

A Duet of SAS and Python Generating XML File for Results Disclosure to ClinicalTrials.gov and EudraCT



PHUSE2022 ET09 Story



Linghui Zhang
Sr. Principal Statistical Programmer

linghui.zhang@jazzpharma.com



Jazz Pharmaceuticals®

Outline

- Overview of Clinical Trial Data Disclosure
- Result information disclosed to public
 - Content of result data
 - Common practice to disclose result data
- The structure of XML file for disclosure
 - XML 101
 - XML schema for result disclosure
- Generation of XML file
 - Programming framework with SAS and Python
 - Generate XML by programming Python
- Conclusion

Clinical Trial Data Disclosure

🕒 Legal requirements

- 42 CFR Part 11 effected in 2017 in US
- Clinical Trial Regulation EU No. 536/20142 effected in 2014 in EU

🕒 Ethical principle

- Provide both the public and the professionals with a comprehensive and informative database of clinical trials
- Promote the better-informed use of medicines
- Accelerate scientific progress and benefit public health in the future

🕒 Sponsor's responsibility

- Register clinical trials & disclose results information publicly
- \$\$\$ penalty if registrations and disclosures are not made within the predefined time periods

🕒 Submission systems

- The ClinicalTrials.gov (CT.gov) Protocol Registration and Results System (PRS)
- European Clinical Trials Database (EudraCT)

Result Data Disclosed to Public

- Four basic elements of result information
- Three categories by requirement:
 - Required
 - Conditionally required
 - Optional
- Discordance between CT.gov and EudraCT
- All the result data is available in CSR.
- References
 - <https://prsinfo.clinicaltrials.gov/>
 - <https://eudract.ema.europa.eu/index.html>

ClinicalTrials.gov (EudraCT)

Participant Flow (Subject disposition)

Started

Completed

Not Completed

Reasons for Not Completed

Baseline Characteristics

Age

Sex/Gender

Race

Outcome Measures (End points)

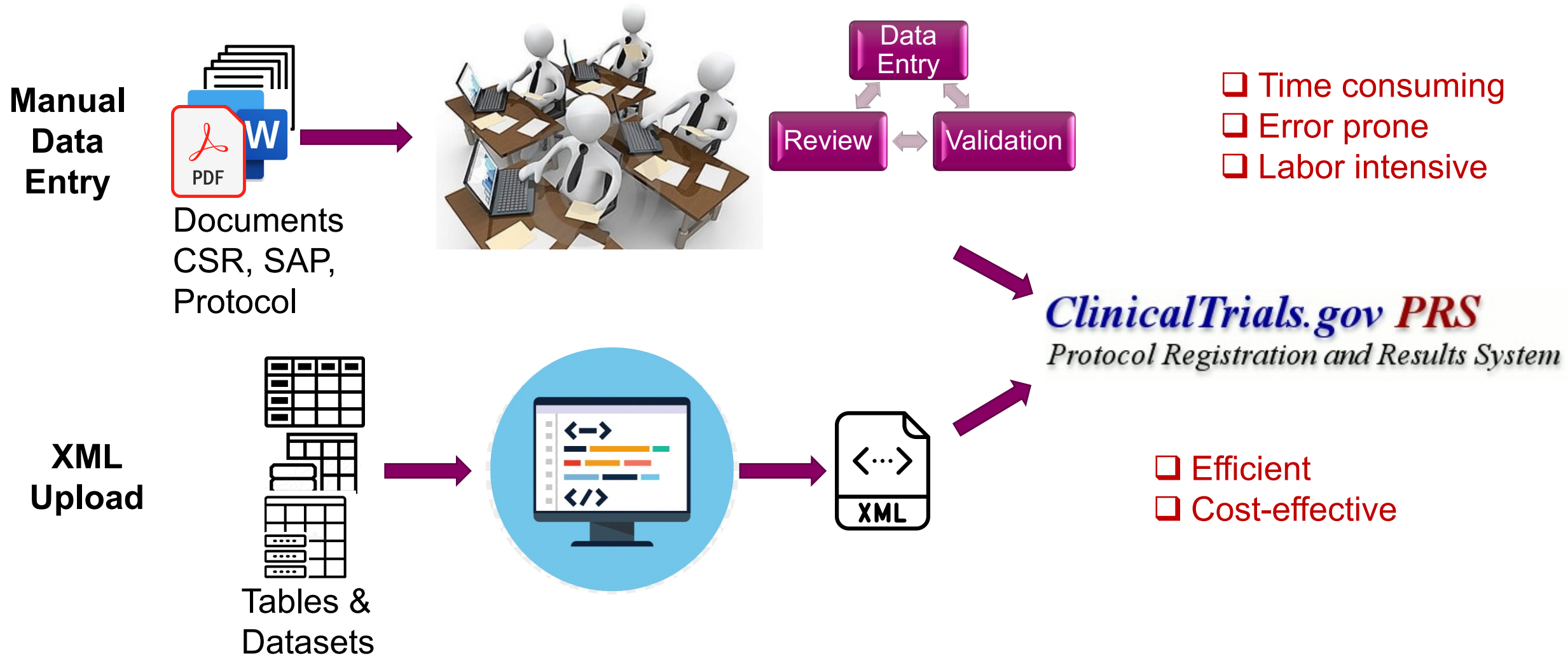
Adverse Event Information (Adverse events)

