

define.xml (noun): Fear Inducing Task for SAS Programmers

Kjersten Offenbecker; Kirsty Lauderdale, Covance, Inc.;
Antonio Cardozo, Spaulding Clinical

ABSTRACT

Faced with creating the define.xml, whether for SDTM or ADaM, programmers tend to shudder with fear. Why is that? Maybe because it tends to be tedious, manual and is not 'defined' very well. What needs to be included, what should it look like and how on earth do you create it? We all know it's a vital piece of any submission and most sponsor companies now require it, even if they have no idea what it is or why they need it. Many companies have a 'home-grown' system (a.k.a. macro) which they use to generate this, but did you know that there is a better way. In our paper, we will show you how to take the fear out of the define.xml creation process.

INTRODUCTION (NOUN): WHERE WE ARE AND WHERE WE ARE GOING TO TAKE YOU

Historically the creation of a define.xml document has been something we programmers absolutely dread, its generally a time-consuming, complicated process and comes when we have very little 'free' time to complete.

Clients tend to think that creating the define.xml is as easy as pressing a button, not really grasping all the ins and outs and involved complexities – value level meta-data, bookmarking and hyperlinking to name a few. Then there is always the knowledge gap – do we need to include every possible value in the code lists, or only the ones that were present in the data? Who do we turn to for advice, and to answer these questions? Does our company have a macro or a tool, or some 'magical' process we should be following?

Surprisingly there actually is a tool we can use, and although it's not as marvelous as 'just press a button', it does pose a nice, simple solution, and because its free, yes - you read that right, your company should not object to you using it!

Pinnacle 21 allows you to create a complaint free – oops we mean, compliant version 2.0 define.xml and define.pdf, and we all know how difficult that usually is! With some minimal user input, some basic information, and a little know-how the process is fast and pretty painless, and this paper will help guide you through this process without your hair turning grey.

We intend to show how to do this using SDTM as our base, however these same principles apply to ADaM as well, so it's like a BOGO deal.

PROGRAMMING SPECIFICATIONS (NOUN): KEY TO A ROBUST DEFINE.XML FILE

The programming specifications tab created by Pinnacle 21, henceforth referenced as P21, can be rather overwhelming the first (and second) time you look at it. There are a huge numbers of tab sheets and each has its own vast array of columns, most of which we have likely never previously entertained thoughts of.

So, to get started we will do a quick overview of each one and guide you in how to populate the applicable information.

define.xml (noun): Fear Inducing Task for SAS Programmers, continued

STUDY TAB

The study tab is a nice easy one to get started with, you should have this information down pat - simply enter the basic information describing your study.

Example – Study Tab

Attribute	Value
StudyName	Protocol Name
StudyDescription	Protocol Title
ProtocolName	Protocol Number
StandardName	SDTM-IG
StandardVersion	3.2
Language	en

DATASETS TAB

This tab contains the high-level overview of each dataset, the name, description, class, structure purpose, key variables, as well as other basic information in the define. Most of this information can be located in the SDTM-IG, as well as in your data.

We cannot stress enough how important it is to ensure that you cross reference the key variables with your data, these keys must identify a unique record within your data and they should be the exact same as the sort order in your dataset.

Example – Datasets Tab (zoom in to see better)

Datas	Description	Class	Structure	Purpose	Key Variables	Repeating	Referenc	e Data	Comment
AE	Adverse Events	EVENTS	One record per adverse event per subject	Tabulation	STUDYID,USUBJID,AEDECOD,AESTDTC	Yes	No		
CM	Concomitant Medications	INTERVENTIONS	One record per recorded medication occurrence or constant-dosing interval	Tabulation	STUDYID,USUBJID,CMTRT,CMSTDTC	Yes	No		
CO	Comments	SPECIAL PURPOSE	One record per comment per subject	Tabulation	STUDYID,RDOMAIN,USUBJID,COSEQ	Yes	No		
DA	Drug Accountability	FINDINGS	One record per drug accountability finding per subject	Tabulation	STUDYID,USUBJID,DATESTCD,DADTC	Yes	No		
DM	Demographics	SPECIAL PURPOSE	One record per subject	Tabulation	STUDYID,USUBJID	No	No		
DS	Disposition	EVENTS	One record per disposition status or protocol milestone per subject	Tabulation	STUDYID,USUBJID,DSDECOD,DSSTDTC	Yes	No		

VARIABLES TAB


This tab is the beating heart of the specifications. It outlines each variable for every dataset your team will be creating and contains all the information we use in programming specifications every day, it's just presented a little differently. A lot of the content is straight-forward, but there are a few sections you want to pay extra attention to. Below is an overview of each column and a couple of pointers on what to watch for.

