Implementation of the CAMSS A-Box

This delivery is accompanied with the implementation of a first CAMSS A-Box. The A-Box is composed of three Graphs.

URL	Description
http://data.europa.eu/2sa/assessments	All the CAMSS Assessments, it includes the answers to the criteria (which are individuals of the CCCEV created in the CAMSS namespace)
http://data.europa.eu/2sa/scenarios	The scenarios and criteria defined for their use in Assessments
http://data.europa.eu/2sa/cssv/rsc	All the specifications and standards identified via CAMSS Assessments and ELIS

Each graph can be obtained as a separate file (an OWL Turtle file). A test Graph Store was also set-up for the loading and testing of the A-Box from a SPARQL endpoint.

All these files, as well as the developments used to produced them, can be publicly downloaded from the CAMSS GitHub[1]. They can be used freely under the EUPL[2] Licensing conditions.

These utilities can be used for the maintenance of the CAMSS Knowledge Graph; e.g., for the addition of new Assessments, scenarios and criteria, as well as standards and specifications.

Artefacts

For a better understanding of how the A-Box of the CAMSS Knowledge Graph is produced, a distinction needs to be done, first, between 'input' artefacts (needed for running a process) and 'output' artefacts (the outcome of the process):

- **Input artefacts**:
    - **Specifications**: In CAMSS, the identification of specifications is done with two main purposes: either for their Assessment (via the CAMSS tools), either to associate them to EIRA ABBs (via ELIS). Specifications extracted from these two types or artefacts are instantiated once and saved into a GraphStore. For the unique and unambiguous identification of the specification, the official URL of the specification[3] is taken as the input for the generation of a SHA-256 hexadecimal number, which is added to the CSSV namespace reserved for resources; e.g.:

[1] See CAMSS GitHu, folder 'util': https://github.com/isa-camss/CAMSS-Ontology/tree/master/util

[2] https://joinup.ec.europa.eu/collection/eupl/about

[3] Including the version whenever it is available in the URL.

http://data.europa.eu/2sa/cssv/rsc/c0e30aa2cfbfa6220770e859721947eb65bfd1f30c280053ea14861a66a06824 ;

- **Criteria**: Each Scenario and CAMSS Toolkit defines, in principle, its own criteria. However some criteria are reused in different combinations of Scenario + ToolKit Version (e.g. EIF-3.0.0 and EIF-3.01). Since each CAMSS assessment specifies one Scenario and its criteria, randomely chosen assessments are picked to extract the different scenarios and criteria and to save them into separate CSV files. For examples and more details, see the CSV files[4] in the CAMSS GitHub. The identifiers of the criteria are also generated hasing the text of the criteria as a SHA256 number.
- **Mappings**: Two means are provided for the generation of the CAMSS A-Box, a Python script and RML mappers[5] (RML stands for RDF Mapping Language, and is a generic mapping language defined to to express customized mapping rules from heterogeneous data structures and serializations to the RDF data model. RML is defined as a superset of the W3C-standardized mapping language R2RML[6]). The RML mappers are provided as one example of reference implementation, but the actual CAMSS A-Box has been generated using the Python utility[7].

- **Output artefacts**:
  - **'Flattened' CSV files**: each CAMSS assessment spread-sheet book is converted to a CSV (one CSV per assessment).
  - **Turtle (TTL) files**: each flattened CSV file is converted into one RDF-OWL Turtle file.**T**he 'camss.py' script can be used to merge the individual TTL files one single TTL file containing all the CAMSS Assessments expressed as OWL. These graphs use the namespace http://data.europa.eu/2sa/assessments.
  - **Scenarios and criteria**: the combinations of the three scenarios and tookit versions (EIF-3.0.0, EIF-3.1.0 and MSP-3.0.0), as well as their criteria, are captured from the flattened CSV files and converted into RDF-OWL Turtle files. The 'camss.py' script can be used to merge the three TTL files containing scenarios and criteria into one single TTL file. These graphs are created in the namespace http://data.europa.eu/2sa/scenarios.
  - **Specifications**: the flattened CSV is also used to convert all the specifications identified therein into one single TTL file. The namespace used for this graph is http://data.europa.eu/2sa/cssv/rsc.

