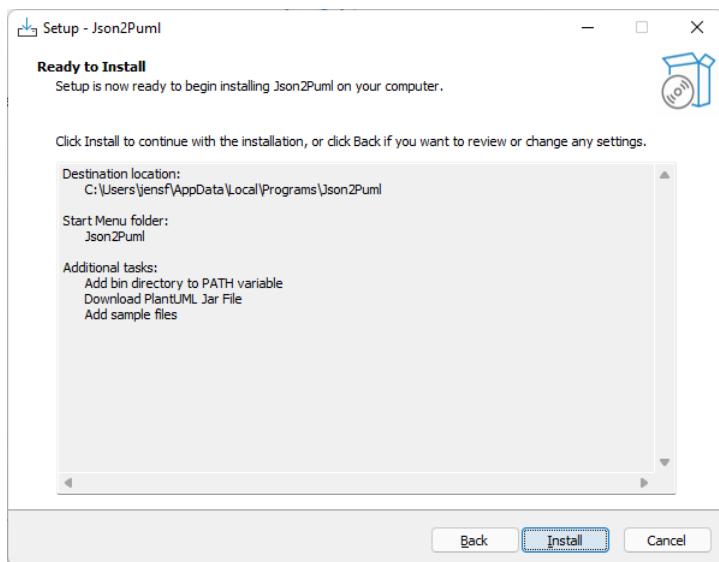


Figure 16 Setup - Summary before installation

### 3.1.6 Status screen while the installation is running

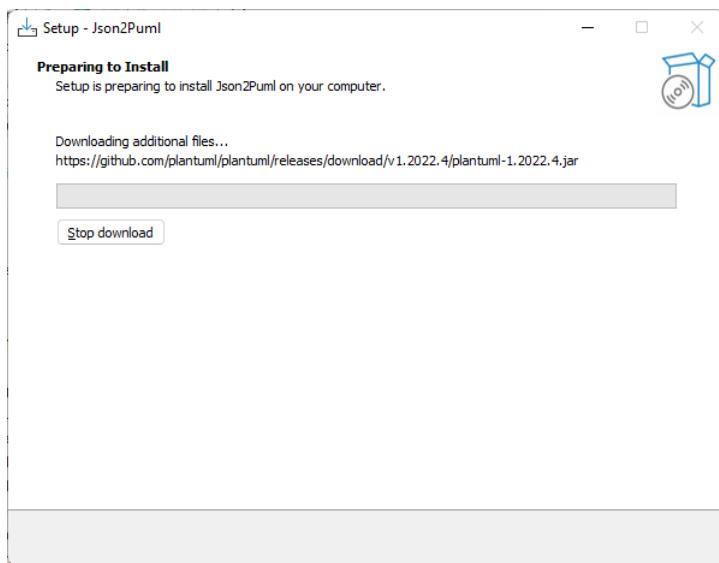


Figure 17 Setup - Status screen while installation

### 3.1.7 Completion screen

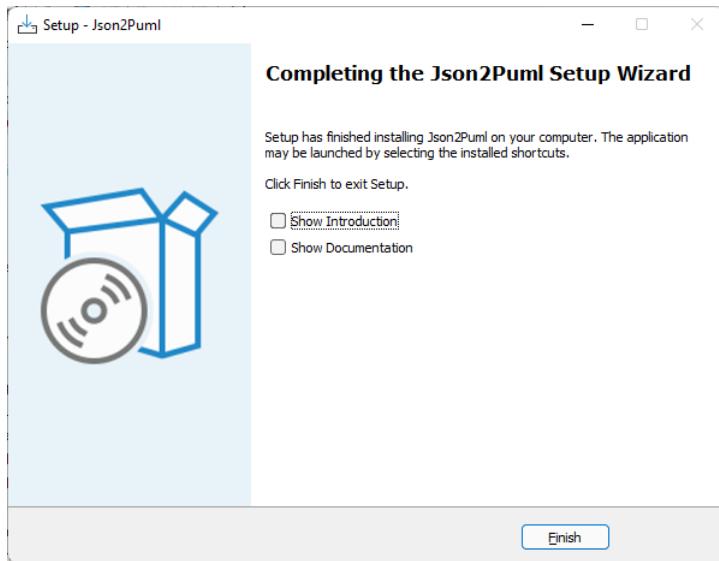


Figure 18 Setup - Completion screen

## 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
json2pumlu.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://www.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

Json2puml This 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.

Example:

```
-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 4.1.2 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 4.1.2 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 4.1.2 Command Line Parameters) or the “outputFormats” parameter in the configuration file (See 6.2.2.11 Output formats).

## 4 How to use

There are multiple types of applications to use json2puml:

- Command line application Windows/Linux<sup>(\*)</sup>
- UI application windows (unsupported, for debugging only)
- Service application Windows/Linux<sup>(\*)</sup>

Linux is officially supported for x86 64-bit and Ubuntu 20.04 LTS, Ubuntu 18.04 LTS, RedHat Enterprise Linux (version 8), and WSL2 (Windows Subsystem for Linux). Other editions should work also, but they are not officially tested.

The Linux based applications can be started from command-line or via predefined docker containers.

The Windows service application can be started from command-line or as a windows service application

### 4.1 Command line

The command line application allows to run the converter based on a set of parameters defined as command line parameters when calling the application.

#### Windows

```
json2puml.exe /inputlistfile:swapi_inputlist_local.json  
/option=detailed -outputformat=png
```

#### Linux

```
./json2puml -inputlistfile:swapi_inputlist_local.json  
-option=detailed -outputformat=png
```

This generates the puml files and the corresponding png images.

#### 4.1.1 Help Screen

The help screen of the command line application is shown when no command line parameter or the help parameter /? -? is used.

```
*****
json2puml v2.1.7.82 - 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>
/definitionfile:<file>
?optionfile:<file>
/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>
/inputlistfile:<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.
/splitidentifier:<identifier>
This option is only valid when generatedetails is activated 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>
/baseoutputpath:<path>
Filename of the generated summary file Base path which will be combined 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>
/openoutput:[<format>]
Format of the generated Puml converters (Allowed values: ,png,svg,pdf) 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>
/generatesummary:<boolean>
This allows to overwrite the generateDetails property of the InputListFile This allows to overwrite the generateSummary property of the InputListFile

/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 InputListFile
/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.
/generatedefaultconfiguration
Flag to define that a default configuration file should be generated.
/debug
Flag to define that a converter log file should be generated parallel to the puml file
/nologfiles
Flag to define that no log files should be generated

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

#### 4.1.2 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.
/serviceport: <portnumber>	Port the service application is listening. Overwrites the global configuration parameter.
/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: 5.3.6 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
	<p>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 6.5.5.4 splitIdentifier)</p> <p>This parameter is only needed for the single file conversion.</p>
<pre>/splitidentifier:     &lt;identifier&gt;</pre>	<p>Name of the property which defines the name/filename of the splitted json element.            (See 6.5.5.4 splitIdentifier)</p> <p>This parameter is only needed for the single file conversion.</p>
<pre>/curlauthenticationfile:     &lt;file&gt;</pre>	<p>CurlAuthenticationFile which contains the central authentication configuration when using the curl automations (See 5.2 Curl pre-processor).</p>
<pre>/curlparameterfile:     &lt;file&gt;</pre>	<p>CurlParameterFile which contains additional parameters to enable a dynamic configuration when using the curl automations (See 5.2 Curl pre-processor).</p>
<pre>/curlauthentication parameterfile:     &lt;file&gt;</pre>	<p>CurlAuthenticationParameterFile which contains additional authentication parameters to enable a dynamic configuration when using the curl automations (See 5.2 Curl pre-processor).</p>
<pre>/curlparameter:     &lt;name&gt;=&lt;value&gt;</pre>	<p>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 5.2 Curl pre-processor).</p>
<pre>/curlauthentication parameter:     &lt;name&gt;=&lt;value&gt;</pre>	<p>Single command line curl authentication parameter, defined as name and value.            The parameter can be used multiple times to define more than one curl authentication parameter.            The command line parameter will overwrite authentication parameter from the parameter file having the same name (See 5.2 Curl pre-processor).</p>
<pre>/outputformat: &lt;format&gt;</pre>	<p>Comma separated list of output formats to be generated.            This parameter overwrites the output format definition in the definition file            (See 6.2.2.11 Output formats)</p>
<pre>/baseoutputpath: &lt;path&gt;</pre>	
<pre>/outputpath: &lt;path&gt;</pre>	Outputpath for the generation of files.

Parameter	Description
	This parameter overwrites the path configured in the definition file and in the input list file. (See 5.5.3 Output path and 6.2.2.7 Output path)
/outputsuffix: <suffix>	Additional name suffix for the generation of files. This parameter overwrites the path configured in the definition file and input list file. (See 5.5.4 Output suffix and 6.2.2.5 Output suffix)
/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. (See 5.5 Generated Files)
/debug	Flag to define that a converter log file should be generated parallel to the puml file

Parameter	Description
/nologfiles	Flag to define that no log files should be generated.
/jobdescription	The job 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.

#### 4.1.3 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 “/inputlistfile” (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.

The following command can be executed in the samples\swapi\data folder

```
json2puml /inputlistfile:swapi_inputlist_local.json
```

The output will look similar to this:

```
*****
json2puml v2.0.10.62 - Command line converter JSON to PUML
*****
Global Configuration (e:\json2puml\configuration\json2pumlconfiguration.json)
baseOutputPath : e:\json2puml\runtime\output\
curlAuthenticationFileName :
    e:\json2puml\configuration\json2pumlcurlauthentication.json
curlPassThroughHeader :
    [1] : traceid
    [2] : x-b3-traceid
InputListFileSearchFolder :
defaultDefinitionFileName :
definitionFileSearchFolder :
    [1] : e:\json2puml\definition\
    [2] : e:\json2puml\samples\swapi\*definition*.json
inputListFileSearchFolder :
    [1] : e:\json2puml\samples\swapi\*inputlist*.json
javaRuntimeParameter :
logFilePath : e:\json2puml\runtime\logs\
putputPath : <job>\<group>\<file>
plantUmlJarFileName :
servicePort : 9090
Current command line parameters: (/inputlistfile:swapi_inputlist_local.json )
/env json2pumlconfigurationfile
    e:\json2puml\configuration\json2pumlconfiguration.json
/inputlistfile
/swapi_inputlist_local.json
/env plantumljarfile
    e:\Json2Puml\plantuml\plantuml-
    1.2023.0.jar
Load configuration files
Load "GlobalConfigurationFile" from
    e:\json2puml\configuration\json2pumlconfiguration.json (24 lines)
Load "CurlAuthentication" from
    e:\json2puml\configuration\json2pumlcurlauthentication.json (15 lines)
Load "InputListFile" from
    e:\json2puml\samples\swapi\data\swapi_inputlist_local.json (24 lines)
Load "ConverterDefinition" from
    e:\json2puml\samples\swapi\swapi_definition.json (462 lines)

Current Configuration
Option default
Group
Detail
OutputPath <job>\<group>\<file>
OutputSuffix
OutputFormats
    png,svg,pdf,puml,json,log,zip,filelist.json
GenerateSummary true
GenerateDetails false
BaseOutputPath e:\json2puml\runtime\output\
FullOutputPath
    e:\json2puml\runtime\output\<job>\<group>\<file>
SummaryFile swapi_all
Input Files
    swapi_data_films.json
    swapi_data_people.json
    swapi_data_planets.json
    swapi_data_species.json
    swapi_data_starships.json
    swapi_data_vehicles.json
Curl Parameter
[ 1/ 1] Convert e:\json2puml\runtime\output\swapi_inputlist_local\swapi_all.json
        to e:\json2puml\runtime\output\swapi_inputlist_local\swapi_all.puml
        puml generated
Execute java -jar "e:\Json2Puml\plantuml\plantuml-1.2023.0.jar"
    "e:\json2puml\runtime\output\swapi_inputlist_local\swapi_all.puml" -svg
        svg generated
Done
```

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

## 4.2 Service Application

The service application is a service application which is listening on http requests to execute various tasks.

The service application is executed once and is running until stopped.

The service application is available for Windows and Linux as a command line application. For Windows exists also a Windows service application which is executed as an operating system service in the background.

### 4.2.1 How to prepare

To run the service application a global configuration file is needed (See 6.1.1 Global configuration file).

Additionally the configuration file variable (See 6.2.3.1 Configuration filename) is needed. It is suggested to use the operating system environment variable `Json2PumlConfigurationFile` to define the global configuration file.

### 4.2.2 How to run

#### 4.2.2.1 Command line

The command line application can be started without any additional parameters:

##### Windows

```
json2pumlservice.exe
```

##### Linux

```
./json2pumlservice
```

The application shows a status message similar to this:

```
** JSON2PUML Server ** v2.0.10.62
Listening on port 9090
CTRL+C to shutdown the server
```

The command line parameters like `/configurationfile` or `/baseoutputpath` are supported.

#### 4.2.2.2 Windows service application

The windows service application is started different then the command line application. The service application needs to be installed and registered as an windows service.

For doing this the following command line parameters are supported:

Parameter	Description
<code>/?</code>	Shows the help screen with all command line parameters

Parameter	Description
/install	Installs and registers the application as a windows service.
/uninstall	Uninstalls and unregisters the application as a windows service.

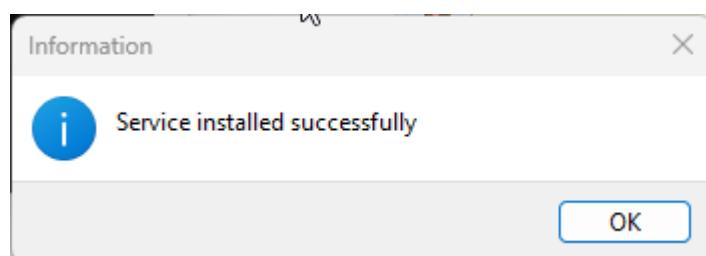
All other command line parameters will be ignored.

**HINT**

For installing or uninstalling the windows service application you must act as a local administrator.

To simplify you can call the install\_json2pumlwindowsservice.cmd and uninstall\_json2pumlwindowsservice.cmd in the installation folder and execute them "as Administrator".

The success of the installation will be signalled with the following message:



After installing the application, you will find it in the list of services:

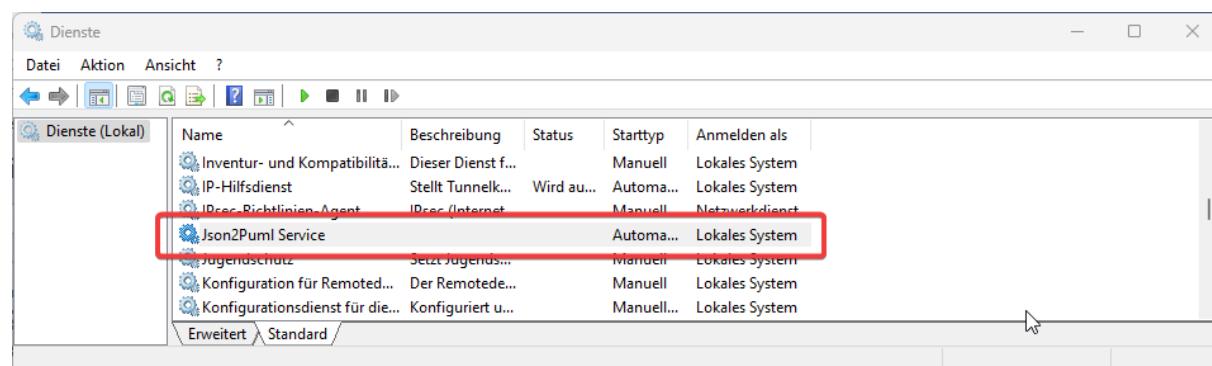


Figure 19 json2puml in windows service list

**HINT**

When using the Windows service application, the environment parameters must be defined as system variable. The application is not executed in the user context, it's always executed in the system context.

#### 4.2.3 How to use it via curl

A simple command to see if the application is successfully running is:

```
curl -X GET <hostname>:<configured port>/api/ inputlistfile
```

When the service application is running on the local machine it can look like:

```
curl -X GET http://localhost:9090/api/inputlistfile
```

You will find additional sample command file for using the service application in the sample\service folder.

#### 4.2.4 Service operations

The detailed parameters of the service operations can be found in the corresponding swagger file json2puml.yaml in the documentation folder.

##### 4.2.4.1 get /heartbeat

This operations returns generic information's about the service and how it is configured.

###### 4.2.4.1.1 Sample curl command

```
curl -X GET http://localhost:9090/api/heartbeat
```

###### 4.2.4.1.2 Sample result data model

Result is a text showing the version of the service and further details.

```
json2puml server v2.2.1.85 (Windows 64Bit x86 Debug)
```

##### 4.2.4.2 get /serviceinformation

This operations returns generic information's about the service and how it is configured.

