Paper OS01

Using odmlib to Create ODM and Define-XML Tools in Python

Sam Hume, CDISC, State College, PA

ABSTRACT

This paper introduces the odmlib Python library for working with ODM and its extensions and spotlights using odmlib to generate a Define-XML v2.1 document. The odmlib library provides an object-oriented interface that simplifies creating and processing ODM or Define-XML documents. It supports serializations in XML and JSON and provides the means to easily create custom ODM or Define-XML extensions. Odmlib also supports conformance checks and schema validation to ensure quality and standards compliance. The odmlib library is available on GitHub under the MIT license. ODM models have been created for ODM v1.3.2, Define-XML v2.0 and v2.1, Dataset-XML v1.0, CT-XML v1.1.1, and Library-XML v1.0. The odmlib project is open to collaborators interested in making contributions such as new features, example programs, additional models, tests, or documentation.

INTRODUCTION

Creating and processing ODM and its extensions like Define-XML can be challenging and repetitive. The odmlib package provides a re-usable Python library that simplifies both creating and processing ODM files. It provides an object-oriented interface to work with ODM that simplifies programming tasks and abstracts away most of the XML knowledge needed to work with ODM documents. It serializes ODM as XML or JSON and supports roundtripping or converting from one format to the other.

Odmlib and the associated example programs can be accessed via public GitHub repositories [1, 2] and all the code has been released as open-source software under the MIT license. Odmlib is also listed on the CDISC Open-Source Alliance (COSA) directory [3]. The odmlib package remains under development but has been used for metadata processing, such as generating a study Define-XML. The remaining development work mainly involves additional testing and implementing any resulting fixes or refinements, as well as improving the documentation.

This paper will highlight the basic features of odmlib for creating or processing ODM and Define-XML files by walking through the steps to create a simple Define-XML file. For those interested in other odmlib use cases and features, visit the references at the end of the article that provide links to documentation, blog posts, and example programs that you might find useful.

CREATING A SIMPLE DEFINE-XML

This section, and most of the paper, highlight the basic steps needed to generate a simple Define-XML v2.1 file. This example provides a foundation from which additional use cases can be explored by exposing readers to the odmlib basics. Once the basics are understood, those interested in learning more can explore some of the other examples and documentation available.

For those who prefer working with code, a Jupyter Notebook containing the code used in this paper is available in the examples repository [6]. A non-notebook version of the code in this paper can be found in my GitHub gists [7]. The code included in this paper has been tested with Python 3.8 and 3.9.

INSTALL ODMLIB

Despite the fact that odmlib remains under development, you don't need to install it from the source. The easiest way to install and start using odmlib is by using pip:

```
pip install odmlib
```

If you have an interest in contributing to odmlib, you may want to install from the source, which can be accomplished by cloning the odmlib GitHub repository [1]. With odmlib installed, the next step is to begin writing the code to create a simple Define-XML v2.1 file.

CREATE THE ODM OBJECT

After installing odmlib, the next step is to import the Define-XML v2.1 model. Many CDISC-developed, ODM models come with odmlib, and we'll focus on the Define-XML v2.1 model in this example. The Define-XML v2.1 odmlib model represents the standard as defined in the specification and makes it possible to create Define-XML elements as objects. To import the Define-XML v2.1 model start your code with the following line:

```
import odmlib.define_2_1.model as DEFINE
```

This example will also use the Python datetime module to set the current datetime:

```
import datetime
current_datetime = datetime.datetime.utcnow().replace(tzinfo=datetime.timezone.utc).isoformat()
```

With the imports taken care of and the current datetime set, we now create the ODM object and assign it to the odm variable. The ODM object from the Define-XML v2.1 model is instantiated by setting values for the desired attributes which must include all those defined as required in the Define-XML v2.1 specification. Certain attributes have a set of permitted values, such as FileType and ODMVersion, while others have specific data types, such as CreationDateTime.

```
odm = DEFINE.ODM(FileOID="DEF.COSA.DEMO",
              AsOfDateTime=current_datetime,
              CreationDateTime=current_datetime,
              ODMVersion="1.3.2",
              FileType="Snapshot",
              Originator="Sam Hume",
              SourceSystem="odmlib",
              SourceSystemVersion="0.1.4",
              Context="Other")
```

Now that we've created the ODM object, which serves as the root element for the Define-XML document, we will continue to create the remaining Define-XML content in the same manner by creating objects.