All-Cause Mortality

Serious Adverse Events

Other Not Including SAEs

Operations to Disclose Result Data



XML (Extensible Markup Language)

- ❶ A file format allowing for data exchange between different operating systems.
- ❷ Data in XML format is stored as text.
- ❸ Tree structure
 - Ordered (hierarchical)
 - Labeled (unique tag name)
- ❹ XML element - the node of the tree
 - Building blocks of an XML
 - An element can contain
 - Text
 - Attributes (Must be Quoted)
 - Other elements

XML prolog (optional)

```
<?xml version="1.0" encoding="UTF-8"?>
```

Root element -

```
<table>
```

Second-level element

```
  <pat source="Medical History"
    category="Demographic">
    <subject>100-01</subject>
    <gender>Male</gender>
    <age>17</age>
    <diagnosis/>
    <treatment date="2022-01-15"/>
  </pat>
</table>
```

Empty element

Tag Name	Attributes	Content
<code><pat</code>	<code>category="Demographic" source="Medical History"</code>	<code></pat></code>
Start/Opening tag "<>"		End/Closing tag "</>"
Element		

XML File Confirming to CT.gov PRS Schemas

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
- <prs:study_collection xmlns:prs="http://clinicaltrials.gov/prs">
```

```
- <clinical_study partial_upload="true">
```

```
- <id_info>
```

```
  <org_name>Jazz</org_name>
```

```
  <org_study_id>CTgov-001</org_study_id>
```

```
</id_info>
```

```
- <rrs:result partial_upload="true" xmlns:rrs="http://clinicaltrials.gov/rrs">
```

```
  + <participantFlow>
```

```
  + <baseline>
```

```
  + <outcomeMeasures>
```

```
  + <reportedEvents>
```

```
</rrs:result>
```

```
</clinical_study>
```

```
</prs:study_collection>
```

Must-haves: <id_info> contains two IDs, which are required to correctly link result data to protocol information in CT.gov

<result> has four parallel sub-elements to disclose study result.

Partial upload: uploading new elements while retaining other elements from the previous version of the record. Normal upload: record is completely replaced by the content in the XML file.

From PRS Result View to XML file

Recruitment Details			
Pre-assignment Details			
Arm/Group Title	Drug-A	Placebo	Total (Not public)
▼ Arm/Group Description	Participants were administered Drug-A...	Participants were administered Placebo...	
Period Title: Open-label Period			
Started	7	7	14
Completed	7	7	14
Not Completed	0	0	0
Period Title: Randomized Withdrawal Period			
Started	7	7	
Completed	6	7	
Not Completed	1	0	1

```

- <participantFlow>
  - <participantFlowGroups>
    + <flowGroup id="ParticipantFlow-ParticipantFlowGroup.1">
    + <flowGroup id="ParticipantFlow-ParticipantFlowGroup.2">
  </participantFlowGroups>
  - <periods>
    + <period>
    + <period>
  </periods>
  <preAssignmentDescription/>
  <recruitmentDetails/>
  <typeUnitsAnalyzed/>
</participantFlow>

```

