Team TheThree - Task 1

Transform the ReqIf (Requirements Interchange Format) file to json format

Requirements

Write a command line program by java or python, the program allows to transform requirements from ReqIf file to JSON file and preserve the values of attributes and object hierarchy.

> Input: Reqif file as sample attachment (Requirements.reqif file)

> Output: JSON file as sample attachment (Json_Output_Sample.json) and program source code

Evaluation

- `[x]` Module Info
- `[x]` All requirements and order
- `[x]` Mandatory attributes
- `[x]` Clean code

Installation

- Python version required `>=3.10`
    - Check your python version:

```
python --version
```

- Upgrade python version:

> `MacOS` : https://macosx-faq.com/how-to-update-python-terminal/

> `Linux` : https://cloudbytes.dev/snippets/upgrade-python-to-latest-version-on-ubuntu-linux

> `Windows`: https://www.geeksforgeeks.org/how-to-update-python-on-windows/

Install the required packages by using `pip`

```
pip install -r requirements.txt
```

Getting Started

```
python main.py -i Requirements.reqif -o Json_Output.json
```

Usage

The program have `2 arguments`. You can check the documentation by using the command

```
python main.py -h
```

```
usage: main.py [-h] [-i INPUT_FILE] [-o OUTPUT_FILE]

options:
  -h, --help            show this help message and exit
  -i INPUT_FILE, --input_file INPUT_FILE
                        Directory to input file. Accepts file *.reqif or
*.xml only
  -o OUTPUT_FILE, --output_file OUTPUT_FILE
                        Directory to output *.json file.
```

Example command:

```
python main.py -i Requirements.reqif -o Json_Output.json
```

Experiment

> In first step

We read and load the file `*.reqif` which such as file `*.xml` by using 3rd party libraries `xmltodict`

> Next step is reading `module info`

We find the following information by using these functions:

```
def find_name_module(data):
    spec = list(find_keys(data, 'SPECIFICATIONS'))[0]['SPECIFICATION']

    return spec.get(NAME_TAG)
```

```
def find_type_module(data):

    return list(
        find_keys(dict(data), 'SPECIFICATION-TYPE'))[0].get(NAME_TAG)
```

We see that:

- The information `Module Name` is stored in `SPECIFICATION`

```
    <SPECIFICATIONS>
            <SPECIFICATION LONG-NAME="ECU_Requirement" LAST-CHANGE="2023-
05-22T03:17:41.877Z" IDENTIFIER="_8f06d09f-ce45-4c07-9d71-a656c6e5c3aa">
    </SPECIFICATIONS>
```

- The information `Module Type` is stored in `SPECIFICATION-TYPE`

```
  <SPECIFICATION-TYPE LONG-NAME="MO_RS" LAST-CHANGE="2020-12-
16T04:03:43.649Z" IDENTIFIER="_8e22da9e-0198-40c8-b577-02cca0635202"
DESC="Module for MO Requirements  PTSA 2.0">
  </SPECIFICATION-TYPE>
```

- Both functions have the same parameters is the data which load in the first step

> Next is find the `List Artifact Info`

- The requirements said that

```
The hierarchy of the objects is defined in section `SPEC-HIERARCHY`.
```

- Look at more details we can see that The top-level element is `CHILDREN,` which contains a list of `SPEC-HIERARCHY` elements.
    - Each `SPEC-HIERARCHY` element represents a level in the hierarchy and contains the following properties:
    - `@LAST-CHANGE`: Represents the timestamp of the last change made to the element.
    - `@IDENTIFIER`: Unique identifier for the element.
    - `OBJECT`: Contains an object with a `SPEC-OBJECT-REF` property, which likely references another object.
    - `CHILDREN`: May contain nested `SPEC-HIERARCHY` elements or a single `SPEC-HIERARCHY` element.
- The structure seems to be recursive, as `SPEC-HIERARCHY` elements can contain more `SPEC-HIERARCHY` elements within their `CHILDREN` property.