CREATE THE REMAINING DEFINE-XML OBJECTS

Next, we will create the Study object and add the global variables to Study. In the Define-XML specification, GlobalVariables variables, like StudyName, have values in the body of the element. In odmlib, you assign values to the body of the element using the _content attribute as shown in the example code below.

```
study = DEFINE.Study(OID="ST.DEFINE.COSA.001")
study.GlobalVariables.StudyName = DEFINE.StudyName(_content="TEST Define-XML ItemGroupDef")
study.GlobalVariables.StudyDescription = DEFINE.StudyDescription(_content="ItemGroupDef 001")
study.GlobalVariables.ProtocolName = DEFINE.ProtocolName(_content="Define-XML ItemGroupDef")
odm.Study = study
```

Assigning the value of the XML element to the _content attribute makes it possible to serialize the Define-XML as JSON since "_content" becomes the name in the name / value pair.

Now we have created a Study object that includes the GlobalVariable objects. The study variable is then assigned to the Study attribute of the ODM object created in the previous step. When an object like Study is created, the attributes are typically set at instantiation. Other Study object attributes, such as those that represent child elements in the Define-XML specification, may be instantiated independently and then assigned to the parent element as shown in the example above. Since the Define-XML v2.1 model follows the specification, we know that Study is a child element of ODM, or in odmlib, Study is an attribute of the ODM class. This can also be determined by examining the odmlib Define-XML v2.1 model as described later in this paper.

Working with odmlib continues in this manner by creating Define-XML objects based on the elements defined in the specification and building the Define-XML hierarchy from them. Next, we create the MetaDataVersion object.

```
mdv = DEFINE.MetaDataVersion(OID="MDV.COSA.IGD.001", Name="ItemGroupDefDemo001",
                             Description="ItemGroupDef COSA Demo", DefineVersion="2.1.0")
```

Next, we create the Standards object, new for Define-XML v2.1. In this case we instantiate the Standard object and append it to the Standard list of objects. The Standard list is an attribute of Standards.

```
mdv.Standards.Standard.append(DEFINE.Standard(OID="STD.1", Name="SDTMIG", Type="IG",
                              Version="3.2", Status="Final"))
mdv.Standards.Standard.append(DEFINE.Standard(OID="STD.2", Name="CDISC/NCI", Type="CT",
                              PublishingSet="SDTM", Version="2021-12-17", Status="Final"))
```

Next, we create an ItemGroupDef object and add the ItemRef child elements.

```
igd = DEFINE.ItemGroupDef(OID="IG.VS", Name="VS",Repeating="Yes", Domain="VS",
                SASDatasetName="VS", IsReferenceData="No", Purpose="Tabulation",
                ArchiveLocationID="LF.VS",
                Structure="One record per vital sign measurement per visit per subject",
                StandardOID="STD.1", IsNonStandard="Yes", HasNoData="Yes")

igd.Description.TranslatedText.append(DEFINE.TranslatedText(_content="Vital Signs", lang="en"))
igd.ItemRef.append(DEFINE.ItemRef(ItemOID="IT.STUDYID", Mandatory="Yes", OrderNumber=1,
                                  KeySequence=1))
igd.ItemRef.append(DEFINE.ItemRef(ItemOID="IT.VS.DOMIAN", Mandatory="Yes", OrderNumber=2))
igd.ItemRef.append(DEFINE.ItemRef(ItemOID="IT.USUBJID", Mandatory="Yes", OrderNumber=3,
                                  KeySequence=2))
igd.ItemRef.append(DEFINE.ItemRef(ItemOID="IT.VS.VSSEQ", Mandatory="Yes", OrderNumber=4))
igd.ItemRef.append(DEFINE.ItemRef(ItemOID="IT.VS.VSTESTCD", Mandatory="Yes", OrderNumber=5,
                                  KeySequence=3))
igd.ItemRef.append(DEFINE.ItemRef(ItemOID="IT.VS.VSTEST", Mandatory="Yes", OrderNumber=6))
```

When creating odmlib objects, odmlib validates that the content is complete and valid based on the Define-XML specification. Before we continue adding objects, we will demonstrate what happens when we attempt to create an object using invalid input. The next block of code shows an attempt to create an ItemRef without the required ItemOID attribute. In this case, odmlib throws a ValueError exception.

```
try:
    ir = DEFINE.ItemRef(Mandatory="Yes", OrderNumber=1)
except ValueError as ve:
    print(f"Error creating ItemRef: {ve}")
```

That's enough ItemRefs for this simple example. Next, let's create a Class object. Then we will add MetaDataVersion and ItemGroupDef to the odmlib hierarchy.

```
igd.Class = DEFINE.Class(Name="FINDINGS")
odm.Study.MetaDataVersion = mdv
odm.Study.MetaDataVersion.ItemGroupDef.append(igd)
```

CREATING AND SAVING A DEFINE-XML XML FILE

With the very simple snippet of a Define-XML file we have defined, we will now use odmlib to write the objects out as a Define-XML XML file.

```
odm.write_xml(odm_file="./data/cosa_define_demo.xml")
```

The namespace definitions needed for the XML serialization of Define-XML are defined in the model and generated automatically.