```

- <period>
  + <completedMilestone>
  <dropWithdrawReasons/>
  <milestones/>
  + <startedMilestone>
  <title>Open-label Period</title>
</period>

```

```

- <flowGroup id="ParticipantFlow-ParticipantFlowGroup.1">
  <description>Participants were administered Drug-A...</description>
  <title>Drug-A</title>
</flowGroup>

```

```

- <completedMilestone>
  <comment/>
  - <milestoneAchievements>
    - <milestoneAchievement>
      <reportingGroupId>ParticipantFlow-ParticipantFlowGroup.1</reportingGroupId>
      <comment/>
      <subjectsAchieve>7</subjectsAchieve>
      <subjectsNaComment/>
      <unitsAchieve/>
    </milestoneAchievement>
    + <milestoneAchievement>
  </milestoneAchievements>
</completedMilestone>

```

- The tabular structure (columns x rows) in the PRS result view is converted to tree (vertical) structure in xml file.
- Two must-have elements under <participantFlow>:
 - <participantFlowGroups> for treatment group
 - <periods> for study period
- Uneven/Unbalanced elements

From CSR Table to PRS Result View

Disposition Table in CSR

	Drug-A (N=7)	Placebo (N=7)	Total (N=14)
Open Label Period			
Completed Study, n (%)			
Yes	7 (100.0)	7 (100.0)	14 (100.0)
No	0	0	0
Randomized Withdrawal Period			
Completed Study, n (%)			
Yes	6 (85.7)	7 (100.0)	13 (92.9)
No	1 (14.3)	0	0

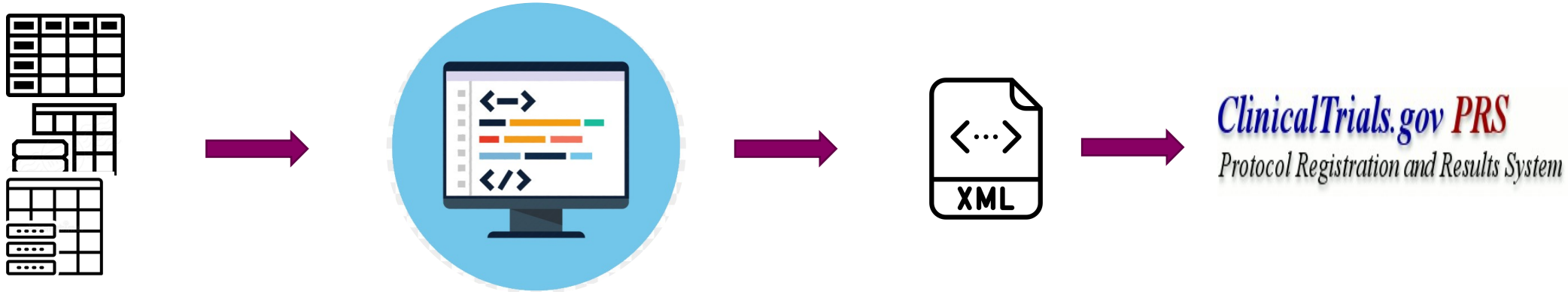
- ☐ Elements Required
- ☐ Elements calculated automatically
 - Not Completed, Percentage, Total
- ☐ CSR tables contain most of the information in the PRS result view.
- ☐ Elements not in CSR Tables

Recruitment Details, Pre-assignment Details, Arm/Group Description

Participant Flow in CT.gov

Recruitment Details			
Pre-assignment Details			
Arm/Group Title	Drug-A	Placebo	Total (Not public)
Arm/Group Description	Participants were administered Drug-A...	Participants were administered Placebo...	
Period Title: Open-label Period			
Started	7	7	14
Completed	7	7	14
Not Completed	0	0	0
Period Title: Randomized Withdrawal Period			
Started	7	7	14
Completed	6	7	13
Not Completed	1	0	1

Process of Generating Result XML File



1. Data Collection

- Recycle the SAS datasets generated for CSR tables
 - Completed – No further derivations
 - Validated – Consistent to the tables
 - Time-efficient – Format manipulation
- Collect description information not available in CSR

2. XML Development

3. XML Upload & Update

SAS Is Not Good at Generating XML File

CSR table dataset: PF01

Period	LINEORD	LINENAME	TRT1	TRT2
1	1	Open Label Period		
1	2	Completed Study, n (%)		
1	3	Yes	7	7
1	4	No	0	0

