json2puml

Based on version v1.3.4

Documentation

1 Content

2 Introduction

2.1 Current Situation

2.1.1 Plain JSON

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

```
{"id": "600000000000004510", "href": "https://party/v4/individual/600000000000004510",
"createdOn": "2022-01-19T13:59:03Z", "lastModifiedOn": "2022-01-19T13:59:03Z",
"givenName": "Karl", "familyName": "Maier", "fullName": "KarlMaier", "birthDate": "1963-
06-01T00:00:00Z", "status": "Active", "contactMedium": [{"id":
"600000000000004510_ContactMedium_2", "mediumType": "postalAddress", "preferred":
true,"validFor": {"startDateTime": "2022-01-17T00:00:00Z"},"characteristic":
{"contactType": "legal", "city": "Berlin", "country": "DEU", "postCode": "80993",
"street1": "FrankfurterStr", "place": {"id": "GeographicAddress_3912", "@referredType":
"GeographicAddress", "@type": "PlaceRef"}}},{"id": "600000000000004510_ContactMedium_20",
"mediumType": "telephone", "preferred": false,"validFor": {"startDateTime": "2022-01-
17T00:00:00Z"},"characteristic": {"contactType": "home", "phoneNumber":
"0123457689"}},{"id": "600000000000004510_ContactMedium_60", "mediumType": "email",
"preferred": true,"validFor": {"startDateTime": "2022-01-17T00:00:00Z"},"characteristic":
{"contactType": "personal", "emailAddress": "jesf@gmx.de", "extensions":
{"verificationStatus": "verified"}}},{"id": "600000000000004510_ContactMedium_75",
"mediumType": "telephone", "preferred": true,"validFor": {"startDateTime": "2022-01-
17T00:00:00Z"},"characteristic": {"contactType": "home", "phoneNumber":
"0123456789"}},{"id": "600000000000004510_ContactMedium_76", "mediumType": "telephone",
"preferred": true,"validFor": {"startDateTime": "2022-01-17T00:00:00Z"},"characteristic":
{"contactType": "mobile", "phoneNumber": "01721234567"}}],"relatedParty": [{"id":
"100000004215", "href": "https:///party/v4/relatedParty/100000004215", "name":
"KarlMaier", "role": "Customer", "@type": "Customer", "@referredType": "Customer",
"validFor": {"startDateTime": "2022-01-17T00:00:00Z"}}]}
```

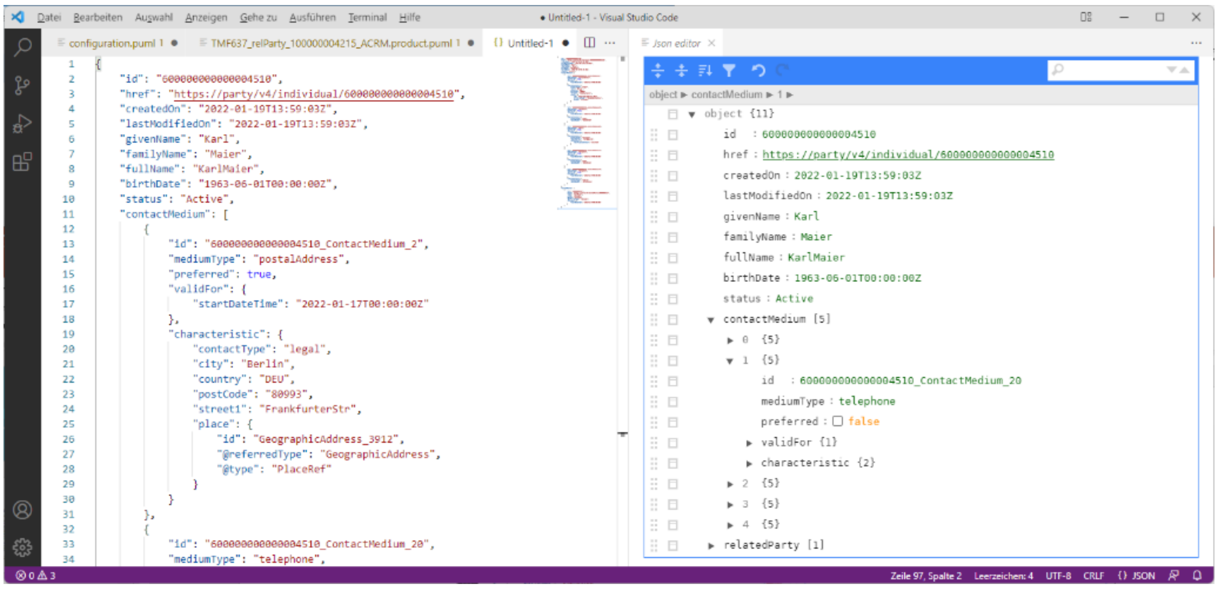
Figure 1: JSON sample

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

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

2.1.2 JSON Editor

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


Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe
• Untitled-1 - Visual Studio Code
configuration.puml 1
TMF637_relParty_100000004215_ACRM.product.puml 1
Untitled-1
Json editor
object ▶ contactMedium ▶ 1 ▶
Zeile 97, Spalte 2 Leerzeichen: 4 UTF-8 CRLF {} JSON

```
{
    "id": "600000000000004510",
    "href": "https://party/v4/individual/600000000000004510",
    "createdOn": "2022-01-19T13:59:03Z",
    "lastModifiedOn": "2022-01-19T13:59:03Z",
    "givenName": "Karl",
    "familyName": "Maier",
    "fullName": "KarlMaier",
    "birthDate": "1963-06-01T00:00:00Z",
    "status": "Active",
    "contactMedium": [
        {
            "id": "600000000000004510_ContactMedium_2",
            "mediumType": "postalAddress",
            "preferred": true,
            "validFor": {
                "startDateTime": "2022-01-17T00:00:00Z"
            },
            "characteristic": {
                "contactType": "legal",
                "city": "Berlin",
                "country": "DEU",
                "postCode": "80993",
                "street1": "FrankfurterStr",
                "place": {
                    "id": "GeographicAddress_3912",
                    "@referredType": "GeographicAddress",
                    "@type": "PlaceRef"
                }
            }
        },
        {
            "id": "600000000000004510_ContactMedium_20",
            "mediumType": "telephone",
```

```
▼ object {11}
    id : 600000000000004510
    href : https://party/v4/individual/600000000000004510
    createdOn : 2022-01-19T13:59:03Z
    lastModifiedOn : 2022-01-19T13:59:03Z
    givenName : Karl
    familyName : Maier
    fullName : KarlMaier
    birthDate : 1963-06-01T00:00:00Z
    status : Active
  ▼ contactMedium [5]
    ▶ 0 {5}
    ▼ 1 {5}
        id : 600000000000004510_ContactMedium_20
        mediumType : telephone
        preferred : ☐ false
      ▶ validFor {1}
      ▶ characteristic {2}
    ▶ 2 {5}
    ▶ 3 {5}
    ▶ 4 {5}
  ▶ relatedParty [1]
```

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:

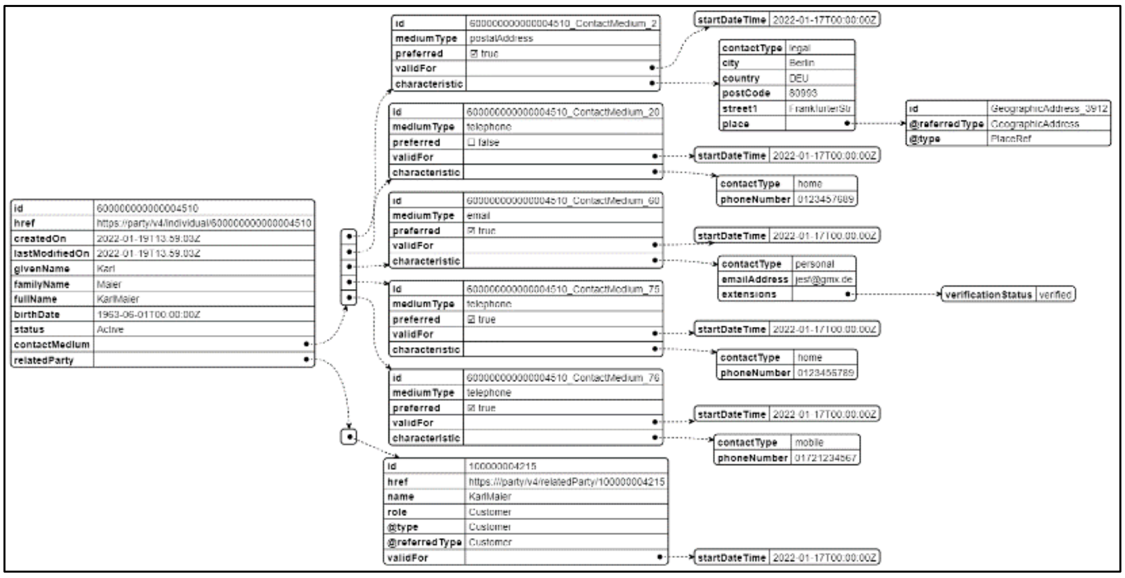

id | 600000000000004510
href | https://party/v4/individual/600000000000004510
createdOn | 2022-01-19T13:59:03Z
lastModifiedOn | 2022-01-19T13:59:03Z
givenName | Karl
familyName | Maier
fullName | KarlMaier
birthDate | 1963-06-01T00:00:00Z
status | Active
contactMedium
relatedParty
id | 600000000000004510_ContactMedium_2
mediumType | postalAddress
preferred | ☑ true
validFor
characteristic
startDateTime | 2022-01-17T00:00:00Z
contactType | legal
city | Berlin
country | DEU
postCode | 80993
street1 | FrankfurterStr
place
id | GeographicAddress_3912
@referredType | GeographicAddress
@type | PlaceRef
id | 600000000000004510_ContactMedium_20
mediumType | telephone
preferred | ☐ false
validFor
characteristic
startDateTime | 2022-01-17T00:00:00Z
contactType | home
phoneNumber | 0123457689
id | 600000000000004510_ContactMedium_60
mediumType | email
preferred | ☑ true
validFor
characteristic
startDateTime | 2022-01-17T00:00:00Z
contactType | personal
emailAddress | jesf@gmx.de
extensions
verificationStatus | verified
id | 600000000000004510_ContactMedium_75
mediumType | telephone
preferred | ☑ true
validFor
characteristic
startDateTime | 2022-01-17T00:00:00Z
contactType | home
phoneNumber | 0123455789
id | 600000000000004510_ContactMedium_76
mediumType | telephone
preferred | ☑ true
validFor
characteristic
startDateTime | 2022-01-17T00:00:00Z
contactType | mobile
phoneNumber | 01721234567
id | 100000004215
href | https:///party/v4/relatedParty/100000004215
name | KarlMaier
role | Customer
@type | Customer
@referredType | Customer
validFor
startDateTime | 2022-01-17T00:00:00Z

Figure 3 PlantUML JSON Example

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

2.2 json2puml

The solution to improve the visualisation of JSON data is

`json2puml`

`json2puml`	is a command line tool developed to generate PlantUML files based JSON files (TMF based).
`json2puml`	has an understanding of how data is structured in TMF and simplifies and visualises the outcome.
`json2puml`	has the possibility to combine the JSON results of multiple API calls into one result set.
`json2puml`	is highly configurable to generate outcomes in different detailed levels.

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

2.3 TMF based examples

2.3.1 Example 1 – TMF 632: Party Individual

Based on the example Data used above the following outputs are possible.