###### 4.2.4.2.1 Sample curl command

```
curl -X GET http://localhost:9090/api/serviceinformation
```

###### 4.2.4.2.2 Result data model

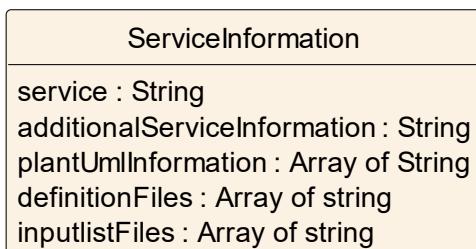


Figure 20 get /serviceinformation result data model

The data structure shows various generic information's about the service and how it is configured.

#### 4.2.4.3 *get /definitionfiles*

This operations returns a list of all configured converter definition files which could be used by the `post /json2pumlRequest*` operations.

##### 4.2.4.3.1 Sample curl command

```
curl -X GET http://localhost:9090/api/definitionfile
```

##### 4.2.4.3.2 Result data model

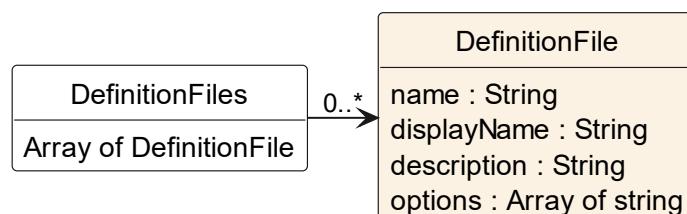


Figure 21 *get /definitionfiles result data model*

The data structure is based on the configuration parameter of the input list file description parameter (See 6.5.2.1 description) and the definition file description found based on the definition parameter (6.5.2.3 definitionFile).

The name property will be filled from the filename of the file.

#### 4.2.4.3.3 Example

A result could look like:

```
[  
 {  
   "name": "tmf_definition.json",  
   "displayName": "TMF Converter Definition",  
   "description": "Converter definition file to generate plantuml files  
     for TMF based data files.\nYou can find further informations  
     here:\nhttps://www.tmfforum.org/oda/about-open-apis/",  
   "options": [  
     "compact", "core", "detailed", "full", "product", "product-detail",  
     "relations", "short"]  
 },  
 {  
   "name": "placeholder_definition.json",  
   "displayName": "JSON Placeholder Definition File",  
   "description": "Converter definition file to generate plantuml files  
     for jsonplaceholder data files.\nThis file is based on the  
     definitions of the public REST API at  
     https://jsonplaceholder.typicode.com/ to show the capabilities  
     of JSON2PUMA.",  
   "options": [  
     "compact", "detailed", "full", "relations", "short"]  
 }  
 ]
```

See 6.3.2.11 definitionFileSearchFolder

#### 4.2.4.4 get /inputlistfiles

This operation returns a list of all configured input list files which could be used by the post /json2pumlRequest\* operations.

The operation has no input parameter.

#### 4.2.4.4.1 Sample curl command

```
curl -X GET http://localhost:9090/api/inputlistfile
```

#### 4.2.4.4.2 Result data model

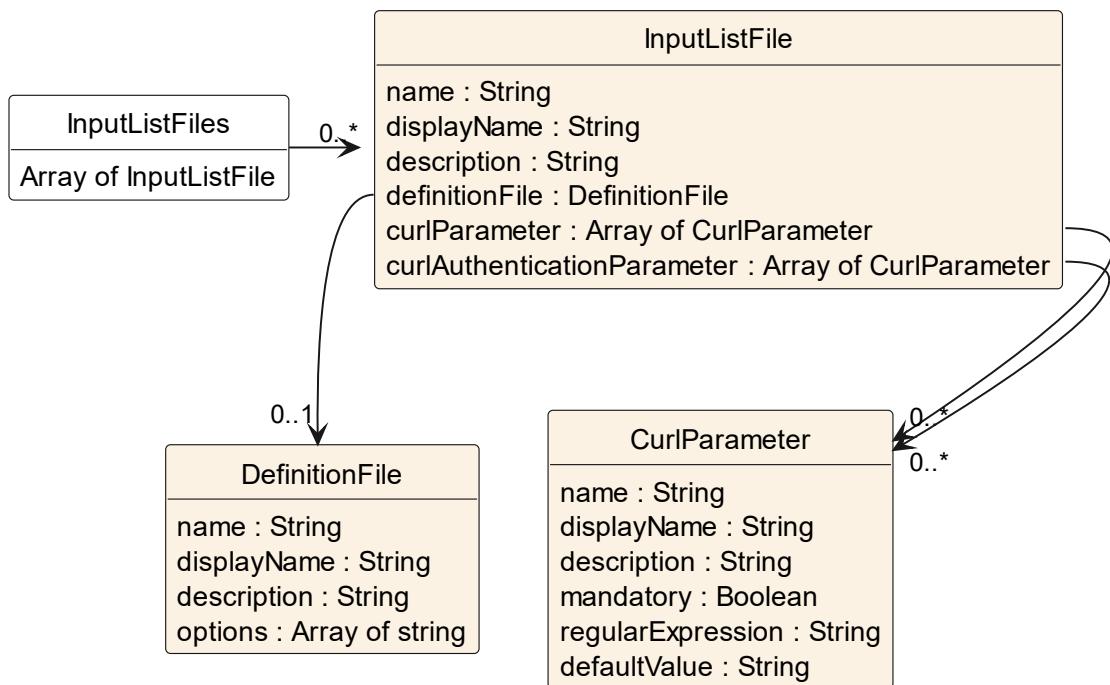


Figure 22 get `/inputlistfiles` result data model

The data structure is based on the configuration parameter of the input list file description parameter (See 6.5.2.1 description) and the definition file description parameter (6.5.2.3 definitionFile)

The name property will be filled from the filename of the file.

#### 4.2.4.4.3 Example

A result could look like:

```
[  
 {  
   "name": "placeholder_inputlist_curl.json",  
   "displayName": "Json Placeholder Data",  
   "description": "This example is based on the JsonPlaceHolder API  
     (\\"https://jsonplaceholder.typicode.com/\").\\nIt delivers data  
     of one test user with corresponding related albums, photos and  
     posts.",  
   "definitionFile": {  
     "name": "placeholder_definition.json",  
     "displayName": "JSON Placeholder Definition File",  
     "description": "Converter definition file to generate plantuml  
       files for jsonplaceholder data files.\\nThis file is baed on the  
       definitions of the public REST API at  
       \\nhttps://jsonplaceholder.typicode.com/ to show the capabilities  
       of JSON2PUML.",  
     "options": [  
       "compact", "detailed", "full", "relations", "short"]  
   },  
   "curlParameter": [  
     {  
       "name": "${userid}",  
       "displayName": "UserId",  
       "description": "ID of the user",  
       "mandatory": "true"  
     }  
   ]  
 },  
 {  
   "name": "swapi_inputlist_film_characters.json",  
   "displayName": "SWAPI Film Character List",  
   "description": "This example is based on the SWAPI API  
     (\\"https://swapi.dev\\").\\nIt delivers a defined film with the  
     characters of the film and the homeworld details of the  
     characters.",  
   "definitionFile": {  
     "name": "swapi_definition.json",  
     "options": [  
       "compact", "detailed", "full", "relations", "short"]  
   },  
   "curlParameter": [  
     {  
       "name": "${id}",  
       "displayName": "Film ID",  
       "description": "Primary id of the film",  
       "mandatory": "true"  
     }  
   ]  
 }]
```

See 6.3.2.12 inputFileSearchFolder

#### 4.2.4.5 post /json2pumlRequest\*

These operations are executing one command similar to a single command line call.  
Based on the operation different result are returned.

#### 4.2.4.5.1 Sample curl command

```
curl -X POST http://localhost:9090/api/json2pumlRequest -d
@jsonplaceholder_parameter_curl.json
```

#### 4.2.4.5.2 Input parameter

The data structure needed as an input parameter is the same as the parameter file definition and can be reused (See 6.4 Parameter file).

The properties `outputFormats`, `generateSummary` and `generateDetails` will be ignored for the operations `post /json2pumlRequestPng` and `post /json2pumlRequestSvg`.

#### Example:

```
{
  "inputListFile": "placeholder_inputlist_curl.json",
  "generateSummary": "true",
  "generateDetails": "true",
  "outputFormats": "svg,log,json,puml,zip",
  "option": "default",
  "curlParameter": [
    {
      "${id}": "1"
    }
  ]
}
```

#### 4.2.4.5.3 post /json2pumlRequest

This operation generates all requested files and returns them as a json structure. The content of the files is return as b64 encoded string and needs to be decoded before they can be used.

##### 4.2.4.5.3.1 Result data model

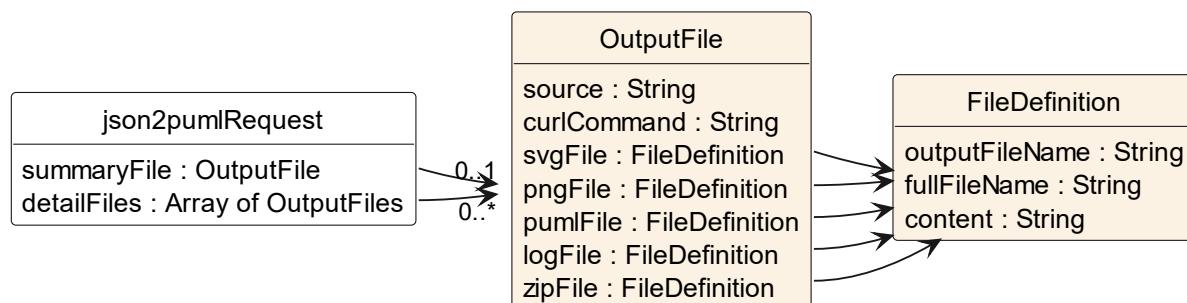


Figure 23 post /json2pumlRequest result data model

#### 4.2.4.5.3.2 Json2pumlRequest

##### 4.2.4.5.3.2.1 summaryFile

Name	Cardinality	Datatype	Default
summaryFile	0..1	OutputFile	
<b>Description</b>			
When a summaryFile is generated then this record contains all details of the generated summary files.			

##### 4.2.4.5.3.2.2 detailsFiles

Name	Cardinality	Datatype	Default
detailsFiles	0..*	OutputFile	
<b>Description</b>			
When a detailFiles are generated then this array contains for all details files an outputFile definition.			

#### 4.2.4.5.3.3 OutputFile

The outputFile contains all information regarding the current generated output file.

##### 4.2.4.5.3.3.1 source

Name	Cardinality	Datatype	Default
Source	1	String	
<b>Description</b>			
Name of the json data file which is basis for the current output file			

##### 4.2.4.5.3.3.2 curlCommand

Name	Cardinality	Datatype	Default
curlCommand	0..1	String	
<b>Description</b>			
If applicable: curl command which is used to fetch the source file.			

##### 4.2.4.5.3.3.3 svgFile

Name	Cardinality	Datatype	Default
svgFile	0..1	FileDefinition	
<b>Description</b>			
When svg is defined as outputFormat: svg result file definition			

4.2.4.5.3.3.4 *pngFile*

Name	Cardinality	Datatype	Default
pngFile	0..1	FileDefinition	
<b>Description</b>			
When <code>png</code> is defined as outputFormat: png result file definition			

4.2.4.5.3.3.5 *pumlFile*

Name	Cardinality	Datatype	Default
pumlFile	0..1	FileDefinition	
<b>Description</b>			
When <code>puml</code> is defined as outputFormat: puml file, which is the base line of the generated images.			

4.2.4.5.3.3.6 *logFile*

Name	Cardinality	Datatype	Default
logFile	0..1	FileDefinition	
<b>Description</b>			
When <code>log</code> is defined as outputFormat: log file, which shows how the json structure has been converted in the puml script file.			

4.2.4.5.3.3.7 *zipFile*

Name	Cardinality	Datatype	Default
zipCommand	0..1	FileDefinition	
<b>Description</b>			
When <code>zip</code> is defined as outputFormat: zip file, which contains all generated files. This property will only be returned as summaryFile (4.2.4.5.3.2.1 <code>summaryFile</code> ).			

## 4.2.4.5.3.4 FileDefinition

For every file the following information will be returned:

4.2.4.5.3.4.1 *outputFileName*

Name	Cardinality	Datatype	Default
outputFileName	1	String	
<b>Description</b>			
Name of the generated file without any folders.			

#### 4.2.4.5.3.4.2 *fullFileName*

Name	Cardinality	Datatype	Default
fullFileName	1	String	
Description			
Name of the generated file including the folder where it's generated.			

#### 4.2.4.5.3.4.3 *content*

Name	Cardinality	Datatype	Default
Content	1	String	
Description			
Base64 encoded content of the generated files. To decode the file any Base64 encoder / algorithm can be used. You can also use: <a href="https://www.base64decode.org/">https://www.base64decode.org/</a>			

#### 4.2.4.5.4 post /json2pumlRequestPng

This operation generates and returns the summary file as a png file.

#### 4.2.4.5.5 post /json2pumlRequestSvg

This operation generates and returns the summary file as a svg file.

#### 4.2.4.5.6 post /json2pumlRequestZip

This operation generates and returns zip file containing all generated summary and details files.

## 4.3 Docker

The docker technology allows to simplify the configuration and execution of Linux based applications in one docker image.

This technology can also be used for json2puml.

As of now there are no pre-configured docker images stored at <https://hub.docker.com>, but all files which are needed to generate the docker images are part of the `docker` folder in the installation folder.

### 4.3.1 Docker image overview

Currently there are three images defined:

- json2pumlbase
- json2puml (based on json2pumlbase)
- json2pumlservice (based on json2pumlbase)

#### 4.3.1.1 json2pumlbase

This image is the base image for the two target images `json2puml` and `json2pumlservice`.

Using the base image simplifies the definition of the two target images and improves the build time. The base image is nearly stable and only needs to be updated when one of the used components is updated.

The base image is based on the following components / packages:

Type	Component / Package	Description
Docker Image	eclipse-temurin:latest	This docker image delivers a current java runtime environment
apk	Graphviz	Graphviz software used by PlantUml
apk	PlantUml	PlantUml runtime
apk	Curl	Curl command line utility

#### 4.3.1.2 json2puml

This image can be used to run a single converter run similar to the command line application.

#### 4.3.1.3 json2pumlservice

This image starts the service application.

### 4.3.2 How to build the images

In the docker folder is a set of utility files defined which can be used to build and maintain the images. They could be used as a starting point for your own configuration.

- `dbuildbase.sh`  
Build the `json2pumlbase` image
- `dbuild.sh`  
Build the `json2puml` image
- `dbuildservice.sh`  
Build the `json2pumlservice` image

#### 4.3.2.1 How to adopt the images

To adopt the images mainly the files in the `docker/src/json2puml/` needs to be updated. This folder is copied into both target docker images.

The `definition` and `inputlist` folders are configured to be returned from the `get /inputlistfiles` and `get /definitionfiles` operations.

As main configuration is `configuration/json2pumlconfiguration.json` configured.

### 4.3.3 How to use the docker images

#### 4.3.3.1 Command line image

This docker image can be used like the command line utility. All command line parameters can be added to the end of the docker command.

```
sudo docker run --rm --name json2puml -option:full --mount  
    type=bind,source=/home/jens/docker/output,  
    target=/json2puml/output -  
    parameterfile:/json2puml/samples/jsonplaceholder/placeholdercurl  
    unixparameter.json
```

The output files will be generated in the local folder /json2puml/output.

#### 4.3.3.2 Service image

To run the service application no further parameters are needed.

```
sudo docker run -d --rm --name json2pumlservice -p 9090:9090  
    json2pumlservice
```

Only the used port needs to be mapped from inside to the outside of the docker image.

#### 4.3.3.3 Mount log and output files outside of the image

To simplify the debugging and the access to the files it is possible to mount the log and output folders to a folder outside of the docker image.

In the main configuration file the following configurations are defined:

```
"baseOutputPath": "/json2puml/output",  
"logFileOutputPath": "/json2puml/logs",
```

By adding the following parameters to the docker start command the folders will be mapped to a folder outside of the image and persist also when the docker image is stopped.

```
sudo docker run --rm --name json2puml  
    --mount type=bind,source=/home/jens/docker/logs,  
    target=/json2puml/logs  
    --mount type=bind,source=/home/jens/docker/output,  
    target=/json2puml/output
```

# 5 How the system is working

## 5.1 Main components

Json2puml consists of 3 main components:

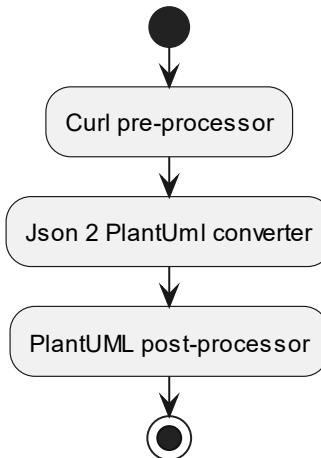


Figure 24 json2puml main components

### 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.

#### 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 6.7 Curl parameter file) and the curl authentication file (See 6.8.3 Curl Parameter)

### 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 6.2.5.1 PlantUML jar filename).

In addition, the `javaRuntimeParameter` variable can be used to configure additional parameters to be added to the java command line (See 6.2.5.2 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-processor 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 a set 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 Using curl parameters in configuration

To use the dynamic parametrisation 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.

