PharmaSUG 2024 - Paper SD-255

Define-XML Conversion: A General Approach on Content Extraction Using Python

Danfeng Fu, MSD China, Shanghai, CN;
Dickson Wanjau, Merck & Co., Inc., Rahway, NJ, USA;
Ben Gao, MSD China, Shanghai, CN

ABSTRACT

Define-XML is a critical component of clinical trial data packages submitted to regulatory agencies as part of a new drug/ biologics application. It provides a standardized way to exchange the metadata for SDTM and ADaM datasets, which are used for analysis and reporting. The rich information embedded in Define-XML is an ideal data source for automating consistency checks against study datasets and documents, e.g. ADRG. SAS is a natural choice for this work due to its widespread use in the pharmaceutical industry, but there are challenges. First, the information needs to be extracted and converted into SAS datasets, all of which are challenging, owing to XML language's natural complexity with nested elements and namespaces. The process is also cumbersome since one needs to create XML map files for parsing the XML structure. Encoding would be another issue to tackle, especially when the Define-XML contains non-ASCII characters.

In this paper, we will introduce a simplified more straightforward approach using Python, leveraging its extensive packages to extract contents from all types of XML schemas, not limited to the CDISC ODM schema. This approach doesn't require the creation of map files and can better handle non-ASCII characters. It also offers a user-friendly method for presenting a structural overview of a selected XML element (sub-elements, contents, and attributes) and flexibility to extract desired contents without relying on a pre-defined map file to create tabulated datasets or stylesheets to visualize the XML contents. This facilitates the automation of cross-document consistency checks and potentially automates other downstream processes, ultimately enhancing submission quality and compliance.

INTRODUCTION

XML files, especially Define-XML, are commonly utilized in clinical study submissions. This paper explores the nature of XML and Define-XML, identifies the challenges encountered in leveraging Define-XML as a rich data source for automation, and illustrates how we employed Python to address these issues.

XML

Extensible Markup Language (XML) is a markup language that is designed to store and transport data, making it a widely used format for data exchange and storage. XML uses a set of rules to encode data in a format that is both human-readable and machine-readable.

In XML, data is enclosed within tags that define its structure and meaning. These tags are created by the user, making XML an extensible language. XML documents can include elements, attributes, text content, and other components. Elements are the building blocks of

XML documents and are enclosed within start and end tags. Attributes provide additional information about the elements.

XML is platform-independent and language-neutral, making it suitable for data interchange between different systems and programming languages. It is commonly used in web services, as well as configuration files, data serialization, and representing structured data.

DEFINE-XML

Define-XML is a standard developed by the Clinical Data Interchange Standards Consortium (CDISC). It serves as a metadata standard for defining and describing the structure of datasets in clinical trials. The CDISC Operational Data Model (ODM) XML schema provides a structured format for capturing and representing metadata about clinical trial datasets, including study design, data collection, variables, and annotations. It serves as the foundation for the Define-XML specification, allowing the standardized definition and documentation of clinical trial data.

Advantages:

1. Standardization: Define-XML provides a standardized way to define and document the structure of datasets, ensuring consistency and interoperability across different systems and organizations.
2. Regulatory Compliance: It aids in regulatory submission package preparation by providing a clear and structured representation of the clinical trial data, facilitating the review process by different stakeholders. It promotes consistency and interoperability in submission package components, making it compliant with internal and regulatory data standards.
3. Improved Understanding: Define-XML provides comprehensive metadata including variable definitions, relationships, and data structures. This helps reviewers understand and interpret the clinical trial data better.

WHY PYTHON

Python is widely favored for extraction of information from XML files due to its robust library support, especially with standard XML processing modules such as xml.dom and xml.sax modules. Modules, such as xml.etree.ElementTree, xml.dom, xml.dom.minidom, xml.sax, are part of the built-in Python standard packages and don't require explicit installation by the users. Robust XML-handling packages such as lxml and data frame-building packages such as Pandas provide more capabilities, thus simplifying the process of parsing and extracting information from XML documents. Pandas' XML API provides the read_xml() function, which provides the ability to specify the parser such as lxml and etree for retrieval of data. It also provides us with the ability to use the Extensible Stylesheet Language Transformations (XSLT) stylesheets. Further, it provides the "encoding" parameter to handle the encoding of a particular XML document ensuring we can programmatically deduce the type of encoding in a particular file, thus facilitating correct parsing of its contents. It also provides us the ability to specify XML Path Language (XPath) expressions, which can be used to tailor our data-retrieval from specific nodes, elements, and attributes in any XML file. XPath is a language used to navigate and query XML documents. These XPath expressions offer a powerful way to navigate through elements and attributes in an XML document.

The readability and simplicity of the Python language provide advantages for developers of all skill levels. Further, Python's open-source nature, large community, and cross-platform compatibility enhance its suitability for many programming tasks in today’s environment. New and improved features are continually added.

CURRENT STATUS ON XML UTILIZATION AND PROCESSING

BUSINESS NEED FOR XML CONTENT EXTRACTION

XML contains structured and hierarchical data, which enables easy navigation, querying, and extraction of its specific data elements. In the pharmaceutical industry, XML formats are widely used in the clinical data lifecycle, i.e., data collection, mapping, analysis reporting, and submission package components. There is a growing need to have automated tools to parse the XML contents, extract desired information, and convert it into formats that can be readily used with analysis & reporting (A&R) tools such as SAS and R for further processing. A common use case is to validate shared content across critical submission deliverables such as Define-XML, cSDRG, ADRG, aCRF, SDTM and ADaM datasets. Typical business use cases suitable for XML content extraction may include:

Define-XML

Define-XML provides a standardized structure to represent metadata at dataset level, variable level and value level governed by the ODM schema. These metadata are referenced among multiple documents, which makes it critical to guarantee cross-document consistency. For example, each ADaM dataset name, its label and structure are mentioned in multiples places within ADRG, e.g., section **5.2 Analysis Datasets**, its sub-sections (**5.2.x**) and section **7.1 ADaM Programs**. The similar scenario applies to cSDRG where section **3.4 Subject Domains** documents each SDTM domain, and sub-sections **3.4.x** is used to list QNAMs in the supplemental domain used in the downstream analysis and reporting. It is more efficient and less error-prone to have automated tools to extract the needed information from Define-XML to perform comprehensive cross-document checking instead of manual spot checks.