Example – Populated rows in the Variables tab.

this is the same row connect the picture from origin column

Order	Dataset	Variable	Label	Data Type	Length	Significant Digits	Format	Mandatory	Codelist	Origin
1	VS	STUDYID	Study Identifier	text	11			Yes		Assigned
2	VS	DOMAIN	Domain Abbreviation	text	2			Yes	DOMAIN	Assigned
3	VS	USUBJID	Unique Subject Identifier	text	20			Yes		Assigned
4	VS	VSSEQ	Sequence Number	integer	2			Yes		Derived
5	VS	VSTESTCD	Vital Signs Test Short Name	text	6			Yes	VSTESTCD	CRF
6	VS	VSTEST	Vital Signs Test Name	text	24			Yes	VSTEST	CRF
7	VS	VSCAT	Category for Vital Signs	text	24			Yes	VSCAT	CRF
8	VS	VSORRES	Result or Finding in Original Units	text	5			No		CRF
9	VS	VSORRESU	Original Units	text	9			No	VSORRESU	CRF
10	VS	VSSTRESC	Character Result/Finding in Std Format	text	5			No		Assigned
11	VS	VSSTRESN	Numeric Result/Finding in Standard Units	float	5	2		No		Assigned

define.xml (noun): Fear Inducing Task for SAS Programmers, continued



Origin	Pages	Method	Predecessor	Role	Comment
Assigned				Identifier	
Assigned				Identifier	
Assigned				Identifier	
Derived		MT.VS.VSSEQ		Identifier	
CRF	14 15			Topic	
CRF	14 15			Synonym Qualifier	
CRF	14 15			Grouping Qualifier	
CRF	14 15			Result Qualifier	
CRF	14 15			Variable Qualifier	
Assigned				Result Qualifier	VS.VSSTRESC
Assigned				Result Qualifier	VS.VSSTRESN

- ☐ **Order:** This represents the order the variables appear in the dataset, starting at 1, by domain, and should continue sequentially. For SDTM, the variables should appear in the order specified in the SDTM-IG.
- ☐ **Dataset:** This is the name of the dataset each variable is present in.
- ☐ **Variable:** This shows the name of the variable, this must match the SDTM-IG and your data.
- ☐ **Label:** This is the label for each variable; this also must match the SDTM-IG and your data.
- **Data Type:** This indicates the possible values that you may see – you may the following three criteria; ‘text’, ‘integer’, or ‘float’, any other value will result in a finding in the P21 compliance report.
- ☐ **Length:** This must match the data exactly (this can be done programmatically in many nifty ways).
- **Significant Digits:** For variables with a data type of ‘float’ you want to specify the number of significant figures represented in your data.
- ☐ **Format:** This is most commonly used to specify date formats within ADaM, but can also be used to further explain any format needed. Please note - do not define code lists here; we will get to that section further down.
- **Mandatory:** This is where we set whether the variable is needed. If so, set to ‘Yes’, otherwise set to ‘No’. This needs to be completed for every variable.
- ☐ **Codelist:** This section links directly, using a reference name, to the CODELISTS tab, where each code list is detailed out. It is a best practice to use the SDTM code list name as a reference whenever possible, for example UNIT, RACE, etc.

Every variable contains a finite number of possible results and should have a code list referenced, both character and numeric (i.e. VISITNUM). Fields that are ‘free text’ and, typically, ‘result’ variables will not have a code list, but this is not always true. Run a check on your data, look at it and see if it makes sense to have a code list.

- **Origin:** Where did the data come from? The CRF, electronic data (eDT) or did you derive it or assign a value to it? You should use one of the 6 pre-defined criteria here; ‘CRF’, ‘eDT’, ‘Assigned’, ‘Protocol’, ‘Derived’ and ‘Predecessor’. If you specify something other than one of these values, the P21 compliance report will flag it.