```
libname result xmlv2 'C:\CTgov\prd1.xml';
data result.prd1 ;
  set pf01(drop=period LINEORD) ;
run ;
```

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
- <TABLE>
  - <PRD1>
    <LINENAME>Open Label Period</LINENAME>
    <TRT1 missing=" "/>
    <TRT2 missing=" "/>
  </PRD1>
+ <PRD1>
- <PRD1>
  <LINENAME>Yes</LINENAME>
  <TRT1>7</TRT1>
  <TRT2>7</TRT2>
</PRD1>
+ <PRD1>
</TABLE>
```

```
- <completedMilestone>
  <comment/>
  - <milestoneAchievements>
    - <milestoneAchievement>
      <reportingGroupId>ParticipantFlow-ParticipantFlowGroup.1</reportingGroupId>
      <comment/>
      <subjectsAchieve>7</subjectsAchieve>
      <subjectsNaComment/>
      <unitsAchieve/>
    </milestoneAchievement>
    + <milestoneAchievement>
      <comment/>
    </milestoneAchievement>
  </milestoneAchievements>
</completedMilestone>
```

```
data _null_ ;
  FILE "prd1complt.xml" MOD LRECL=1000;
  set pf01 ;
  array trtvar $ trt1 trt2 ;
  if Period=1 & compress(LINENAME)='Completed Study, n (%)' then do ;
    put "<completedMilestone>";
    put " <comment/>";
    put " <milestoneAchievements>";
    do i=1 to 2 ;
      grpId=compress("ParticipantFlow-ParticipantFlowGroup."||put(i,best.)) ;
      put " <milestoneAchievement>";
      put " <reportingGroupId>" grpId "</reportingGroupId>";
      put " <comment/>";
      put " <subjectsAchieve>" trtvar[i] "</subjectsAchieve>";
      put " <subjectsNaComment/>";
      put " <unitsAchieve/>";
      put " </milestoneAchievement>";
    end ;
    put " </milestoneAchievements>";
    put "</completedMilestone>";
  end ;
run ;
```

Difficult for SAS:

- × Output uneven structures
- × Define XML element
- × Form tree structure
- × Debug
- × Display tree view



- ❶ Object-oriented programming
- ❷ Define objects
 - Numbers, strings, files
 - Format library
 - Dataset (Dataframe)
 - An element of XML tree
 - XML tree
- ❸ Create programming model based on the concept of objects.

Use Python to Generate XML

```
import os
import pandas as pd
from lxml import etree as et
```

} Import libraries/packages/functional modules

```
dtdir = r"C:\CTgov\results"
outdir= r"C:\CTgov\results\xml"
```

} Define Input/Output path

```
pfperiod=pd.read_sas(os.path.join(dtdir,"pf02period.sas7bdat"),
                      format='sas7bdat',encoding='utf-8')
pfgrpprd=pd.read_sas(os.path.join(dtdir,"pf03grpprd.sas7bdat"),
                     format='sas7bdat',encoding='utf-8')
```

} Import SAS datasets to Python

```
dt_period=pd.DataFrame({'PERIODTITLE': pfperiod.PERIODTITLE})
dt_grpprd=pd.DataFrame({
    'GRPID'      : pfgrpprd.GRPID,
    'GRPTITLE'   : pfgrpprd.GRPTITLE,
    'GRPDESC'    : pfgrpprd.GRPDESC,
    'NCOMPLT'    : pfgrpprd.NCOMPLT,
    'NSTART'     : pfgrpprd.NSTART
})
```