To enhance the overall quality of Define-XML, derivations/comments can be extracted and processed programmatically to identify unclear or ambiguous language, or descriptions containing programming statements not relevant for the reviewers. In addition, metadata can be extracted from Define-XML to check that it is consistent with the submitted datasets.

SDTM aCRF Annotations

The PDF annotations in an SDTM annotated Case Report Form (aCRF) can be exported to a file with .xfdf extension, which is an XML Forms Data Format (XFDF) generated with Adobe Acrobat software. XFDF contains the form element descriptions and values, and the names and values of text fields in the PDF form. The text fields can be imported to facilitate aCRF annotation updates or other validation purposes, for example, CRF page number validation against Define-XML.

Other Customized Uses

Excel spreadsheet is a popular format among all professions due to its ease of use and user-friendly interface. XML files can be rendered in Excel if they adhere to the structure defined by the Office Open XML schema. This schema consists of various XML elements and attributes, which represent different aspects of the spreadsheet, such as worksheets, tables, formulas, formatting, styles, and more. This seamless XML and Excel integration facilitates the storage and exchange of structured data in a platform-independent way. At the same time, it allows people with little or no XML knowledge to view and manipulate the files in their familiar approach. In addition, Microsoft Word documents (.docx) are represented in XML format following the Open XML standard.

XML is being used as the data format to upload clinical trial results onto public disclosure databases such as ClinicalTrial.gov and EU Clinical Trials Register (EU CTR).

An automated tool is needed to analyze the XML structure and extract the desired information to streamline downstream processing.

STATUS AND POTENTIAL PROBLEM

Since SAS is so widely used in the pharmaceutical industry, particularly in the analysis and reporting area, processing XML files in SAS becomes a natural choice. The XML Mapper, a tool in the SAS software suite, provides a way to understand the structure of the XML document and map the XML elements to SAS variables. It allows users to visually navigate through an XML document, create an XMLMap, and save the XMLMap to a file. XMLMap files are used by the SAS XML library engine to read XML documents and save the pre-specified information from XML into SAS datasets.

However, most statistical programmers supporting clinical trial A&R don’t use XML very often. Understanding what an XML file is, as well as its structure and content, and how to read it, is not an easy task for statistical programmers. Using the SAS XML Mapper and generating XMLMap files requires a deep understanding of the XML format, including the tree structure of XML, namespace definition and usage, and its various terminologies.

Creating a correct XMLMap file is also quite challenging. Searching for each element visually is inefficient; manually dragging elements to establish data structure relationships is prone to errors. Using an XMLMap as a template for different users within an organization works well in general but leads to problems in some specific scenarios. Another likely scenario is that most people often use the SAS English version (encoding=latin1), while the majority of XML encodings are non-latin, which can easily lead to the failure to read some non-Latin characters in XML.

PROPOSED SOLUTION

Allowing users to understand the content structure of the XML file from the beginning, rather than relying on another XMLMap file and running a program to see the information, will help them more easily obtain the content they need. If users can have unique elements and attributes, the relative relationship between elements (such as parent elements and child elements), and what content each type of element contains when they first get the XML, it will be of great help to their understanding of the XML file and their ability to process it correctly in later steps.

We leverage advantages of various packages mentioned in the Why Python Section, and write functions using these packages to extract the information we want from the XML files. These functions serve various purposes within XML processing, such as extracting essential information from the XML file, and retrieving the complete content of the XML file, encompassing elements, attributes, and other XML components. Certain functions facilitate the visualization of structural relationships between different elements. Lastly, specific functions are designed to fulfill particular requirements as needed.

We developed additional user-defined functions to address the specific needs of our organization. The functions we designed are applicable to each individual XML file provided by the user. They do not require any auxiliary files to read the content and display the relative relationships within. These functions can not only be called within applications developed in Python, but they can also be made into server-side APIs, so that they can be invoked from other programming languages as needed.

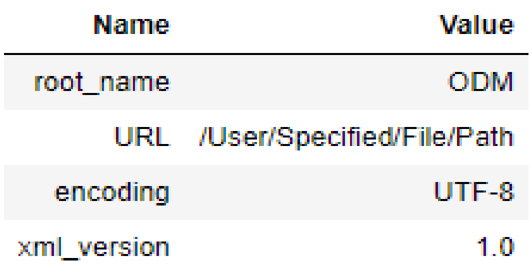
FUNCTIONS AND DEMONSTRATIONS

XML BASIC INFORMATION

Upon receiving an XML document, it is helpful for the users to quickly have a high-level overview of XML structure such as XML version, encoding, namespaces, summary of unique elements and corresponding attributes. Building on top of the Python lxml package, several Python functions were defined to implement each specific functionality as listed below. This basic information will be used in implementing other customized use cases.

XML Version, Encoding and Namespaces

The XML declaration is enclosed within *<?xml ... ?>* tags at the beginning of an XML document (typically as the first line), which gives the version and encoding of the XML document. The XML declaration helps software tools and systems accurately process and interpret the XML data. The Python function that we developed uses the **DocInfo** class in **lxml.etree** module to obtain the XML version, encoding, XML file path and root element name. If DOCTYPE is declared with the XML, it can also be returned from the class property. See Figure 1. Basic Info Extracted from Define-XML.

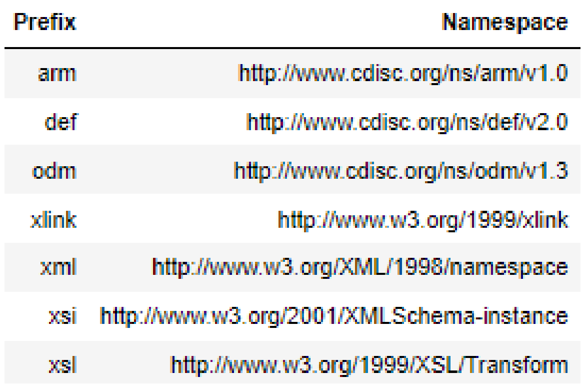


Name	Value
root_name	ODM
URL	/User/Specified/File/Path
encoding	UTF-8
xml_version	1.0

Figure 1. Basic Info Extracted from Define-XML

XML namespaces are primarily used to avoid naming conflicts when different vocabularies or schemas are combined within a single XML document. Different organizations or working groups can define their own vocabularies within their namespaces, so it is important to know the specific namespaces associated with each element and attribute. For example, CDISC Define-XML standard document defines the namespaces for the ODM element (root).