Example:

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

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

The content of the following configuration properties could be enhanced by curl parameters:

- `inputListFile.curlOptions` and `singleInputFile.curlOptions`
- `singleInputFile.inputFile` and  
`singleInputFile.curlFileSuffix`
- `singleInputFile.curlBaseUrl` and `inputListFile.curlBaseUrl`
- `singleInputFile(curlExecuteEvaluation`
- `singleInputFile.curlUrl`
- `inputListFile.curlUrlAddon`
- `inputListFile.summaryFile`
- `inputListFile.group` and `inputListFile.detail` and  
`inputListFile.job`

### 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 (6.5.2.13 summaryFileName) and for the input file name of the SingleListFile (6.5.5.1 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`,  
`singleInputFile.curlOptions`, `curlPassThroughHeader`,  
`curlUserAgentInformation`, `curlSpanIdHeader` **and**  
`curlTraceIdHeader`.
- <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:

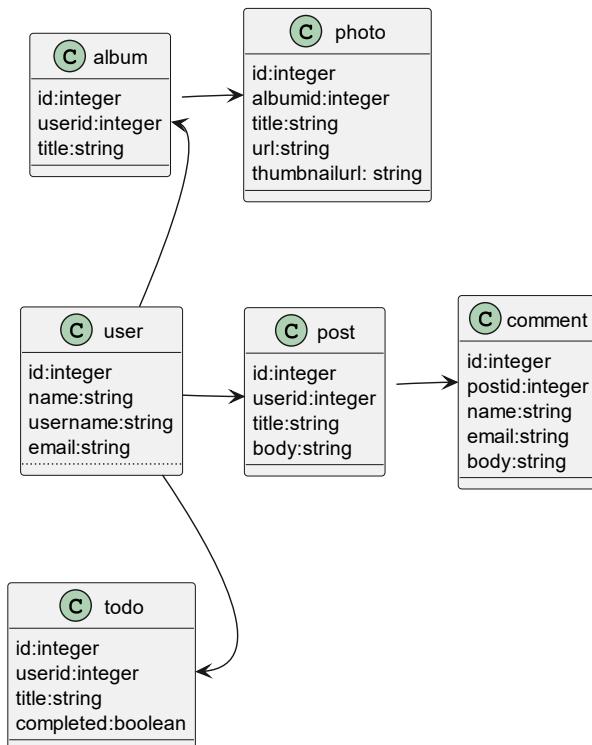


Figure 25 jsonplaceholder data model

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
<code> \${userid}</code>	1
<code> \${name}</code>	Leanne Graham
<code> \${albumid}</code>	1
<code> \${albumid}</code>	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 incident",  
    "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 authentication 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.

The important difference is that the content of curl authentication parameters are not written into the log files or into the summary file list.

To simplify the definition of curl authentication parameters the curl authentication file can be used to configure the parameters per base URL.

Based on the current base URL of the current file the curl authentication file is searched and only these parameters which are defined for the matching URL will be used for replacing (See also 5.2.6.4 Curl authentication file and 6.8 Curl authentication file)

).

### 5.2.4.2 OAuth Example

One common scenario for API security is the OAuth standard.

In this case the first step is to get a security token from an OAuth API call and this token is used in all further API calls of the same system.

Such a scenario can easily be configured in the json2puml curl pre-processor.

The following samples of the input list file and the curl authentication file showing how this could be configured.

### Curl authentication file:

```
[  
  { "baseUrl": "https://mytmfserver.com",  
    "parameter": [  
      {"${authentication}": "${Json2PumltMFAuthentication}"}]  
  },  
  { "baseUrl": "https://www.myotherserver.com",  
    "parameter": [  
      {"${userid}": "json2puml"}, {"${password}": "json2puml"}]  
  }  
]
```

### Input list file:

```
{  
  "definitionFile": "tmfdefinition.json",  
  "curlBaseUrl": "https://mytmfserver.com",  
  "summaryFileName": "summary_customer_${customer_number}",  
  "input": [  
    {  
      "inputFile": "oauthToken.json",  
      "leadingObject": "token",  
      "curlFormatOutput": "true",  
      "curlUrl": "oauth2/v1/accesstoken?grant_type=client_credentials",  
      "curlOptions": "--header \"Authorization: Basic  
      ${authentication}\" --request POST",  
      "curlCache": "3500",  
      "generateOutput": "false",  
      "curlOutputParameter": [  
        {"${access_token}": "access_token"}  
      ]  
    },  
    {  
      "inputFile": "customer.json",  
      "leadingObject": "customer",  
      "curlFormatOutput": "true",  
      "curlUrl": "customerManagement/v1/customer/${customer_number}",  
      "curlOptions": "--header \"Authorization: Bearer ${access_token}\"  
      --request GET",  
      "curlCache": "0",  
      "generateOutput": "true"  
    },  
    {  
      "inputFile": "billingaccount.json",  
      "leadingObject": "account",  
      "curlFormatOutput": "true",  
      "curlUrl":  
        "accountManagement/v1/billingAccount/?filters=relatedParty.id==$  
        {customer_number}",  
      "curlOptions": "--header \"Authorization: Bearer ${access_token}\"  
      --request GET",  
      "curlCache": "0",  
      "generateOutput": "true"  
    }  
  ]  
}
```

### Step 1:

With the first configured file the OAuth token for this run will be fetched.

```
{  
    "inputFile": "oauthToken.json",  
    "leadingObject": "token",  
    "curlFormatOutput": "true",  
    "curlUrl": "oauth2/v1/accesstoken?grant_type=client_credentials",  
    "curlOptions": "--header \"Authorization: Basic  
    ${authentication}\" --request POST",  
    "curlCache": "3500",  
    "generateOutput": "false",  
    "curlOutputParameter": [  
        {  
            "${access_token}": "access_token"  
        }  
    ]  
},
```

The generated curl command will look like:

```
curl --header \"Authorization: Basic ${authentication}\" --request  
POST -url  
https://mytmfserer.com/oauth2/v1/accesstoken?grant\_type=client\_credentials -o oauthToken.json
```

The parameter \${authentication} will be fetched from the curl authentication file from the record with the base url matching the current base URL. The found value is \${Json2PumlTMFAuthentication}. This value is then checked against the operating system environment variables, and here a value like "myHiddenSecretPassword" may be configured.

This will lead to the following curl command

```
curl --header \"Authorization: Basic myHiddenSecretPassword\" --  
request POST -url  
https://mytmfserer.com/oauth2/v1/accesstoken?grant\_type=client\_credentials -o oauthToken.json
```

As the returned token is valid for one hour the "curlCache": 3590 configuration is used to reduce the handling time a little bit, when the input list file is executed multiple times.

The fetched result file oauthToken.json may look like:

```
{
  "refresh_token_expires_in" : "0",
  "token_type" : "Bearer",
  "issued_at" : "1667420998563",
  "client_id" : "HLwYzbGBiiP1Iyib1vLJ4GusgkkvyW30",
  "access_token" : "aemPdLssnGoLxhEmRlsxtJNUkSCB",
  "application_name" : "123cfgaf-579f-4bed-a92e-f0cc59af1234",
  "expires_in" : "3599",
  "refresh_count" : "0",
  "status" : "approved"
}
```

Now the configuration of the curlOutputParameter

```
"curlOutputParameter": [
  {
    "${access_token}": "access_token"
  }
]
```

is needed to get the returned bearer token out of the file.

```
 ${access_token} = aemPdLssnGoLxhEmRlsxtJNUkSCB
```

This bearer token is then used in the following API calls which are fetching the wanted data.

The configuration "generateOutput": "false" prevents additionally that the content of the oauthToken.json file is integrated into the summary result file.

## Step 2:

```
{
  "inputFile": "customer.json",
  "leadingObject": "customer",
  "curlFormatOutput": "true",
  "curlUrl": "customerManagement/v1/customer/${customer_number}",
  "curlOptions": "--header \"Authorization: Bearer ${access_token}\"",
  "curlCache": "0",
  "generateOutput": "true"
},
```

The curl command will look like:

```
curl --header \"Authorization: Bearer ${access_token}\" --request GET
      -url
      https://mytmfserver.com/customerManagement/v1/customer/${customer_number} -o customer.json
```

Based on the command line curl parameter \${customer\_number}=123456 and the fetched parameter \${access\_token} = aemPdLssnGoLxhEmRlsxtJNUkSCB the executed command line will look like:

```
curl --header \"Authorization: Bearer aemPdLssnGoLxhEmRlsxtJNUkSCB\" -  
-request GET -url  
https://mytmfserver.com/customerManagement/v1/customer/123456 -o  
customer.json
```

### Step 3:

This can be repeated for the next file (and any further files)

```
{  
    "inputFile": "billingaccount.json",  
    "leadingObject": "account",  
    "curlFormatOutput": "true",  
    "curlUrl":  
        "accountManagement/v1/billingAccount/?filters=relatedParty.id==$  
        {customer_number}",  
    "curlOptions": "--header \"Authorization: Bearer ${access_token}\""  
        "--request GET",  
    "curlCache": "0",  
    "generateOutput": "true"  
}
```

This will lead to the following final curl command:

```
curl --header \"Authorization: Bearer aemPdLssnGoLxhEmRlsxtJNUkSCB\" -  
-request GET -url  
https://mytmfserver.com/accountManagement/v1/billingAccount/?filt  
ers=relatedParty.id==123456 -o billingaccount.json
```

## 5.2.5 OpenTelemetry Trace/Span-Id

json2puml supports the OpenTelemetry (<https://opentelemetry.io/>) traceability of api calls by transporting and generating trace and span id's when fetching data via curl.

For this the trace-id and span-id header can be configured in the global configuration and in the input list file (see 6.2.4.2 Curl Trace Id Header and 6.2.4.1 Curl Span Id Header).

When the parameters are defined the uuid's will be generated and added as request header to the curl command.

### 5.2.5.1 Reuse of trace id in service application

When in the service application the trace id header is used when calling the /json2pumlRequest\* service no new trace id will be used, the existing one will be used for calling curl.

This will only work for the curlPassThroughHeader and curlTraceIdHeader of the global configuration. The curlTraceIdHeader of the input list file cannot be used for this purpose.

## 5.2.6 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 authentication file

### 5.2.6.1 *Curl parameter via command line*

#### 5.2.6.1.1 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.

#### 5.2.6.1.2 Curl authentication parameter

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

### 5.2.6.2 *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 6.7 Curl parameter file

### 5.2.6.3 *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 6.5.5.13 curlOutputParameter

#### 5.2.6.4 *Curl authentication file*

The curl authentication 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 5.2.4 API Security / OAuth

#### 5.2.6.5 *Environment variables*

For further flexibility it is also possible to fill the curl parameter values via environment variables of the operating system.

It allows to work with source code repository-based configuration files and runtime environment tools like docker and Kubernetes to use one configuration files in multiple environments.

This is supported in the following area:

- `curlbaseurl`
- `curlparameter.value`

## 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 converter follows the following high-level structure.

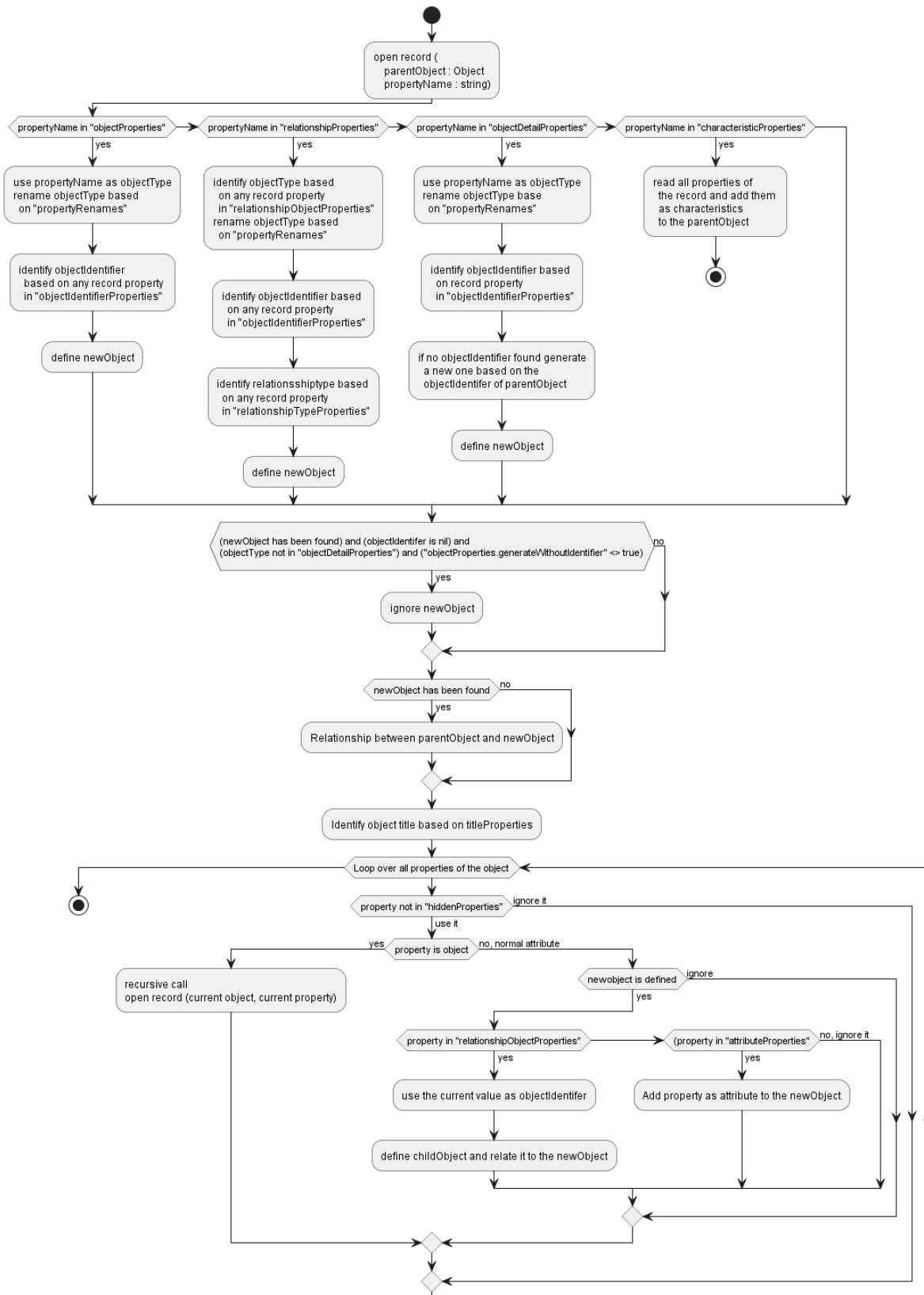


Figure 26: Basic flow how the converter is working

### 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 6.6.8.4 `identifyObjectsByTypeAndIdent`)
- The type of an object will be identified by the following rules:
  - o First the name of the record property or if it is the leading record, the name of the command line parameter “`leadingObject`”.
  - o If the JSON record contains a property matching one of the “`objectTypeProperties`” (See also 6.6.8.11 `objectTypeProperties`) the corresponding value is used.
  - o The type of the object can be mapped to consolidated values using the “`objectTypeRenames`” parameter (See 6.6.8.12 `objectTypeRenames`). This allows to harmonize the object names.
  - o 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 6.6.9.2 `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 define based on the parameter “`relationshipTypeArrowFormats`” (See 6.6.8.18 `relationshipTypeArrowFormats`) how the arrows should be painted.

