
**Paper TT17**

# Leveraging Python to Generate Define.xml v2.0

Kevin Miller, Gilead Sciences, Foster City, California
Lawrence Sleeper, Catalyst Clinical Systems, Durham, North Carolina

## ABSTRACT

In our current SAS-dominated world of statistical programming of datasets and define.xml files, Gilead has sought to implement emerging Python technology as an efficient alternative. Specifically, we have leveraged an existing standard metadata template commonly used in industry today for CDISC validation as the starting point for our new Python utility. We have also built in quality-by-design functionality (i.e. upfront quality control checks) so that the user can even more efficiently generate a define.xml v2.0 compliant file for regulatory submissions, allowing for inclusion of analysis results metadata (ARM), automated CRF page numbering, and other benefits of automation. This paper will discuss the various Python modules used and provide the general process flow for use of this Python utility for future submissions.

## INTRODUCTION

Python is increasingly proving itself as a crucial and beneficial tool in the garages of programmers. However, there is a lack of built-in, ready-to-drive modules for statistical programmers to quickly leverage its power. Our goal is to lay the foundation for this opportunity by first providing the framework for an open source community driven method for generating a regulatory compliant (FDA Standards Catalog) version 2.0 define.xml via a Python package and an excel-driven metadata specification file. With Python's versatility and ability to interact with multiple processes, the opportunity exists to broaden the capability of the package and include additional tools in the future. Some examples that may prove useful and bring an added benefit may be; aCRF interaction with population and capture of annotations, a method to quickly compare specification files and provide differences, assisting in creation of SDTM/ADaM program creation, and other submission related repetitive and manual tasks. In full mindfulness of the notion of keeping programming opensource and providing full clarity into the methods of generation of a define xml 2.0 document we have developed the "Trials" module for python. Our hope is to gather feedback from the community; building a userbase and a communal voice to develop, test, and ultimately utilize the package and its modules.

## LEVERAGING PYTHON

A large range of tools exist currently to assist in the generation of a define.xml. Unfortunately, none of them are truly owned by the community of users who need to create the document. In many cases this forces a programming team to rely on a company or product for the generation, or to develop an inhouse process. Python is an open source software (and depending on the package licensed, this can include commercial usage) and is a community-based platform relying heavily on users and programmers to constantly improve the language and its functionality. The modules themselves are useful building blocks in accomplishing daunting tasks, saving time and complexity. Developed and published packages can be dependent on other modules/packages and can leverage the functionality without re-inventing the wheel. The ease of installing, writing, and understanding python makes it an ideal language to develop a tool designed to be owned and developed by its users. This module, Trials, adds on to the capabilities and openness of the python language and presents a solution to an unmet need.

### UTILIZING OPEN SOURCE

Most of the heavy lifting needed to create a define.xml has already been created in the python community. A large amount of a user's interaction in creating a define.xml revolves around importing/modifying Excel xlsx metadata sheets and creating/manipulating XML markup documents. In the case of the Trials module, we have chosen to rely on well documented, supported, and easy to integrate packages and modules. For excel interaction we are utilizing XlsxWriter, and for xml interaction we have chosen lxml. Interacting with the data inside of python is best done by using pandas and its dataframe storage. Below are the descriptions, implementation and origins of these modules.

#### XlsxWriter

"XlsxWriter (Xlsxwriter Module) is a Python module that can be used to write text, numbers, formulas and hyperlinks to multiple worksheets in an Excel 2007+ XLSX file. It supports features such as formatting and many more". We are relying on this module to create a metadata specification template for the user to interact with, enabling the define creation process.

**lxml**

"lxml (lxml module) is a Pythonic binding for the **libxml2** and **libxslt** libraries. It is unique in that it combines the speed and feature completeness of these libraries with the simplicity of a native Python API, mostly compatible but superior to the **ElementTree** API." This package allows effortless creation of xml style output and is vital in the final xml creation.