The lxml package provides robust namespace support for XPath, e.g., XPath axes, predicates and functions. With a single function call *xpath('//namespace::*')*, all the namespaces in the XML document can be retrieved and returned into a Python list for manipulation into other data structures (See Figure 2. All Namespaces Extracted from Define-XML for a Pandas DataFrame example).



Prefix	Namespace
arm	http://www.cdisc.org/ns/arm/v1.0
def	http://www.cdisc.org/ns/def/v2.0
odm	http://www.cdisc.org/ns/odm/v1.3
xlink	http://www.w3.org/1999/xlink
xml	http://www.w3.org/XML/1998/namespace
xsi	http://www.w3.org/2001/XMLSchema-instance
xsl	http://www.w3.org/1999/XSL/Transform

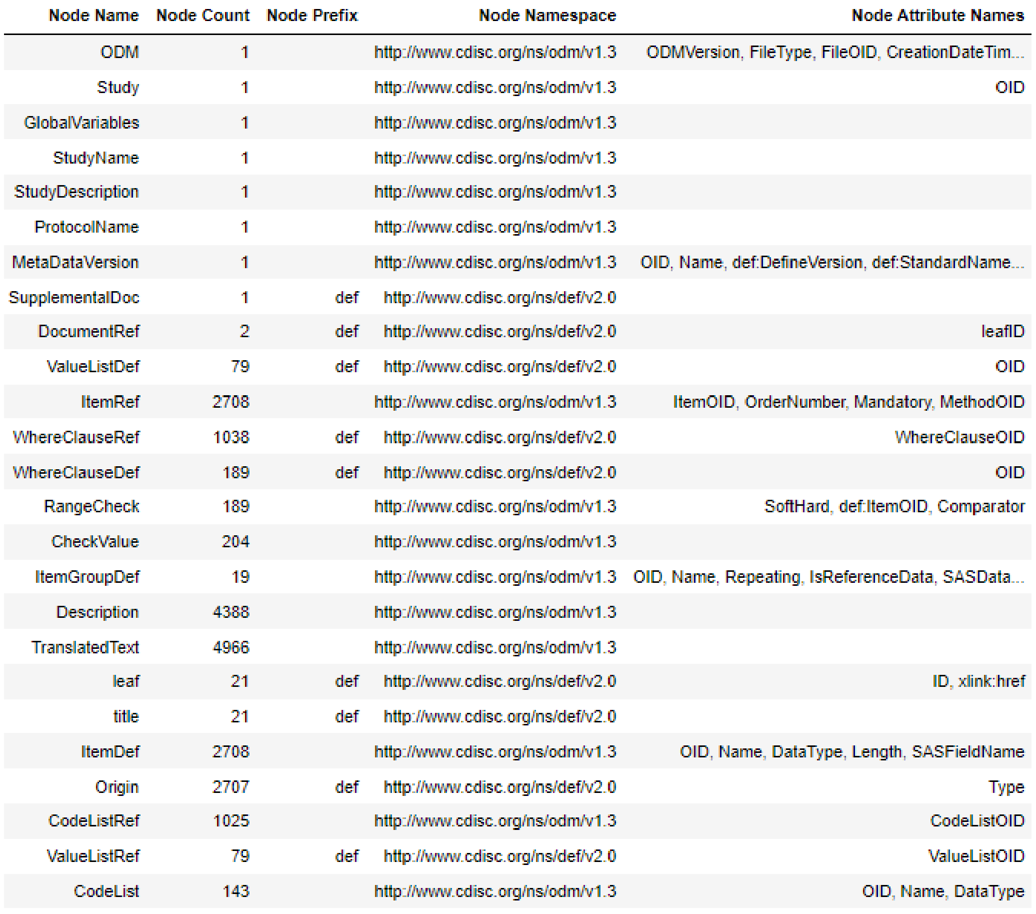
Figure 2. All Namespaces Extracted from Define-XML

Unique Elements and Attributes

Elements are the fundamental components of an XML document's structure. An element can contain other elements as its content, which creates a hierarchical and nested structure. A summary table can be used to show an overview of the unique elements, element counts, namespace, and attributes associated with each element. Directly inspecting and scrolling through the raw XML file is time-consuming and challenging due to the lack of visualization and the absence of styling and formatting.

The table in Figure 3. Define-XML's Unique Elements and Its Namespaces and Attributes lists the unique Element (Node Name) and the number of each element in the whole XML document. The namespace for each element is split into 2 parts: the namespace prefix (Node

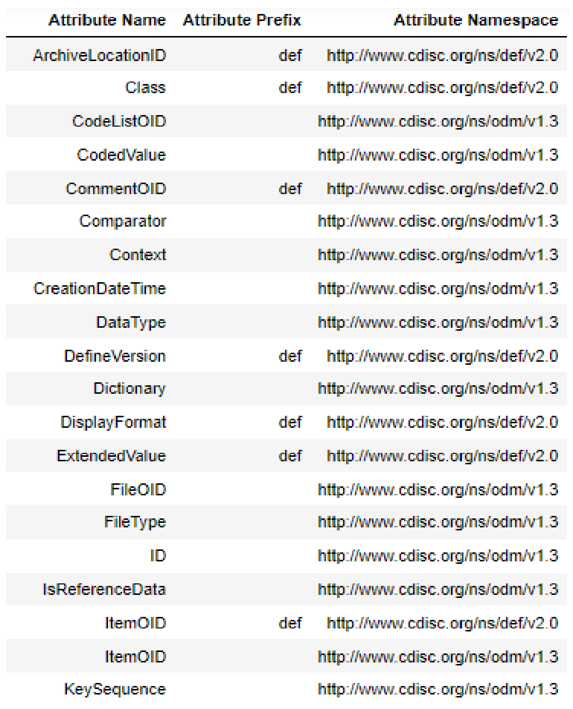
Prefix) and the URI (Node namespace). If any attributes are defined for an element, the attribute names will be displayed in the last column, concatenated by a comma. This summary table at the element level forms the basis for developing other features, offering the capability to efficiently access information from multiple aspects. For example, if we seek information about a specific Node Name, we can easily obtain a significant portion of the essential details related to that Node Name from this table. Users are not required to manually search for relevant information within the source XML file, saving them valuable time and effort.