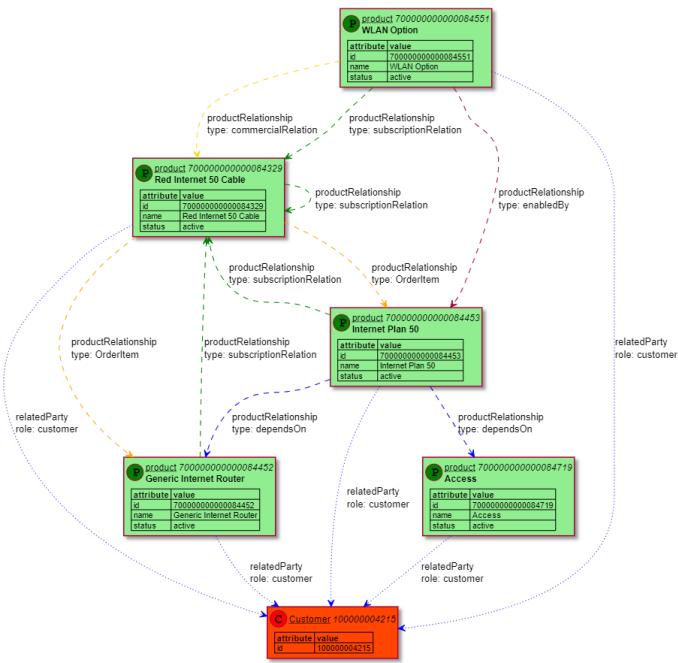


Figure 27 Object relationship example

### 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 6.6.8.15 `objectDetailProperties`)

Detailobjects will be visualized with a diamond at the relationship:

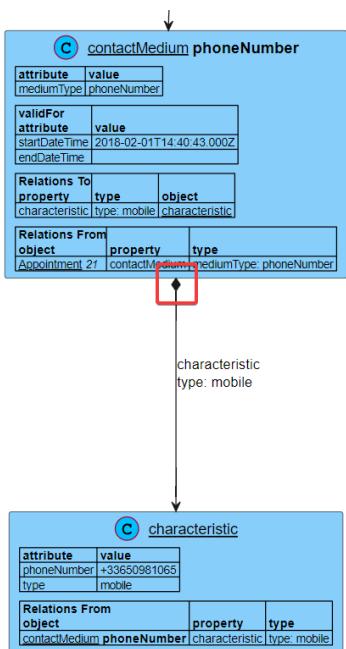


Figure 28 Visualisation of a detailed object

### 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 be 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 6.6.8.20 characteristicProperties)

### Example:

#### Json Data

```
{  
    "height" : {  
        "meters" : 70,  
        "feet" : 229.6  
    },  
    "diameter" : {  
        "meters" : 3.7,  
        "feet" : 12  
    },  
    "mass" : {  
        "kg" : 549054,  
        "lb" : 1207920  
    },  
    "landing_legs" : {  
        "number" : 4,  
        "material" : "carbon fiber"  
    },  
    "payload_weights" : [  
        {  
            "id" : "leo",  
            "name" : "Low Earth Orbit",  
            "kg" : 22800,  
            "lb" : 50265  
        },  
        {  
            "id" : "gto",  
            "name" : "Geosynchronous Transfer Orbit",  
            "kg" : 8300,  
            "lb" : 18300  
        },  
        {  
            "id" : "mars",  
            "name" : "Mars Orbit",  
            "kg" : 4020,  
            "lb" : 8860  
        }  
    ],  
    "name" : "Falcon 9",  
    "type" : "rocket",  
    "active" : true,  
    "stages" : 2,  
    "boosters" : 0  
}
```

### Characteristic Configuration:

```
"characteristicProperties": [  
    "height",  
    "mass",  
    "diameter",  
    "thrust",  
    {"parentProperty": "payload_weights", "type": "list",  
     "nameProperty": "name", "valueProperty": "kg", "infoProperty":  
     "lb"}  
,
```

### Generated Image:

<b>height</b>												
<table border="1"> <thead> <tr> <th>attribute</th><th>value</th></tr> </thead> <tbody> <tr> <td>meters</td><td>70</td></tr> <tr> <td>feet</td><td>229.6</td></tr> </tbody> </table>	attribute	value	meters	70	feet	229.6						
attribute	value											
meters	70											
feet	229.6											
<b>diameter</b>												
<table border="1"> <thead> <tr> <th>attribute</th><th>value</th></tr> </thead> <tbody> <tr> <td>meters</td><td>3.7</td></tr> <tr> <td>feet</td><td>12</td></tr> </tbody> </table>	attribute	value	meters	3.7	feet	12						
attribute	value											
meters	3.7											
feet	12											
<b>mass</b>												
<table border="1"> <thead> <tr> <th>attribute</th><th>value</th></tr> </thead> <tbody> <tr> <td>kg</td><td>549054</td></tr> <tr> <td>lb</td><td>1207920</td></tr> </tbody> </table>	attribute	value	kg	549054	lb	1207920						
attribute	value											
kg	549054											
lb	1207920											
<b>landing_legs</b>												
<table border="1"> <thead> <tr> <th>attribute</th><th>value</th></tr> </thead> <tbody> <tr> <td>number</td><td>4</td></tr> <tr> <td>material</td><td>carbon fiber</td></tr> </tbody> </table>	attribute	value	number	4	material	carbon fiber						
attribute	value											
number	4											
material	carbon fiber											
<b>payload_weights</b>												
<table border="1"> <thead> <tr> <th>name</th><th>kg</th><th>lb</th></tr> </thead> <tbody> <tr> <td>Low Earth Orbit</td><td>22800</td><td>50265</td></tr> <tr> <td>Geosynchronous Transfer Orbit</td><td>8300</td><td>18300</td></tr> <tr> <td>Mars Orbit</td><td>4020</td><td>8860</td></tr> </tbody> </table>	name	kg	lb	Low Earth Orbit	22800	50265	Geosynchronous Transfer Orbit	8300	18300	Mars Orbit	4020	8860
name	kg	lb										
Low Earth Orbit	22800	50265										
Geosynchronous Transfer Orbit	8300	18300										
Mars Orbit	4020	8860										

### **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 4.1.2 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 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 command line parameter or as part of the single file definition.

### Example

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

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

### 5.3.7 Generated Outcome

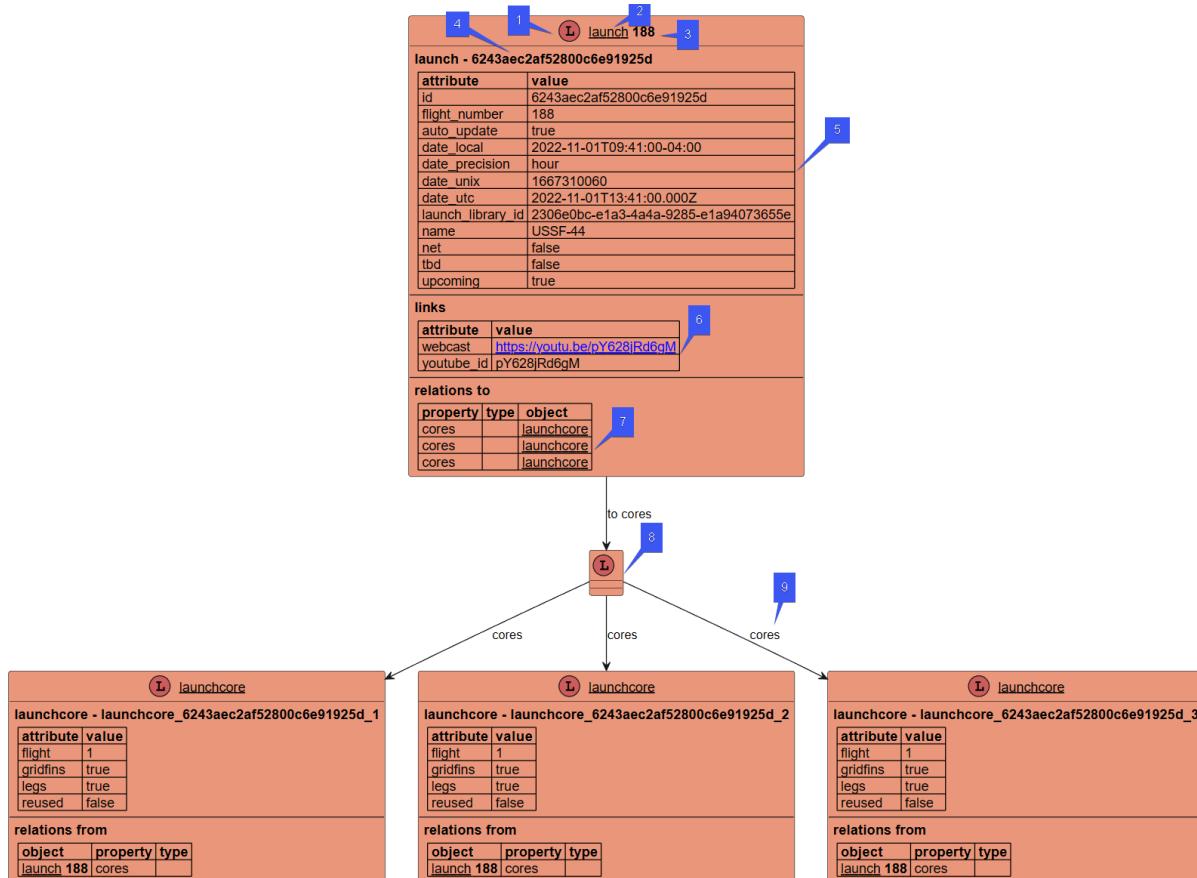


Figure 29 Generated Output Example

1. Object Icon  
Color is based on 6.6.13.2 iconColor
2. Object Type  
Based on 6.6.8.11 objectTypeProperties, 6.6.8.10 objectProperties
3. Object Title  
Based on 6.6.8.14 objectTitleProperties
4. Object Ident  
Based on 6.6.8.13 objectIdentifierProperties
5. List of Object Attributes  
Based on 6.6.8.9 attributeProperties
6. Characteristics  
Based on 6.6.8.20 characteristicProperties
7. From / To Relationship Lists
8. Group Box  
Based on 6.6.8.19 groupProperties

## 9. Relationship / Type

Based on 6.6.8.16 relationshipProperties and 6.6.8.17  
relationshipTypeProperties

## 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 5.3.6 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 5.2 Curl pre-processor ).

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.

#### 5.4.3.1 Generate Summary

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

See 6.2.2.9 Generate Summary

#### 5.4.3.2 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.

See 6.2.2.10 Generate Details

## 5.5 Generated Files

All files generated by the system will be stored based on the following pattern:

```
<baseOutputPath>\<outputPath>\  
<filename><curlFileSuffix><outputSuffix>.<Extension>
```

The `<curlFileSuffix>` is only used when a curl result file will be generated.

### 5.5.1 Variable replacement

The output path and output suffix configuration can be configured using a set of dynamic variables which allows to structure the generated directories and filenames based on additional parameters or file details.

Each found replace string will be replaced by the corresponding variable contents.

Possible Parameters:

- `<file>`  
will be replaced with the value of the `file` variable  
When the `curlFileSuffix` parameter is used, then this will be not included into the replaced value, only filename coming from the `inputFile` parameter will be used.  
(See 6.2.2.1 File)
- `<job>`  
will be replaced with the value of the `job` variable  
(See 6.2.2.2 Job)
- `<group>`  
will be replaced with the value of the `group` variable  
(See 6.2.2.3 Group)
- `<detail>`  
will be replaced with the value of the `detail` variable  
(See 6.2.2.4 Detail)
- `<option>`  
will be replaced with the value of the `option` variable  
(See 6.2.1.2 Option)

Additionally, it is possible to use the curl parameters which are known at this point in time.

The string replacements are supported for the following configuration parts:

- `outputPath` (5.5.3 Output path)
- `outputSuffix` (5.5.4 Output suffix)
- `summaryFileName` (5.5.5 Summary file)

Example:

Configuration	<pre>"group" : "\${launch}" "outputPath" : "&lt;job&gt;\&lt;group&gt;\&lt;file&gt;" "outputSuffix" : ".&lt;option&gt;" "curlFileSuffix" : ".\${rocketid}"</pre>				
Input	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td><code>baseOutputPath</code></td> <td>C:\data\output\</td> </tr> </tbody> </table>	Parameter	Value	<code>baseOutputPath</code>	C:\data\output\
Parameter	Value				
<code>baseOutputPath</code>	C:\data\output\				

	<b>inputFile</b>	rocket.json
	<b> \${rocketid}</b>	4711
	<b>inputListFile</b>	spacex_inputlist_launches.json
	<b> \${launch}</b>	latest
	<b>option</b>	Full
<b>Variables</b>	<b>Parameter</b>	<b>Value</b>
	<b>group</b>	latest
	<b>outputpath</b>	spacex_inputlist_launches\latest
	<b>outputSuffix</b>	.full
<b>Generated File</b>	C:\data\output\spacex_inputlist_launches\latest\rocket.4711.full.png	

### 5.5.2 Base Output path

The base output path is defined as variable `baseOutputPath` (See 6.2.2.6 Base output path).

Based on the variable definition the value can be based on command line parameters or global configurations. If none of these is defined the base output path will be calculated based on the path of the parameter or input files.

All files generated in one run will be generated in this base output path.

### 5.5.3 Output path

The base output path is defined as variable `outputPath` (See 6.2.2.7 Output path).

Using the parameter “`outputPath`” it is possible to structure the generated files based on a various set of variables.

The output path can be defined absolute or relative.

- An absolute path will be used without any changes
- A relative path will be added to the `baseOutputPath` variable.

#### Examples

<b>baseOutputPath</b>	<b>outPutPath</b>	<b>Result</b>
C:\data\json	output	C:\data\json\output
C:\data\json	D:\myoutput	D:\myoutput

The generator tries to create the output path if it does not exist. If this fails an error will be raised.

### 5.5.4 Output suffix

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

The generated output suffix is based on the variable output suffix (See 6.2.2.5 Output suffix)

The output suffix allows the similar replacement like the output path using `<job>`, `<group>`, `<option>` and `<detail>`.

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.

### 5.5.5 Summary file

A summary file will be generated when more than one json files will be handled and the `generateSummary` variable (See 6.2.2.9 Generate Summary) is active.

This summary file name will be calculated based on `summaryfilename` variable (See 6.2.2.8 Summary filename).

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

### 5.5.6 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 were generated with the current `json2puml` execution.

### 5.5.7 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`”.

### 5.5.8 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`”.

The defined output option will be based on the variable 6.2.2.11 Output formats.

# 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)

See 6.3 Global configuration file

### 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 pre-processor without modifying the parameter file and having for every call separate curl parameter files.

See 6.7 Curl parameter file

### 6.1.7 Curl authentication file

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

See 6.8 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 *jobDescription*

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

Rank	Place	Detail
1	Command line	<code>/jobdescription</code>
2	Parameter file	<code>{jobdescription}</code>
3	Input list file	<code>{jobdescription}</code>

#### 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	<code>{option name}</code>
2	Format option file	Constant “single”
3	Command line	<code>/option</code>
4	Parameter file	<code>{option}</code>
5	Input list file	<code>{option}</code>
6	Converter definition file	<code>{defaultOption}</code>

## 6.2.2 Generated folder and file names

### 6.2.2.1 File

The variable can be used in the `output path` and `output suffix` variables as a replace variable `<file>` of the current handled input file. The filename will be returned without folder names and the file extention.

Rank	Place	Detail
1	Command line	/inputfile
2	Parameter file	Inputfiles.inputfile
3	Input list file	Input.inputFile

### 6.2.2.2 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
2	Parameter file	{job}
3	Input list file	{job}
4	Input list file	<Name of input list file>

### 6.2.2.3 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	Input list file	{group}

### 6.2.2.4 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}

### 6.2.2.5 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}

#### 6.2.2.6 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
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.

#### 6.2.2.7 Output path

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

#### 6.2.2.8 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"

#### 6.2.2.9 Generate Summary

The variable defines if a summary of all input files should be generated into one combined file.

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

Rank	Place	Detail
4	Default	When generateSummary and generateDetails are not defined then generateSummary is true when the number of input files is greater than one.

#### 6.2.2.10 Generate Details

The variable defines if for every input file a separate result file should be generated.

Rank	Place	Detail
1	Command line	/generatedetails
2	Parameter file	{generateDetails}
3	Input list file	{generateDetails}
4	Default	When generateSummary and generateDetails are not defined then generateDetails is true when the number of input files is one.

#### 6.2.2.11 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}

The outputFormats can be defined as a comma separated list or as an array of strings.

The following format names are supported:

Format	Description	PlantUML parameter
<b>png</b>	Generates a png graphic file	-png
<b>svg</b>	Generates a svg graphic file	-svg
<b>pdf</b>	Generates a pdf file containing the graphic	-pdf
<b>puml</b>	Adds the generated PlantUml script file to the zip file.	
<b>json</b>	Adds the json source file (when generated / fetched via curl) to the zip file.	
<b>log</b>	Adds the converter log file to the zip file.	
<b>zip</b>	Generates a zip file as a summary containing all generated files	
<b>filelist</b>	Generates a json file containing the list of generated files. When the source file is curl based the curl command will also be added to the list file.	

	The list file will be added to the zip file.	
--	----------------------------------------------	--

### 6.2.3 Configuration files

#### 6.2.3.1 Configuration filename

The variable defines the name of the global configuration file (See 6.1.1 Global configuration file ).

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 6.1.7 Curl authentication file).

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 6.1.3 Input list file).

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

#### 6.2.3.4 Converter definition filename

The variable defines the name of the converter definition file (See 6.1.4 Converter definition file).

Rank	Place	Detail
1	Command line	/definitionfile
2	Parameter file	{definitionfile}
3	Input list file	{definitionfile}
4	Global configuration file	{defaultDefinitionFileName}

## 6.2.4 Curl Configuration

### 6.2.4.1 Curl Span Id Header

The variable defines the request header variable sends with a unique span id. When this variable is defined a UUID based values is generated and added as response header.

Rank	Place	Detail
1	Input list file	{curlSpanIdHeader}
2	Global configuration file	{curlSpanIdHeader}

### 6.2.4.2 Curl Trace Id Header

The variable defines the request header variable sends with a unique trace id. When this variable is defined a UUID based values is generated and added as response header.

Rank	Place	Detail
1	Input list file	{curlTraceIdHeader}
2	Global configuration file	{curlTraceIdHeader}

### 6.2.4.3 Curl User Agent Information

The variable defines with which information the curl command will be called for the –user-agent parameter.