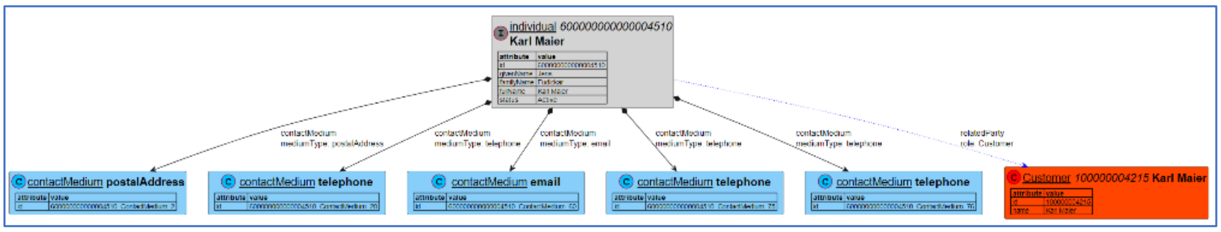


Figure 4 TMF 632 Example 1

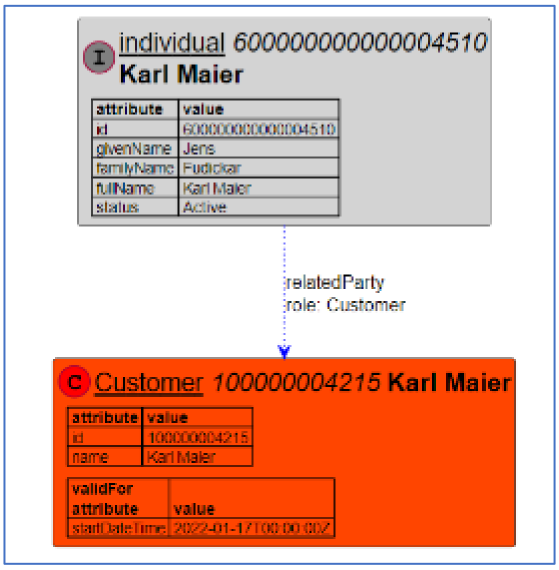


Figure 5 TMF 632 Example 2

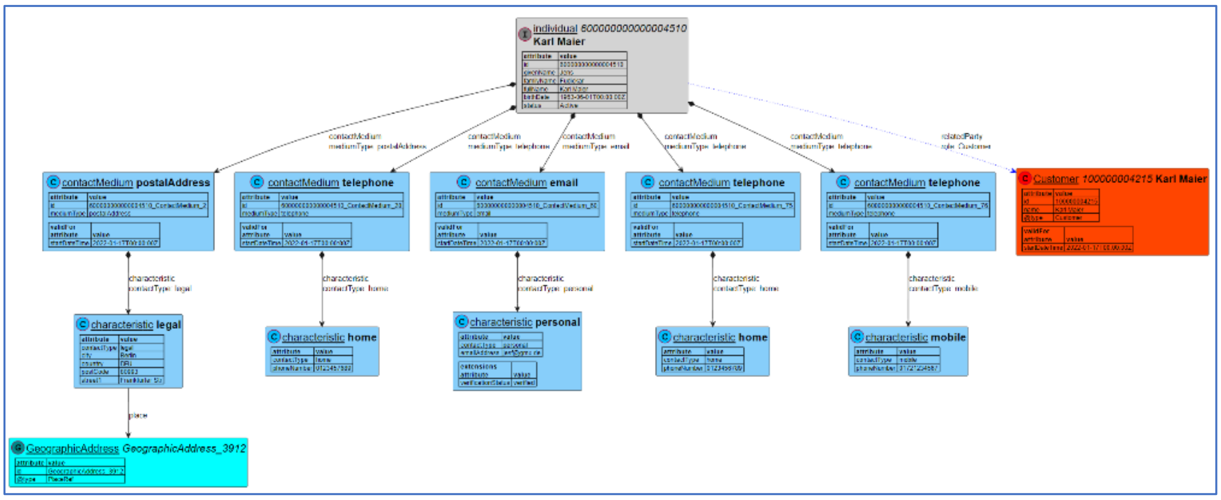


Figure 6 TMF 632 Example 3

2.3.2 Example 2 - TMF 637: All products of a customer

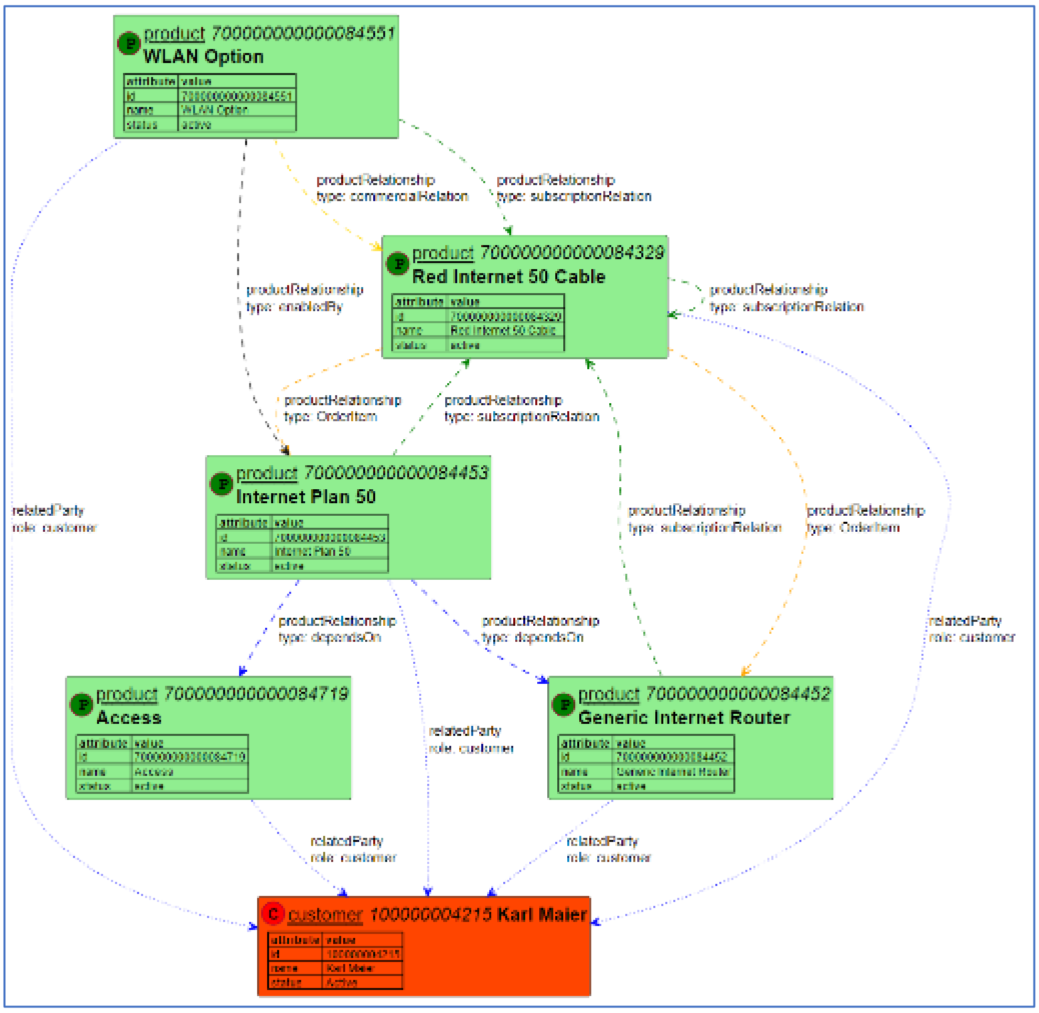

product 7000000000000084551
WLAN Option
attribute | value
id | 7000000000000084551
name | WLAN Option
status | active
productRelationship type: commercialRelation
productRelationship type: subscriptionRelation
product 7000000000000084329
Red Internet 50 Cable
attribute | value
id | 7000000000000084329
name | Red Internet 50 Cable
status | active
productRelationship type: enabledBy
productRelationship type: subscriptionRelation
productRelationship type: OrderItem
productRelationship type: subscriptionRelation
product 7000000000000084453
Internet Plan 50
attribute | value
id | 7000000000000084453
name | Internet Plan 50
status | active
relatedParty role: customer
productRelationship type: subscriptionRelation
productRelationship type: OrderItem
productRelationship type: dependsOn
productRelationship type: dependsOn
relatedParty role: customer
product 7000000000000084719
Access
attribute | value
id | 7000000000000084719
name | Access
status | active
product 7000000000000084452
Generic Internet Router
attribute | value
id | 7000000000000084452
name | Generic Internet Router
status | active
relatedParty role: customer
relatedParty role: customer
relatedParty role: customer
customer 100000004215 Karl Maier
attribute | value
id | 100000004215
name | Karl Maier
status | Active