Node Name	Node Count	Node Prefix	Node Namespace	Node Attribute Names
ODM	1		http://www.cdisc.org/ns/odm/v1.3	ODMVersion, FileType, FileOID, CreationDateTim...
Study	1		http://www.cdisc.org/ns/odm/v1.3	OID
GlobalVariables	1		http://www.cdisc.org/ns/odm/v1.3	
StudyName	1		http://www.cdisc.org/ns/odm/v1.3	
StudyDescription	1		http://www.cdisc.org/ns/odm/v1.3	
ProtocolName	1		http://www.cdisc.org/ns/odm/v1.3	
MetaDataVersion	1		http://www.cdisc.org/ns/odm/v1.3	OID, Name, def:DefineVersion, def:StandardName...
SupplementalDoc	1	def	http://www.cdisc.org/ns/def/v2.0	
DocumentRef	2	def	http://www.cdisc.org/ns/def/v2.0	leafID
ValueListDef	79	def	http://www.cdisc.org/ns/def/v2.0	OID
ItemRef	2708		http://www.cdisc.org/ns/odm/v1.3	ItemOID, OrderNumber, Mandatory, MethodOID
WhereClauseRef	1038	def	http://www.cdisc.org/ns/def/v2.0	WhereClauseOID
WhereClauseDef	189	def	http://www.cdisc.org/ns/def/v2.0	OID
RangeCheck	189		http://www.cdisc.org/ns/odm/v1.3	SoftHard, def:ItemOID, Comparator
CheckValue	204		http://www.cdisc.org/ns/odm/v1.3	
ItemGroupDef	19		http://www.cdisc.org/ns/odm/v1.3	OID, Name, Repeating, IsReferenceData, SASData...
Description	4388		http://www.cdisc.org/ns/odm/v1.3	
TranslatedText	4966		http://www.cdisc.org/ns/odm/v1.3	
leaf	21	def	http://www.cdisc.org/ns/def/v2.0	ID, xlink:href
title	21	def	http://www.cdisc.org/ns/def/v2.0	
ItemDef	2708		http://www.cdisc.org/ns/odm/v1.3	OID, Name, DataType, Length, SASFieldName
Origin	2707	def	http://www.cdisc.org/ns/def/v2.0	Type
CodeListRef	1025		http://www.cdisc.org/ns/odm/v1.3	CodeListOID
ValueListRef	79	def	http://www.cdisc.org/ns/def/v2.0	ValueListOID
CodeList	143		http://www.cdisc.org/ns/odm/v1.3	OID, Name, DataType

Figure 3. Define-XML's Unique Elements and Its Namespaces and Attributes

In addition to attributes associated with each unique element above, another table shown in Figure 4. Unique Attributes in Define-XML is also provided to display the unique attributes and their namespaces (prefix and name) within the whole document. This attribute table can be cross-referenced if the namespace name (URI) information is needed from the element table.



Attribute Name	Attribute Prefix	Attribute Namespace
ArchiveLocationID	def	http://www.cdisc.org/ns/def/v2.0
Class	def	http://www.cdisc.org/ns/def/v2.0
CodeListOID		http://www.cdisc.org/ns/odm/v1.3
CodedValue		http://www.cdisc.org/ns/odm/v1.3
CommentOID	def	http://www.cdisc.org/ns/def/v2.0
Comparator		http://www.cdisc.org/ns/odm/v1.3
Context		http://www.cdisc.org/ns/odm/v1.3
CreationDateTime		http://www.cdisc.org/ns/odm/v1.3
DataType		http://www.cdisc.org/ns/odm/v1.3
DefineVersion	def	http://www.cdisc.org/ns/def/v2.0
Dictionary		http://www.cdisc.org/ns/odm/v1.3
DisplayFormat	def	http://www.cdisc.org/ns/def/v2.0
ExtendedValue	def	http://www.cdisc.org/ns/def/v2.0
FileOID		http://www.cdisc.org/ns/odm/v1.3
FileType		http://www.cdisc.org/ns/odm/v1.3
ID		http://www.cdisc.org/ns/odm/v1.3
IsReferenceData		http://www.cdisc.org/ns/odm/v1.3
ItemOID	def	http://www.cdisc.org/ns/def/v2.0
ItemOID		http://www.cdisc.org/ns/odm/v1.3
KeySequence		http://www.cdisc.org/ns/odm/v1.3

Figure 4. Unique Attributes in Define-XML

XML ADVANCED INFORMATION

Basic XML information, including elements, attributes, and namespaces, provides standalone insights about the XML file. However, the relationships between these components represent more critical information within the XML file. Without understanding these dependencies, the values remain incomplete, even with all the separate contents at hand.

Element – Element Relationship

Python's xml.etree.ElementTree module effectively identifies these relationships by iterating through every element in the file. A detected element object encompasses all its child elements, enabling the establishment of a parent-child relationship. As the initial step in constructing the element dependency map, we loop through the top-level element until no child elements are detected. This process results in a collection of all identified pairs, thereby establishing the element dependency map. Figure 5. Unique Elements' Relationship shows the relationships of all elements from a define.xml file.

This dependency map will help users to establish the hierarchical structure of the whole XML document. If they are interested in a specific element, they can navigate from the root element to it by following along the lines. Within Figure 6. Unique Element's and Unique Attributes' Relationship, the attribute elements (blue nodes) are also clearly represented by yellow lines linking to each element node. Coupled with the customized XPath search functionality, the node-attribute tables can be built for user selected nodes, not limited to the root.