**Pandas**

"Pandas (Pandas module) is a software library written for the **Python** programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series" This is a ubiquitous package that is well known and synonymous with python. It allows us to easily import and manipulate our metadata spec file, iterate through dataframe rows, and even interact with SAS and XPT data formats.

These three modules are dependencies of the define module and are the backbone of the overall define generation. By utilizing their capabilities, we can efficiently produce a define xml without re-inventing the basic interaction functions.

## HOW IT WORKS

**CREATING SPEC**

The backbone of creating a define.xml is the content. For our initial development the format of the spec mimics what is currently used by Pinnacle 21 for define creation. The module has the capability to create a completely empty template specification file or specifying the location of your SAS or XPT format data files to pre-populate a spec with available metadata information. Fields like datasets, variables, labels, lengths, and formats can be populated. Unfortunately, there is a lot of information that cannot currently be populated, such as codelist, where clauses, and valuelist items. By using the xlsxwriter functions the module can leverage what information we can read to create an easily user manipulated file that is also machine readable and reproducible. Below are two screen shots from the package created metadata file.

The Study Sheet:

| | A | B | C | D |
|---|---|---|---|---|
| 1 | Attribute | Value | | |
| 2 | StudyName | Python Define | | |
| 3 | StudyDescription | Make define 2.0 with Python | | |
| 4 | ProtocolName | define | | |
| 5 | StandardName | define | | |
| 6 | StandardVersion | define | | |
| 7 | Language | define | | |
| 8 | SDTMCTVersion | 0.0.01 | | |
| 9 | ADaMCTVersion | 0.0.01 | | |
| 10 | | | | |

Study | Datasets | Variables | ValueLevel | WhereClauses | Codelists | Dictionaries | Methods | C ...

The Variables Sheet:

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | Order | Dataset | Variable | Label | Data Type | Length | Significant Digits |
| 2 | 1 | AE | STUDYID | Study Identifier | text | 200 | |
| 3 | 2 | AE | DOMAIN | Domain Abbreviation | text | 10 | |
| 4 | 3 | AE | USUBJID | Unique Subject Identifier | text | 200 | |
| 5 | 4 | AE | AESEQ | Sequence Number | integer | 8 | |
| 6 | 5 | AE | AESPID | Sponsor-Defined Identifier | text | 200 | |
| 7 | 6 | AE | AETERM | Reported Term for the Adverse Event | text | 200 | |
| 8 | 7 | AE | AELLT | Lowest Level Term | text | 200 | |
| 9 | 8 | AE | AELLTCD | Lowest Level Term Code | integer | 8 | |

Study | Datasets | Variables | ValueLevel | WhereClauses | Codelists | Dictionaries | Methods

**IMPORTING SPEC**

The module imports the spec sheet-by-sheet and creates working dataframes, which are then manipulated and passed on to the xml creation functions.

**EXPORTING DEFINE SPEC**

Each dataframe from the imported metadata excel file is passed to an xml writing function in the following order:

- Description
- Origin
- Global Variables
- Metadata Info
- Supplemental Documents
- Valuelist
- Where clauses
- Item Groups
- Item Definitions
- Codelist
- Method Definitions
- Comments
- External Documents