Figure 7 TMF 637 Example 1

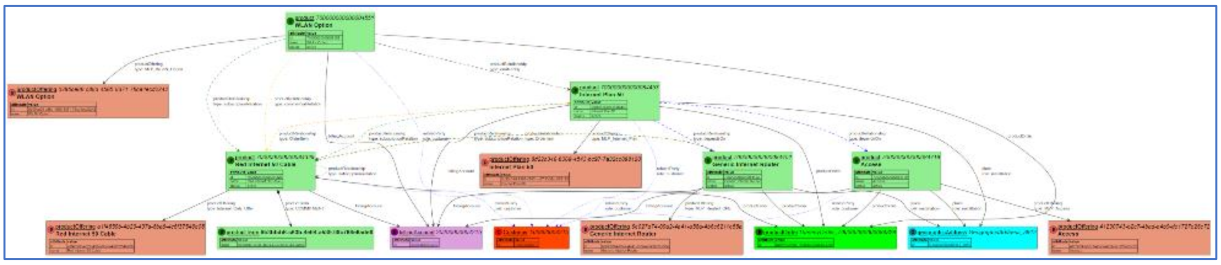


Figure 8 TMF 637 Example 2

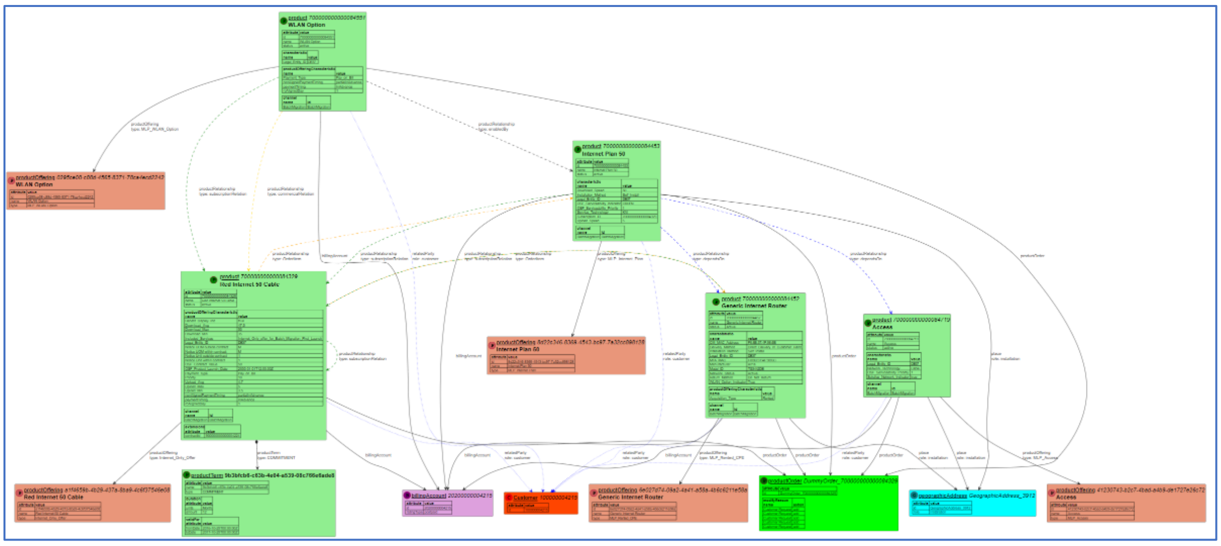


Figure 9 TMF 637 Example 3

2.3.3 Example 3 - TMF 629,632,637,666,670 for one customer

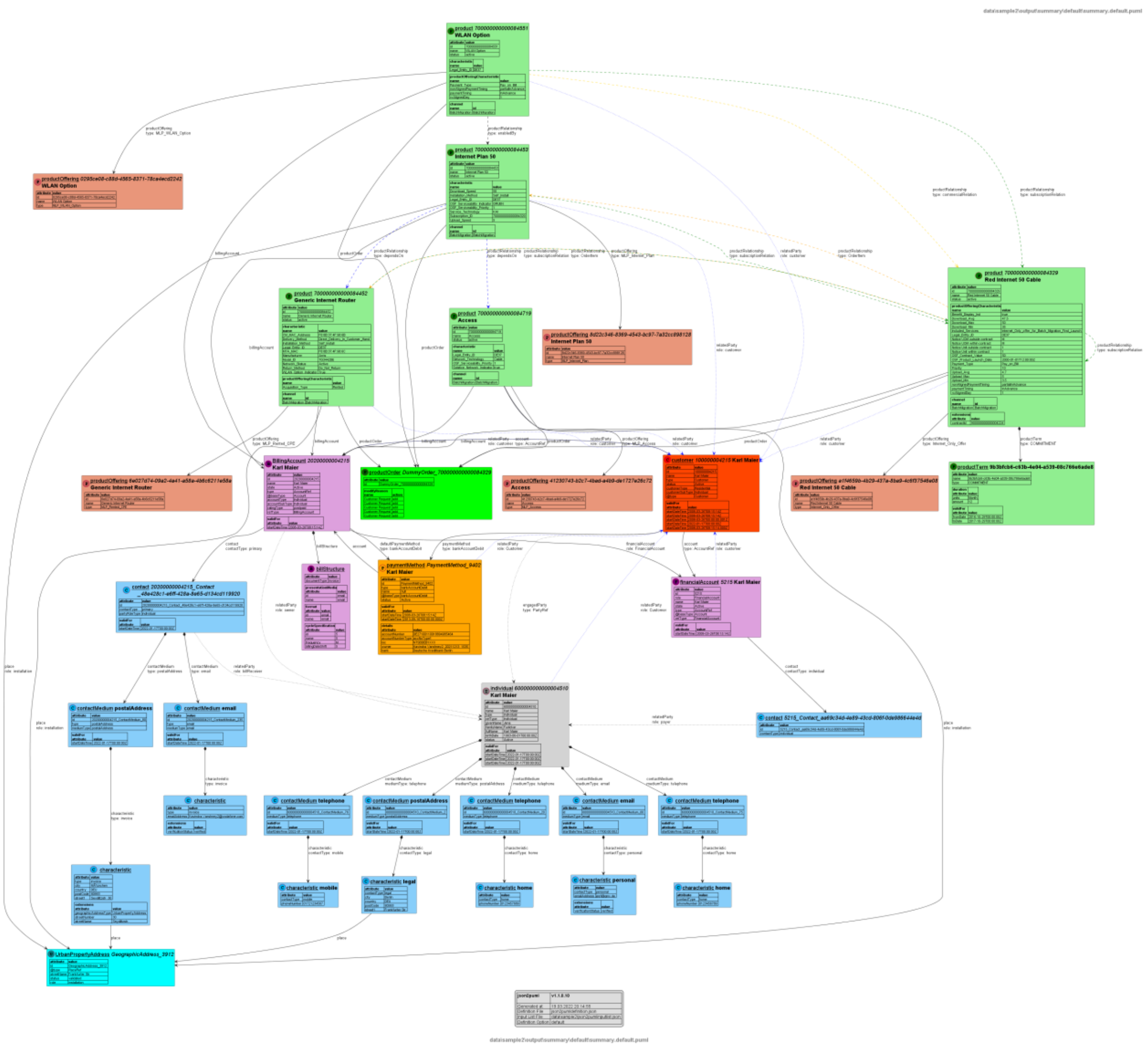


Figure 10 TMF 629,632,637,666,670

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

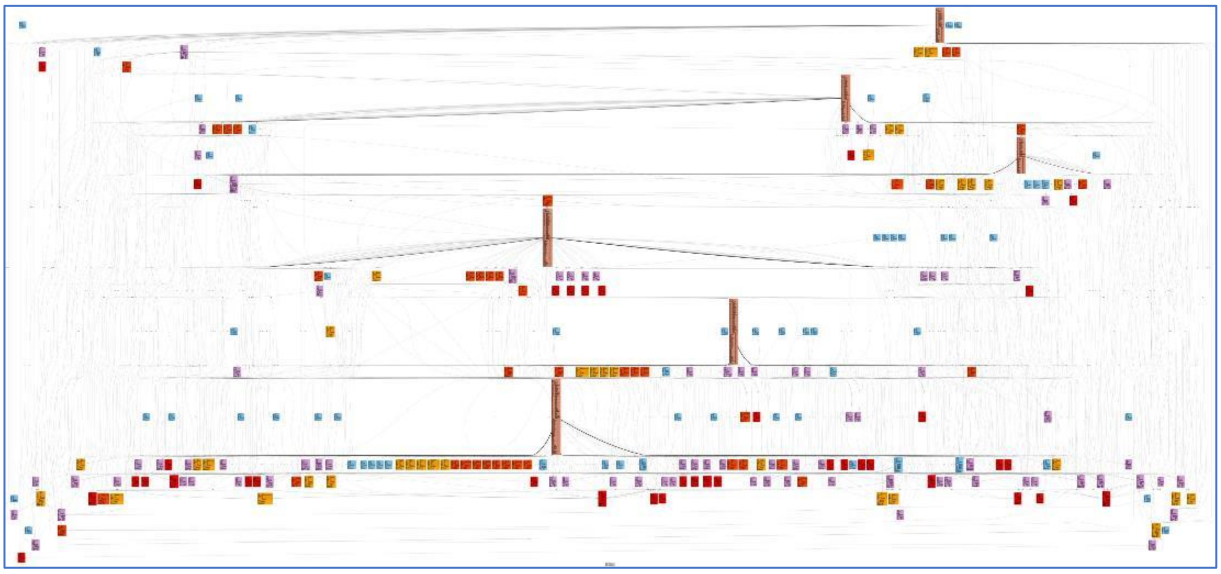
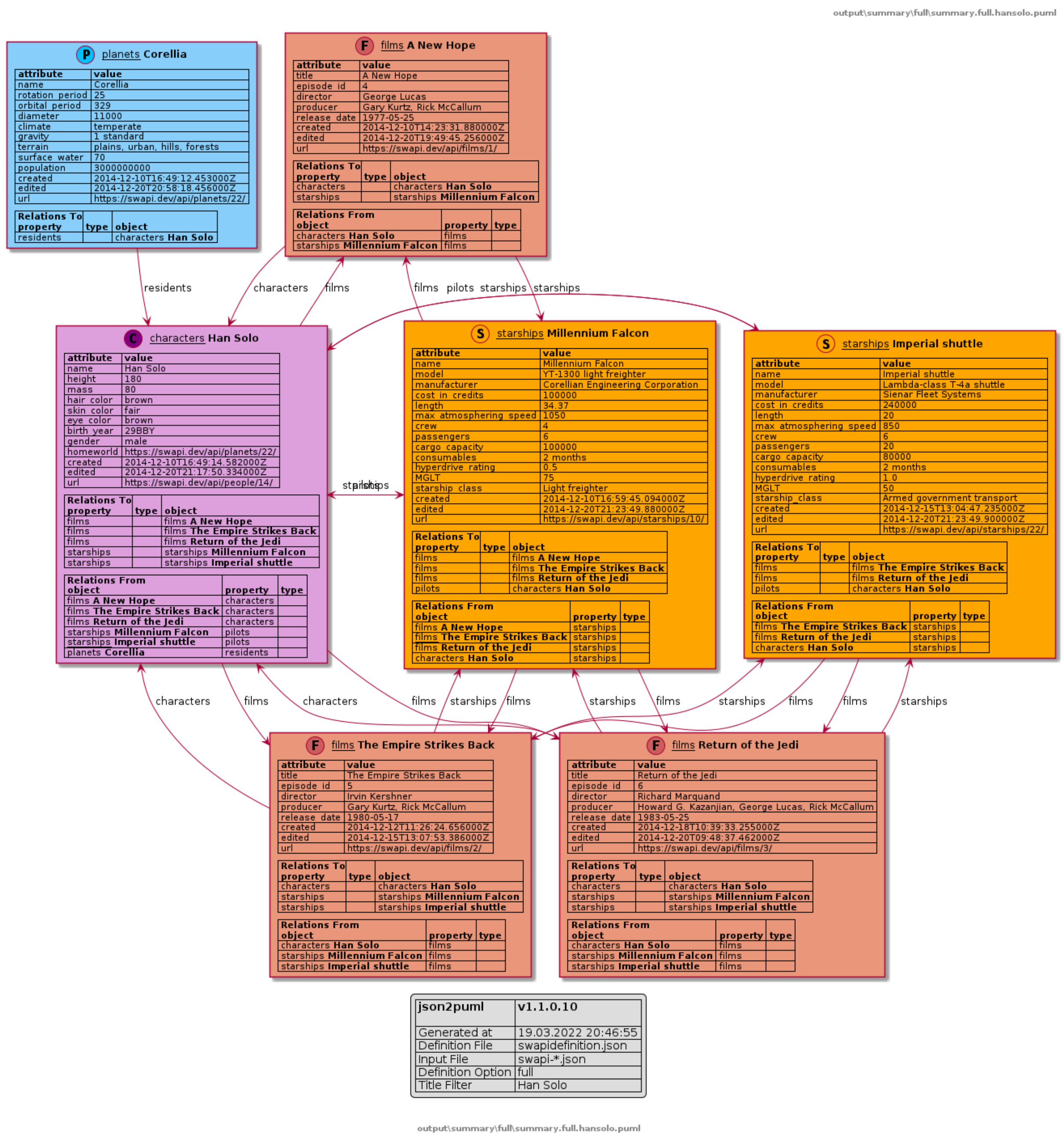


Figure 11 swapi.dev - all data - 6 Films, 82 Persons, 60 Planets, 37 Species, 36 Star ships, 39 Vehicles

2.4.2 Example 2 – Han Solo



P planets **Corellia**

attribute	value
name	Corellia
rotation period	25
orbital period	329
diameter	11000
climate	temperate
gravity	1 standard
terrain	plains, urban, hills, forests
surface water	70
population	3000000000
created	2014-12-10T16:49:12.453000Z
edited	2014-12-20T20:58:18.456000Z
url	https://swapi.dev/api/planets/22/

Relations To property	type	object
residents		characters **Han Solo**

F films **A New Hope**

attribute	value
title	A New Hope
episode id	4
director	George Lucas
producer	Gary Kurtz, Rick McCallum
release date	1977-05-25
created	2014-12-10T14:23:31.880000Z
edited	2014-12-20T19:49:45.256000Z
url	https://swapi.dev/api/films/1/

Relations To property	type	object
characters		characters **Han Solo**
starships		starships **Millennium Falcon**

Relations From object	property	type
characters **Han Solo**	films	
starships **Millennium Falcon**	films	

C characters **Han Solo**

attribute	value
name	Han Solo
height	180
mass	80
hair color	brown
skin color	fair
eye color	brown
birth year	29BBY
gender	male
homeworld	https://swapi.dev/api/planets/22/
created	2014-12-10T16:49:14.582000Z
edited	2014-12-20T21:17:50.334000Z
url	https://swapi.dev/api/people/14/

Relations To property	type	object
films		films **A New Hope**
films		films **The Empire Strikes Back**
films		films **Return of the Jedi**
starships		starships **Millennium Falcon**
starships		starships **Imperial shuttle**

Relations From object	property	type
films **A New Hope**	characters	
films **The Empire Strikes Back**	characters	
films **Return of the Jedi**	characters	
starships **Millennium Falcon**	pilots	
starships **Imperial shuttle**	pilots	
planets **Corellia**	residents	

S starships **Millennium Falcon**

attribute	value
name	Millennium Falcon
model	YT-1300 light freighter
manufacturer	Corellian Engineering Corporation
cost in credits	100000
length	34.37
max atmosphering speed	1050
crew	4
passengers	6
cargo capacity	100000
consumables	2 months
hyperdrive rating	0.5
MGLT	75
starship class	Light freighter
created	2014-12-10T16:59:45.094000Z
edited	2014-12-20T21:23:49.880000Z
url	https://swapi.dev/api/starships/10/

Relations To property	type	object
films		films **A New Hope**
films		films **The Empire Strikes Back**
films		films **Return of the Jedi**
pilots		characters **Han Solo**

Relations From object	property	type
films **A New Hope**	starships	
films **The Empire Strikes Back**	starships	
films **Return of the Jedi**	starships	
characters **Han Solo**	starships	

S starships **Imperial shuttle**

attribute	value
name	Imperial shuttle
model	Lambda-class T-4a shuttle
manufacturer	Sienar Fleet Systems
cost in credits	240000
length	20
max atmosphering speed	850
crew	6
passengers	20
cargo capacity	80000
consumables	2 months
hyperdrive rating	1.0
MGLT	50
starship_class	Armed government transport
created	2014-12-15T13:04:47.235000Z
edited	2014-12-20T21:23:49.900000Z
url	https://swapi.dev/api/starships/22/

Relations To property	type	object
films		films **The Empire Strikes Back**
films		films **Return of the Jedi**
pilots		characters **Han Solo**

Relations From object	property	type
films **The Empire Strikes Back**	starships	
films **Return of the Jedi**	starships	
characters **Han Solo**	starships	

F films **The Empire Strikes Back**

attribute	value
title	The Empire Strikes Back
episode id	5
director	Irvin Kershner
producer	Gary Kurtz, Rick McCallum
release date	1980-05-17
created	2014-12-12T11:26:24.656000Z
edited	2014-12-15T13:07:53.386000Z
url	https://swapi.dev/api/films/2/

Relations To property	type	object
characters		characters **Han Solo**
starships		starships **Millennium Falcon**
starships		starships **Imperial shuttle**

Relations From object	property	type
characters **Han Solo**	films	
starships **Millennium Falcon**	films	
starships **Imperial shuttle**	films	

F films **Return of the Jedi**

attribute	value
title	Return of the Jedi
episode id	6
director	Richard Marquand
producer	Howard G. Kazanjian, George Lucas, Rick McCallum
release date	1983-05-25
created	2014-12-18T10:39:33.255000Z
edited	2014-12-20T09:48:37.462000Z
url	https://swapi.dev/api/films/3/

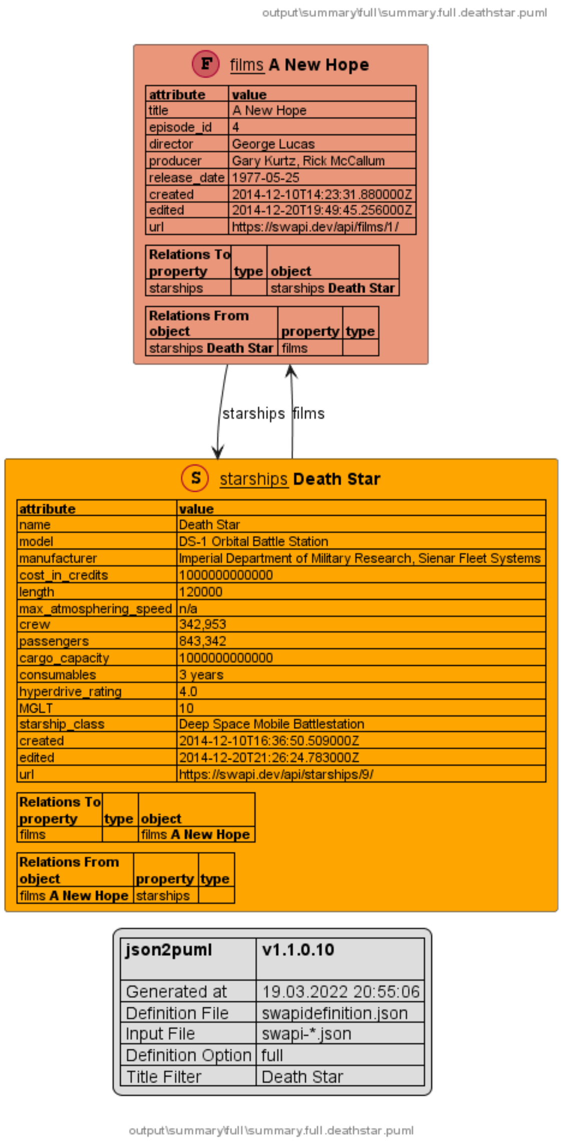
Relations To property	type	object
characters		characters **Han Solo**
starships		starships **Millennium Falcon**
starships		starships **Imperial shuttle**

Relations From object	property	type
characters **Han Solo**	films	
starships **Millennium Falcon**	films	
starships **Imperial shuttle**	films	

json2puml	**v1.1.0.10**
Generated at	19.03.2022 20:46:55
Definition File	swapidefinition.json
Input File	swapi-*.json
Definition Option	full
Title Filter	Han Solo