Rank	Place	Detail
1	Input list file	{curlUserAgentInformation}
2	Global configuration file	{curlUserAgentInformation}
3	Default value	json2puml <applicationtype> v<application version> - <input list file name>

## 6.2.5 System Environment

### 6.2.5.1 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

Rank	Place	Detail
2	Global configuration file	{plantumlJarFile}
3	Environment	%PlantUmlJarFile%

### 6.2.5.2 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

GlobalConfiguration
<pre>additionalServiceInformation : String baseOutputPath : String outputPath : String curlAuthenticationFileName : String curlPassThroughHeader : Array of String curlSpanIdHeader : String curlTraceIdHeader : String curlUserAgentInformation : String defaultDefinitionFileName : String definitionFileSearchFolder : Array of String inputListFileSearchFolder : Array of String javaRuntimeParameter : String logFilePath : String plantUmlJarFileName : String servicePort : Integer</pre>

Figure 30 Data model global configuration file

### 6.3.2 Global Configuration

#### 6.3.2.1 additionalServiceInformation

Name	Cardinality	Datatype	Default
additionalServiceInformation	0..1	String	
Description			

Name	Cardinality	Datatype	Default
<p>This property defines the content of the additionalServiceInformation attribute in the <code>get /serviceinformation</code> api operation.</p>			
<p>The property can also be defined using environment variables in the curl variable format.</p>			
<p><b>6.3.2.2 (See 4.2.4.1 <code>get /heartbeat</code>)</b></p>			
<p>This operations returns generic information's about the service and how it is configured.</p>			
<p><b>6.3.2.2.1 Sample curl command</b></p>			
<pre>curl -X GET http://localhost:9090/api/heartbeat</pre>			
<p><b>6.3.2.2.2 Sample result data model</b></p>			
<p>Result is a text showing the version of the service and further details.</p>			
<pre>json2puml server v2.2.1.85 (Windows 64Bit x86 Debug)</pre>			
<p><code>get /serviceinformation</code>)</p>			

### 6.3.2.3 `baseOutputPath`

Name	Cardinality	Datatype	Default
baseOutputPath	0..1	String	
<p><b>Description</b></p>			
<p>This defines the base directory in the operating system where the output files should be generated.</p>			
<p>When this parameter is not defined, the base output path will be calculated based on the current parameter or input list file.</p>			
<p>(See 6.2.2.6 Base output path and 5.5.2 Base Output path)</p>			

### 6.3.2.4 `outputPath`

Name	Cardinality	Datatype	Default
outputPath	0..1	String	
<p><b>Description</b></p>			
<p>This defines the default outputPath variable.</p>			

Name	Cardinality	Datatype	Default
(See 5.5.3 Output path and 6.2.2.7 Output path)			

### 6.3.2.5 curlAuthenticationFile

Name	Cardinality	Datatype	Default
curlAuthenticationFile	0..1	String	
<b>Description</b>			
This defines the path to the curlAuthenticationFile variable.  (See 6.2.3.2 Curl Authentication filename and 6.8 Curl authentication file)			

### 6.3.2.6 curlPassThroughHeader

Name	Cardinality	Datatype	Default
curlPassThroughHeader	0..*	String	
<b>Description</b>			
This parameter is only valid for the service applications. It defines a list of http-headers which are read from the incoming service call and handed over to the outgoing service calls. This could be used for example to transport trace-id's following the open telemetry standard.			
<b>Example:</b>			
<pre>"curlPassThroughHeader": ["traceid", "x-b3-traceid"],</pre>			
(See 5.2.2 Curl command)			

### 6.3.2.7 curlSpanIdHeader

Name	Cardinality	Datatype	Default
curlSpanIdHeader	0..1	String	
<b>Description</b>			
This parameter defines a http request header for which a UUID based span id (following the open telemetry standard) will be defined and send via the curl command.			
(See 5.2.2 Curl command and 6.2.4.1 Curl Span Id Header)			

### 6.3.2.8 curlTraceIdHeader

Name	Cardinality	Datatype	Default
curlTraceIdHeader	0..1	String	
<b>Description</b>			
<p>This parameter defines a http request header for which a UUID based trace id (following the open telemetry standard) will be defined and send via the curl command.</p> <p>When the header is one of the <code>curlPassThroughHeader</code> the new trace id will only be generated when no corresponding header has been sent to the json2puml service application.</p> <p>(See 5.2.2 Curl command and 6.2.4.2 Curl Trace Id Header)</p>			

### 6.3.2.9 curlUserAgentInformation

Name	Cardinality	Datatype	Default
curlUserAgentInformation	0..*	String	
<b>Description</b>			
<p>This parameter defines the information which is handed off to the <code>-user-agent</code> parameter of the curl commands.</p> <p>When the parameter is not defined a default value “<code>json2puml &lt;application type&gt; v&lt;version&gt; - &lt;input list file name&gt;</code>” will be used.</p> <p><b>Example:</b>  <code>json2puml service v2.1.4.79 - placeholder_inputlist_curl.json</code></p> <p>(See 5.2.2 Curl command and 6.2.4.3 Curl User Agent Information))</p>			

### 6.3.2.11 defaultDefinitionFileName

Name	Cardinality	Datatype	Default
defaultDefinitionFileName	0..1	String	
<b>Description</b>			
<p>The parameter defines a default value for the converter definition file variable.</p> <p>(See 6.1.4 Converter definition file and 6.2.3.4 Converter definition filename)</p>			

### 6.3.2.12 definitionFileSearchFolder

Name	Cardinality	Datatype	Default
definitionFileSearchFolder	0..*	String	

Name	Cardinality	Datatype	Default
<b>Description</b>			
This parameter is used in two different scenarios:			
1. converter definition filename variable: This parameter defines a list of folders for searching the defined definition file. The parameter is used when the input file is not found base on the defined filename variable. Then all folders defined in this parameter are checked if the file can be found. This allows to define a definition file only by it's name without knowing in which exact folder it is placed.			
2. Support the <code>get /definitionfilelist</code> operation of the service application The resultset of this operation is based on all files found in the configured folders and filter conditions.			
<b>Example:</b>			
<pre>"definitionFileSearchFolder": [     "e:\\json2puml\\definition\\",     "e:\\json2puml\\samples\\tmf\\*definition*.json",     "e:\\json2puml\\samples\\jsonplaceholder\\*definition*.json",     "e:\\json2puml\\samples\\swapi\\*definition*.json",     "e:\\json2puml\\samples\\tvmaze\\*definition*.json" ]</pre>			
(See 6.1.4 Converter definition file and 6.2.3.4 Converter definition filename)			

### 6.3.2.13 *inputListFileSearchFolder*

Name	Cardinality	Datatype	Default
inputListFileSearchFolder	0..*	String	
<b>Description</b>			
This parameter is used in two different scenarios:			
1. input list filename variable: This parameter defines a list of folders for searching the defined definition file. The parameter is used when the input file is not found base on the defined filename variable. Then all folders defined in this parameter are checked if the file can be found. This allows to define a definition file only by it's name without knowing in which exact folder it is placed.			
2. Support the <code>get /inputlistfilelist</code> operation of the service application The resultset of this operation is based on all files found in the configured folders and filter conditions.			
<b>Example:</b>			

Name	Cardinality	Datatype	Default
<pre>"inputListFileSearchFolder": [     "e:\\\\ json2puml\\\\samples\\\\curl\\\\*inputlist*.json",     "e:\\\\ json2puml\\\\samples\\\\jsonplaceholder\\\\*inputlist*.json",     "e:\\\\ json2puml\\\\samples\\\\swapi\\\\*inputlist*.json",     "e:\\\\ json2puml\\\\samples\\\\tvmaze\\\\*inputlist*.json"]</pre>			
(See 6.1.3 Input list file and 6.2.3.3 Input list filename)			

### 6.3.2.14 *javaRuntimeParameter*

Name	Cardinality	Datatype	Default
javaRuntimeParameter	0..1	String	
Description			
This parameter defines the <code>javaRuntimeParameter</code> variable.			
(See 6.2.5.2 Java runtime parameter)			

### 6.3.2.15 *logFilePathOutputPath*

Name	Cardinality	Datatype	Default
logFilePathOutputPath	0..1	String	
Description			
This parameter defines the folder where the global log files will be generated.			

### 6.3.2.16 *plantUmlJarFileName*

Name	Cardinality	Datatype	Default
plantUmlJarFileName	0..1	String	
Description			
This parameter defines the <code>plantumlJarFileName</code> variable.			
(See 6.2.5.1 PlantUML jar filename)			

### 6.3.2.17 *servicePort*

Name	Cardinality	Datatype	Default
servicePort	0..1	Integer	
Description			
This parameter defines http port on which the microservice is listening for requests. This parameter is used only for the service applications.			

Name	Cardinality	Datatype	Default
The parameter can also be defined as environment variable using the curl variable format.			
<u>Example:</u> <pre>"servicePort": "\${JSON2PUML_SERVICEPORT}"</pre>			

## 6.4 Parameter file

The parameter file allows to put most of the command line parameters into one configuration file. This allows to simplify and standardise the execution of converter runs (See 4.1.2 Command Line Parameters).

Additionally the parameter file can be used as input for the calls to the service application using curl (See 4.2 Service Application)

### 6.4.1 Data model

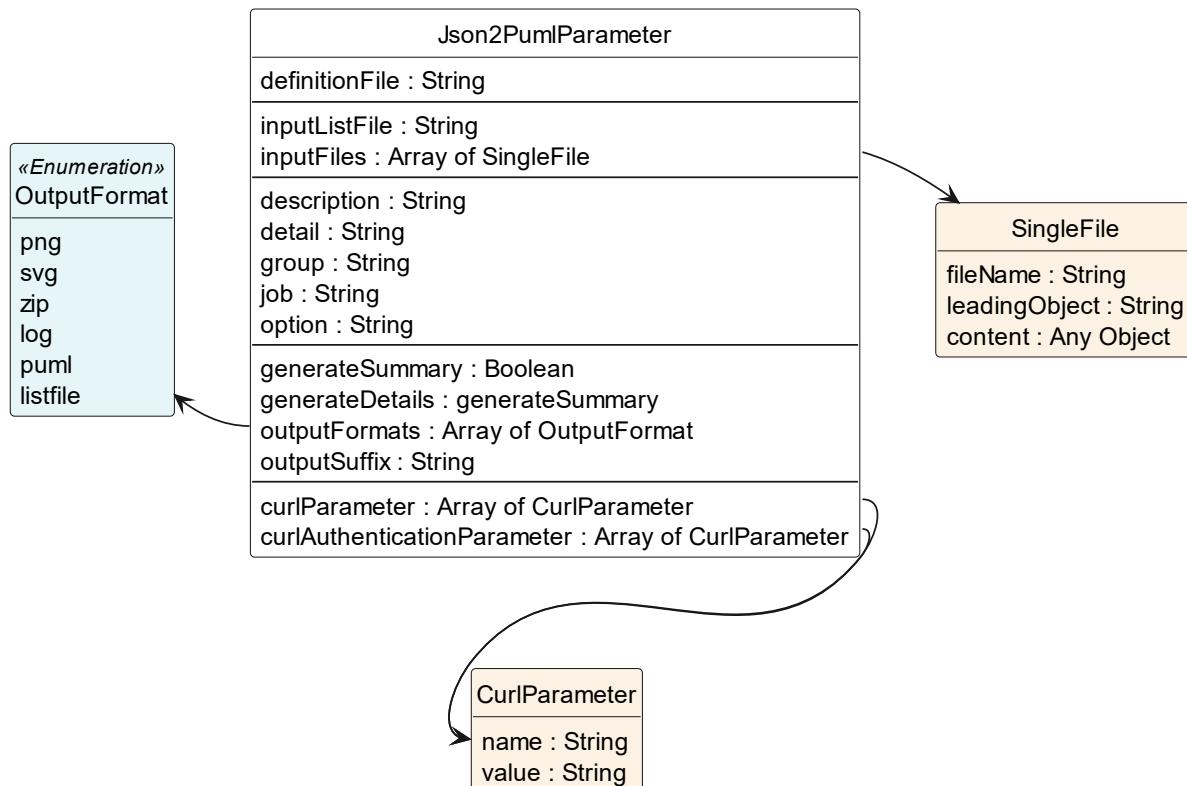


Figure 31 Data model parameter file

### 6.4.2 Json2PumlParameter

#### 6.4.2.1 `definitionFile`

Name	Cardinality	Datatype	Default
<code>definitionFile</code>	0..1	String	
<b>Description</b>			

Name	Cardinality	Datatype	Default
<p>The parameter defines the converter definition file variable.            The converter definition file will be used by the json2puml converter and contains the mappings for the input files.            (See 6.1.4 Converter definition file and 6.2.3.4 Converter definition filename)</p>			

#### 6.4.2.2 group

Name	Cardinality	Datatype	Default
group	0..1	String	
Description			
<p>The parameter defines the group variable.            The group variable can be used for a customized generation of output folders and filenames.            (See 6.2.2.3 Group and 5.5 Generated Files)</p>			

#### 6.4.2.3 detail

Name	Cardinality	Datatype	Default
detail	0..1	String	
Description			
<p>The parameter defines the detail variable.            The detail variable can be used for a customized generation of output folders and filenames.            (See 6.2.2.4 Detail and 5.5 Generated Files)</p>			

#### 6.4.2.4 job

Name	Cardinality	Datatype	Default
job	0..1	String	
Description			
<p>The parameter defines the job variable.            The job variable can be used for a customized generation of output folders and filenames.            (See 6.2.2.2 Job and 5.5 Generated Files)</p>			

#### 6.4.2.5 jobDescription

Name	Cardinality	Datatype	Default
jobDescription	0..1	String	
Description			

Name	Cardinality	Datatype	Default
<p>The parameter defines the description variable.            The description variable is added to the legend of the generated image and allows to add an additional description to the image.            (See 6.2.1.1 jobDescription)</p>			

#### 6.4.2.6 *option*

Name	Cardinality	Datatype	Default
option	0..1	String	
<p>Description</p> <p>The parameter defines the option variable.            The option variable is used to define which converter definition is used to convert the input files.            (See 6.2.1.2 Option and 6.6.2 Identifying an Option )</p>			

#### 6.4.2.7 *inputListFile*

Name	Cardinality	Datatype	Default
inputListFile	0..1	String	
<p>Description</p> <p>The parameter defines the input list file variable.            The input list file variable defines which input list file is used to identify the files to be converted.            (See 6.1.3 Input list file and 6.2.3.3 Input list filename)</p>			

#### 6.4.2.8 *inputFiles*

Name	Cardinality	Datatype	Default
inputListFile	0..*	SingleFile	
<p>Description</p> <p>The parameter allows to define a fixed list of files / file contents to be converted.            Each record in this will be seen as a single file definition.            (See 6.4.3 Single File)</p>			

#### 6.4.2.9 *generateSummary*

Name	Cardinality	Datatype	Default
generateSummary	0..1	Boolean	
<p>Description</p>			

Name	Cardinality	Datatype	Default
This flag defines the generateSummary variable and is used to define if a combined summary file for all input files should be generated or not. (See 6.2.2.8 Summary filename)			

#### 6.4.2.10 *generateDetails*

Name	Cardinality	Datatype	Default
generateDetails	0..1	Boolean	
Description			
This flag defines the generateDetails variable and is used to define if for every input file a single result file should be generated or not. (See 6.2.2.10 Generate Details)			

#### 6.4.2.11 *outputFormats*

Name	Cardinality	Datatype	Default
outputFormats	0..*	String	
Description			
This parameter defines the outputFormats variable. The outputFormats variable determines which files should be generated. (See 6.2.2.11 Output formats)			

#### 6.4.2.12 *outputSuffix*

Name	Cardinality	Datatype	Default
outputSuffix	0..1	String	
Description			
This parameter defines the outputSuffix variable. The outputSuffix variable is used to add dynamic suffix to any generated output file. This allows for example to add the converter option to the files to generate the files with different options without overwriting the files. (See 6.2.2.5 Output suffix and 5.5.4 Output suffix)			

#### 6.4.2.13 *curlParameter*

Name	Cardinality	Datatype	Default
curlParameter	0..*	CurlParameter	
Description			
This parameter allows to define curl parameters for configuring the curl pre-processor.			

Name	Cardinality	Datatype	Default
(See 5.2 Curl pre-processor)			

#### 6.4.2.14 curlAuthenticationParameter

Name	Cardinality	Datatype	Default
curlAuthenticationParameter	0..*	CurlParameter	
Description			
This parameter allows to define curl authentication parameters for configuring the curl pre-processor. (See 5.2 Curl pre-processor)			

### 6.4.3 Single File

The single file object defines a single file to be converted.

There are two ways to define the file:

- Using a file existing on the local filesystem
- Defining the json data of a file as content.

When the content parameter contains a valid json structure this data is used as file content and the filename parameter will be used to determine the name of the file which then will be used for the generation of the output files.

The content parameter is mainly defined to support the microservice based application where already existing files needs to be transferred to the json2puml server (See 4.2 Service Application).

When no valid json content is defined the file defined in the filename parameter must be existing on the local filesystem, otherwise the file will be ignored.

#### 6.4.3.1 fileName

Name	Cardinality	Datatype	Default
fileName	1	String	
Description			
This parameter defines the filename of the file			

#### 6.4.3.2 leadingObject

Name	Cardinality	Datatype	Default
leadingObject	0..1	String	
Description			
This parameter defines the object type of the top level objects of this file.			