[4] See folder https://github.com/isa-camss/CAMSS-Ontology/tree/master/util/py/out/ass/csv

[5] RML specification site: https://rml.io/specs/rml/

[6] R2RML specification site: https://www.w3.org/TR/r2rml/

[7] All the functionality has been condensed into one single file to facilitate its use and distribution. See file camss.py file in folder 'util':

- **List of Assessments**: a CSV containing a basic set of metadata of all CAMSS Assessments being extracted and converted: the Assessment ID, Scenario ID, the tool version used to perform the Assessment, the Assessment title, and the date of the Assessment.

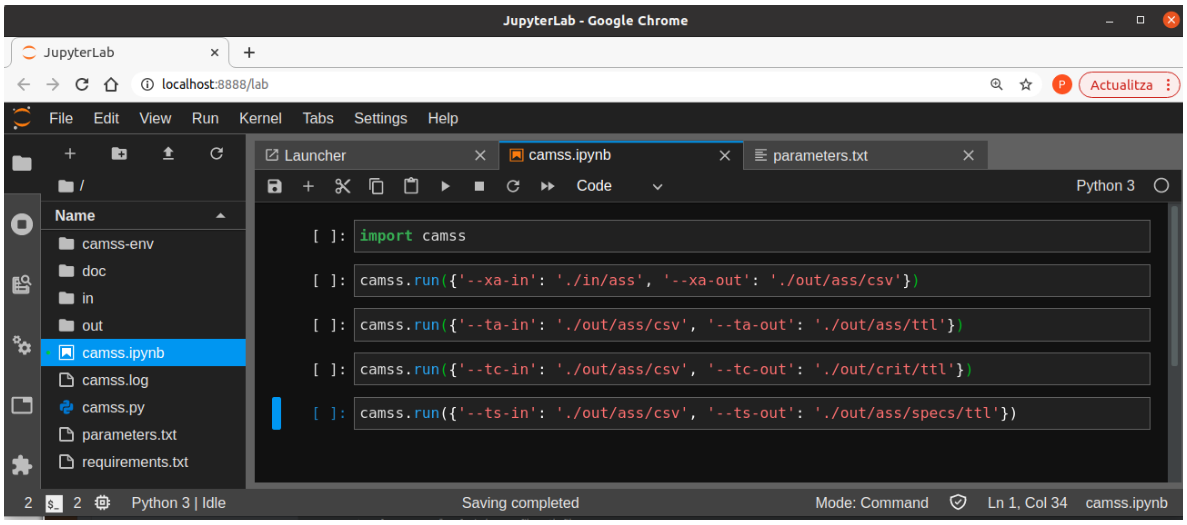
Software

- **camss.py**: Given the input artefacts mentioned above, this scripts provides classes and functions to generate the outputs enumerated in the previous section. If you clone the GitHub repository, you need a folder containing the CAMSS Assessments (as spread-sheets) and the **camss.py** module[8].

  Make sure that all the dependencies (third party-dependencies) are correctly installed in your Python's virtual environment (a 'requirements.txt' file is provided with these dependencies).

  To see which parameters to use, see the file 'parameters.txt' or just run 'python camss.py', or 'python camss.py --help'. These show how to invoke the different functionalities from a command-line environment.