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

2.4.3 Example 3 – Death Star


output\summary\full\summary.full.deathstar.puml
F
films A New Hope
starships
films
S
starships Death Star
output\summary\full\summary.full.deathstar.puml

films **A New Hope**

attribute	value
title	A New Hope
episode_id	4
director	George Lucas
producer	Gary Kurtz, Rick McCallum
release_date	1977-05-25
created	2014-12-10T14:23:31.880000Z
edited	2014-12-20T19:49:45.256000Z
url	https://swapi.dev/api/films/1/

Relations To		
property	**type**	**object**
starships		starships **Death Star**

Relations From		
object	**property**	**type**
starships **Death Star**	films	

starships **Death Star**

attribute	value
name	Death Star
model	DS-1 Orbital Battle Station
manufacturer	Imperial Department of Military Research, Sienar Fleet Systems
cost_in_credits	1000000000000
length	120000
max_atmosphering_speed	n/a
crew	342,953
passengers	843,342
cargo_capacity	1000000000000
consumables	3 years
hyperdrive_rating	4.0
MGLT	10
starship_class	Deep Space Mobile Battlestation
created	2014-12-10T16:36:50.509000Z
edited	2014-12-20T21:26:24.783000Z
url	https://swapi.dev/api/starships/9/

Relations To		
property	**type**	**object**
films		films **A New Hope**

Relations From		
object	**property**	**type**
films **A New Hope**	starships	

json2puml	**v1.1.0.10**
Generated at	19.03.2022 20:55:06
Definition File	swapidefinition.json
Input File	swapi-*.json
Definition Option	full
Title Filter	Death Star

Figure 13 swapi.dev - /titlefilter = "Han Solo"

3 Installation

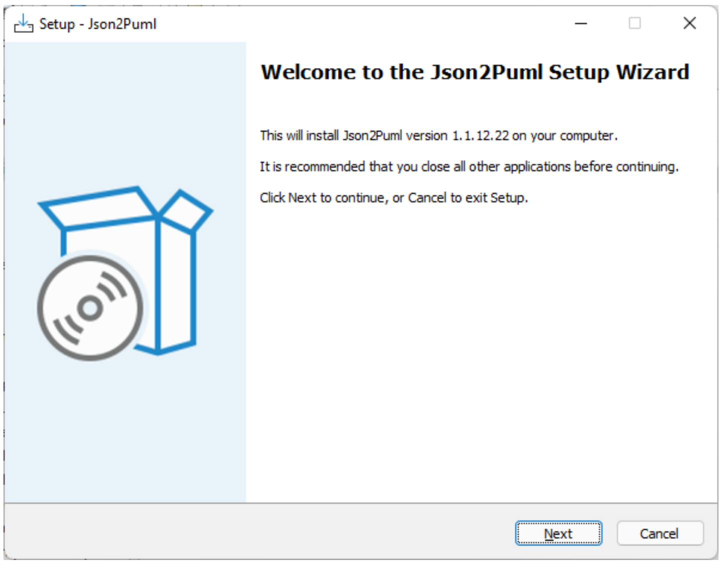
`json2puml` comes with an automated setup program which copies the main files to a user defined directory, optionally downloads the PlantUML jar file and configures the system to simplify the usage.

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

3.1 Steps

The setup is following the following steps:

3.1.1 Welcome Screen



Setup - Json2Puml

Welcome to the Json2Puml Setup Wizard

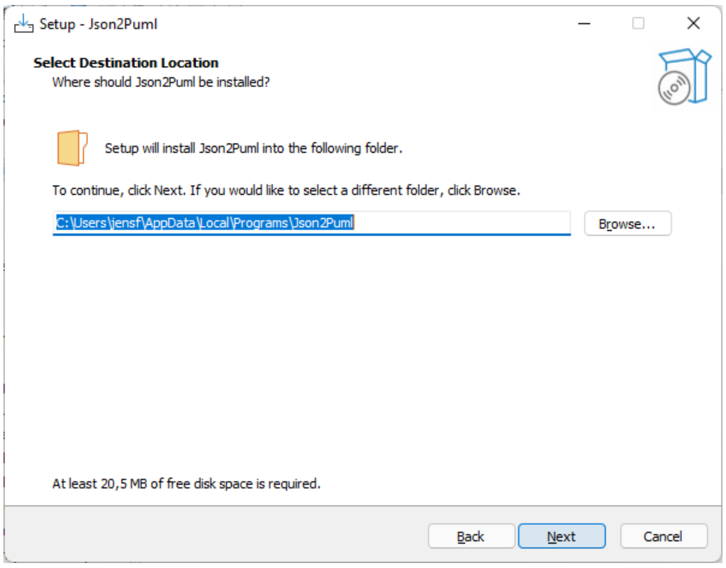
This will install Json2Puml version 1.1.12.22 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Figure 14 Setup - Welcome Screen

3.1.2 Definition of the installation folder



Setup - Json2Puml

Select Destination Location

Where should Json2Puml be installed?

Setup will install Json2Puml into the following folder.

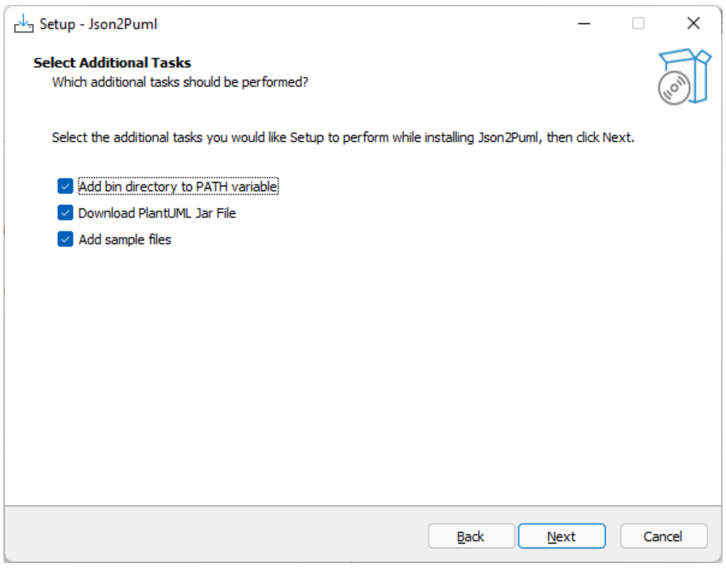
To continue, click Next. If you would like to select a different folder, click Browse.

C:\Users\jensf\AppData\Local\Programs\Json2Puml

At least 20,5 MB of free disk space is required.

Figure 15 Setup - Definition of installation folder

3.1.3 Definition of installation options



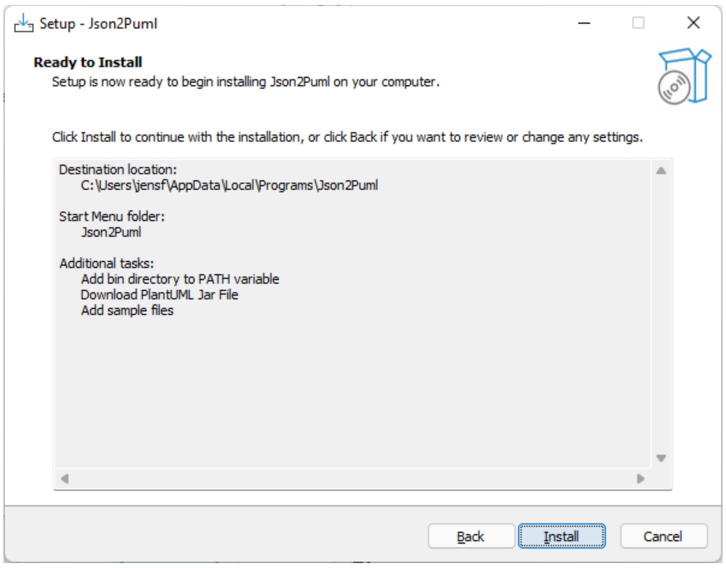
Select Additional Tasks
Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing Json2Puml, then click Next.

- [x] Add bin directory to PATH variable
- [x] Download PlantUML Jar File
- [x] Add sample files

Figure 16 Setup - Definition of installation options

3.1.4 Summary before installation



Ready to Install
Setup is now ready to begin installing Json2Puml on your computer.

Click Install to continue with the installation, or click Back if you want to review or change any settings.

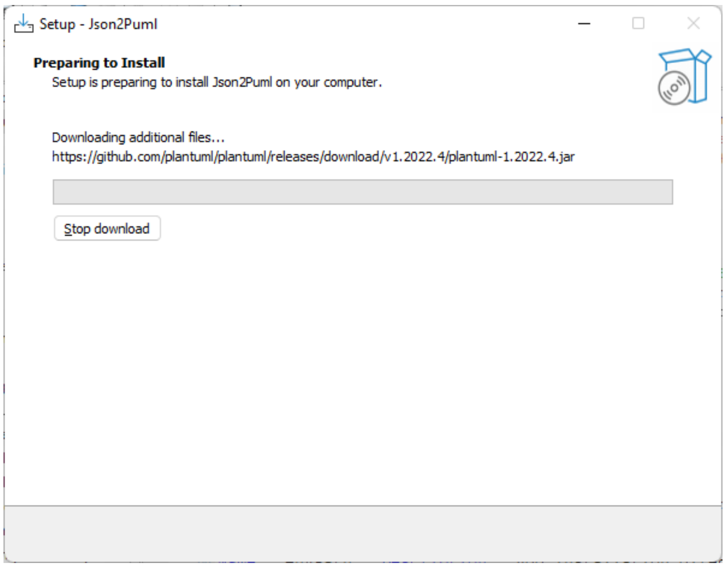
```
Destination location:
    C:\Users\jensf\AppData\Local\Programs\Json2Puml

Start Menu folder:
    Json2Puml

Additional tasks:
    Add bin directory to PATH variable
    Download PlantUML Jar File
    Add sample files
```

Figure 17 Setup - Summary before installation

3.1.5 Status screen while the installation is running

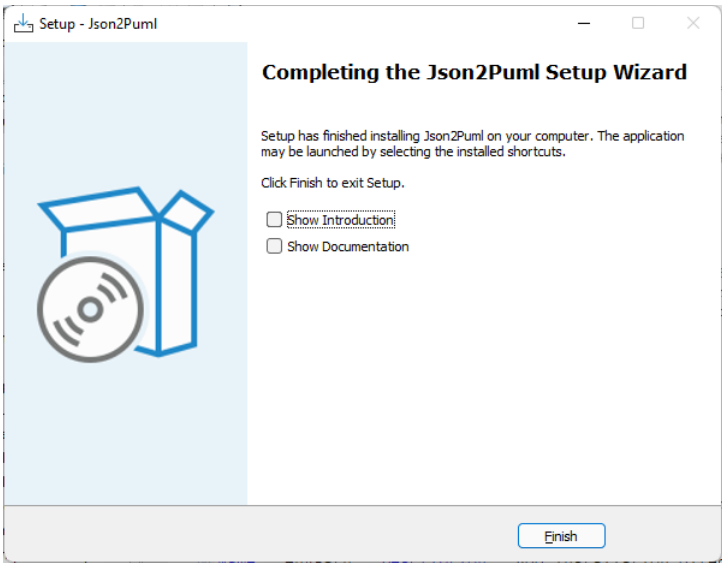

Setup - Json2Puml

Preparing to Install
Setup is preparing to install Json2Puml on your computer.

Downloading additional files...
https://github.com/plantuml/plantuml/releases/download/v1.2022.4/plantuml-1.2022.4.jar

Figure 18 Setup - Status screen while installation

3.1.6 Completion screen


Setup - Json2Puml

Completing the Json2Puml Setup Wizard

Setup has finished installing Json2Puml on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

☐ Show Introduction
☐ Show Documentation

Figure 19 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	Binary of json2puml	
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	Main executable
release-notes.txt	documentation	Version history
json2puml introduction.pdf	documentation	Short introduction presentation
json2puml documentation.pdf	documentation	Full documentation
plantuml<version>.jar	plantuml	Java executable to convert the generated puml files into the graphical output.
*definition.json	samples	Examples of definition 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 defintions:

- TMForum (http://ww.tmforum.org)
- SWAPI - The Star Wars API (https://swapi.dev/)

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 specific authentication parameter when using curl (See also 6.6.3 Curl support)	No, manual action needed

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.exe` 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

This is done calling the following command line:

```
Java.exe -jar <parameter> <plantum.jar> <configuration.puml> <format>
```

3.6.3.1 <parameter>

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

<u>Example:</u>

```
-DPLANTUML_LIMIT_SIZE=8192
```

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

1. json2puml command line parameter
   (see 4.2 Command Line Parameters)
2. json2puml definition file
   (see 7.2.5.2 javaRuntimeParameter)

3.6.3.2 <plantuml.jar>

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

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

1. json2puml command line parameter
   (see 4.2 Command Line Parameters)
2. json2puml definition file
   (see 7.2.5.1 plantUmlJarFile)
3. User environment parameter `%PlantUmlJarFile%`
   (see 3.5 Environment Parameters)

3.6.3.3 <configuration.puml>

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

3.6.3.4 <format>

This is the option to define the output format.
This parameter will automatically determined based on the command line parameter `/outputformat` (see 4.2 Command Line Parameters) or the "`outputFormats`" parameter in the configuration file (See 7.2.8.11 outputFormats).

4 How to use

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

This generates the puml file and the corresponding png image.