- In the implementation, we get the `SPEC-OBJECT-REF` as a list followed by the `SPEC-HIERARCHY` order element

```
def get_spec_object_ref_hierarchy(data, hierarchy=None):
    if hierarchy is None:
        hierarchy = []

    if isinstance(data, list):
        for item in data:
            get_spec_object_ref_hierarchy(item, hierarchy)

    elif isinstance(data, dict):
        object_ref = data.get("OBJECT", {}).get("SPEC-OBJECT-REF")
        if object_ref:
            hierarchy.append(object_ref)

        children = data.get("CHILDREN")

        if children:
            get_spec_object_ref_hierarchy(
                children.get('SPEC-HIERARCHY'), hierarchy)

    return hierarchy
```

- After got the hierarchy of the objects, we can iterate over all the content of the objects which contain in the `SPEC-OBJECTS`
- We can got the type of the `SPEC-OBJECT` which stored in the

```
<TYPE>
    <SPEC-OBJECT-TYPE-REF>
        {REFERENCE}
    </SPEC-OBJECT-TYPE-REF>
</TYPE>
```

and easily mapping the `{REFERENCE}` by looking up the `SPEC-OBJECT-TYPE`

In each `SPEC-OBJECT` it have 4 `VALUES` attributes is

- `ATTRIBUTE-VALUE-XHTML`
- `ATTRIBUTE-VALUE-DATE`
- `ATTRIBUTE-VALUE-STRING`
- `ATTRIBUTE-VALUE-ENUMERATION`

And following the description:

```
Attributes are specified in sub section {SPEC-OBJECT-TYPE} of {SPEC-TYPES},
the name is defined in {LONG-NAME}. Each attribute consists of {DEFINITION}
and {THE-VALUE}. The {DEFINITION} references to section {DATATYPES}. {THE-
VALUE} contains the value of attribute
```

For each `ATTRIBUTE`, we mapping the `VALUE` and the `DEFINITION` by the function with parameters is `spec_attrs` is the `SPEC-ATTRIBUTES` in the `SPEC-OBJECT-TYPE` abd `spec_obj_values` is the `VALUE` of the `SPEC-OBJECT`

```
def mapping_attr_definition(spec_attrs, spec_obj_values, attr_key):
    result = []

    attrs = spec_obj_values.get(attr_key)
    attr_name = attr_key.split('-')[-1]

    DEFINITION_TAG = f'ATTRIBUTE-DEFINITION-{attr_name}'

    if isinstance(attrs, dict):
        attrs = [attrs]

    for attr in attrs:
        for def_attr in spec_attrs.get(DEFINITION_TAG):
            attr_ref = attr['DEFINITION'][f'{DEFINITION_TAG}-REF']

            if attr_ref == def_attr.get(IDENTIFIER_TAG):
                key = def_attr.get(NAME_TAG)
                value = attr.get('THE-VALUE')

                if value is None:
                    value = attr.get('@THE-VALUE')

                key, value = refactor_key_value(key, value, attr_name,
attr)

                if key is not None:
                    result.append({key: value})

    return result
```

Parallel with mapping `VALUE` and `DEFINITION` attributes we also refactor the `key` and `value` attributes which will save in the result.

```
def refactor_key_value(key, value, attr_name, attr):
    match key:
        # XHTML
        # - ReqIF.ChapterName is ReqIF.Text in the "Heading"
        case 'ReqIF.Text':
            key = 'ReqIF.Text'
            value = xmltodict.unparse(value, pretty=True)[39:]
        case 'ReqIF.Name':
            key = 'Title'
            value = value.get('div', {}).get('#text', '')
        case 'ReqIF.ChapterName':
            key = 'ReqIF.Text'
            value = xmltodict.unparse(value, pretty=True)[39:]
        case 'ReqIF.Description':
            key = 'Description'
            value = value.get('div', {}).get('#text', '')

        # DATE
        case 'ReqIF.ForeignCreatedOn':
            key = 'Created On'
        case 'ReqIF.ForeignModifiedOn':
            key = 'Modified On'

        # STRING
        case 'ReqIF.ForeignID':
            key = 'Identifier'
            value = int(value)
        case 'ReqIF.ForeignCreatedBy':
            key = 'Creator'
        case 'ReqIF.ForeignModifiedBy':
            key = 'Contributor'

        # ENUMERATION
        # - Artifact Format not needed to collect
        case 'Artifact Format':
            key = None

        # OTHERWISE
        case _:
            if attr_name == 'STRING':
                key = key

            elif attr_name == 'ENUMERATION':
                value = find_enum_value(
                    attr['VALUES']['ENUM-VALUE-REF'])

            else:
                key = None

    return key, value
```