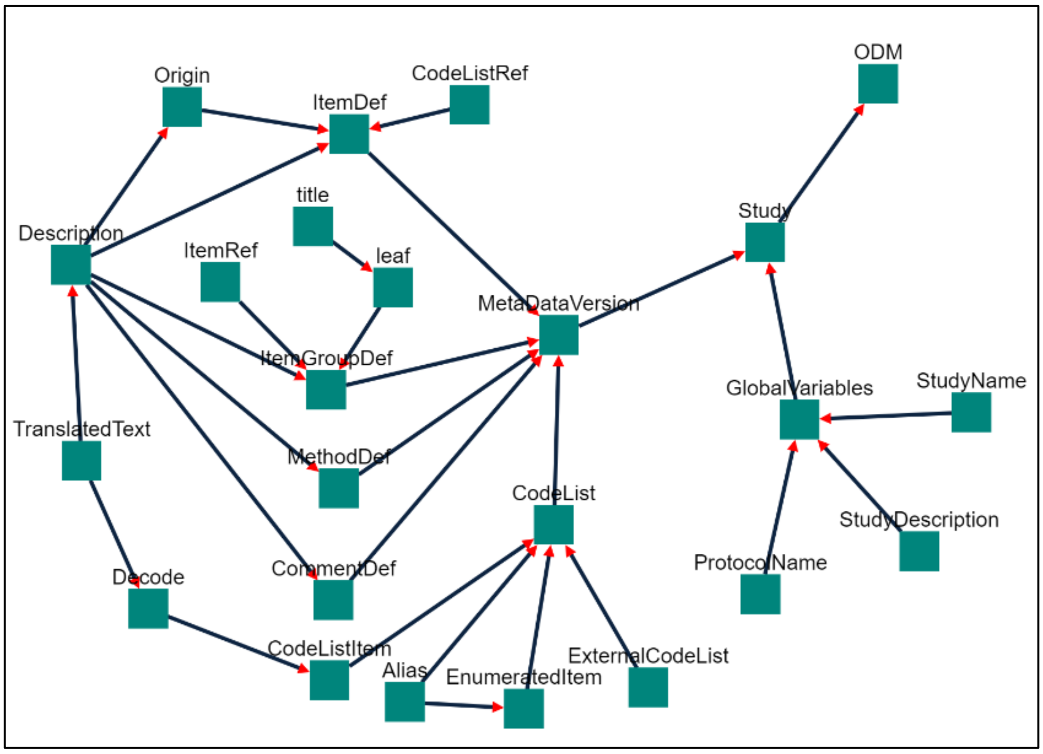


Figure 5. Unique Elements' Relationship

Element – Attribute Relationship

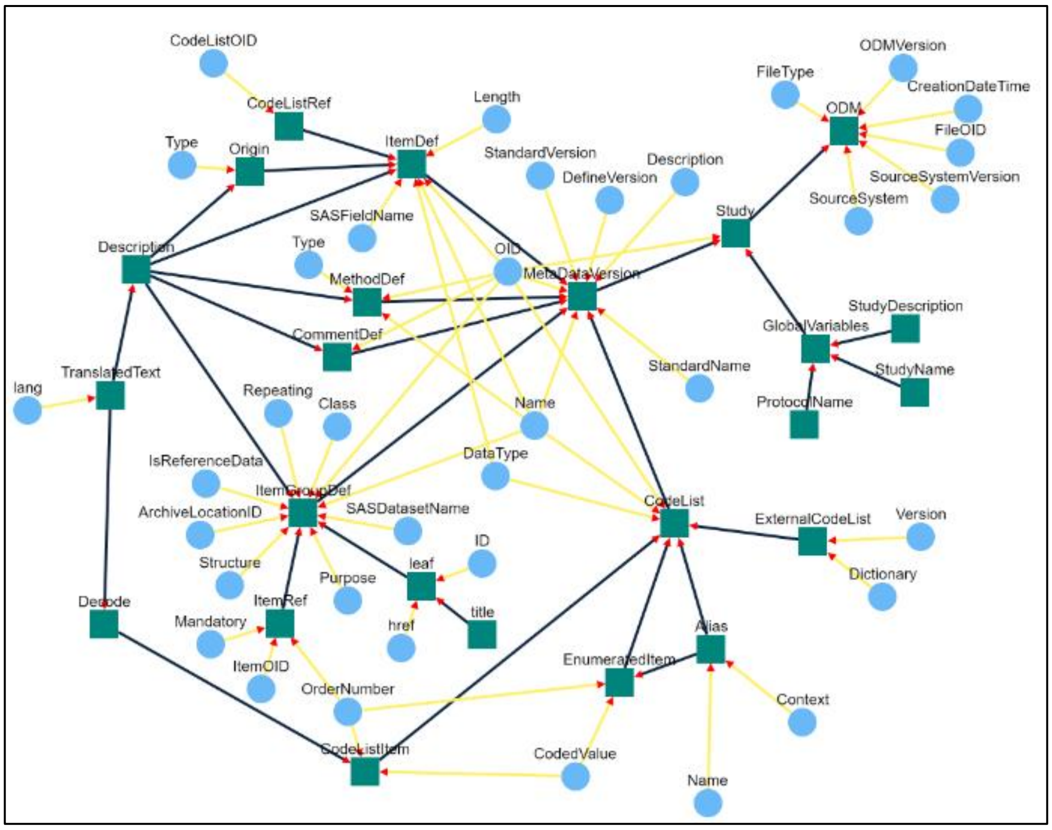


Figure 6. Unique Element's and Unique Attributes' Relationship

XML CUSTOMIZED CONTENTS

Utilizing the powerful functions of the above solutions, we can quickly extract specific content from the specific XML used in our daily work. In this way, the work of extracting content from XML will become more intuitive, user-friendly, and efficient. In contrast to the past, when working with XML files felt like navigating a black box, now every user can confidently manage XML content with a clear understanding of the information it contains. Below are two types of extraction that can be customized:

Dynamic Extract & Content Preview

One of the features we developed is to assist users in defining their own XPath to extract content in real time, thereby allowing them to browse the extracted content to determine if it meets their needs. Defining XPath requires a certain understanding, but the functionality we designed can minimize the user's requirements for XPath syntax understanding, allowing them to focus more on the content they need.