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

4.1 Help Screen

```
json2puml v1.2.1.24 - 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.

/definitionfile:<file>        Definitionfile which contains the configuration of the mapper
/optionfile:<file>            Optionfile which contains only the configuration of one option which then will
                              be used for generation
/option:<name>                Name of the option group of the definitionfile which should be used to generate
                              the files
/alloptions                   Flag to generate for all defined options
/formatdefinitionfiles        Flag to reformat the used definition files.

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

/curlauthenticationfile:<file> CurlAuthenticationFile which contains the central authentication
                              Configuration when using the curl automations
/curlparameterfile:<file>     CurlParameterFile 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.

/outputpath:<path>            Output path for the generation of files. This parameter overwrites the path
                              configured in the definition file
/outputformat:<format>        Format of the generated Puml converters (Allowed values: png,svg,pdf)
/openoutput:[<format>]        Flag to define if the generated files should be opened after the generation.
                              The files will be opened using the default program to handle the file format.
                              Optional the files to be opened can be restricted by the format types (Allowed
                              values: png,svg,pdf,puml)

/generatedetails:<boolean>    This allows to overwrite the generateDetails property of the inputlistfile
/generatesummary:<boolean>    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

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

4.2 Command Line Parameters

Parameter	Description
`/?`	Shows the help screen with all command line parameters
`/plantumljarfile: <file>`	Defines an alternative path to the PlantUML jar file. If defined this parameter overwrites the corresponding parameter in the definition file.
`/javaruntimeparameter: <param>`	Additional parameter which will be added to the java call when calling the PlantUML jar file to generate the output formats.
`/definitionfile: <file>`	Definition file which contains the configuration of the converter. When the parameter is not defined or the

Parameter	Description
	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.
`/alloptions`	Generate the outcome for every option defined in the configuration file
`/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: 6.6.1 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: 6.6.2 List File)
`/leadingobject: <name>`	Name of the property which should be used as highest level of the JSON objects in the input files. (See: 8.1 leadingObject) This parameter is only needed for the single file conversion.
`/splitinputfile`	Flag to define if the inputfile should be split up in single files. This option is splitting the input file in separate files when the leading structure 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 7.1.3.10 splitIdentifier) This parameter is only needed for the single file conversion.

Parameter	Description
`/splitidentifier: <identifier>`	Name of the property which defines the name/filename of the splitted json element. (See 7.1.3.10 splitIdentifier) This parameter is only needed for the single file conversion.
`/curlauthenticationfile: <file>`	CurlAuthenticationFile which contains the central authentication configuration when using the curl automations (See 6.6.3 Curl support).
`/curlparameterfile: <file>`	CurlParameterFile which contains additional variables to enable a dynamic configuration when using the curl automations (See 6.6.3 Curl support).
`/curlparameter:<name>=<value>`	Single curl command line parameter, defined as name and value. The parameter can be used multiple times to define more than one curl parameter. The command line parameter will overwrite parameter from the parameter file having the same name (See 6.6.3 Curl support).
`/outputformat: <format>`	Comma separated list of output formats to be generated. This parameter overwrites the output format definition in the definition file (See 7.2.8.11 outputFormats)
`/outputpath: <path>`	Outpath for the generation of files. This parameter overwrites the path configured in the definition file. (See 5.1 Output path and 7.2.8.9 outputPath)
`/openoutput: [<format>]`	Flag to define if the generated files should be opened after the generation. The files will be opened using the default program to handle the file format. Optional the list of files to be opened can be restricted by the format types (Allowed values: png, svg, pdf, puml)
`/generatedetails: <boolean>`	Defines if an output should be generated for any included JSON input file. This parameter is only relevant in the list mode, If defined this parameter overwrites the corresponding parameter in the input list file.
`/generatesummary: <boolean>`	Defines if a summary output should be generated as a combination of all included JSON input file.

Parameter	Description
	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 6.5 Filtering)
`/titlefilter: <filter>`	Value to filter/allow only objects where the title matches to this filter value. (see 6.5 Filtering)
`/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
`/generateoutputdefinition`	Flag to define if the merged generator definition should be stored in the output folder.
`/debug`	Flag to define that a converter log file should be generated parallel to the puml file

4.3 QuickStart

The minimum parameter you have to define to use json2puml are the “`/definitionfile`” (which contains the definition how to convert) and the “`/inputfile`” or the “`/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.

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

The output will look similar to this:

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

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


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

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

5 Generated Files

5.1 Output path

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

The output path is a parameter of the current definition option. (see 7.2.8 Single)

The output path can be defined absolute or relative.

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

Example relative path:

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

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

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

Parameters:

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

Example:

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

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

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

5.2 Output suffix

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

The output suffix is a parameter of the current definition option. (see 7.2.8 Single)

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

5.3 Summary file

A summary file will be generated when the list file (see 6.6.2 List File)is used and the `generateSummary` option (see 4.2 Command Line Parameters and 7.2.8 Single) is active.

The default base name of the summary file is `summary.json`. This name will be enhanced with the output suffix of the current definition. The name can be configured when using the operational mode "List File" (see 6.6.2 List File) by using the `summaryFileName` option (see 7.1.2.7 summaryFileName) of the list file.

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

5.4 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 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`".

To generate the additional images an installed java runtime environment must be installed and executable by calling "java.exe".

The path to the needed PlantUML jar file will be defined on the command line parameter /plantumjarfile. (see 4.2 Command Line Parameters).

6 How the system is working

6.1 Basic Flow

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

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

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

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

The program follows the following high-level structure.

Based on the found objects and relations a PlantUML file is generated.

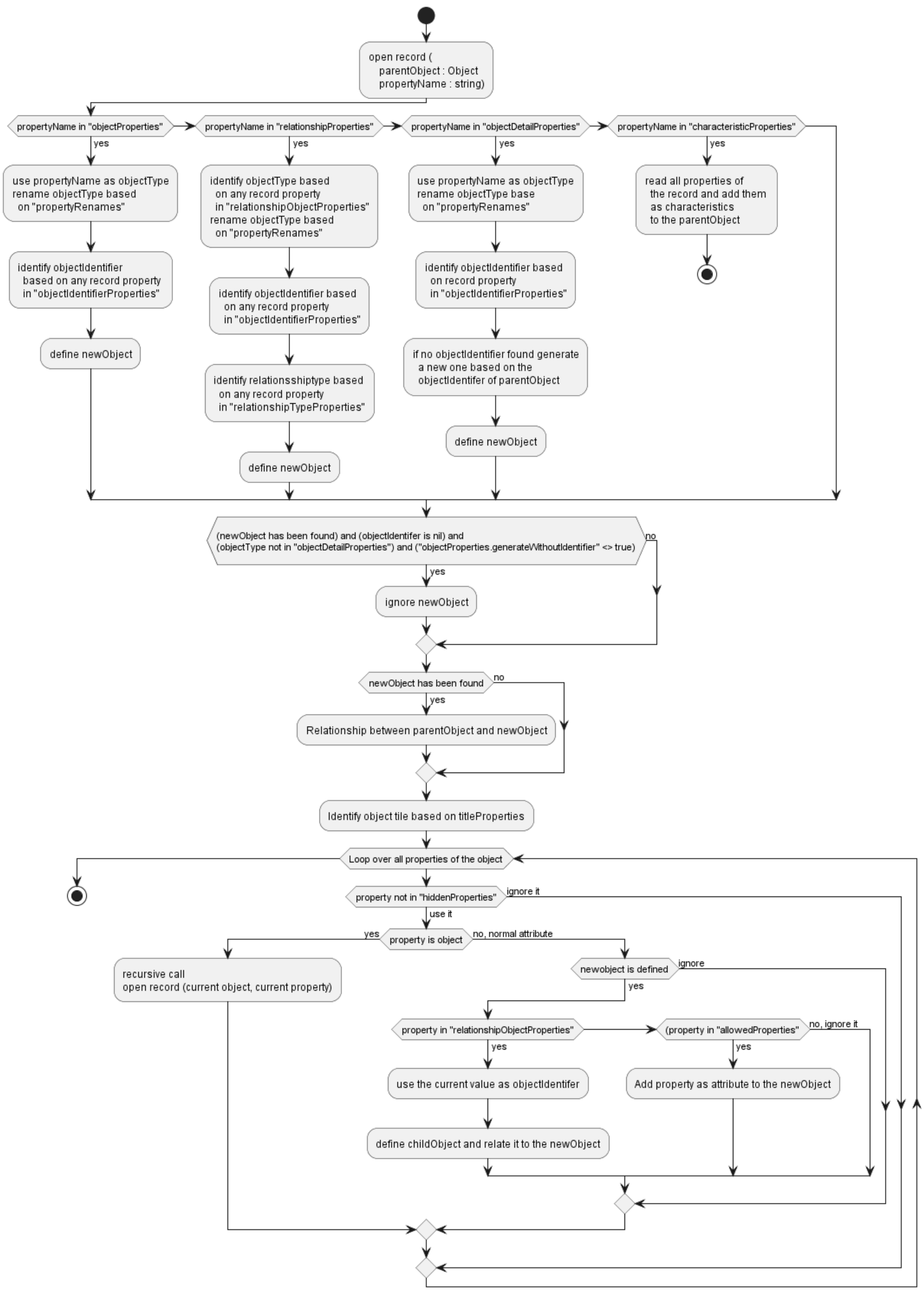

open record (parentObject : Object propertyName : string)
propertyName in "objectProperties"
yes
use propertyName as objectType rename objectType based on "propertyRenames"
identify objectIdentifier based on any record property in "objectIdentifierProperties"
define newObject
propertyName in "relationshipProperties"
yes
identify objectType based on any record property in "relationshipObjectProperties" rename objectType based on "propertyRenames"
identify objectIdentifier based on any record property in "objectIdentifierProperties"
identify relationsshiptype based on any record property in "relationshipTypeProperties"
define newObject
propertyName in "objectDetailProperties"
yes
use propertyName as objectType rename objectType base on "propertyRenames"
identify objectIdentifier based on record property in "objectIdentifierProperties"
if no objectIdentifier found generate a new one based on the objectIdentifer of parentObject
define newObject
propertyName in "characteristicProperties"
yes
read all properties of the record and add them as characteristics to the parentObject
(newObject has been found) and (objectIdentifer is nil) and (objectType not in "objectDetailProperties") and ("objectProperties.generateWithoutIdentifier" <> true)
no
yes
ignore newObject
newObject has been found
no
yes
Relationship between parentObject and newObject
Identify object tile based on titleProperties
Loop over all properties of the object
property not in "hiddenProperties"
ignore it
use it
yes
property is object
no, normal attribute
recursive call open record (current object, current property)
newobject is defined
ignore
yes
property in "relationshipObjectProperties"
yes
use the current value as objectIdentifer
define childObject and relate it to the newObject
(property in "allowedProperties"
no, ignore it
yes
Add property as attribute to the newObject

Figure 20: Basic flow how the converter is working

6.2 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 7.2.8.2 identifyObjectsByTypeAndIdent)
- The type of an object will be identified by the following rules:
    - First the name of the record property or if it is the leading record, the name of the command line parameter “`leadingObject`”.
    - If the JSON record contains a property matching one of the “`objectTypeProperties`” (see also 7.2.8.15 objectTypeProperties) the corresponding value is used.
    - The type of the object can be mapped to consolidated values using the “`objectTypeRenames`” parameter (see **Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.**). This allows to harmonize the object names.
    - If the objects are unique based on the identifier only it can happen that the object is found at different places in the JSON structure.
      In this case it is possible that object type is different for the different places.
      E.g., in TMF there is the pattern of the “`relatedParty`”, which is a reference to a party object. If this is pointing to an `individual` object then this JSON object could have the object type “`individual`”, but both are sharing the same “`identifier`”.
      In this case the order of the elements in the configuration parameter “`objectProperties`” is used to define if the object type can be overwritten or not.
      E.g., the objectProperties list contains the value (in this order) “`individual, party, relatedParty, engagedParty`”. When an object is found at two places with the types “`relatedParty`” and “`individual`” the object will have the type “`individual`”.

6.3 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 (6.2 Object Identification). If an object with the identification is defined/used at many places in the JSON structures multiple relations will be shown to this object.

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

Based on the name of the property and the relationship type it is possible to defined based on the parameter "`relationshipTypeArrowFormats`" (see 7.2.8.22 relationshipTypeArrowFormats) how the arrows should be painted.

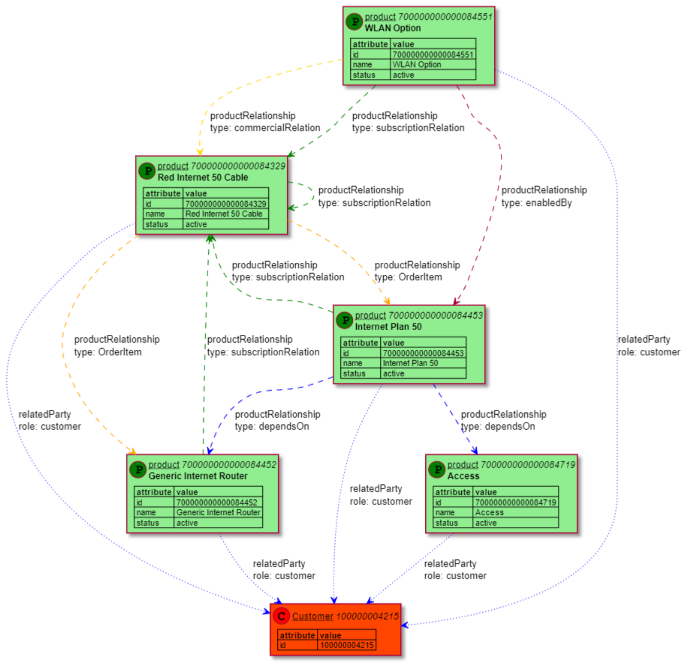

P product 700000000000084551
WLAN Option
attribute | value
id | 700000000000084551
name | WLAN Option
status | active
productRelationship type: commercialRelation
productRelationship type: subscriptionRelation
P product 700000000000084329
Red Internet 50 Cable
attribute | value
id | 700000000000084329
name | Red Internet 50 Cable
status | active
productRelationship type: subscriptionRelation
productRelationship type: enabledBy
productRelationship type: subscriptionRelation
productRelationship type: OrderItem
P product 700000000000084453
Internet Plan 50
attribute | value
id | 700000000000084453
name | Internet Plan 50
status | active
productRelationship type: OrderItem
productRelationship type: subscriptionRelation
relatedParty role: customer
relatedParty role: customer
productRelationship type: dependsOn
productRelationship type: dependsOn
P product 700000000000084452
Generic Internet Router
attribute | value
id | 700000000000084452
name | Generic Internet Router
status | active
relatedParty role: customer
P product 700000000000084719
Access
attribute | value
id | 700000000000084719
name | Access
status | active
relatedParty role: customer
relatedParty role: customer
C Customer 100000004215
attribute | value
id | 100000004215

Figure 21 Object relationship example

6.4 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.

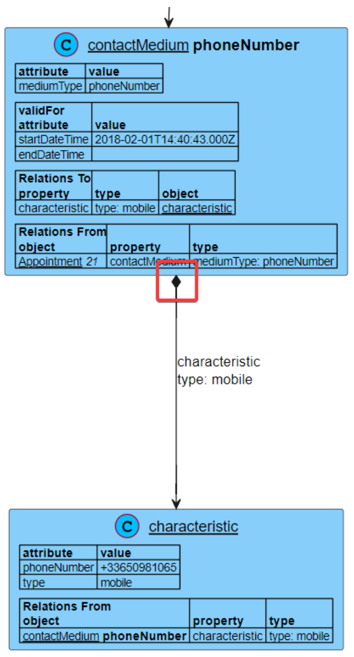
6.4.1 Detail Objects

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 7.2.8.16 objectDetailProperties)

Detailobjects will be visualized with a diamond at the relationship:



attribute	value
mediumType	phoneNumber

validFor attribute	value
startDateTime	2018-02-01T14:40:43.000Z
endDateTime	

Relations To property	type	object
characteristic	type: mobile	characteristic

Relations From object	property	type
Appointment *21*	contactMedium	mediumType: phoneNumber

attribute	value
phoneNumber	+33650981065
type	mobile

Relations From object	property	type
contactMedium **phoneNumber**	characteristic	type: mobile

Figure 22 Visualisation of a detailed object

6.4.2 Characteristics properties

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

There are two different type of characteristic properties:

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

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

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

Characteristic properties will be identified based on the parameter "`characteristicProperties`". (see 7.2.8.25 characteristicProperties)

6.5 Filtering

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

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

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

6.6 Operational Modes

6.6.1 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 8.1 leadingObject

6.6.2 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 6.6.3 Curl support).

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

For further details see have a look at 7.1 List File

6.6.3 Curl support

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 only possible using the list file.

With the list file it is possible to define for every single file definition of the list file that this file should be fetched dynamically before starting the converter.

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.

Based on the global and the single file parameters the following curl command is executed before converting the files.