To view the XML file, open the file and print it to the screen.

```
with open("./data/cosa_define_demo.xml", 'r') as file:
    cosa_odm = file.read()
print(cosa_odm)
```

VALIDATING THE DEFINE-XML FILE

Now that the Define-XML file has been created, it can be validated against the XML schema published with the standard. The following code demonstrates how to validate the Define-XML file we just created. If you have followed the example the Define-XML file will validate successfully. You will need to update the code to reference the schema file on your system.

```
from odmlib import odm_parser as P
schema_file = "./schema/cdisc-define-2.1/define2-1-0.xsd"

validator = P.ODMSchemaValidator(schema_file)
try:
    validator.validate_file("./data/cosa_define_demo.xml")
    print("define-XML schema validation completed successfully...")
```

```
except P.OdmlibSchemaValidationError as ve:
    print(f"schema validation errors: {ve}")
```

LOADING THE DEFINE-XML FILE

In addition to providing an object-oriented way to create Define-XML, odmlib can also be used to load and process an existing Define-XML file. To do this we need to import the Define-XML loader and load the document.

```
from odmlib import define_loader as DL, loader as LD
define_uri = "http://www.cdisc.org/ns/def/v2.1"
loader = LD.ODMLoader(DL.XMLDefineLoader(model_package="define_2_1", ns_uri=define_uri))
loader.open_odm_document("./data/cosa_define_demo.xml")
```

PROCESSING THE DEFINE-XML

Now that the Define-XML file has been loaded, using the same object-oriented style we used to create the Define-XML we can access and process content. The following code prints out the Study GlobalVariables created earlier in this example.

```
odm = loader.load_odm()
print(f"Study OID is {odm.Study.OID}")
print(f"Study Name is {odm.Study.GlobalVariables.StudyName}")
print(f"Study Description is {odm.Study.GlobalVariables.StudyDescription}")
print(f"Protocol Name is {odm.Study.GlobalVariables.ProtocolName}")
```

TRANSFORMING THE DEFINE-XML TO A DICTIONARY OR TO JSON

In addition to XML, odmlib supports serializations in JSON and as a Python dictionary. The following code demonstrates serializing the previously created content as a Python dictionary and printing it to the screen.

```
cosa_dict = odm.to_dict()
print(cosa_dict)
```

Similarly, the Define-XML can be converted to JSON.

```
cosa_json = odm.to_json()
```

Odmlib can convert JSON to XML or XML to JSON, known as roundtripping, which enables developers to work in JSON and convert it to XML should they prefer JSON.

UNDERSTANDING MODELS

Odmlib includes models for ODM v1.3.2 and several ODM extensions, such as Define-XML v2.1, Dataset-XML v1.0.1, CT-XML v1.1.1, and others. Additionally, you can create your own models or ODM extensions as demonstrated in the library_xml example that provides the model for working with the CDISC Library ODM media type. The models follow the specifications, so in many cases referencing the standard specification provides enough information to work with odmlib. Alternatively, you can reference the odmlib model class definitions that specify the models. For example, the CommentDef element is part of the Define-XML namespace and is defined in the model as:

```
class CommentDef(OE.ODMElement):
    namespace = "def"
    OID = T.OID(required=True)
    Description = T.ODMObject(element_class=Description)
    DocumentRef = T.ODMListObject(element_class=DocumentRef, namespace="def")
```

Every element of the model inherits from the ODMElement class. This class allows programmers to define models that generate and validate ODM and Define-XML documents without writing any code to support the logic. Define-XML the model and ODMElement take care of transforming the content into an XML document or JSON file.

The above example highlights some basic elements of an odmlib model. In cases where the default namespace is used, no namespace needs to be defined. In this case, CommentDef is part of the Define-XML namespace as indicated with the namespace = “def” line. Developers need not worry about namespaces when working with odmlib since when the Define-XML file is generated, it will add the appropriate namespaces based on the model. Developers can define their own namespaces to create their own extensions to ODM or Define-XML.