For example, using the results from XML Basic Information, we can:

- help users automatically find the prefix or URI of the namespace corresponding to the Node Name.
- help users confirm whether the content they need comes from the XML’s Node or attribute.

Using the results from XML Advanced Information, we can:

- help users understand the hierarchical relationship between elements and attributes.
- check whether the XPath provided by the user is reasonable in terms of relative relationships.

After users provide the correct XPath, the content is directly made available for preview. Users can continue to update the XPath until they achieve the desired result.

Pre-Define XPath for Target Contents

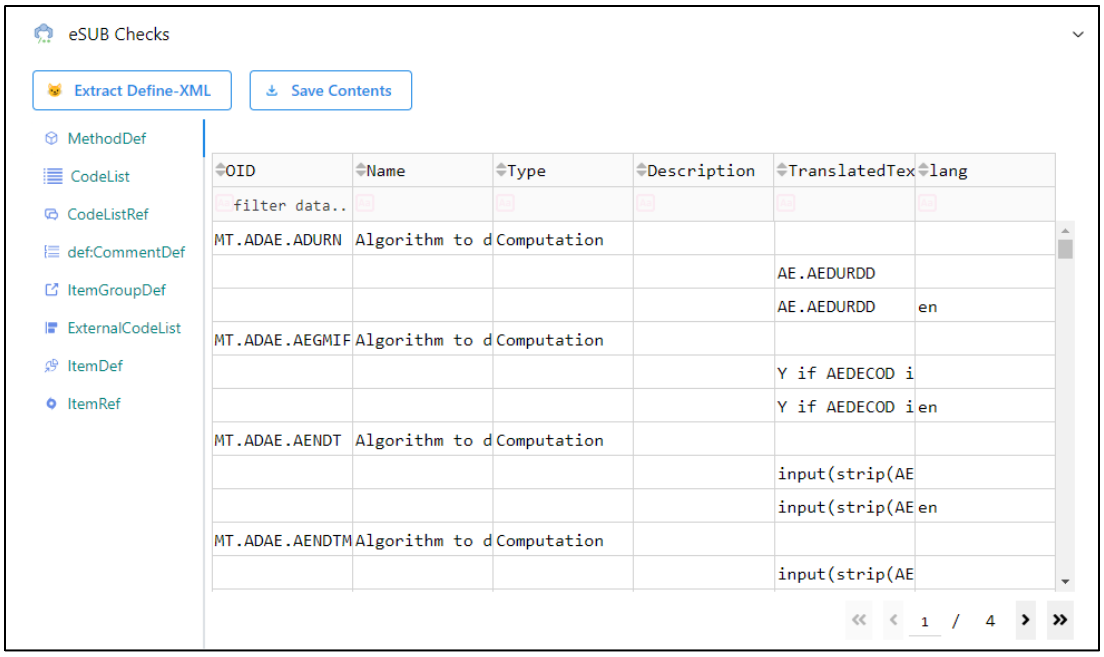
Below, we have outlined three use cases that demonstrate how utilizing pre-defined XPath can aid in extracting desired content.

Define-XML

As mentioned above, the created define.xml or submission package of each clinical trial need to be validated. Extracting content programmatically improves efficiency.

It is implemented using extensions to the CDISC Operational Data Model (ODM) XML schema. There are around 150 possible elements mentioned in ODM Specification[2] and 98 possible elements mentioned under ODM-Study Element. Tailoring each element and attribute to gather all essential information during the extraction phase could entail a considerable amount of manual effort and also carries a high risk of errors.

In our design, pre-specified XPath can be used to extract contents from both ADaM and SDTM define.xml. For example, all contents for elements whose node name is “MethodDef” or “ItemDef” or “ItemRef”, etc will be extracted into separate tables. See Figure 7. Data from define.xml or eSUB Data Check for an example of the MethodDef table. These tables can be subsequently saved to SAS datasets and to be used in SAS environment to implement consistency checks with source data. The advantage of using XPath is that they are easier to create and update compared to a template mapper file that covers the entire document.



OID	Name	Type	Description	TranslatedTex	lang
filter data..					
MT.ADAE.ADURN	Algorithm to d	Computation			
				AE.AEDURDD	
				AE.AEDURDD	en
MT.ADAE.AEGMIF	Algorithm to d	Computation			
				Y if AEDECOD i	
				Y if AEDECOD i	en
MT.ADAE.AENDT	Algorithm to d	Computation			
				input(strip(AE	
				input(strip(AE	en
MT.ADAE.AENDTM	Algorithm to d	Computation			
				input(strip(AE	

Figure 7. Data from define.xml or eSUB Data Check

XML for Excel

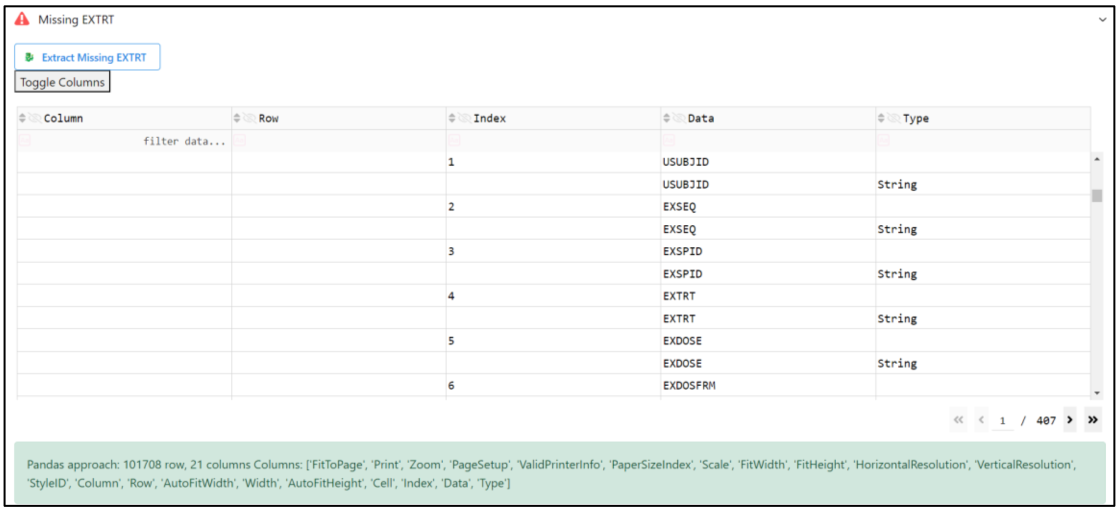
The drawback of SAS output Excel files lacking the ability to apply styles in Excel can be overcome by utilizing the ODS Tagsets.ExcelXP module in SAS. The difference is that the former outputs an Excel file while the latter outputs an XML file (which utilizes the Microsoft Office 2003 XML schema and can be opened directly in Microsoft Excel). In the case where the content within an XML file needs to be read and utilized, this serves as another prime example of the demand for XML file parsing among SAS users.