```
curl <global curlOptions> <single curlOptions> -o <inputFile> --url
      <global curlBaseUrl>/<single curlUrl><global curlUrlAddon>
```

For this purpose, the curl command must be available in the search path of the computer.

6.6.3.1 Curl parametrisation

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

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

For this the curl parameter must be defined in the configured curl parameters in the following pattern "`{$<name>}`". The search of the name is case sensitive.

Example:

- Name : id
- Value : 1
- curlUrl : "/users/{$id}"

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

Curl parameters can be defined based on the following ways:

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

6.6.3.1.1 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 7.3 Curl parameter file

6.6.3.1.2 Curl command line parameter

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

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

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

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

6.6.3.1.3 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 7.1.3.7 curlOutputParameter

6.6.3.1.4 Curl authorisation file

The curl authorisation file allows to separate the user specific authorisation 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.

See 7.4 Curl authentication file

6.6.3.2 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 (7.1.2.7 summaryFileName) and for the input file name of the SingleListFile (7.1.3.1 inputFile).

As a restriction: For a single file the `curlOutParameter` of this file cannot be used for the filename of this file.

6.6.3.3 Curl Example

This example shows how two files are fetched from a api using a dynamic parameter from the curl parameter file and personalized authorisation from the curl authorisation file.

6.6.3.3.1 List file

```
{
    "generateDetails": "false",
    "generateSummary": "true",
    "summaryFileName": "summary_{$username}",
    "curlOptions": "--noproxy \"*\" --token {$token}",
    "curlBaseUrl": "https://curlsample.com/",
    "input": [
        {
            "inputFile": "todos_{$id}.json",
            "leadingObject": "todo",
            "curlUrl": "todos?userId={$id}"
        },
        {
            "inputFile": "users_{$id}.json",
            "leadingObject": "user",
            "curlUrl": "/users/{$id}",
            "curlOutputParameter": {
                "username": "name"
            }
        }
    ]
}
```

6.6.3.3.2 Curl parameter file

```
{
      "id": "1"
}
```

6.6.3.3.3 Curl authentication file

```
[
    {
        "baseUrl": "https://curlsample.com",
        "parameter": {
            "${token}": "dasdgasdasdjasdsad==",
            "${clientsecret}": "jens",
            "${clientid}": "sdfsddasweqwe00"
        }
    },
    {
        "baseUrl": "http://unknown.com",
        "parameter": {
            "${authentication}": "jens"
        }
    }
]
```

6.6.3.3.4 Result

First curl command:

```
curl --noproxy "*" --token dasdgasdasdjasdsad== -o todos_1.json
      --url https://curlsample.com/ todos?userId=1
```

The output file will be named `todos_1.json`.

Second curl command:

```
curl --noproxy "*" --token dasdgasdasdjasdsad== -o users_1.json
      --url https://curlsample.com/users/1
```

The output file will be named `users_1.json`.

The summary file will be named `summary_john.json` based on the username content of the api call result in the file `users_1.json`.

6.6.4 Detail / Summary generation

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

Both modes can be independently activated or deactivated.

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

1. Command line parameter /generateDetails & /generateSummary
2. The corresponding parameters of the single list file (see 7.1.2.1 generateSummary and 7.1.2.2 generateDetails)

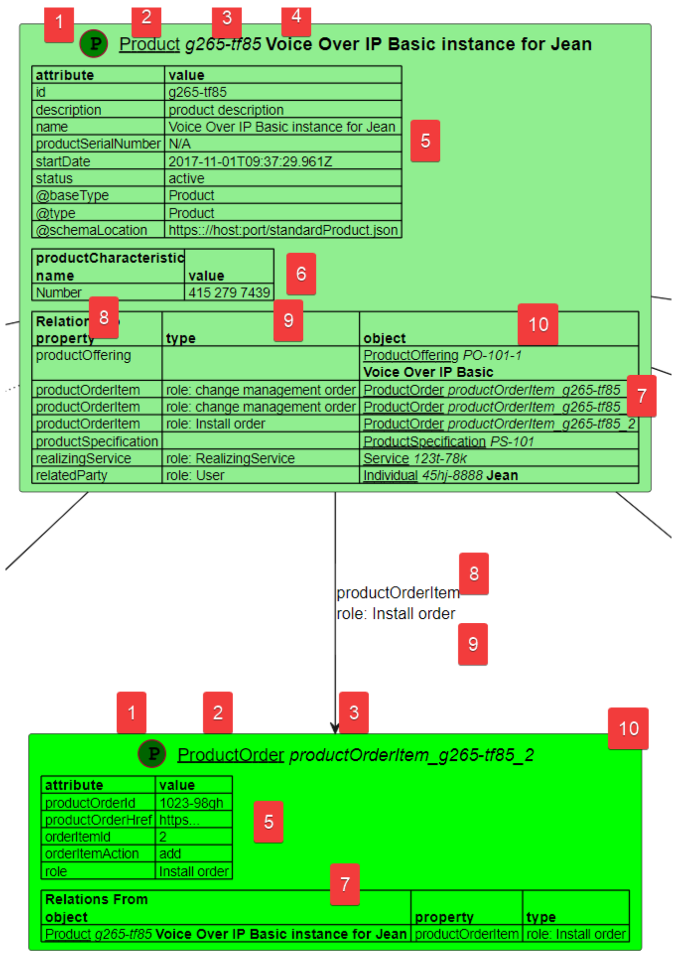
6.6.4.1 Generate Details

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

6.6.4.2 Generate Summary

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

6.7 Generated Outcome



Product *g265-tf85* **Voice Over IP Basic instance for Jean**

attribute	**value**
id	g265-tf85
description	product description
name	Voice Over IP Basic instance for Jean
productSerialNumber	N/A
startDate	2017-11-01T09:37:29.961Z
status	active
@baseType	Product
@type	Product
@schemaLocation	https:://host:port/standardProduct.json

productCharacteristic name	**value**
Number	415 279 7439

Relation property	**type**	**object**
productOffering		ProductOffering *PO-101-1* **Voice Over IP Basic**
productOrderItem	role: change management order	ProductOrder *productOrderItem_g265-tf85*
productOrderItem	role: change management order	ProductOrder *productOrderItem_g265-tf85*
productOrderItem	role: Install order	ProductOrder *productOrderItem_g265-tf85_2*
productSpecification		ProductSpecification *PS-101*
realizingService	role: RealizingService	Service *123t-78k*
relatedParty	role: User	Individual *45hj-8888* **Jean**

ProductOrder *productOrderItem_g265-tf85_2*

attribute	**value**
productOrderId	1023-98gh
productOrderHref	https...
orderItemId	2
orderItemAction	add
role	Install order

Relations From object	**property**	**type**
Product *g265-tf85* **Voice Over IP Basic instance for Jean**	productOrderItem	role: Install order

Figure 23 Generated Output Example

1. Object Icon
   Color is based on 7.2.13.2 iconColor
2. Object Type
   Based on 7.2.8.15 objectTypeProperties, 7.2.8.13 objectProperties
3. Object Ident
   Based on 7.2.8.18 objectIdentifierProperties
4. Object Title
   Based on 7.2.8.14 objectTitleProperties
5. List of Object Attributes
   Based on 7.2.8.12 attributeProperties
6. Characteristics
   Based on 7.2.8.25 characteristicProperties
7. From / To Relationship Lists
8. Relationship Property
9. Relationship Type
   Based on 7.2.8.21 relationshipTypeProperties
10. Object information of the related object

7 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.

7.1 List File

This file allows to define a list of files to imported into a grouped definition.

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 6.6.3 Curl support.

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

7.1.1 Data model

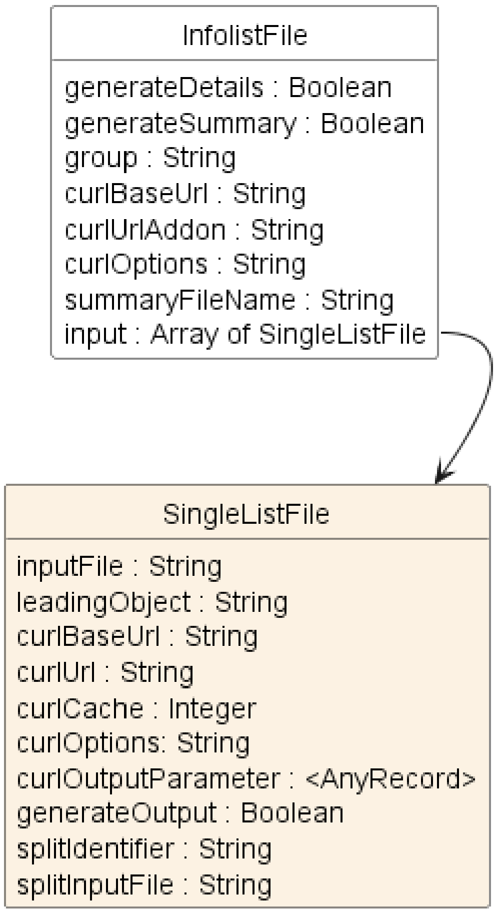


Figure 24: Data model list file

7.1.2 InfoListFile

7.1.2.1 generateSummary

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

7.1.2.2 generateDetails

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

7.1.2.3 curlBaseUrl

Name	Cardinality	Datatype	Default
curlBaseUrl	0..1	String	
Description			
Base URL which will be added before all curlUrl parameters (7.1.3.3 curlBaseUrl) of the inputfile to define the full URL. This allows to simplify the URL definition when multiple files from the same URL have to be fetched. This parameter can be overwritten by the single file curlBaseUrl parameter (7.1.3.3 curlBaseUrl) See 6.6.3 Curl support.			

7.1.2.4 curlUrlAddon

Name	Cardinality	Datatype	Default
curlUrlAddon	0..1	String	
Description			
Additional URL part which will be added after all curlUrl parameters (7.1.3.3 curlBaseUrl) of the inputfile to define the full URL. This allows to simplify the URL definition when common definitions like authentications needs to be added to all URL's of the list file. See 6.6.3 Curl support.			

7.1.2.5 curlOptions

Name	Cardinality	Datatype	Default
curlOptions	0..1	String	
Description			
Additional parameter which will be added to all curl commands of the input list. This could be used to add common additional parameters like authentication information to the curl command. See 6.6.3 Curl support.			

7.1.2.6 Group

Name	Cardinality	Datatype	Default
Group	0..1	String	
Description			
Optional name to group the generated files by enhancing the generated file names and folders with the group (see 7.2.8.9 outputPath and 7.2.8.10 outputSuffix). This could be useful if multiple list files are used in one folder.			

7.1.2.7 summaryFileName

Name	Cardinality	Datatype	Default
summaryFileName	0..1	String	summary.json
Description			
Name of the generated summary file. This name will be enhanced with the output suffix of the current definition (see also 5.3 Summary file). When using the curl option, the filename can be enhanced with curl parameter values. See 6.6.3 Curl support.			

7.1.2.8 Input

Name	Cardinality	Datatype	Default
Input	0..*	SingleListFile	
Description			
List of file definitions to be imported. See 7.1.3 SingleListFile			

7.1.3 SingleListFile

7.1.3.1 inputFile

Name	Cardinality	Datatype	Default
inputFile	0..1	String	
Description			
Filter to find the JSON input files. The path to the files can be relative or absolute defined. Relative paths will be expanded based on the source path of the list input file. When using the curl option, the filename is not allowed to have filters. When using the curl option, the filename can be enhanced with curl parameter values. See 6.6.3 Curl support			

7.1.3.2 leadingObject

Name	Cardinality	Datatype	Default
leadingObject	0..1	string	
Description			
Name of the leadingObject to be used for all input files found based on the “`inputFile`” parameter. See 8.1 leadingObject			

7.1.3.3 curlBaseUrl

Name	Cardinality	Datatype	Default
curlBaseUrl	0..1	string	
Description			
Base URL which will be added before the curlUrl parameters (7.1.3.3 curlBaseUrl) of the inputfile to define the full URL. This parameter overwrites the curlBaseUrl parameter of the list filed (7.1.2.3 curlBaseUrl) See 6.6.3 Curl support			

7.1.3.4 curlUrl

Name	Cardinality	Datatype	Default
curlUrl	0..1	string	
Description			
Url to fetch the json file from. See 6.6.3 Curl support			

7.1.3.5 curlOptions

Name	Cardinality	Datatype	Default
curlOptions	0..1	string	
Description			
Additional parameters added to the curl command when fetching the json data. See 6.6.3 Curl support.			

7.1.3.6 curlCache

Name	Cardinality	Datatype	Default
curlCache	0..1	Integer	0
Description			
Number of seconds while the generated curl result file will be cached and not refreshed. See 6.6.3 Curl support			

7.1.3.7 curlOutputParameter

Name	Cardinality	Datatype	Default
curlOutputParameter	0..1	<anyRecord>	0
Description			
The curlOutputParameter can any be any valid json record structure. This parameter defines which additional curl parameter should be read from the current output file and been added to the list of curl parameter. For each element of this record the name will be used as curl parameter name and the value will be used to fetch the curl parameter value from the current output file. For reading the value the path can be defined using a recursive `<name>.<subname>.<subsubname>` pattern. Example output file:			

Name	Cardinality	Datatype	Default

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

Example curlOutputParameter:

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

This would lead to the following curl parameters:

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

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

See 6.6.3 Curl support

7.1.3.8 generateOutput

Name	Cardinality	Datatype	Default
generateOutput	0..1	Boolean	True
Description			
Flag to define if an output should be generated for this file or that it should be included into the summary file. This parameter could when an oAuth call is used to fetch an access token with the first curl command and then use this token as parameter for all further curl commands. In this case the result of the oAuth token generation should not be included into the generated output. See 6.6.3 Curl support			

7.1.3.9 splitInputFile

Name	Cardinality	Datatype	Default
splitInputFile	0..1	Boolean	
Description			
This option can be used to split datafiles into more detailed datafiles. This option is only possible/valid when the datafile has a JSON array on the highest level. If the top-level JSON structure is not an array the parameter will be ignored. This option is only valid when generatedetails (see 0 generateDetails) is activated. A common scenario for this use case is search/get results returning multiple objects of the same type. This option allows to generate detail data files and based on that generate separate images for every instance of the result set. The generated filename of the splited files will be enhanced with an identifier and sequence number (to make the filename unique). The identifier will be based on the “`splitIdentifier`” parameter (see 7.1.3.10 splitIdentifier).			

7.1.3.10 splitIdentifier

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

Example code block:

```
{
    "id": "875153820701008909",
    "ticketType": [
        {
            "name": "Incident"
        },
        {
            "name": "Admin"
        }
    ],
```

7.1.4 Simple List File Example

```json
{
  "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"
    }
  ]
}
```

7.2 Configuration File

This configuration 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.

7.2.1 Merging of options

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

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

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

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

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

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

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

<u>Example Merge Mode:</u>

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

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

<u>Example Replace Mode:</u>

```
{
  "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 7.2.11 ObjectFormat)

7.2.2 Identifying an Option

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

The command line parameter `/option` (see 4.2 Command Line Parameters) is used first. If this parameter is not defined the parameter “`defaultOption`” from the configuration file is used (see 7.2.4.3 defaultOption).

This value is then used to search in the “`options`” list of the configuration file (based on the “`optionName`” property of the options list (see 7.2.4.1 options).

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

7.2.3 Data model

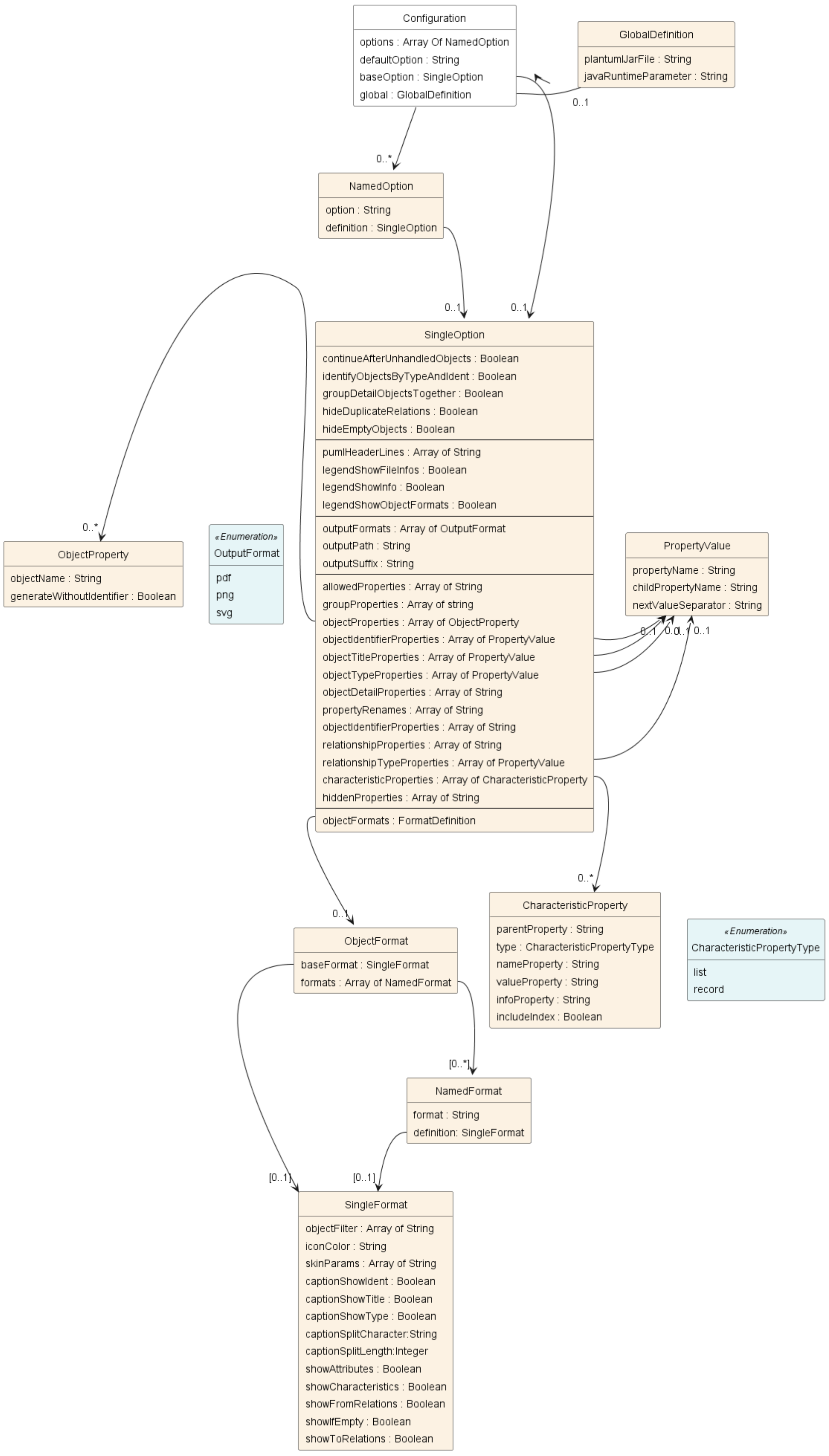

Configuration
options : Array Of NamedOption
defaultOption : String
baseOption : SingleOption
global : GlobalDefinition
GlobalDefinition
plantumlJarFile : String
javaRuntimeParameter : String
0..1
0..*
NamedOption
option : String
definition : SingleOption
0..1
0..1
SingleOption
continueAfterUnhandledObjects : Boolean
identifyObjectsByTypeAndIdent : Boolean
groupDetailObjectsTogether : Boolean
hideDuplicateRelations : Boolean
hideEmptyObjects : Boolean
pumlHeaderLines : Array of String
legendShowFileInfos : Boolean
legendShowInfo : Boolean
legendShowObjectFormats : Boolean
outputFormats : Array of OutputFormat
outputPath : String
outputSuffix : String
allowedProperties : Array of String
groupProperties : Array of string
objectProperties : Array of ObjectProperty
objectIdentifierProperties : Array of PropertyValue
objectTitleProperties : Array of PropertyValue
objectTypeProperties : Array of PropertyValue
objectDetailProperties : Array of String
propertyRenames : Array of String
objectIdentifierProperties : Array of String
relationshipProperties : Array of String
relationshipTypeProperties : Array of PropertyValue
characteristicProperties : Array of CharacteristicProperty
hiddenProperties : Array of String
objectFormats : FormatDefinition
0..*
ObjectProperty
objectName : String
generateWithoutIdentifier : Boolean
«Enumeration»
OutputFormat
pdf
png
svg
PropertyValue
propertyName : String
childPropertyName : String
nextValueSeparator : String
0..1
0..1
0..1
0..1
0..*
CharacteristicProperty
parentProperty : String
type : CharacteristicPropertyType
nameProperty : String
valueProperty : String
infoProperty : String
includeIndex : Boolean
«Enumeration»
CharacteristicPropertyType
list
record
0..1
ObjectFormat
baseFormat : SingleFormat
formats : Array of NamedFormat
[0..*]
NamedFormat
format : String
definition: SingleFormat
[0..1]
[0..1]
SingleFormat
objectFilter : Array of String
iconColor : String
skinParams : Array of String
captionShowIdent : Boolean
captionShowTitle : Boolean
captionShowType : Boolean
captionSplitCharacter:String
captionSplitLength:Integer
showAttributes : Boolean
showCharacteristics : Boolean
showFromRelations : Boolean
showIfEmpty : Boolean
showToRelations : Boolean

Figure 25 Main Configuration Model

7.2.4 Configuration

7.2.4.1 options

Name	Cardinality	Datatype	Default
Options	0..*	NamedOption	
Description			
This array defines the list of possible options which could be used for analysing and formatting the output. Which option is used can be configured via command line parameter `/option` (see 4.2 Command Line Parameters). If the command line parameter is not used the parameter `defaultOption` is used instead (see 7.2.4.3 defaultOption).			

7.2.4.2 baseOption

Name	Cardinality	Datatype	Default
baseOption	0..1	SingleOption	
Description			
This array defines the list of possible options which could be used for analysing and formatting the output. Which option is used can be configured via command line parameter `/option` (see 4.2 Command Line Parameters). If the command line parameter is not used the parameter `defaultOption` is used instead (see 7.2.4.3 defaultOption).			

7.2.4.3 defaultOption

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

7.2.4.4 global

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

7.2.5 GlobalDefinition

7.2.5.1 plantUmlJarFile

Name	Cardinality	Datatype	Default
plantUmlJarFile	0..1	String	
Description			
Name and path of the PlantUML jar file which will be used to generate the PlantUML output files. For further details see 5.5 PlantUML output files.			

7.2.5.2 javaRuntimeParameter

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

7.2.6 NamedOption

Record to define an entry in the options list. Each entry is defined by a `optionName` and the corresponding `definition`.
(See 7.2.1 Merging of option)

7.2.6.1 optionName

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

7.2.6.2 baseOption

Name	Cardinality	Datatype	Default
definition	0..1	SingleOption	
Description			

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

7.2.7 PropertyValue

Record to define how the related information should be calculated.

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

- 7.2.8.14 objectTitleProperties
- 7.2.8.15 objectTypeProperties
- 7.2.8.18 objectIdentifierProperties
- 7.2.8.21 relationshipTypeProperties

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

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

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

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

<u>Example:</u>

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

<u>Explanation:</u>

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`” .

<u>Samples:</u>

No	JSON	Outcome

1a	`"characteristic":`
	`{`
	`"name": "Product",`
	`"value": "Voice Over IP Basic",`
		"Voice Over IP Basic"

1b	`"product":`
	`{`
	`"name": "Prodcuct",`
	`"value": "Voice Over IP Basic",`
		N/A
		(because of the not matching object type product)

2	`"product":`
	`{`
	`"id": "g265-tf85",`
	`"name": "VOIP",`
	`"fullName": "Voice Over IP Basic"`
	`"productSerialNumber": "N/A",`
		“VOIP”
		“`fullName`” is ignored because the “`name`” is coming first in the definition list.