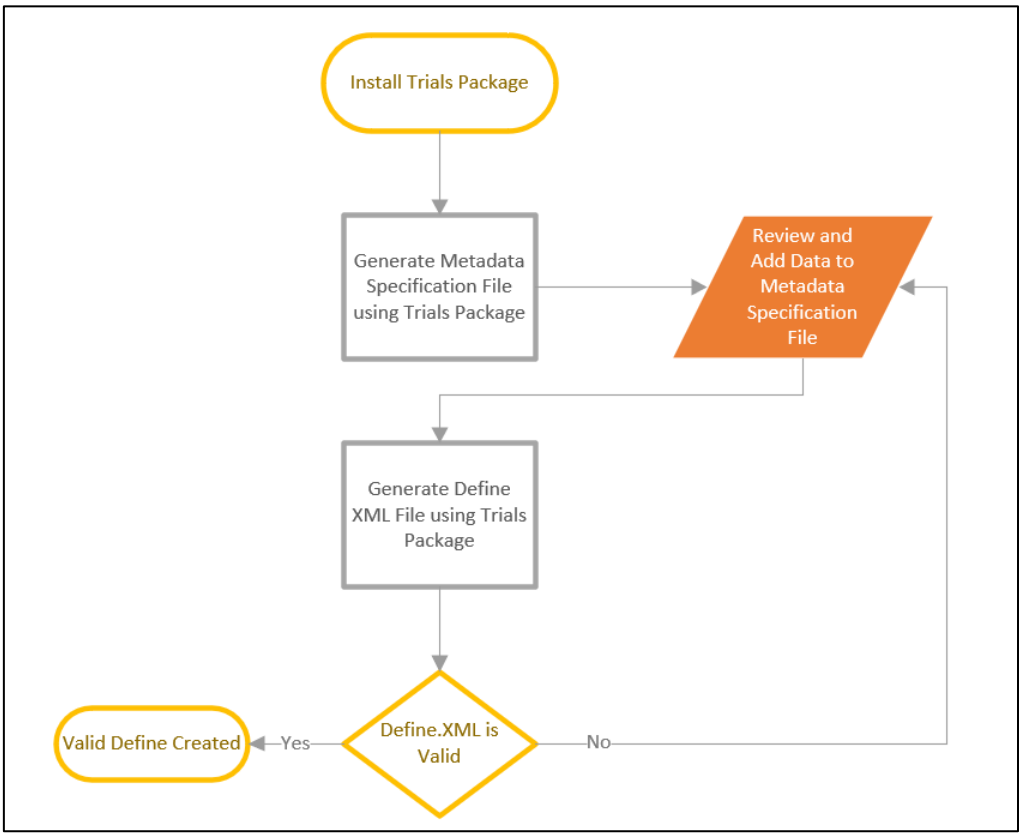
Outline of the XML style and output:

```
<ODM>
    <Study>
    <GlobalVariables>
        <StudyName>
        <StudyDescription>
        <ProtocolName>
    <MetaDataVersion>
        <def:AnnotatedCRF>
            <def:DocumentRef>
    <def:SupplementalDoc>
        <def:DocumentRef>
    <def:ValueListdef>
        <ItemRef>
            <def:WhereClauseRef>
    <def:WhereClauseDef>
        <RangeCheck>
            <CheckValue>
    <ItemGroupDef>
        <Description>
            <TranslatedText>
        <ItemRef>
        <def:leaf>
        <def:title>
```

```
<ItemDef>
    <Description>
        <TranslatedText>
    <CodeListRef>
    <def:Origin>
        <def:DocumentRef>
            <def:PDFPageRef>
    <def:ValueListRef>
<CodeList>
    <EnumeratedItem>
        <Alias>
    <CodeListItem>
        <Decode>
            <TranslatedText>
        <Alias>
    <ExternalCodeListItem>
    <Alias>
<MethodDef>
    <Description>
        <TranslatedText>
    <def:DocumentRef>
        <def:PDFPageRef>
<def:CommentDef>
    <Description>
        <TranslatedText>
    <def:DocumentRef>
        <def:PDFPageRef>
    <def:leaf>
<def:titlle>
```

**PROCESS FLOW**

A generalized overview of the expected usage and flow of the modules and interaction in generating a define.xml




**A NOTE ABOUT STYLESHEETS**

The stylesheet is a vital component to the define.xml, it translates the XML to the web browser about how to apply style rules to the XML elements. The module currently uses the Define-XML v2.0 style sheet provided by PhUSE Define XML 2.0 Stylesheet Recommendations Working Group (PhUSE Wiki). We assume that the stylesheet is placed in the same folder as the created define.xml. Sponsor-specific updates can be made to this stylesheet to accommodate your organization needs.