Implementing the extraction of specific parts from this type of XML, such as the content of a specific sheet or column, can be achieved through various approaches in Python. For example, in Pandas, by defining the XPath, you can extract all the content of a certain sheet or a certain column in Excel. Green nodes in below figure (Figure 9. Element-Attribute Relationships for XML Created by ODS Tagsets) shows elements’ relationships in an XML for Excel.

Example of XPath to extract all contents in a worksheet element (under namespace “xmlns”) which has an attribute called “Name” (under namespace “ss”) and attribute’s value contains string “EXTRT_missing”.

XPath = "//xmlns:Worksheet[contains(@ss:Name,'EXTRT_missing')]//"*

Figure 8. Missing EXTRT Sheet from XML shows target contents from a “EXTRT_missing” worksheet.



Missing EXTRT

Extract Missing EXTRT

Toggle Columns

Column	Row	Index	Data	Type
filter data...				
		1	USUBJID	
			USUBJID	String
		2	EXSEQ	
			EXSEQ	String
		3	EXSPID	
			EXSPID	String
		4	EXTRT	
			EXTRT	String
		5	EXDOSE	
			EXDOSE	String
		6	EXDOSFRM	

« ‹ 1 / 407 › »

Pandas approach: 101708 row, 21 columns Columns: ['FitToPage', 'Print', 'Zoom', 'PageSetup', 'ValidPrinterInfo', 'PaperSizeIndex', 'Scale', 'FitWidth', 'FitHeight', 'HorizontalResolution', 'VerticalResolution', 'StyleID', 'Column', 'Row', 'AutoFitWidth', 'Width', 'AutoFitHeight', 'Cell', 'Index', 'Data', 'Type']

Figure 8. Missing EXTRT Sheet from XML

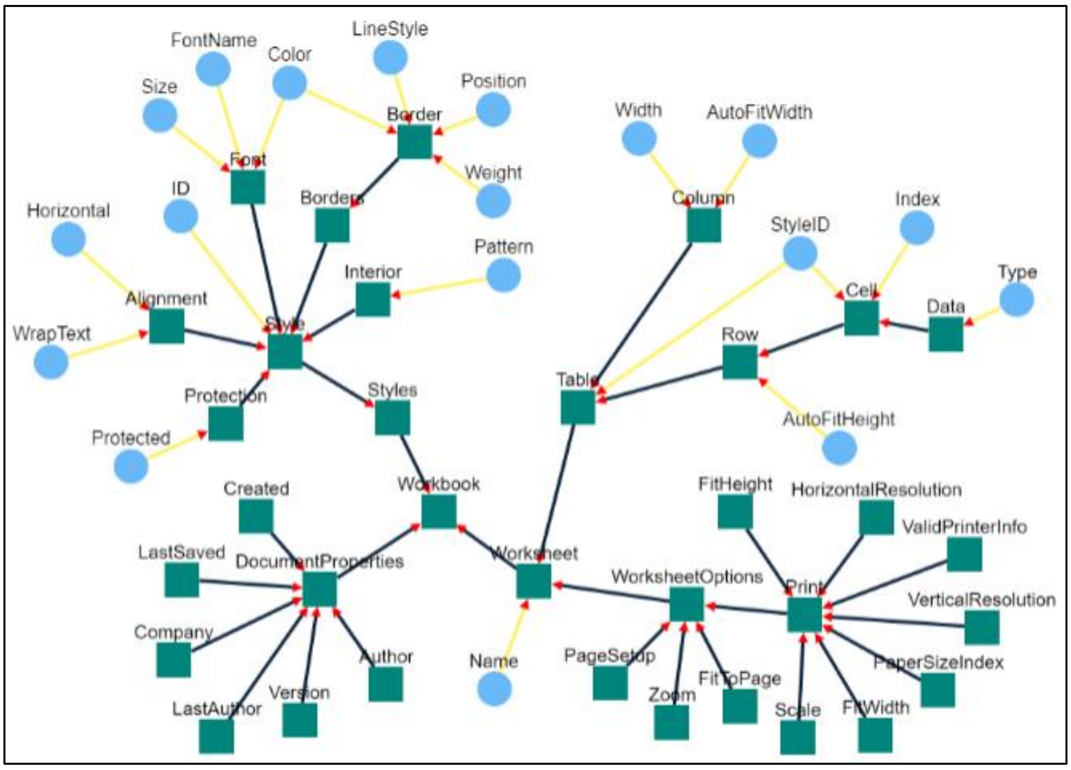


Figure 9. Element-Attribute Relationships for XML Created by ODS Tagsets

Extraction of aCRF Page Numbers

The SDTM aCRF mappings are inherently PDF annotations that can be exported into the XFDF (XML Forms Data Format). XFDF files are structured as an XML hierarchy containing elements and attributes, following the XML standard syntax. The root element is typically <*xfdf*>, and within it, various elements represent form fields, annotations, and other related information. XFDF annotations contain full information to recreate the annotation in a PDF document, including size and position on the page, color, and attached comments. For SDTM aCRF, the page on which each annotation is put can be used to crosscheck the page numbers in define.xml. The annotated texts are the content of XML element <*p*>, and the page information is stored in the attribute named *page* inside element <*freetext*>, which is an ancestor of element <*p*>. See below Figure 10 Element Relationships for XML from aCRF XFDF file for the element structure of PDF annotations.