There are cases when more than one of these values is applicable, especially true in supplemental data, in these instances specify all origins separated by a space. Having more than one origin will result in a P21 compliance report issue since P21 does not currently allow for more than one origin.

- **Pages:** If you set your origin to ‘CRF’ then you should specify the pages from the annotated CRF (aCRF) in this section. If the variable appears on more than one page, separate the page numbers with a space. This piece can be populated programmatically, if you wish.
- **Method:** This is often what programmers refer to as the “Derivation”, it’s basically the how-to guide on creating a specific variable. This column works much like the **Codelist** column where it is a reference name that links to the METHODS tab, where the actual derivation lives. When creating the

reference name, it is a best practice to create as MT.<DATASET NAME>.<VARIABLE NAME>, for example MT.VS.VSBLFL.

In general, the method column will be empty for variables where the origin is populated with 'CRF', 'eDT' or 'Assigned'. There will be exceptions to this, but for the most part this should only be populated when the origin is set to 'Derived'.

- **Predecessor:** This is rarely used in SDTM but will definitely show up in ADaM. If a variable is directly populated from an SDTM variable, you will specify the origin as 'Predecessor' and populate the variable in this column, for example, AE.AESTDTC. This also works great for core variables in ADaM, such as ADSL.TRT01P.
- **Role:** This should match the role for each variable from the SDTM-IG.
- **Comment:** This is similar to the Codelist and Method columns where it uses a reference to link to information found on the COMMENTS tab sheet. This is typically additional information that will help a reviewer understand what was done outside of the derivation. When creating the reference name, it is a best practice to create as COM.<DATASET NAME>.<VARIABLE NAME>, for example COM.VS.VSBLFL.

VALUELEVEL TAB

This tab is used for any dataset which contains xxTESTCD, QNAM or PARAMCD, to further explain how these values are created and used within the domain. Each value of xxTESTCD, QNAM or PARAMCD will have a row within this tab.

Example – Populated rows in the ValueLevel tab.

this is the same row connect the picture from origin column

Order	Dataset	Variable	Where Clause	Description	Data Type	Length	Significant	Form	Mandator	Codeli	Origin
1	DA	DAORRES	DA.DATESTCD.EQ.DISAMT	Dispensed Amount	integer	2			Yes		CRF
2	DA	DAORRES	DA.DATESTCD.EQ.RETAMT	Returned Amount	integer	1			Yes		CRF
1	EG	EGORRES	EG.EGTESTCD.EQ.INTP	Interpretation	text	8			Yes		CRF
2	EG	EGORRES	EG.EGTESTCD.EQ.EGHRMN	ECG Mean Heart Rate	integer	3			Yes		CRF
3	EG	EGORRES	EG.EGTESTCD.EQ.PRMEAN	Summary (Mean) PR Interval	integer	3			Yes		CRF
4	EG	EGORRES	EG.EGTESTCD.EQ.PWAVEDUR	Summary (Mean) P Wave Duration	integer	3			Yes		CRF
5	EG	EGORRES	EG.EGTESTCD.EQ.QRSDUR	Summary (Mean) QRS Duration	integer	3			Yes		CRF
6	EG	EGORRES	EG.EGTESTCD.EQ.QTMEAN	Summary (Mean) QT Interval	integer	3			Yes		CRF
7	EG	EGORRES	EG.EGTESTCD.EQ.HRMEAN	ECG Mean Heart Rate	integer	3			Yes		CRF

Origin	Pages	Method	Predecessor	Comment
CRF	25			
CRF	25			
CRF	17			
CRF	17			
CRF	17			
CRF	18			
CRF	18			
CRF	18			
CRF	17			

- **Order:** This represents the order the variables appear in the dataset, starting at 1, by domain, and should continue sequentially.
- **Dataset:** This should be populated with the dataset name.
- **Variable:** This should be populated with the result variable name (usually xxORRES/xxSTRESC, QVAL or AVAL/AVALC).
- **Where clause:** This should be completed using the following structure; <dataset name>.<(xxTESTCD/QNAM/PARAMCD)>.EQ.<variable name>, for example VS.VSTESTCD.EQ.HEIGHT. This will match with the ID column on the WHERECLAUSES tab.
- **Data type:** Please refer to the data type on the VARIABLES tab sheet and insert that value into this section.