> Finally

Store the result in file `*.json` by calling `json.dump()`

```
  json_data = json.dumps({
        "Module Name": find_name_module(data_dict),
        "Module Type": find_type_module(data_dict),
        "List Artifact Info": find_list_artifact_info(data_dict)
    })

    with open(OUT_SRC, "w") as json_file:
        json_file.write(json_data)
```

Demo

Here is our result which runs in the sample file `Requirements.reqif`


```
{
  "Module Name": "ECU_Requirement",
  "Module Type": "MO_RS",
  "List Artifact Info": [
    {
      "Attribute Type": "Heading",
      "Contributor": "kit7fe",
      "Created On": "2019-10-08T06:18:45.677Z",
      "Creator": "kit7fe",
      "Description": "",
      "Identifier": 629021,
      "Modified On": "2019-10-08T06:18:45.677Z",
      "ReqIF.Text": "<div
xmlns=\"http://www.w3.org/1999/xhtml\">\n\t<p>General Overview / Document
Scope</p>\n</div>",
      "Title": "General Overview / Document Scope"
    },
    {
      "Attribute Type": "Heading",
      "Contributor": "kit7fe",
      "Created On": "2019-10-08T06:18:45.662Z",
      "Creator": "kit7fe",
      "Description": "",
      "Identifier": 629020,
      "Modified On": "2019-10-08T06:18:45.662Z",
      "ReqIF.Text": "<div
xmlns=\"http://www.w3.org/1999/xhtml\">\n\t<p>Document Scope</p>\n</div>",
      "Title": "Document Scope"
    },
    {
      "Attribute Type": "Information",
      "Contributor": "kit7fe",
      "Created On": "2019-10-08T06:18:45.662Z",
```

```
      "Creator": "kit7fe",
      "Description": "",
      "Identifier": 629016,
      "Modified On": "2019-10-08T06:18:45.662Z",
      "ReqIF.Text": "<div
xmlns=\"http://www.w3.org/1999/xhtml\">\n\t<p>&lt;put below a first
description of the scope for ECU requirement specification&gt;
</p>\n</div>",
      "Title": "<put below a first description of the scope for software
requirement specification>"
    },
    {
      "Attribute Type": "Heading",
      "Contributor": "kit7fe",
      "Created On": "2019-10-08T06:18:45.662Z",
      "Creator": "kit7fe",
      "Description": "",
      "Identifier": 629012,
      "Modified On": "2019-10-08T06:18:45.662Z",
      "ReqIF.Text": "<div
xmlns=\"http://www.w3.org/1999/xhtml\">\n\t<p>Document Specific
Glossary</p>\n</div>",
      "Title": "Document Specific Glossary"
    },
    {
      "Attribute Type": "Information",
      "Contributor": "kit7fe",
      "Created On": "2019-10-08T06:18:45.677Z",
      "Creator": "kit7fe",
      "Description": "",
      "Identifier": 629013,
      "Modified On": "2019-10-08T06:18:45.677Z",
      "ReqIF.Text": "<div
xmlns=\"http://www.w3.org/1999/xhtml\">\n\t<p>&lt;put below a definition of
first glossary specific terms&gt;</p>\n</div>",
      "Title": "<put below a definition of first glossary specific terms>"
    },
    {
      "Attribute Type": "Heading",
      "Contributor": "kit7fe",
      "Created On": "2019-10-08T06:18:45.662Z",
      "Creator": "kit7fe",
      "Description": "",
      "Identifier": 629019,
      "Modified On": "2019-10-08T06:18:45.662Z",
      "ReqIF.Text": "<div
xmlns=\"http://www.w3.org/1999/xhtml\">\n\t<p>System
Requirements</p>\n</div>",
      "Title": "System Requirement"
    },
    {
      "Attribute Type": "Information",
      "Contributor": "kit7fe",
      "Created On": "2019-10-08T06:18:45.677Z",
```

```
      "Creator": "kit7fe",
      "Description": "",
      "Identifier": 629017,
      "Modified On": "2019-10-08T06:18:45.677Z",
      "ReqIF.Text": "<div
xmlns=\"http://www.w3.org/1999/xhtml\">\n\t<p>\n\t\t<br></br>\n\t\t<br>
</br>\n&lt;infos relevant for the complete chapter&gt;Note to the
template:- the example requirements below are independent of each other and