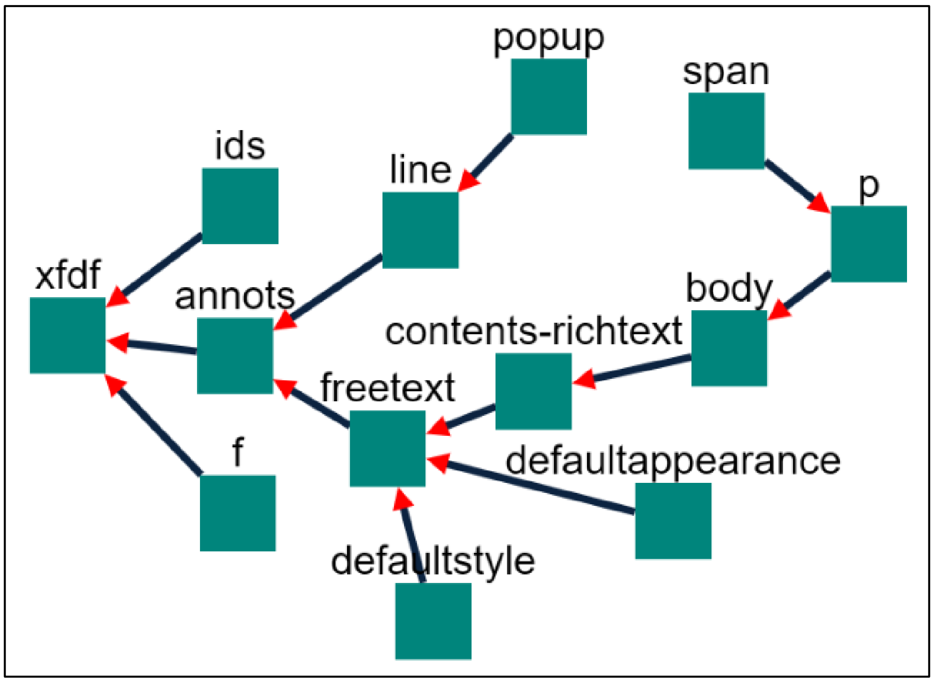


Figure 10 Element Relationships for XML from aCRF XFDF file

With the unique attributes presented on the XML basic information table, we can search the keyword “page” to locate the page attribute and its enclosing element. Then we can utilize the node tree graph described in the XML content section to get a visual representation of the relationships and hierarchies for the <*freetext*> element within the whole node tree.

DISCUSSION

1. XPath Understanding

XPath plays a crucial role in our approach, enabling several functionalities. While most of the XPath syntax is straightforward for Python users, some aspects may not be as intuitive. One feature of our solution is to assist users in defining their own XPath expressions for extracting content of interest. Another feature allows for content extraction from pre-specified XPath, enabling the saved content to be used for other purposes such as consistency checks for submission deliverables or identifying “Missing EXTRT” worksheet, as previously mentioned.

Example XPath to extract all contents from multiple elements by a given node name.

XPath = "//{namespace}:{localnodename}|//{namespace}:{localnodename}//"*

2. Namespaces for different levels

According to W3C XML specification, the scope of a namespace declaration extends from the beginning of the start-tag in which it appears to the end of the corresponding end-tag, excluding the scope of any inner declarations with the same name (default or prefixed). In this case, inner declarations will overwrite the previous declarations. It is likely for users to encounter XML files in which the same element name (local name) is bound to different namespace names (URI) or the default namespace is bound to different URIs in nested elements. The lxml package supports XPath definition in below pattern so that those ambiguities can be avoided.

XPath = "//[local-name()='<node name>' and namespace-uri()='<uri>']"*

3. Extracted Content from Pandas Approach

When reading data from an XML file and writing it into a Pandas DataFrame, a record typically represents an element or node in the XML file. Specifically, each row (or record) usually corresponds to an element in the XML file, while each column corresponds to an attribute or sub-element of that element. This may vary depending on the structure of the XML file and the parsing method.

Although all content from XML can be extracted, the output format may not always meet the requirements of the data format you need. For example, the content of a parent-level element is not retained in the record output of a child-level element. Sometimes these contents are related and require further processing to reflect this association. Therefore, you can add processing in Python to transform this data into another format, or you can process the data into the format you want after outputting it to another format.

CONCLUSION

The tool discussed here was developed as a proof of concept to showcase the benefits of Python in clinical trials analysis and reporting (A&R). By using this tool, the need for auxiliary files like XML map files to parse XML files will be eliminated, thereby simplifying the process. This helps statistical programmers and A&R standards developers understand the XML files they have at hand, quickly grasp the basic information of the XML, sort out the relationships between various elements in the file, and have access to the relevant information that can be used downstream. We believe this is crucial for better handling and utilizing the file. This tool helps users understand the XML structure and decide for themselves what content to extract and whether the extracted content is correct.

Ultimately, we will continue to strive to make these useful functions more accessible to a wider audience, benefiting various developers and users in the end.

REFERENCES

1. "ODM v2.0 XML Schema". Available at https://github.com/cdisc-org/DataExchange-ODM
2. "ODM v2.0 Specification". Available at https://wiki.cdisc.org/display/PUB/General+Elements
3. "Extensible Markup Language (XML) 1.0 (Fifth Edition)". Available at https://www.w3.org/TR/xml/
4. "XML Forms Data Format Specification". Available at https://www.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/XFDF_Spec_3.0_2012.pdf

ACKNOWDEDGMENTS

We would like to thank Amy Gillespie, Susan Kramlik, Suhas Sanjee, Guowei Wu, Majdoub Haloui, Jane Liao and Nicole Zhang for their review and valuable comments on this paper.

CONTACT INFORMATION

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

Danfeng Fu
MSD China
+86 (21) 22118613
dan.feng.fu@merck.com

Dickson Wanjau
Merck & Co., Inc., Rahway, NJ, USA
+1 (610) 2910922
dickson.wanjau@merck.com

Ben Gao
MSD China
+86 (21) 22112893
peng.gao4@merck.com