- ☐ **Length:** This should be a direct copy from the length populated on the VARIABLES tab sheet.
- ☐ **Significant digit:** Lookup the significant digit values on the VARIABLES tab sheet and insert that value into this section.
- **Format:** This is most commonly used to specify date formats within ADaM but can also be used to further explain any format needed. Please do not define code lists here – we will get to that section further down.
- **Mandatory:** Typically, this is set to “No” for this tab.
- ☐ **Codelist:** This is exactly like the Codelist column on the VARIABLES tab sheet.
- ☐ **Origin:** This is a direct copy from the Origin column on the VARIABLES tab sheet and should match the value present on xxORRES/xxSTRESC, QVAL or AVAL/AVALC. If they are not the same, the P21 compliance report will generate an issue.
- ☐ **Pages:** If your origin is CRF then you should specify the pages from the annotated CRF (aCRF) here. If the variable appears on more than one page, separate the page numbers with a space. This section can be populated programmatically.
- ☐ **Method:** This is exactly like the Method column on the VARIABLES tab sheet.
- ☐ **Predecessor:** This is exactly like the Predecessor column on the VARIABLES tab sheet.
- ☐ **Comment:** This is exactly like the Comment column on the VARIABLES tab sheet.

WHERECLAUSES TAB

For each value level in the VALUELEVEL tab sheet, there is a corresponding where clause in the WHERECLAUSES tab.

Example – WhereClauses Tab

ID	Dataset	Variable	Comparator	Value
DA.DATESTCD.EQ.DISPAM	DA	DATESTCD	EQ	DISPAMT
DA.DATESTCD.EQ.RETAMT	DA	DATESTCD	EQ	RETAMT
EG.EGTESTCD.EQ.INTP	EG	EGTESTCD	EQ	INTP
EG.EGTESTCD.EQ.EGHRMN	EG	EGTESTCD	EQ	EGHRMN
EG.EGTESTCD.EQ.PRMEAN	EG	EGTESTCD	EQ	PRMEAN
EG.EGTESTCD.EQ.PWAVED	EG	EGTESTCD	EQ	PWAVEDUR
EG.EGTESTCD.EQ.QRSDUR	EG	EGTESTCD	EQ	QRSDUR
EG.EGTESTCD.EQ.QTMEAN	EG	EGTESTCD	EQ	QTMEAN
EG.EGTESTCD.EQ.HRMEAN	EG	EGTESTCD	EQ	HRMEAN

- ☐ **ID:** This is populated from the where clause column of the VALUELEVEL tab sheet. These must match exactly.
- ☐ **Dataset:** This is completed with the dataset name insertion.
- ☐ **Variable:** Set this to the result variable name, generally xxTESTCD, QNAM or PARAMCD.
- **Comparator:** Populate this with the way that the value needs to be compared, usually this is set to “EQ” for equals.
- ☐ **Value:** This should be populated with the actual results value that corresponds to the variable, for example xxORRES/xxSTRESC, QVAL or AVAL/AVALC.

CODELISTS TAB

The codelists tab sheet is used to show the reviewer all the codelists, and the values of each, represented in your data. For variables with predefined codelists on the CRFs you must present all possible values listed on the CRF even if they do not appear within your data.

Although a large portion of this task can be completed programmatically, you must double check this part since not all expected values may be represented in your data. For those values that are not in the data, you will need to add them manually to the lists.

The most efficient way to start this process is by checking if the code list exists from the CDISC Controlled Terminology SDTM/ ADaM spec, and then copying those values into the code list tab.

Example – Codelists Tab

ID	Name	NCI Codelist Code	Data Type	Order	Term	NCITermCode	Decoded Value
AEACN	Action Taken with Study Treatment	C66767	text	1	DOSE NOT CHANGED	C49504	
AEACN	Action Taken with Study Treatment	C66767	text	2	DRUG WITHDRAWN	C49502	
AEACN	Action Taken with Study Treatment	C66767	text	3	NOT APPLICABLE	C48660	
AEACN	Action Taken with Study Treatment	C66767	text	4	DRUG INTERRUPTED	C49501	
AEREL	Causality		text	1	No		
AEREL	Causality		text	2	Yes		
AERELNST	Relationship to Non-Study Treatment		text	1	No		
AESEV	Severity/Intensity Scale for Adverse Event	C66769	text	1	MILD	C41338	
AESEV	Severity/Intensity Scale for Adverse Event	C66769	text	2	MODERATE	C41339	
AESEV	Severity/Intensity Scale for Adverse Event	C66769	text	3	SEVERE	C41340	
AGEU	Age Unit	C66781	text	1	YEARS	C29848	