## A COMMENT ABOUT VALIDATION

There is much discussion in the industry regarding creating a “truly” validated python program. Since the define.xml is a submission related document, its validation and accurateness is controlled via standard operating procedures that ensure accuracy and validity. Every organization has their own operating procedures, processes, and methods for validation, what is and isn’t required, and a programmer should ensure that they are followed. Inside of our organization we ensure that each requirement is tested and documented and reviewed. We have the bonus of validating our output with existing in-house SAS-based define.xml creation programming method and other vendor creation methods. We can compare our Python output to these outputs and validate. Without a doubt, any document that is being used in submission activities should require having an experienced associate who is familiar with the requirements and standards review the final define.xml output.

## WHERE TO GO FROM HERE

As with any python module, the reliability, functionality and usability directly correlate to the amount of usage the module gets. As usage increases and more utilization opportunities arise, the amount of effort to update and maintain the module code also grows. It comes down to needing users and programmers that are interested and willing to commit time to use, maintain, and collaborate with the package to increase its functionality and reliability, especially since there are a multitude of ideas and improvements that could be implemented and added to the package. If there is interest from the community in pursuing a community, working-group owned define creation module; expanding the


abilities requires creating an active developer, tester, subject matter expert, and user group. The next items our team hopes to begin implementing are the inclusion of the ability to create Analysis Results Metadata (ARM) and creating the quality-by-design functionality checks.

## CONCLUSION

The define xml creation is a crucial deliverable associated with submission of clinical trials to regulatory agencies. While each organization will certainly have their preferences and standards the Trials module offers a free and quick entry into the creation of the define.xml. While we were able to create a useful tool for our own use, it was also an opportunity for our organization to identify areas where python can be applied to a pivotal process. From our current state of development, we see the Python approach as more efficient, easier to maintain, and more flexible. Our hope is to allow the community another method of define.xml creation, and to further enhance and develop the define creation process, while incorporating capabilities in the package for related define.xml creation tasks.

## REFERENCES

1. FDA Standards Catalog. (2019 Nov 8). Study Data for Submission to CDER and CBER. Retrieved from - https://www.fda.gov/industry/study-data-standards-resources/study-data-submission-cder-and-cber

2. Define XML 2.0. (2014 Mar 5). CDISC Define-XML Specification Version 2.0. Retrieved from - https://www.cdisc.org/standards/data-exchange/define-xml

3. XlsxWriter module (2020 Jan 12). Creating Excel files with Python and XlsxWriter. Retrieved from - https://xlsxwriter.readthedocs.io/

4. lxml module (2019 Oct 10). lxml - XML and HTML with Python Overview. Retrieved from - https://lxml.de/

5. Pandas module (2019 Oct 31). Pandas Getting started. Retrieved from - https://pandas.pydata.org/

6. PhUSE Wiki. (2018 Aug 31). Define-XML V2.0 Completion Guidelines & Style Sheet Recommendations. Retrieved from - https://www.phusewiki.org/wiki/index.php?title=Define-XML_V2.0_Completion_Guidelines_%26_Style_Sheet_Recommendations

7. ARM (2015 Jan 27). Analysis Results Metadata (ARM) v1.0 for Define-XML v2.0 Retrieved from - https://www.cdisc.org/standards/foundational/define-xml/analysis-results-metadata-arm-v10-define-xml-v20

## ACKNOWLEDGMENTS

Many thanks for the numerous creators, testers and developers of both python and its versatile modules. The open source nature of the programming language allows for great freedom in creating many useful tools. We also acknowledge our team at Gilead including Dhananjay Chhatre, Pei Sern, Greg Campbell, and Barry Ashby.



## CONTACT INFORMATION

Your comments and questions are valued and encouraged.
Visit the current home of the package at: https://github.com/kevinjacksonm/trials
Contact the authors at:

**Kevin Miller**
Gilead Sciences
333 Lakeside Drive
Foster City, CA 94404
Email: kevin.miller@gilead.com

**Lawrence Sleeper**
Catalyst Clinical Systems 333 Lakeside Drive
7780 Brier Creek Parkway, Suite 310
Raleigh, NC 27617
Email: LarrySleeper@catalystcr.com