  Alternatively, a Jupyter Notebook[9] is provided to show and document everyone of these functionalities from a Python interpreter console. The figure below shows how to execute some of these entries from a Jupyter Lab notebook:



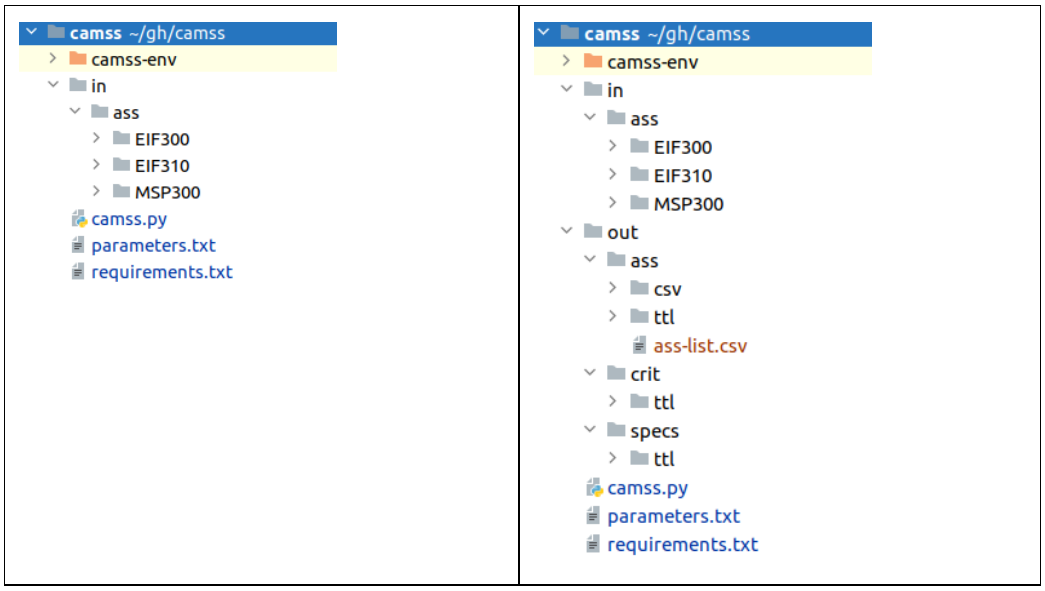
```
[ ]: import camss
[ ]: camss.run({'--xa-in': './in/ass', '--xa-out': './out/ass/csv'})
[ ]: camss.run({'--ta-in': './out/ass/csv', '--ta-out': './out/ass/ttl'})
[ ]: camss.run({'--tc-in': './out/ass/csv', '--tc-out': './out/crit/ttl'})
[ ]: camss.run({'--ts-in': './out/ass/csv', '--ts-out': './out/ass/specs/ttl'})
```

Figure 1: Running the 'camss.py' pipe-line from a Jupyter Lab notebook

The examples provided in the 'parameters.txt' file or when running 'python camss.py –help', should produce a structure of directories and files like these:

[8] The 'camss.py' module: https://github.com/isa-camss/CAMSS-Ontology/blob/master/util/py/camss.py

[9] Jupyter notebook: https://github.com/isa-camss/CAMSS-Ontology/blob/master/util/py/camss.ipynb



```
camss ~/gh/camss
  camss-env
  in
    ass
      EIF300
      EIF310
      MSP300
  camss.py
  parameters.txt
  requirements.txt
```

```
camss ~/gh/camss
  camss-env
  in
    ass
      EIF300
      EIF310
      MSP300
  out
    ass
      csv
      ttl
      ass-list.csv
    crit
      ttl
    specs
      ttl
  camss.py
  parameters.txt
  requirements.txt
```

Figure 2: Directory structure 'before' and 'after' executing the examples of 'camss.py –help'

- **RMLMapper**: Some RML files are also provided to illustrate how the conversion from CSV files can be mapped into RDF using the RML.io specification and the RMLMapper Java tool. This implies that aJVM must be installed for the RMLMapper to run properly. For this version of the CAMSS utilities, Java 1.8 was successfully used. Accompanying these RML files, Jupiter Notebooks are provided to show how to invoke the RMLMapper Java jar from Python.

All the software developed in the context of this project is available under the EUPL Licence[10] conditions. Before reusing it make sure that any other dependency of this development from other software reused therein is duly respected[11].

[10] EUPL Licence in Joinup: https://joinup.ec.europa.eu/collection/eupl/about

[11] Check licences of RMLMapper, Pandas, other (a 'Requirements.txt' file is provided so dependencies with third-party libraries can be easily checked).