- ☐ **ID:** Populate this from the code list column on the VARIABLES or VALUELEVEL tab sheet. The values **must** match exactly.
- **Name:** Populate similar to the labels for the codelist, for example “Units”, “Race Codes”, etc.
- ☐ **NCI Code List Code:** For code lists from the CDISC Controlled Terminology, you should ensure you use the value from the Code list code column on the SDTM/ADaM Terminology spreadsheet (referencing Column B). This will be the same for all values for a given code list.

Please note: if the code list is not from CDISC Controlled Terminology spreadsheet, this field should be left blank.
- **Data Type:** This is set depending on the variable that is using the code list. It should be set to one of the three expected criteria; “integer”, “float” or “text”.
- ☐ **Order:** This represents the order the values will appear in the dataset, it should start at 1 for each code list and run sequentially.
- ☐ **Term:** This will contain the value of the variable that requires the decode, it will come directly from the data or is represented on the CRF.
- ☐ **NCI Term Code:** For code lists from the CDISC Controlled Terminology, please enter the value from the Code column on the SDTM/ADaM Terminology spreadsheet (reference Column A). This will be unique for each value for a given code list.

It is important to note that if there is not a value in the CDISC Controlled Terminology spreadsheet, this field should be left blank.
- ☐ **Decoded Value:** This will be set to actual value corresponding to the code list. This can be created programmatically and may be blank if not applicable.

DICTIONARIES TAB

This tab is very similar to the CODELIST tab sheet, but it outlines all the dictionaries used in the study. The two most commonly used dictionaries are MedDRA and WHO Drug, but you should also include the ISO dictionaries, and/or the dictionaries referenced in TS domain, for example the ISO8601 and/or NDF-RT.

Example – Dictionaries Tab

ID	Name	Data Type	Dictionary	Version
DRGDICT	Concomitant Medication Dictionary	text	DRGDICT	2017-03
ISO8601	Date Dictionary	text	ISO8601	
MedDRA	Adverse Event Dictionary	text	MedDRA	20.1
NDF-RT	Pharmacological Class Dictionary	text	NDF-RT	2017-06-14
SNOMED	Trial Indication Dictionary	text	SNOMED	2017-03-01
UNII	Treatment Dictionary	text	UNII	2017-04-28

- ☐ **ID:** This should be populated from the code list column on the VARIABLES or VALUELEVEL tab sheet. The values must match exactly.
- ☐ **Name:** Populate with a description of the dictionary.
- ☐ **Data Type:** Type of the data. Usually text.
- ☐ **Dictionary:** What dictionary is it.
- ☐ **Version:** The version of the dictionary.

METHODS TAB

This tab is where all the meat is located; it tells the reviewer how variables are derived and/or assigned. This section should be crafted without using SAS code, to ensure we still allow independent validation processes to be followed, however using pseudo-code is acceptable. The ultimate goal is to have a derivation that is clear and descriptive.

Example – Methods Tab

ID	Name	Type	Description	Expression Conte	Expression Co	Document	Pages
MT.CM.CMENDY	Algorithm to derive MT.CM.CMENDY	Computation	Set to (CMENDTDC-DM.RFSTDTC) + (CMENDTDC >= DM.RFSTDTC).				
MT.CM.CMSEQ	Algorithm to derive MT.CM.CMSEQ	Computation	Sort by key variables and assign sequential integer by USUBJID.				
MT.CM.CMSTDY	Algorithm to derive MT.CM.CMSTDY	Computation	Set to (CMSTDTC-DM.RFSTDTC) + (CMSTDTC >= DM.RFSTDTC).				
MT.CM.EPOCH	Algorithm to derive MT.CM.EPOCH	Computation	Set to 'SCREENING' when CMSTDTC <= SE.SESTDTC when SE.ELEMENT='SCREENING'. Set to 'OPEN LABEL TREATMENT' when SE.SESTDTC <= CMSTDTC <= SE.SEENDTC when SE.ELEMENT = 'OPEN LABEL TREATMENT'. Set to 'BLINDED TREATMENT' when SE.SESTDTC <= CMSTDTC <= SE.SEENDTC when SE.ELEMENT = 'BLINDED TREATMENT'. Set to 'FOLLOW-UP' when SE.SESTDTC <= CMSTDTC <= SE.SEENDTC when SE.ELEMENT = 'FOLLOW-UP'.				