No	JSON	Outcome

3	
	```
	"ticketType": [
	        {
	            "name": "Incident"
	        },
	        {
	            "name": "Admin"
	        },
	        {
	            "name": "Escalation"
	        }
	]
	```
		“Incident.Admin.Escalation”

4	
	```
	"transition": {
	        "sourceStatus": {
	            "name": "Draft"
	        },
	        "targetStatus": {
	            "name": "New"
	        },
	}
	```
		“Draft->New”

7.2.7.1 propertyName

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

7.2.7.2 childPropertyName

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

Name	Cardinality	Datatype	Default

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

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

propertyName	childPropertyName	nextValueSeparator
escalation.ticket	ticketType.name	.

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

7.2.7.3 *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.			

7.2.8 SingleOption

7.2.8.1 *continueAfterUnhandledObjects*

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

7.2.8.2 *identifyObjectsByTypeAndIdent*

Name	Cardinality	Datatype	Default
identifyObjectsByTypeAndIdent	0..1	Boolean	False
Description			
Defines if the objects should be identified and linked based on the object type and the object ident of the object or only based on the object ident. To use the object ident only is only possible/valid if the identifiers are unique over all object types. When including the type also the type must be included in the data of an included property. (See 6.2 Object Identification)			

7.2.8.3 *hideDuplicateRelations*

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

7.2.8.4 *hideEmptyObjects*

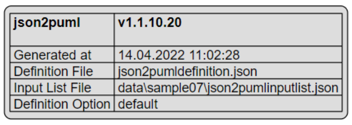
Name	Cardinality	Datatype	Default
hideEmptyObjects	0..1	Boolean	False
Description			
Defines if empty objects (without any attributes, characteristics and relationships) should be hidden or not.			

7.2.8.5 *groupDetailObjectsTogether*

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

7.2.8.6 *legendShowInfo*

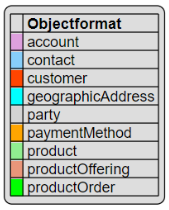
Name	Cardinality	Datatype	Default
legendShowInfo	0..1	Boolean	True
Description			
Defines if the common information block should be generated into the legend of the drawing. Example:			



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

7.2.8.7 *legendShowObjectFormats*

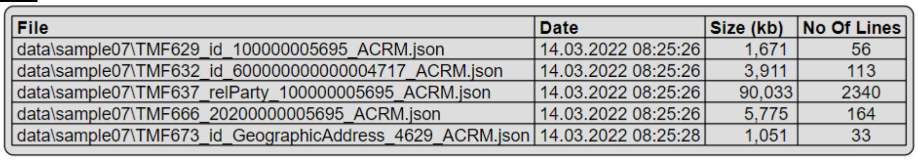
Name	Cardinality	Datatype	Default
legendShowObjectFormats	0..1	Boolean	True
Description			
Defines if the object format color list should be generated into the legend of the drawing. Example:			



	Objectformat
🟪	account
🟦	contact
🟥	customer
🟦	geographicAddress
⬜	party
🟧	paymentMethod
🟩	product
🟧	productOffering
🟩	productOrder

7.2.8.8 *legendShowFileInfos*

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



File	Date	Size (kb)	No Of Lines
data\sample07\TMF629_id_100000005695_ACRM.json	14.03.2022 08:25:26	1,671	56
data\sample07\TMF632_id_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

7.2.8.9 outputPath

Name	Cardinality	Datatype	Default
outputPath	0..1	String	
Description			
The output path could be used to separate the generated files from the original source files. The output path could be defined absolute or relative to the source file of the generation. To structure the directories, it is possible to include the current option, group and detail into the folder name. The following replacements are supported: - <option> Replaced by the name of the use configuration option - <group> Replaced by the corresponding value from the list definition file (see 7.1.2 InfoListFile) or the corresponding command line parameter `/group` (see 4.2 Command Line Parameters) - <detail> Replaced by the corresponding command line parameter `/detail` (see 4.2 Command Line Parameters) Example: `"outputPath": "output\\<group>\\<option>"`			

7.2.8.10 outputSuffix

Name	Cardinality	Datatype	Default
outputSuffix	0..1	String	
Description			
The output suffix will be added to filename of any generated file. This could be used to separate the generated files from the original source files and to separate them if multiple options or filters have been used to generate them. To structure the files, it is possible to include the current option, group and detail into the folder name. The following replacements are supported: - <option> Replaced by the name of the use configuration option - <group> Replaced by the corresponding value from the list definition file (see 7.1.2 InfoListFile) or the corresponding command line parameter `/group` (see 4.2 Command Line Parameters) - <detail> Replaced by the corresponding command line parameter `/detail` (see 4.2 Command Line Parameters) Example: `"outputSuffix": ".<group>.<option>”` Input file : `"tmf666_based.json“` will lead to a filename			

Name	Cardinality	Datatype	Default
"tmf666_based.crm.full.json" based on the group "crm" and the option "full".			

7.2.8.11 *outputFormats*

Name	Cardinality	Datatype	Default
outputFormats	0..*	OutputFormat	
Description			
List of image formats the generator should create based on the created puml file. The pre-configured value can be overwritten by the command line parameter /outputFormats. (see 4.2 Command Line Parameters) For further details see 5.5 PlantUML output files.			

7.2.8.12 *attributeProperties*

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

7.2.8.13 *objectProperties*

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

Name	Cardinality	Datatype	Default

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

7.2.8.14 objectTitleProperties

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

7.2.8.15 objectTypeProperties

Name	Cardinality	Datatype	Default
objectTypeProperties	0..*	PropertyValue	
Description			
List of properties which should be handled as the type of an object. The value of the property will be used as object type of the object. Using the “`objectTypeRenames`” parameter (see 7.2.8.17 objectTypeRenames) the object type could be renamed. (See 6.2 Object Identification)			

7.2.8.16 objectDetailProperties

Name	Cardinality	Datatype	Default
objectDetailProperties	0..*	String	
Description			
List of properties which will be handled as detail object. (see 6.4 Detail Objects / Characteristic properties)			

7.2.8.17 *objectTypeRenames*

Name	Cardinality	Datatype	Default
objectTypeRenames	0..*	String	
Description			
List of name replacements to be used for object type. (See 6.2 Object Identification) The format of the rename property is: “`<oldname>=<newname>`”. <u>Example:</u> `"objectTypeRenames": [ "compositeProduct=product", "defaultPaymentMethod=paymentMethod", "place=geographicAddress", "email=contactMedium", ]`			

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

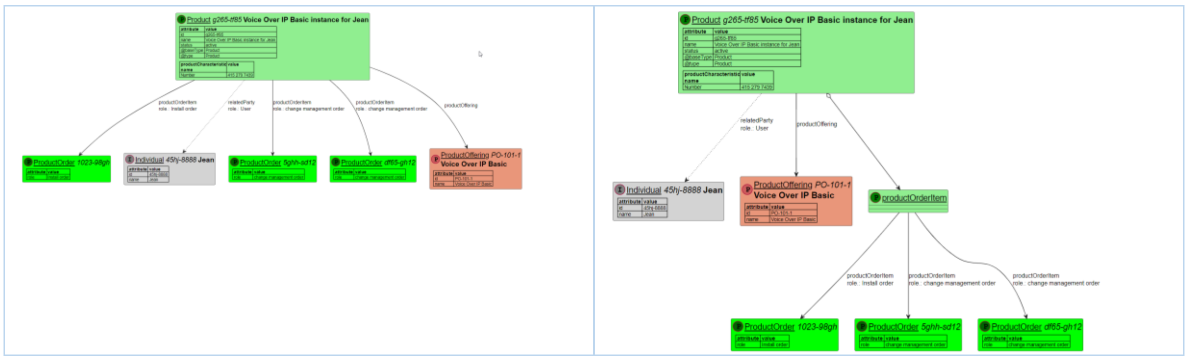
7.2.8.18 *objectIdentifierProperties*

Name	Cardinality	Datatype	Default
objectIdentifierProperties	0..*	PropertyValue	
Description			
List of properties which should be handled as the identifier of an object. The value of the property will be used as identifier of the object. (See 6.2 Object Identification)			

7.2.8.19 *groupProperties*

Name	Cardinality	Datatype	Default
groupProperties	0..*	String	
Description			
A group property allows to structure the generated output in that way that an additional blank object is created which collects the relations of the related detail objects of the same name. <u>Example:</u>			

Without GroupProperty	**With GroupProperty**

Name	Cardinality	Datatype	Default
			

7.2.8.20 *relationshipProperties*

Name	Cardinality	Datatype	Default
relationshipProperties	0..*	String	
Description			
List of properties which should be handled to identify typed relations to another object. A relationship property can, but most not be defined as a separate object. Based on the parameter "`relationshipTypeProperties`" a type of the relationship could be identified. (See 6.3 Object Relationships)			

7.2.8.21 *relationshipTypeProperties*

Name	Cardinality	Datatype	Default
relationshipTypeProperties	0..*	String	
Description			
List of properties which should be handled as the type of a relationship. (See 6.3 Object Relationships)			

7.2.8.22 *relationshipTypeArrowFormats*

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

Name	Cardinality	Datatype	Default

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

(See 6.3 Object Relationships)

7.2.8.23 hiddenProperties

Name	Cardinality	Datatype	Default
hiddenProperties	0..*	String	
Description			
List of properties which will stop the recursive handling of the converter. (see 6.1 Basic Flow)			

7.2.8.24 pumlHeaderLines

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

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

7.2.8.25 characteristicProperties

Name	Cardinality	Datatype	Default
characteristicProperties	0..*	CharacteristicProperty	
Description			
List of properties which will be handled as characteristics. (see 6.4 Detail Objects / Characteristic properties)			

7.2.8.26 *objectFormats*

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

7.2.9 ObjectProperty

7.2.9.1 *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 "`objectTypeRenames`" parameter (see **Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.**) the object type could be renamed. (See 6.2 Object Identification)			

7.2.9.2 *generateWithoutIdentifier*

Name	Cardinality	Datatype	Default
generateWithoutIdentifier	0..1	String	false
Description			
Flag to define if the objects of this type should be generated without having an identifier. In this the identifier will automatically generated based on the identifier of the parent object (enhanced with a unique number) (See 6.2 Object Identification)			

7.2.10 CharacteristicProperty

The characteristic property defines how a property which is identified as a characteristic should be handled. (see 6.4.2 Characteristics properties)

7.2.10.1 *parentProperty*

Name	Cardinality	Datatype	Default
parentProperty	1	String	
Description			

Name	Cardinality	Datatype	Default
Name of the property which should be handled as a characteristic			

7.2.10.2 *type*

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

7.2.10.3 *nameProperty*

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

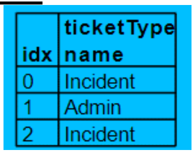
7.2.10.4 *valueProperty*

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

7.2.10.5 *infoProperty*

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

7.2.10.6 includeIndex

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

	ticketType
idx	name
0	Incident
1	Admin
2	Incident

7.2.11 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 7.2.1 Merging of options)

7.2.11.1 baseFormat

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

7.2.11.2 formats

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

7.2.12 NamedFormat

7.2.12.1 formatName

Name	Cardinality	Datatype	Default
formatName	1	7.2.13string	
Description			
Name of the format definition. This name will be used as PlantUML creole to identify the format.			

7.2.12.2 definition

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

7.2.13 SingleFormat

<u>Example:</u>

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

7.2.13.1 objectFilter

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

7.2.13.2 *iconColor*

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

7.2.13.3 *skinParams*

Name	Cardinality	Datatype	Default
skinParams	0..*	string	
Description			

List of skinparams which will be added to the beginning of the PlantUML file to define the format.
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
}
```

7.2.13.4 *captionShowIdent*

Name	Cardinality	Datatype	Default
captionShowIdent	0..1	Boolean	True

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

7.2.13.5 *captionShowTitle*

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

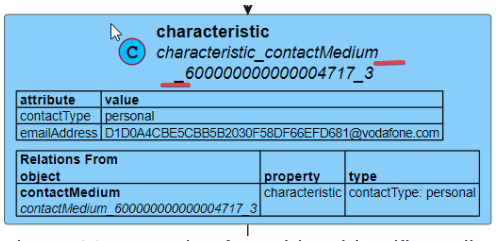
7.2.13.6 *captionShowType*

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

7.2.13.7 *captionSplitCharacter*

Name	Cardinality	Datatype	Default
captionSplitCharacter	0..1	String	
Description			

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

Name	Cardinality	Datatype	Default
 Figure 26 Example of an object identifier split			

characteristic
*characteristic_contactMedium_*
_6000000000000004717_3

attribute	value
contactType	personal
emailAddress	D1D0A4CBE5CBB5B2030F58DF66EFD681@vodafone.com

Relations From object	property	type
contactMedium *contactMedium_6000000000000004717_3*	characteristic	contactType: personal

7.2.13.8 *captionSplitLength*

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

7.2.13.9 *showAttributes*

Name	Cardinality	Datatype	Default
showAttributes	0..1	Boolean	True
Description			
Flag to define if the attribute list should be generated for the object. (see 6.7 Generated Outcome)			

7.2.13.10 *showCharacteristics*

Name	Cardinality	Datatype	Default
showCharacteristics	0..1	Boolean	True
Description			
Flag to define if the characteristics lists should be generated for the object. (see 6.7 Generated Outcome)			

7.2.13.11 *showFromRelations*

Name	Cardinality	Datatype	Default
showFromRelations	0..1	Boolean	True
Description			

Name	Cardinality	Datatype	Default
Flag to define if the from relation list should be generated inside of the object. (see 6.7 Generated Outcome)			

7.2.13.12 showIfEmpty

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

7.2.13.13 showToRelations

Name	Cardinality	Datatype	Default
showAttributes	0..1	Boolean	True
Description			
Flag to define if the to relation list should be generated inside of the object. (see 6.7 Generated Outcome)			

7.3 Curl parameter file

7.4 Curl authentication file

8 Global Definitions

8.1 leadingObject

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

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

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

Example

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

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