are showing the different possibilities of the requirements
structure\t</p>\n</div>",
      "Title": "<infos relevant for the complete chapter>"
    },
    {
      "Attribute Type": "Heading",
      "Contributor": "kit7fe",
      "Created On": "2019-10-08T06:18:45.646Z",
      "Creator": "kit7fe",
      "Description": "",
      "Identifier": 629022,
      "Modified On": "2019-10-08T06:18:45.646Z",
      "ReqIF.Text": "<div xmlns=\"http://www.w3.org/1999/xhtml\">\n\t<p>ECU
1st Requirement</p>\n</div>",
      "Title": "ECU 1st Requirement"
    },
    {
      "Allocation": "SW Allocation",
      "Attribute Type": "MO_FUNC_REQ",
      "CRQ": "RQONE03587423",
      "Contributor": "MIG1COB",
      "Created On": "2019-10-08T06:18:45.662Z",
      "Creator": "kit7fe",
      "Description": "",
      "Identifier": 629015,
      "Modified On": "2023-05-23T02:41:23.510Z",
      "ReqIF.Text": "<div
xmlns=\"http://www.w3.org/1999/xhtml\">\n\t<p>&lt;description of the
requirement in requirements language&gt;
</p>\n\t<p>\n\t\t<b>VEHICLE_SYSTEM_BEHAVIOUR</b>\n\t\t<b>CONSTRAINT</b>\n\t
\t<b>IMPACT</b>\n\t\t<b>INFO</b>\n\t\t<b>ASSUMPTION</b>\n\t\t<br>
</br>\n\t\t<br></br>\n\t\t<br></br>\n\t\t<br></br>\n\t\t<br></br>\n\t\t<br></br>\n\t\t<br>
</br>\n\t\t<br></br>\n\t\t<br></br>\n\t\t<br></br>\n\t\t<br></br>\n\t\t<br></br>\n\t\t<br>
</br>\n\t\t<br></br>\n\t\t<br></br>\n\t\t<br></br>\n\n&lt;Optional:
description of desired vehicle behaviour (\"development
target\")&gt;\u00a0\u00a0 \u00a0 \u00a0&lt;Optional: constraints on the
solution space for the requirement&gt;\u00a0 \u00a0&lt;Optional:
description of possible cross-functional impact of the requirement, or
impact on other components&gt;&lt;Optional: additional informations about
the requirement:- know-how- background- HW dependencies related to the
system requirement- internal signals- etc&gt;&lt;Optional: assumptions on
the requirement&gt;\t</p>\n\t<p></p>\n</div>",
      "Safety Classification": "ASIL A",
      "Status": "NEW/CHANGED",
      "Title": "<description of the requirement in requirements language>",
      "VAR_FUNC_SYS": "VAR_Func_sys value",
```

```
      "Verification Criteria": "Test Environment:\nTest Bench/Lab-car with
hardware setup\n\nSuccess Criteria: Verify whether the signal value is
correct or not"
    },
    {
      "Attribute Type": "Heading",
      "Contributor": "kit7fe",
      "Created On": "2019-10-08T06:18:45.677Z",
      "Creator": "kit7fe",
      "Description": "",
      "Identifier": 629018,
      "Modified On": "2019-10-08T06:18:45.677Z",
      "ReqIF.Text": "<div
xmlns=\"http://www.w3.org/1999/xhtml\">\n\t<p>System Non Functional
Requirements</p>\n</div>",
      "Title": "System Non Fonctional Requirements"
    },
    {
      "Allocation": "Non_Func Allocation",
      "Attribute Type": "MO_NON_FUNC_REQ",
      "CRQ": "RQONE03587423",
      "Contributor": "MIG1COB",
      "Created On": "2019-10-08T06:18:45.677Z",
      "Creator": "kit7fe",
      "Description": "",
      "Identifier": 629014,
      "Modified On": "2023-05-23T02:56:54.487Z",
      "ReqIF.Text": "<div
xmlns=\"http://www.w3.org/1999/xhtml\">\n\t<p>&lt;description of the non
functional requirement in requirements language&gt;</p>\n</div>",
      "Safety Classification": "ASIL B",
      "Status": "NEW/CHANGED",
      "Title": "<description of the non functional requirement in
requirements language>",
      "VAR_FUNC_SYS": "Var_func_sys value 2",
      "Verification Criteria": "Non Func Test Environment:\nTest Bench/Lab-
car with hardware setup\n\nSuccess Criteria: Verify whether the signal
value is correct or not"
    }
  ]
}
```


- And same as file `Json_Output_Sample.json`.