Name	Cardinality	Datatype	Default
(See 5.3.6 leadingObject)			

#### 6.4.3.3 content

Name	Cardinality	Datatype	Default
Content	0..1	AnyObject	
<b>Description</b>			
The content parameter can be any valid JSON structure which then will be used as the content of the defined file.			

#### 6.4.4 Curl Parameter

The curl parameter will be used by the curl pre-processor to fetching data based on the defined curl parameters (See 5.2 Curl pre-processor).

The curl parameter can be defined in three different formats:

1. String  
“<parameter name>=<parameter value>”
2. Simple json record  
{ “<parameter name>”: “<parameter value>” }
3. Full json record  
{ “name”: “<parameter name>”,  
“value”: “<parameter value>” }

##### 6.4.4.1 name

Name	Cardinality	Datatype	Default
name	1	String	
<b>Description</b>			
This parameter defines the name of the of the curl parameter. Curl parameter names are defined using the following pattern: \${<name>} . In the configuration they can defined only using the name or using the full pattern.			

##### 6.4.4.2 value

Name	Cardinality	Datatype	Default
Value	0..1	String	
<b>Description</b>			
This parameter defines the value of the curl parameter which is replacing the parameter name in the curl commands.			

Name	Cardinality	Datatype	Default
When the value contains \${<name>} strings, these strings will be replaced with operating system environment variables based on <name>.			

## 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 5.2 Curl pre-processor.

### 6.5.1 Data model

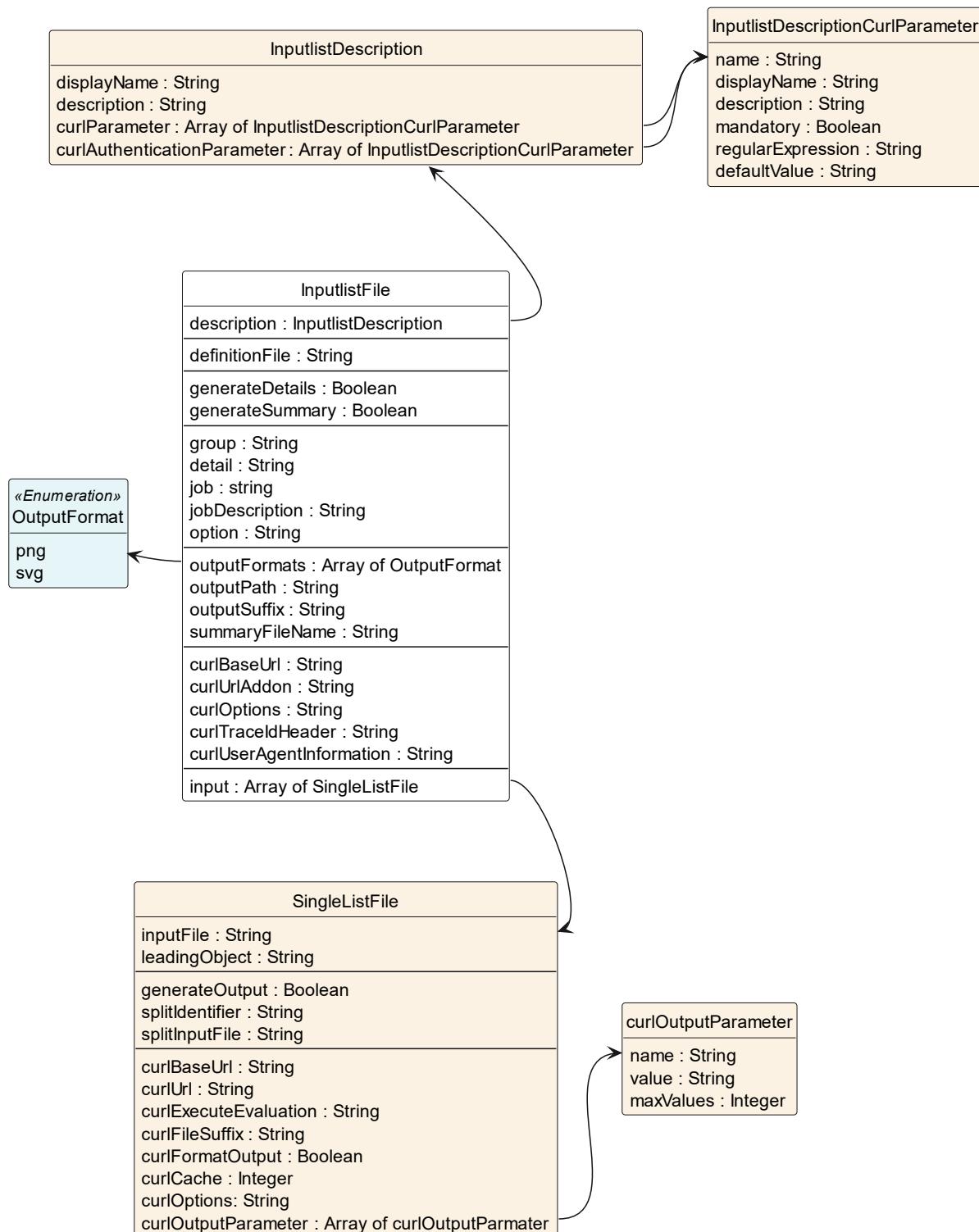


Figure 32: Data model input list file

## 6.5.2 InputListFile

### 6.5.2.1 description

Name	Cardinality	Datatype	Default
Description	0..1	InputListDescription	
<b>Description</b>			
Object to describe the purpose and use of this input list file. This description is used in <code>get /inputlistfiles</code> operation (See 4.2.4.4 <code>get /inputlistfiles</code> ) of the service application. (See 6.5.3 InputListDescription)			

### 6.5.2.2 generateSummary

Name	Cardinality	Datatype	Default
generateSummary	0..1	Boolean	
<b>Description</b>			
Flag to define if a combined summary file of all input files should be generated. (See 6.2.2.8 Summary filename)			

### 6.5.2.3 definitionFile

Name	Cardinality	Datatype	Default
definitionFile	0..1	String	
<b>Description</b>			
This property defines the definition file variable. It allows to define a default converter definition file to be used in this input list file. (See 6.2.3.4 Converter definition filename)			

### 6.5.2.4 generateDetails

Name	Cardinality	Datatype	Default
generateDetails	0..1	Boolean	
<b>Description</b>			
Flag to define if the generator should be executed for every single input file. (See 6.2.2.10 Generate Details)			

### 6.5.2.5 group

Name	Cardinality	Datatype	Default
group	0..1	String	

Name	Cardinality	Datatype	Default
<b>Description</b>			
<p>The parameter defines the group variable.            The group variable can be used for a customized generation of output folders and filenames.            (See 6.2.2.3 Group and 5.5 Generated Files)</p>			

#### 6.5.2.6 *detail*

Name	Cardinality	Datatype	Default
detail	0..1	String	
<b>Description</b>			
<p>The parameter defines the detail variable.            The detail variable can be used for a customized generation of output folders and filenames.            (See 6.2.2.4 Detail and 5.5 Generated Files)</p>			

#### 6.5.2.7 *job*

Name	Cardinality	Datatype	Default
job	0..1	String	
<b>Description</b>			
<p>The parameter defines the job variable.            The job variable can be used for a customized generation of output folders and filenames.            (See 6.2.2.1 File and 5.5 Generated Files)</p>			

#### 6.5.2.8 *jobDescription*

Name	Cardinality	Datatype	Default
jobDescription	0..1	String	
<b>Description</b>			
<p>The parameter defines the description variable.            The description variable is added to the legend of the generated image and allows to add an additional description to the image.            (See 6.2.1.1 jobDescription)</p>			

#### 6.5.2.9 *option*

Name	Cardinality	Datatype	Default
option	0..1	String	

Name	Cardinality	Datatype	Default
<b>Description</b>			
<p>The parameter defines the option variable.            The option variable is used to define which converter definition is used to convert the input files.            (See 6.2.1.2 Option and 6.6.2 Identifying an Option )</p>			

#### 6.5.2.10 *outputPath*

Name	Cardinality	Datatype	Default
outputPath	0..1	String	
<b>Description</b>			
<p>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:</p> <ul style="list-style-type: none"> <li>- &lt;option&gt; Replaced by the name of the use configuration option</li> <li>- &lt;group&gt; Replaced by the corresponding value from the list definition file (See 6.5.2 InputListFile) or the corresponding command line parameter /group (See 4.1.2 Command Line Parameters)</li> <li>- &lt;detail&gt; Replaced by the corresponding value from the list definition file (See 6.5.2 InputListFile) or the corresponding command line parameter /detail (See 4.1.2 Command Line Parameters)</li> </ul> <p><b>Example:</b>  <code>"outputPath": "output\\&lt;group&gt;\\&lt;option&gt;"</code></p>			

#### 6.5.2.11 *outputFormats*

Name	Cardinality	Datatype	Default
outputFormats	0..*	OutputFormat	
<b>Description</b>			
<p>This parameter defines the outputFormats variable.            The outputFormats variable determines which files should be generated.            (See 6.2.2.11 Output formats)</p>			

### 6.5.2.12 *outputSuffix*

Name	Cardinality	Datatype	Default
outputSuffix	0..1	String	
<b>Description</b>			
<p>This parameter defines the outputSuffix variable.            The outputSuffix variable is used to add dynamic suffix to any generated output file.            This allows for example to add the converter option to the files to generate the files with different options without overwriting the files.            (See 6.2.2.5 Output suffix and 5.5.4 Output suffix)</p>			

### 6.5.2.13 *summaryFileName*

Name	Cardinality	Datatype	Default
summaryFileName	0..1	String	summary.json
<b>Description</b>			
<p>The parameter defines the summary file variable.            Based on the name of generated summary will be build.            (See 6.2.2.8 Summary filename and 5.5.5 Summary file)</p>			

### 6.5.2.14 *curlBaseUrl*

Name	Cardinality	Datatype	Default
curlBaseUrl	0..1	String	
<b>Description</b>			
<p>Base URL which will be added before all curlUrl parameters (6.5.5.7 curlUrl) 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 (6.5.5.6 curlBaseUrl)            (See 5.2 Curl pre-processor)</p>			

### 6.5.2.15 *curlUrlAddon*

Name	Cardinality	Datatype	Default
curlUrlAddon	0..1	String	
<b>Description</b>			
<p>Additional URL part which will be added after all curlUrl parameters (6.5.5.7 curlUrl) 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 5.2 Curl pre-processor)</p>			

#### 6.5.2.16 curlOptions

Name	Cardinality	Datatype	Default
curlOptions	0..1	String	
<b>Description</b>			
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 5.2 Curl pre-processor)			

#### 6.5.2.17 curlSpanIdHeader

Name	Cardinality	Datatype	Default
curlSpanIdHeader	0..1	String	
<b>Description</b>			
This parameter defines a http request header for which a UUID based span id (following the open telemetry standard) will be defined and send via the curl command. (See 5.2.2 Curl command and 6.2.4.1 Curl Span Id Header)			

#### 6.5.2.18 curlTraceIdHeader

Name	Cardinality	Datatype	Default
curlTraceIdHeader	0..1	String	
<b>Description</b>			
This parameter defines a http request header for which a UUID based trace id (following the open telemetry standard) will be defined and send via the curl command. When the header is one of the <code>curlPassThroughHeader</code> the new trace id will only be generated when no corresponding header has been sent to the json2puml service application.			

#### 6.5.2.19 (See 5.2.2 Curl command and 6.2.4.1 Curl Span Id Header)

The variable defines the request header variable sends with a unique span id. When this variable is defined a UUID based values is generated and added as response header.

Rank	Place	Detail
1	Input list file	{curlSpanIdHeader}
2	Global configuration file	{curlSpanIdHeader}

Name	Cardinality	Datatype	Default
Curl Trace Id Header)			

#### 6.5.2.20 curlUserAgentInformation

Name	Cardinality	Datatype	Default
curlPassThroughHeader	0..*	String	
Description			

This parameter defines the information which is handed off to the –user-agent parameter of the curl commands.

When the parameter is not defined a default value “json2puml <application type> v<version> - <input list file name>” will be used.

**Example:**

```
json2puml service v2.1.4.79 - placeholder_inputlist_curl.json
```

#### 6.5.2.21 (See 5.2.2 Curl command and 6.2.4.1 Curl Span Id Header

The variable defines the request header variable sends with a unique span id. When this variable is defined a UUID based values is generated and added as response header.

Rank	Place	Detail
1	Input list file	{curlSpanIdHeader}
2	Global configuration file	{curlSpanIdHeader}

#### 6.5.2.22 Curl Trace Id Header

The variable defines the request header variable sends with a unique trace id. When this variable is defined a UUID based values is generated and added as response header.

Rank	Place	Detail
1	Input list file	{curlTraceIdHeader}
2	Global configuration file	{curlTraceIdHeader}

Curl User Agent Information)

### 6.5.2.23 *Input*

Name	Cardinality	Datatype	Default
Input	1..*	SingleListFile	
Description			
List of file definitions to be converted. (See 6.5.3 SingleListFile)			

### 6.5.3 InputListDescription

The description object is used by the `get /inputlistfiles` operation of the service application. It allows to give a description of the input list file and how it could be used and parametrized.

#### 6.5.3.1 *displayName*

Name	Cardinality	Datatype	Default
displayName	0..1	string	
Description			
Information to be displayed instead of the filename of the input list file. This allows to give a more descriptive name to be shown.			

#### 6.5.3.2 *description*

Name	Cardinality	Datatype	Default
Description	0..1	string	
Description			
Additional description for the current input list file. This could be shown additionally to help a user understanding the purpose of the file (e.g. in a Mouse Over Notification)			

#### 6.5.3.3 *curlParameter*

Name	Cardinality	Datatype	Default
curlParameter	0..*	InputListDescriptionCurlParameter	
Description			
This parameter explains which curl parameter are support or mandatory when working with the input list file. (See 6.5.4 InputListDescriptionCurlParameter)			

#### 6.5.3.4 curlAuthenticationParameter

Name	Cardinality	Datatype	Default
curlAuthenticationParameter	0..*	InputListDescriptionCurlParameter	
<b>Description</b>			
This parameter explains which curl authentication parameter are support or mandatory when working with the input list file. (See 6.5.4 InputListDescriptionCurlParameter)			

#### 6.5.4 InputListDescriptionCurlParameter

This object is used to describe curl (authentication) parameters which can be used or a mandatory when working with the current input list file.

##### 6.5.4.1 name

Name	Cardinality	Datatype	Default
name	1	String	
<b>Description</b>			
This parameter defines the name of the of the curl parameter. Curl parameter names are defined using the following pattern: \${<name>} . In the configuration they can defined only using the name or using the full pattern.			

##### 6.5.4.2 displayName

Name	Cardinality	Datatype	Default
displayName	0..1	String	
<b>Description</b>			
Information to be displayed instead of the name of the curl parameter. This allows to give a more descriptive name to be shown.			

##### 6.5.4.3 description

Name	Cardinality	Datatype	Default
Description	0..1	string	
<b>Description</b>			
Additional description for the current curl parameter. This could be shown additionally to help a user understanding the purpose of the parameter (e.g. in a Mouse Over Notification)			

#### 6.5.4.4 *mandatory*

Name	Cardinality	Datatype	Default
mandatory	0..1	Boolean	
Description			
Flag to define if the curl parameter is mandatory or not for the execution of the input list file.			

#### 6.5.4.5 *regularExpression*

Name	Cardinality	Datatype	Default
regularExpression	0..1	String	
Description			
The regular expression can be used to support a UI to restrict the input of the parameter values based on the regular expression (e.g. ensure that a customer number is only based on numbers and has exactly 10 digits).			

#### 6.5.4.6 *defaultValue*

Name	Cardinality	Datatype	Default
defaultValue	0..1	String	
Description			
When a default value is defined and the parameter is not defined at runtime the default value will be used when executing the script. The default value is only supported for the normal curl parameters (6.5.3.3 curlParameter) and not for the curl authentication parameters (6.5.3.4 curlAuthenticationParameter).			

### 6.5.5 SingleListFile

#### 6.5.5.1 *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.			

Name	Cardinality	Datatype	Default
See 0			
Curl			

#### 6.5.5.2 *leadingObject*

Name	Cardinality	Datatype	Default
leadingObject	0..1	string	
<b>Description</b>			
Name of the leadingObject to be used for all input files found based on the "inputFile" parameter.			
See 5.3.6 leadingObject			

#### 6.5.5.3 *generateOutput*

Name	Cardinality	Datatype	Default
generateOutput	0..1	Boolean	True
<b>Description</b>			
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			

#### 6.5.5.4 *splitIdentifier*

Name	Cardinality	Datatype	Default
splitIdentifier	0..1	String	
<b>Description</b>			
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 a JSON array the different values of this array will be concatenated to one identifier.			

Name	Cardinality	Datatype	Default
<u>Example:</u>			
<pre>{     "id": "875153820701008909",     "ticketType": [         {             "name": "Incident"         },         {             "name": "Admin"         }     ], }</pre>			
The parameter "splitIdentifier":"ticketType.name") would lead to an identifier "Incident.Admin".			