After the namespace, the element attributes are defined. Odmlib supports a wide range of attribute types to match the types represented in the ODM and Define-XML specifications. CommentDef has just one attribute, the OID. In this case, it is defined as type OID and is a required attribute. If you attempt to instantiate a CommentDef class without the OID specified, it will throw an exception.

Next the child elements are defined as objects. These may be defined as a single object or a list of objects. The specification for each object is defined elsewhere in the model and is referenced by the element_class attribute. If the child object belongs to a different namespace, that is indicated by the namespace attribute.

The odmlib models define a class for each element in the standard specification. Developers instantiate these classes to generate XML, JSON, or a Python dictionary. Developers typically build a hierarchy from the instantiated objects to generate a complete ODM or Define-XML document as is demonstrated in example programs like xlsx2define2-1.

RUNNING THE DEFINE-XML EXAMPLE PROGRAMS

As noted previously, the odmlib_examples repository [2] contains a number of useful example programs. The two most relevant examples for this paper are xlsx2define2-1.py and define2-1-to-xlsx.py. These programs generate a Define-XML from a metadata spreadsheet and generate a metadata spreadsheet from a Define-XML file, respectively. They are example programs and not intended to be production software, but they have been used to generate Define-XML files and are a useful way to learn more about odmlib. Once odmlib and any other modules listed in the requirements.txt file are installed, these programs can be executed from the command-line. The following shows an example command-line to run the xlsx2define2-1.py program:

```
python xlsx2define2-1.py -e ./data/odmlib-define-metadata.xlsx -d ./data/odmlib-define.xml
```

We'll use the following options when running the application:

- -e provides the name and location of the spreadsheet file to use as input.
- -d provides the name and location of the Define-XML file to generate as output.

The xlsx2define2-1.py comes with an example metadata spreadsheet so that you can run the application. You can also generate a metadata spreadsheet from an existing Define-XML file using the define2-1-to-xlsx example, update the spreadsheet metadata to match your needs, and then generate the new Define-XML using xlsx2define2-1. The define2-1-to-xlsx example now includes an option to generate a Define-XML v2.1 metadata spreadsheet from a Define-XML v2.0 file making it simpler to upgrade from Define-XML v2.0 to v2.1.

Please note that these examples are not intended to indicate that using a spreadsheet is the ideal long-term solution. Using spreadsheets happens to be a common use case and one that makes for a useful, standalone example. Odmlib works equally well for generating Define-XML files when retrieving metadata from a metadata repository or other data store.

ADDITIONAL ODMLIB USE CASES

Creating a Define-XML is a use case of broad interest, but there are many other uses of ODM and ODM extensions including:

- Generating ODM CRF definitions for study setup
- Exchanging data using ODM
- Processing CDISC Controlled Terminology
- Generating a Define-XML for use as a study specification
- Generating standards metadata from content retrieved from the CDISC Library using the ODM media type
- Representing datasets using Dataset-XML

ODM supports a broad range of use cases, and odmlib makes it easier to work with ODM for Python programmers.

ODMLIB NEXT STEPS

The odmlib module has been demonstrated to work by passing its unit tests, by developing several example programs, and by using it in production applications. That said, odmlib will receive additional testing and use prior to an official version 1.0 release. The documentation will also be expanded and improved.

ODM v2.0 will be supported in a future release once it gets closer to being released as a final standard. An odmlib model and examples will be developed to help test the ODM v2.0 specification as well as to add support for this new model to odmlib.

CONCLUSION

Although its testing continues, odmlib has been used for generating and validating Define-XML documents in several projects. Once testing and documentation have been completed, an initial production version will be released.

REFERENCES

[1] Odmlib GitHub repository: https://github.com/swhume/odmlib
[2] Odmlib Examples repository: https://github.com/swhume/odmlib_examples
[3] Odmlib documentation: https://swhume.github.io/odmlib/
[4] COSA Directory: https://cosa.cdisc.org/
[5] Blog posts on odmlib: https://swhume.github.io/blog-home.html
[6] Example Define-XML Jupyter Notebook: https://github.com/swhume/odmlib_examples/tree/master/notebooks
[7] Gist Define-XML example code: https://gist.github.com/c686b40a6356c48f42b3bd2bef53c980

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Sam Hume
CDISC
shume@cdisc.org
https://swhume.github.io/