} Define dataframes and variables

- <participantFlow>

- <participantFlowGroups>

- <flowGroup id="ParticipantFlow-ParticipantFlowGroup.1">

<description>Participants were administered Drug-A...</description>

<title>Drug-A</title>

</flowGroup>

+ <flowGroup id="ParticipantFlow-ParticipantFlowGroup.2">

</participantFlowGroups>

+ <periods>

<preAssignmentDescription/>

<recruitmentDetails/>

<typeUnitsAnalyzed/>

</participantFlow>

Advantages of Python over SAS:

- ✓ Generate uneven structures
- ✓ Define XML element
- ✓ Form tree structure
- ✓ Easy to debug
- ✓ Display tree view

rootFlow=et.Element('participantFlow')

Define the root element <participantFlow>

rootRptGrps=et.SubElement(rootFlow,'participantFlowGroups')

Define the sub-element
under <participantFlow>

for row in dt_grpprd.iterrows():

rootFlowRptGrp=et.SubElement(rootRptGrps,'flowGroup')

flowGrpDesc=et.SubElement(rootFlowRptGrp,'description')

flowGrpTitle=et.SubElement(rootFlowRptGrp,'title')

rootFlowRptGrp.attrib["id"]=str(row[1]['GRPID'])

flowGrpTitle.text=str(row[1]['GRPTITLE'])

flowGrpDesc.text=str(row[1]['GRPDESC'])

Apply for-loop on even
sub-element <flowGroup>
under
<participantFlowGroups>

```

import os
from lxml import etree as et
outdir= r"C:\CTgov\results\xml"
ae = et.parse(os.path.join(outdir,"aeterm.xml"))
pf = et.parse(os.path.join(outdir,"ptflow.xml"))
om = et.parse(os.path.join(outdir,"outcome.xml"))
bs = et.parse(os.path.join(outdir,"baseline.xml"))
participantFlow = pf.getroot()
baseline          = bs.getroot()
outcomeMeasures  = om.getroot()
reportedEvents   = ae.getroot()

```

Import XML files

Get the root information
of the xml files

```

xmlns_uris = {'rrs':'http://clinicaltrials.gov/rrs'}
xmlns_upis = {'prs':'http://clinicaltrials.gov/prs'}
rootPRS=et.Element('{%s}study_collection'%xmlns_upis['prs'],nsmmap=xmlns_upis)
rootClinical=et.SubElement(rootPRS,'clinical_study')
rootClinical.attrib["partial_upload"]=str("true")
rootIDInfo = et.SubElement(rootClinical,'id_info')
OrgName     = et.SubElement(rootIDInfo,'org_name')
OrgStudyID  = et.SubElement(rootIDInfo,'org_study_id')
OrgName.text='Jazz'
OrgStudyID.text='CTgov-001'

```

Define the namespace of prefix

Define root element
<study_collection>, its
attribute and sub-elements

Define <result> under <clinical_study>

```

rootResult=et.SubElement(rootClinical,'{%s}result'%xmlns_uris['rrs'],nsmmap=xmlns_uris)
rootResult.attrib["partial_upload"]=str("true")
rootResult.insert(0,participantFlow)
rootResult.insert(1,baseline)
rootResult.insert(2,outcomeMeasures)
rootResult.insert(3,reportedEvents)
tree = et.ElementTree(rootPRS)
tree.write(os.path.join(outdir,"CTgov-001.xml"), pretty_print=True,
           xml_declaration=True, encoding="utf-8")

```

Pile the sub-elements under <result>

Define and Output to tree view

Conclusions

- 🔄 Design and implement an end-to-end disclosure strategy
 - Exclude manual data entry
 - Disclose result accurately and efficiently
 - Recycle SAS datasets generated for CSR tables
 - Reduce the workload on validation and review due to manual data entry
 - Facilitate review cycles, quality control checks, final disclosure.
 - Simplify the workflows throughout the disclosure lifecycle
- 🔄 Unite the open source of Python with the analytic power of SAS
 - SAS – Procedure-oriented programming language
 - Leader and industry standard in analytics
 - Programming framework of data management and analytics of clinical trial studies
 - Python – Object-oriented programming language
 - Open source - tremendous functional modules
 - Integrate SAS with Python to gain more efficiency and streamline the analysis process