#### 6.5.5.5 splitInputFile

Name	Cardinality	Datatype	Default
splitInputFile	0..1	Boolean	
<b>Description</b>			
<p>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.</p> <p>This option is only valid when generatedetails (See 6.2.2.10 Generate Details) is activated.</p> <p>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.</p> <p>The generated filename of the splitted 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 6.5.5.4 splitIdentifier).</p>			

#### 6.5.5.6 curlBaseUrl

Name	Cardinality	Datatype	Default
curlBaseUrl	0..1	string	
<b>Description</b>			
<p>Base URL which will be added before the curlUrl parameter (6.5.5.7 curlUrl) of the inputfile to define the full URL.</p> <p>This parameter overwrites the curlBaseUrl parameter of the list filed (6.5.2.14 curlBaseUrl).</p> <p>(See 5.2 Curl pre-processor)</p>			

### 6.5.5.7 curlUrl

Name	Cardinality	Datatype	Default
curlUrl	0..1	string	
Description			
Url to fetch the json file from. (See 5.2 Curl pre-processor)			

### 6.5.5.8 curlExecuteEvaluation

Name	Cardinality	Datatype	Default
curlExecuteEvaluation	0..1	string	
Description			
The curlExecuteEvaluation allows to define a condition if the curl command should be executed or not.			
The content of the property is evaluated and when the result of this evaluation is true the curl command will be executed.			
Inside of the evaluation command curl parameters could be used to do the evaluation based on input parameters or of curl output parameters.			
This allows to prevent unneeded or unwanted curl commands to be executed.			
Examples:			
Evaluation	Description		
"\${core}" <> ""	The curl command will be executed when the variable \${core} is filled.		
("\${launch}" = "next") or ("\${launch}" = "latest")	The curl command will be executed when the variable \${launch} is filled with next or latest.		
(See 5.2 Curl pre-processor)			

### 6.5.5.9 curlFileSuffix

Name	Cardinality	Datatype	Default
curlFileSuffix	0..1	string	
Description			
The curlFileSuffix will be added to the filename of the generated outputfile. This allows to have more flexibility in generating the output files. The curlFileSuffix can contain curl parameter replacement to build the filenames based on the current parameter values (e.g. customer number which is the input criteria of the api). (See 5.2 Curl pre-processor)			

#### 6.5.5.10 curlFormatOutput

Name	Cardinality	Datatype	Default
curlFormatOutput	0..1	Boolean	False
<b>Description</b>			
Flag to define if the curl output file should be formatted. (See 5.2 Curl pre-processor)			

#### 6.5.5.11 curlCache

Name	Cardinality	Datatype	Default
curlCache	0..1	Integer	0
<b>Description</b>			
Number of seconds while the generated curl result file will be cached and not refreshed. (See 5.2 Curl pre-processor)			

#### 6.5.5.12 curlOptions

Name	Cardinality	Datatype	Default
curlOptions	0..1	string	
<b>Description</b>			
Additional parameters added to the curl command when fetching the json data. (See 5.2 Curl pre-processor)			

#### 6.5.5.13 curlOutputParameter

Name	Cardinality	Datatype	Default
curlOutputParameter	0..1	<anyRecord>	0
<b>Description</b>			
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.			
<u>Example output file:</u>			

Name	Cardinality	Datatype	Default
<pre>{   "id": 1,   "name": "Leanne Graham",   "email": "Sincere@april.biz",   "location": {     "zipcode": "92998-3874",     "geo": {       "lat": "-37.3159",       "lng": "81.1496"     }   } }</pre>			

Example curlOutputParameter:

<pre>{   "name": "name",   "geo_lat": "location.geo.lat",   "geo_lng": "location.geo.lng" }</pre>
---------------------------------------------------------------------------------------------------

This would lead to the following curl parameters:

Parameter	Value
<code> \${name}</code>	Leanne Graham
<code> \${geo_lat}</code>	-37.3159
<code> \${geo_lng}</code>	81.1496

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

(See 5.2 Curl pre-processor)

### 6.5.6 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 is 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 6.6.11 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 4.1.2 Command Line Parameters) is used first. If this parameter is not defined the parameter “defaultOption” from the configuration file is used (See 6.6.4.5 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 6.6.4.3 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

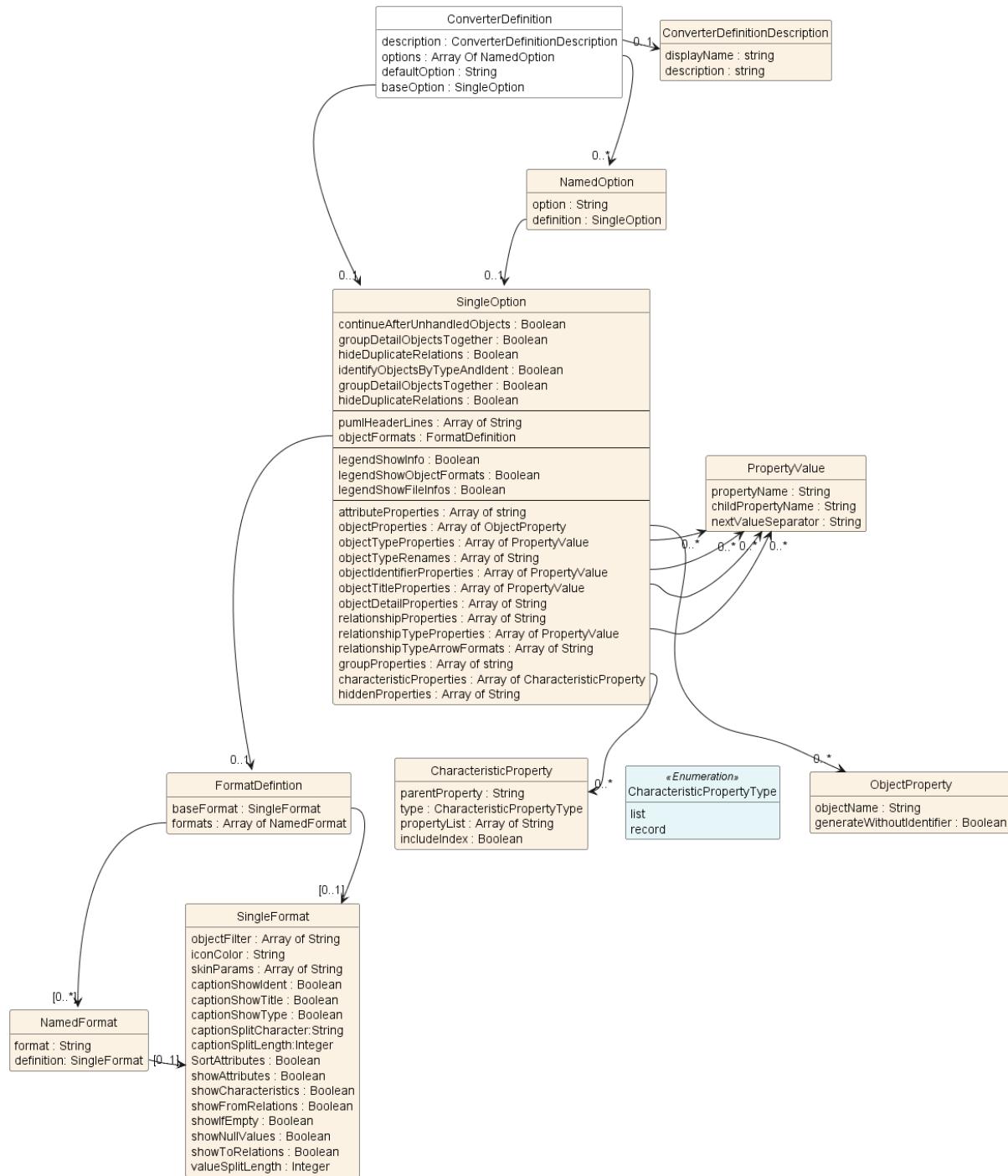


Figure 33 Main Configuration Model

## 6.6.4 Converterdefinition

### 6.6.4.1 *description*

Name	Cardinality	Datatype	Default
Description	0..*	ConverterDefintionDescription	
Description			

Name	Cardinality	Datatype	Default
<b>6.6.4.2 Object to describe the purpose and use of this input list file. This description is used in get /definitionfiles operation (See 4.2.4.1 get /heartbeat)</b>			
This operations returns generic information's about the service and how it is configured.			
<b>6.6.4.2.1 Sample curl command</b>			
<pre>curl -X GET http://localhost:9090/api/heartbeat</pre>			
<b>6.6.4.2.2 Sample result data model</b>			
Result is a text showing the version of the service and further details.			
<pre>json2puml server v2.2.1.85 (Windows 64Bit x86 Debug)</pre>			
<b>6.6.4.3 get /serviceinformation</b>			
This operations returns generic information's about the service and how it is configured.			
<b>6.6.4.3.1 Sample curl command</b>			
<pre>curl -X GET http://localhost:9090/api/serviceinformation</pre>			
<b>6.6.4.3.2 Result data model</b>			
<pre>ServiceInformation service : String additionalServiceInformation : String plantUmlInformation : Array of String definitionFiles : Array of string inputlistFiles : Array of string</pre>			
<p><i>Figure 20 get /serviceinformation result data model</i></p> <p>The data structure shows various generic information's about the service and how it is configured.  <b>get /definitionfiles</b>) of the service application.  (See 6.6.5 ConverterDefintionDescription)</p>			

#### 6.6.4.4 *options*

Name	Cardinality	Datatype	Default
Options	0..*	NamedOption	
<b>Description</b>			
This array defines the list of possible options which could be used for analysing and formatting the output. Which option is used will be defined via the variable <code>option</code> (See 6.2.1.2 Option)			

#### 6.6.4.5 *baseOption*

Name	Cardinality	Datatype	Default
baseOption	0..1	SingleOption	
<b>Description</b>			
The property defines the base converter definition option. This definition is merged with the defined runtime option. (See 6.6.1 Merging of options)			

#### 6.6.4.6 *defaultOption*

Name	Cardinality	Datatype	Default
defaultOption	0..1	String	
<b>Description</b>			
Name of the option which should be used when option is not defined via any other way (See 6.2.1.2 Option).			

### 6.6.5 ConverterDefintionDescription

Record to define the information which are returned when the definition file is found and returned for the service operation `get /definitionFile` or `get /inputListFile` (See 4.2.4 Service operations)

)

#### 6.6.5.1 *displayName*

Name	Cardinality	Datatype	Default
displayName	0..1	string	
<b>Description</b>			
Information to be displayed instead of the filename of the converter definition file. This allows to give a more descriptive name to be shown.			

### 6.6.5.2 *description*

Name	Cardinality	Datatype	Default
Description	0..1	string	
<b>Description</b>			
Additional description for the current converter definition file. This could be shown additionally to help a user understanding the purpose of the file (e.g. in a Mouse Over Notification)			

### 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	
<b>Description</b>			
Name of the option			

#### 6.6.6.2 `baseOption`

Name	Cardinality	Datatype	Default
definition	0..1	SingleOption	
<b>Description</b>			
The definition contains the option specific values which should be merged with 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`
- 6.6.8.11 `objectTypeProperties`
- 6.6.8.13 `objectIdentifierProperties`
- 6.6.8.17 `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.

### Example:

```
"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	<pre>"characteristic": {   "name": "Product",   "value": "Voice Over IP Basic",</pre>	"Voice Over IP Basic"
1b	<pre>"product": {   "name": "Product",   "value": "Voice Over IP Basic",</pre>	N/A (because of the not matching object type product)
2	<pre>"product": {   "id": "g265-tf85",   "name": "VOIP",   "fullName": "Voice Over IP Basic"   "productSerialNumber": "N/A",</pre>	"VOIP" "fullName" is ignored because the "name" is coming first in the definition list.
3	<pre>"ticketType": [   {     "name": "Incident"   },   {     "name": "Admin"   },   {     "name": "Escalation"   } ]</pre>	"Incident.Admin.Escalation"
4	<pre>"transition": {   "sourceStatus": {     "name": "Draft"   },   "targetStatus": {     "name": "New"   }, }</pre>	"Draft->New"

### 6.6.7.1 propertyName

Name	Cardinality	Datatype	Default
propertyName	1	String	
<b>Description</b>			
<p>Name of the property where the information is searched.            The name can be defined as &lt;objectType&gt;. &lt;propertyName&gt; or as &lt;propertyName&gt;. If no object type is defined the propertyName is checked against all object types.</p>			

Name	Cardinality	Datatype	Default
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.

Example:

```
"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
<b>Description</b>			
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 groupDetailObjectsTogether

Name	Cardinality	Datatype	Default
groupDetailObjectsTogether	0..1	Boolean	False
<b>Description</b>			
This parameter allows to generate "together" objects in PlantUML to group the detail objects (See 6.6.8.15 objectDetailProperties) 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.			

### 6.6.8.3 hideDuplicateRelations

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

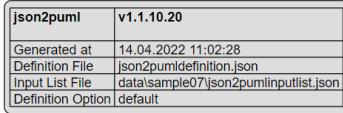
### 6.6.8.4 identifyObjectsByTypeAndIdent

Name	Cardinality	Datatype	Default
identifyObjectsByTypeAndIdent	0..1	Boolean	False
<b>Description</b>			
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)			

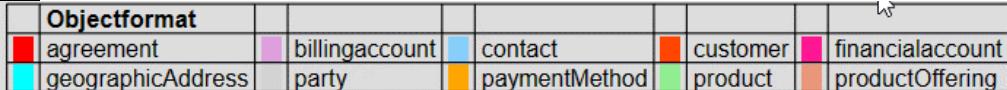
#### 6.6.8.5 hideEmptyObjects

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

#### 6.6.8.6 legendShowInfo

Name	Cardinality	Datatype	Default										
legendShowInfo	0..1	Boolean	True										
<b>Description</b>													
Defines if the common information block should be generated into the legend of the drawing.													
<b>Example:</b>													
 <p>The screenshot shows a legend block with the following content:</p> <table border="1"> <tr> <td>json2puml</td> <td>v1.1.10.20</td> </tr> <tr> <td>Generated at</td> <td>14.04.2022 11:02:28</td> </tr> <tr> <td>Definition File</td> <td>json2pumldefinition.json</td> </tr> <tr> <td>Input List File</td> <td>data\sample07\json2pumlinputlist.json</td> </tr> <tr> <td>Definition Option</td> <td>default</td> </tr> </table>				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
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																					
<b>Description</b>																								
Defines if the object format color list should be generated into the legend of the drawing.																								
<b>Example:</b>																								
 <p>The screenshot shows a legend block with the following content:</p> <table border="1"> <tr> <td><b>Objectformat</b></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>agreement</td> <td>billingaccount</td> <td>contact</td> <td>customer</td> <td>financialaccount</td> <td></td> <td></td> </tr> <tr> <td>geographicAddress</td> <td>party</td> <td>paymentMethod</td> <td>product</td> <td>productOffering</td> <td></td> <td></td> </tr> </table>				<b>Objectformat</b>							agreement	billingaccount	contact	customer	financialaccount			geographicAddress	party	paymentMethod	product	productOffering		
<b>Objectformat</b>																								
agreement	billingaccount	contact	customer	financialaccount																				
geographicAddress	party	paymentMethod	product	productOffering																				

#### 6.6.8.8 legendShowFileInfos

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

Name	Cardinality	Datatype	Default
<u>Example:</u>			
<b>File</b>	<b>Date</b>	<b>Size (kb)</b>	<b>No Of Lines</b>
data\sample07\TMF629_id_100000005695_ACRM.json	14.03.2022 08:25:26	1,671	56
data\sample07\TMF632_id_600000000000004717_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 *attributeProperties*

Name	Cardinality	Datatype	Default
attributeProperties	0..*	String	
<b>Description</b>			
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.			

#### 6.6.8.10 *objectProperties*

Name	Cardinality	Datatype	Default
objectProperties	0..*	ObjectProperty	
<b>Description</b>			
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. <b>Example:</b>			
<pre>"objectProperties": [     "account",     "party",     {"objectName": "productTerm",         "generateWithoutIdentifier": "true"},     {"objectName": "contactMedium",         "generateWithoutIdentifier": "true"} ],</pre>			

#### 6.6.8.11 *objectTypeProperties*

Name	Cardinality	Datatype	Default
objectTypeProperties	0..*	PropertyValue	
<b>Description</b>			
<p>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 6.6.8.12 objectTypeRenames)          the object type could be renamed.          (See 5.3.2 Object Identification)</p>			

#### 6.6.8.12 *objectTypeRenames*

Name	Cardinality	Datatype	Default
objectTypeRenames	0..*	String	
<b>Description</b>			
<p>List of name replacements to be used for object type. (See 5.3.2 Object Identification)          The format of the rename property is:          “&lt;oldname&gt;=&lt;newname&gt;”.</p> <p><b>Example:</b></p> <pre>"objectTypeRenames": [     "compositeProduct=product",     "defaultPaymentMethod=paymentMethod",     "place=geographicAddress",     "email=contactMedium", ]</pre>			

#### 6.6.8.13 *objectIdentifierProperties*

Name	Cardinality	Datatype	Default
objectIdentifierProperties	0..*	PropertyValue	
<b>Description</b>			
<p>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)</p>			

#### 6.6.8.14 *objectTitleProperties*

Name	Cardinality	Datatype	Default
objectTitleProperties	0..*	PropertyValue	

Name	Cardinality	Datatype	Default
<b>Description</b>			
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 *objectDetailProperties*

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

#### 6.6.8.16 *relationshipProperties*

Name	Cardinality	Datatype	Default
relationshipProperties	0..*	String	
<b>Description</b>			
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)			

#### 6.6.8.17 *relationshipTypeProperties*

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

#### 6.6.8.18 *relationshipTypeArrowFormats*

Name	Cardinality	Datatype	Default
relationshipTypeArrowFormats	0..*	PropertyValue	
<b>Description</b>			
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:			

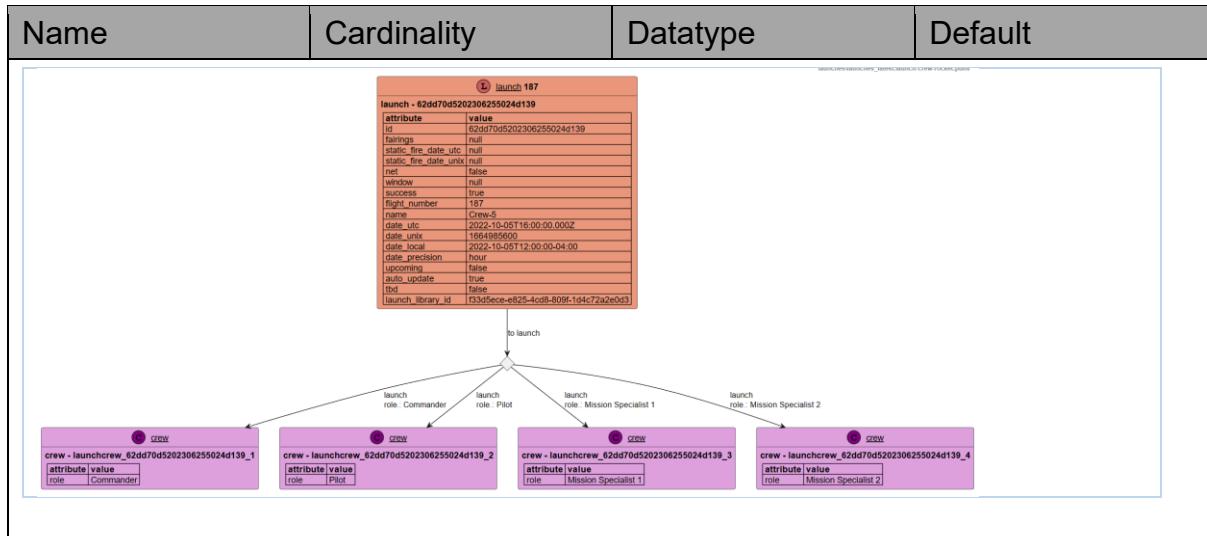
Name	Cardinality	Datatype	Default
<relationship pattern>[.<relationship type>]=<PlantUML arrow format>			
<b>Example:</b>			

```
"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)

#### 6.6.8.19 *groupProperties*

Name	Cardinality	Datatype	Default
groupProperties	0..*	String	
<b>Description</b>			
A group property allows to structure the generated output in that way that an additional diamond shape is created which groups the relations of the related detail objects of the same type.			
<b>Example:</b>			
<b>Without GroupProperty</b>			
<p>The diagram illustrates a launch object with various attributes like name, date UTC, and success status. Four crew members are associated with this launch, each assigned a specific role: Commander, Pilot, Mission Specialist 1, and Mission Specialist 2.</p>			
<b>With GroupProperty</b>			



#### 6.6.8.20 characteristicProperties

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

#### 6.6.8.21 hiddenProperties

Name	Cardinality	Datatype	Default
hiddenProperties	0..*	String	
<b>Description</b>			
List of properties which will stop the recursive handling of the converter.			

#### 6.6.8.22 pumlHeaderLines

Name	Cardinality	Datatype	Default
pumlHeaderLines	0..*	String	
<b>Description</b>			
The “pumlHeaderlines” will be written at the beginning of the generated PlantUML file. This allows to have generic formatting definitions. <u>Example:</u>			

Name	Cardinality	Datatype	Default
<pre>"pumlHeaderLines": [   "hide stereotype",   "skinparam HeaderFontSize 18" ],</pre>			

#### 6.6.8.23 *objectFormats*

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

### 6.6.9 ObjectProperty

The object properties can be written in a short and in a long format.

The short format contains only the `objectName` as a string (without property name). The `generateWithoutIdentifier` will then be the default value `false`. The long format is a json record containing both properties. This allows to simplify the definition of the object properties.

Example:

```
"characteristicProperties": [
  "*pad.images",
  "height",
  "reddit",
  {"parentProperty": "xdragon", "type": "record", "includeIndex": true},
  {"parentProperty": "payload_weights", "type": "list",
   "nameProperty": "name", "valueProperty": "kg", "infoProperty": "lb"}
],
```

#### 6.6.9.1 *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 “ <code>objectTypeRenames</code> ” parameter (See 6.6.8.12 <code>objectTypeRenames</code> ) the object type could be renamed.			

Name	Cardinality	Datatype	Default
(See 5.3.2 Object Identification)			

#### 6.6.9.2 *generateWithoutIdentifier*

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

#### 6.6.10 CharacteristicProperty

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

The characteristic properties can be written in a short and in a long format.

The short format contains only the `parentName` as a string (without property name). The `type` will then be the default value `record` and `includeIndex` will then be the default values `false`. The long format is a json record containing all properties. This allows to simplify the definition of the characteristic properties.

##### Example:

```
"objectProperties": [
    "launch",
    "crew",
    {"objectName": "launchcrew", "generateWithoutIdentifier": "true"},
    {"objectName": "launchcore", "generateWithoutIdentifier": "true"},
    "capsule",
    "payload",
    "core"
]
```

#### 6.6.10.1 *parentProperty*

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

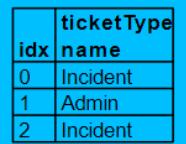
### 6.6.10.2 type

Name	Cardinality	Datatype	Default
Type	1	Characteristic.PropertyType	record
<b>Description</b>			
Type of the characteristic Possible values: - list - record			

### 6.6.10.3 propertyList

Name	Cardinality	Datatype	Default
propertyList	0..*	String	
<b>Description</b>			
List of properties which should be shown as a table. The order of the shown columns is based on the order in this list and the order when the fields appeared the first time. When this list is empty, all fields will be handled.			
Wildcards are also supported. In combination with wildcards there is additionally the option to exclude a property from the result by adding the prefix “-“ before the property name.			
<b>Example:</b> <pre>{"parentProperty": "contactMedium.characteristic", "type": "record",   "propertyList": "*", "-@type"},</pre> <p>In this example all properties of the contactMedium.characteristic property will be handled, except the “@type” property.</p>			

### 6.6.10.4 includeIndex

Name	Cardinality	Datatype	Default									
includeIndex	0..1	Boolean	false									
<b>Description</b>												
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”.												
<b>Example:</b>  <table border="1"> <thead> <tr> <th>idx</th> <th>ticketType</th> </tr> <tr> <th>name</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Incident</td> </tr> <tr> <td>1</td> <td>Admin</td> </tr> <tr> <td>2</td> <td>Incident</td> </tr> </tbody> </table>				idx	ticketType	name	0	Incident	1	Admin	2	Incident
idx	ticketType											
name												
0	Incident											
1	Admin											
2	Incident											

### 6.6.11 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)

#### 6.6.11.1 *baseFormat*

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

#### 6.6.11.2 *formats*

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

### 6.6.12 NamedFormat

#### 6.6.12.1 *formatName*

Name	Cardinality	Datatype	Default
formatName	1	6.6.13string	
<b>Description</b>			
Name of the format definition. This name will be used as PlantUML creole to identify the format.			

#### 6.6.12.2 *definition*

Name	Cardinality	Datatype	Default
Definition	0..1	SingleFormat	
<b>Description</b>			

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

### 6.6.13 SingleFormat

Example:

```
"baseFormat": {
  "objectFilter": [],
  "iconColor": null,
  "skinParams": ["BackgroundColor=STRATEGY"],
  "captionShowIdent": false,
  "captionShowTitle": true,
  "captionShowType": true,
  "captionSplitCharacter": "_",
  "captionSplitLength": 0,
  "showAttributes": true,
  "showCharacteristics": true,
  "showFromRelations": true,
  "showIfEmpty": false,
  "showNullValues": false,
  "showToRelations": true,
  "sortAttributes": true,
  "valueSplitLength": 50
}
```

#### 6.6.13.1 *objectFilter*

Name	Cardinality	Datatype	Default
objectFilter	0..*	string	
<b>Description</b>			
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 <code>baseFormat</code> definition.			

#### 6.6.13.2 *iconColor*

Name	Cardinality	Datatype	Default
iconColor	0..1	string	
<b>Description</b>			
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.			

### 6.6.13.3 skinParams

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

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
}
```

### 6.6.13.4 captionShowIdent

Name	Cardinality	Datatype	Default
captionShowIdent	0..1	Boolean	True
<b>Description</b>			
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 <i>italic</i> .			

### 6.6.13.5 captionShowTitle

Name	Cardinality	Datatype	Default
captionShowTitle	0..1	Boolean	True
<b>Description</b>			

Name	Cardinality	Datatype	Default
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 <b>bold</b> .			

#### 6.6.13.6 *captionShowType*

Name	Cardinality	Datatype	Default
captionShowType	0..1	Boolean	True
<b>Description</b>			
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 <u>underlined</u> .			

#### 6.6.13.7 *captionSplitCharacter*

Name	Cardinality	Datatype	Default
captionSplitCharacter	0..1	String	
<b>Description</b>			
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 than the "captionSplitLength".			
<b>Example:</b>			
<pre>"captionSplitCharacter": " ", "captionSplitLength": "20",</pre>			
Would lead to:			
Figure 34 Example of an object identifier split			

#### 6.6.13.8 *captionSplitLength*

Name	Cardinality	Datatype	Default
captionSplitLength	0..1	Integer	0

Name	Cardinality	Datatype	Default
<b>Description</b>			
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 than the defined value.			

#### 6.6.13.9 *showAttributes*

Name	Cardinality	Datatype	Default
showAttributes	0..1	Boolean	True
<b>Description</b>			
Flag to define if the attribute list should be generated for the object. (See 5.3.6 5.3.7 Generated Outcome)			

#### 6.6.13.10 *showCharacteristics*

Name	Cardinality	Datatype	Default
showCharacteristics	0..1	Boolean	True
<b>Description</b>			
Flag to define if the characteristics lists should be generated for the object. (See 5.3.6 5.3.7 Generated Outcome)			

#### 6.6.13.11 *showFromRelations*

Name	Cardinality	Datatype	Default
showFromRelations	0..1	Boolean	True
<b>Description</b>			
Flag to define if the FROM relation list should be generated inside of the object. (See 5.3.6 5.3.7 Generated Outcome)			

#### 6.6.13.12 *showIsEmpty*

Name	Cardinality	Datatype	Default
showIsEmpty	0..1	Boolean	False
<b>Description</b>			
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 its identifier as a single sub object of another object.			

#### 6.6.13.13 *showNullValues*

Name	Cardinality	Datatype	Default
showNullValues	0..1	Boolean	False
<b>Description</b>			
Flag to define if null values should be included in the generated output.			

#### 6.6.13.14 *showToRelations*

Name	Cardinality	Datatype	Default
showAttributes	0..1	Boolean	True
<b>Description</b>			
Flag to define if the TO relation list should be generated inside of the object. (See 5.3.6 5.3.7 Generated Outcome)			

#### 6.6.13.15 *sortAttributes*

Name	Cardinality	Datatype	Default
sortAttributes	0..1	Boolean	True
<b>Description</b>			
Flag to define if attributes list and the characteristic lists should be sorted over the first column or not.			

## 6.7 Curl parameter file

### 6.7.1 Data model

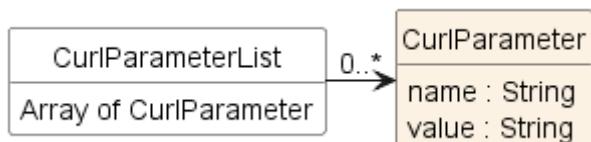


Figure 35 Curl parameter file data model

The curl parameter list is a list of name value pairs which are defining the values to be used by the curl pre-processor (See 5.2.1 Curl parametrisation)

### 6.7.2 Curl Parameter

The curl parameter will be used by the curl pre-processor to fetching data based on the defined curl parameters (See 5.2 Curl pre-processor).

The curl parameter can be defined in three different formats:

4. String  
“<parameter name>=<parameter value>”
5. Simple json record  
{ “<parameter name>”: “<parameter value>” }
6. Full json record  
{ “name”: “<parameter name>”,  
“value”: “<parameter value>” }

#### 6.7.2.1 name

Name	Cardinality	Datatype	Default
name	1	String	
<b>Description</b>			
This parameter defines the name of the curl parameter. Curl parameter names are defined using the following pattern: \${<name>} . In the configuration they can be defined only using the name or using the full pattern.			

#### 6.7.2.2 value

Name	Cardinality	Datatype	Default
Value	0..1	String	
<b>Description</b>			
This parameter defines the value of the curl parameter which is replacing the parameter name in the curl commands.			

## 6.8 Curl authentication file

The curl authentication file allows to predefine curl authentication parameters based on a base URL. Separating these parameters in an additional file is a simple way to ensure that client credentials are not handled in normal configuration files (See 5.2.6.4 Curl authentication file)

### 6.8.1 Data model

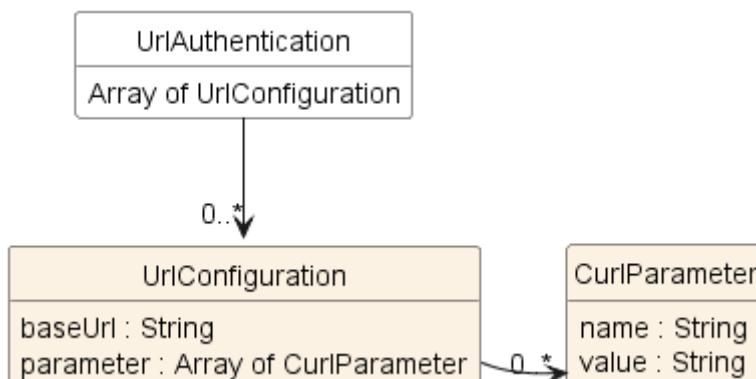


Figure 36 Main Configuration Model

## 6.8.2 UrlConfiguration

### 6.8.2.1 baseUrl

Name	Cardinality	Datatype	Default
baseUrl	1	String	
<b>Description</b>			
Defines the baseUrl value which will be compared with the curlBaseUrl value from the inputlist or inpurlistFile (See ) to find the corresponding curl authentication parameters.			

### 6.8.2.2 value

Name	Cardinality	Datatype	Default
Value	0..1	String	
<b>Description</b>			
This parameter defines the value of the curl parameter which is replacing the parameter name in the curl commands.			

## 6.8.3 Curl Parameter

The curl parameter will be used by the curl pre-processor to define url based authentication parameters (See 5.2 Curl pre-processor).

The curl parameter can be defined in three different formats:

7. String  
“<parameter name>=<parameter value>”
8. Simple json record  
{ “<parameter name>”: “<parameter value>” }
9. Full json record  
{ “name”: “<parameter name>”,  
“value”: “<parameter value>” }

### 6.8.3.1 name

Name	Cardinality	Datatype	Default
name	1	String	
<b>Description</b>			
This parameter defines the name of the of the curl parameter. Curl parameter names are defined using the following pattern: \${<name>} . In the configuration they can defined only using the name or using the full pattern.			

### 6.8.3.2 *value*

Name	Cardinality	Datatype	Default
Value	0..1	String	
Description	This parameter defines the value of the curl parameter which is replacing the parameter name in the curl commands.		

# 7 Appendix

## 7.1 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 in windows service list .....	34
Figure 20 get /serviceinformation result data model .....	35
Figure 21 get /definitionfiles result data model .....	36
Figure 22 get /inputlistfiles result data model.....	38
Figure 23 post /json2pumlRequest result data model.....	40
Figure 24 json2puml main components .....	46
Figure 25 jsonplaceholder data model.....	50
Figure 26: Basic flow how the converter is working .....	64
Figure 27 Object relationship example .....	66
Figure 28 Visualisation of a detailed object .....	67
Figure 29 Generated Output Example .....	70
Figure 30 Data model global configuration file.....	82
Figure 31 Data model parameter file .....	88
Figure 32: Data model input list file .....	94
Figure 33 Main Configuration Model.....	112
Figure 34 Example of an object identifier split .....	132
Figure 35 Curl parameter file data model .....	134
Figure 36 Main Configuration Model.....	135