- ☐ **ID:** This should be populated from the Methods column on the VARIABLES or VALUELEVEL tab sheet. The values must match exactly.
- **Name:** This needs to be populated following the set criteria; “Algorithm to derive: <ID value>”, for example “Algorithm to derive: MT.CM.CMENDY”
- **Type:** This represents what type of algorithm this is. Generally, this is set to “Computation”, however it also can be set to “Imputation”.
- **Description:** This is the derivation part, the part that enables us to create the specified variable. For SDTM ensure that you are not referencing any raw variables that are not present in the SDTM anywhere. For ADaM if you are using an SDTM variable, please be sure to include the domain when referencing, for example AE.AESEV='MILD'.
- ☐ **Expression Content:** This can be left blank

- ☐ **Expression Code:** This can be left blank
- ☐ **Document:** This is used to when the derivation is very complicated and needs more explanation. For these, you add the extra detail to a separate document and reference that document in this section. This field will then be an identifier that is also referenced on the DOCUMENTS tab sheet, in the form of COM.VS.VSBLFL, for example. This typically comes into play with ADaM datasets since some of those derivations can be very lengthy.
- ☐ **Pages:** Please ensure any page numbers references are separated by a space.

COMMENTS TAB

This tab is used to add additional information that may be useful for a reviewer.

Example – Comments Tab

ID	Description	Document	Pages
CO.COEVAL	'INVESTIGATOR'.		
CO.IDVAR	Set to one of the following based on the identifying variable: 'S_ODAE', 'DASEQ', 'LBSEQ'.		
CO.IDVARVAL	Set to AE.AESPID when IDVAR='S_ODAE' raw data. Set to DA.DASEQ when IDVAR='DASEQ'. Set to LB.LBSEQ when IDVAR='LBSEQ'.		
CO.RDOMAIN	Set to one of the following based on related domain: 'EX', 'DA', 'LB'.		

- ☐ **ID:** This should be populated from the comments column on the VARIABLES or VALUELEVEL tab sheet. The values **must** match exactly
- ☐ **Description:** This is just like the Methods column on the METHODS tab sheet.
- ☐ **Document:** If you need to reference a separate document in order to explain an item in greater detail, create an ID here following the same guidelines as the Documents column in the METHODS tab sheet.
- **Pages:** If your origin is set to 'CRF' then specify the pages from the annotated CRF (aCRF) in this section. If the variable appears on more than one page separate the page numbers with a space. This piece can be populated programmatically.

DOCUMENTS TAB

This tab is where all the external documents are referenced. These will be linked to the define file for easy access by the reviewer.

Example – Documents Tab

ID	Title	Href
acrf	Annotated Case Report Form	acrf.pdf
sdrgr	Study Data Reviewer's Guide	sdrgr.pdf

- ☐ **ID:** You should create a unique identifier for each external document. You should keep it simple for this.
- ☐ **Title:** Create a document title that clearly depicts what the document is, this text will appear on the define file, so make this meaningful.
- ☐ **Href:** You will populate this as the name of the document, but in file reference format. The file needs to reside in the same location as the data and the define, otherwise P21 will not be able to find it but it will still create a link.

SPECIFICATION AUTOMATION (VERB): HOW TO MAKE SPECIFICATION CREATION EASIER

As noted previously, there are a lot of i's to dot and t's to cross in the specifications. P21 does a wonderful job of telling you when you've missed something, however it's not the most transparent in what that something is. By adding in some automation and programmatic cross checks, we can minimize those risks. Applying automation can help populate areas of the specifications, such as the CODELISTS, VALUELEVEL and WHERECLAUSES tab sheet, and using programmatic cross checks can verify some variable aspects.

AUTOMATION EXAMPLE: CODELISTS TAB POPULATION

Populating the CODELISTS tab sheet can be a beast of a task. Finding all the values in the data, finding the NCI Codelist and NCI Term Code, populating all the xxTESTCD/VISITNUM decoded values is definitely a mind numbing and time-consuming process. If we automate this process, our programming scripts take on the brunt of the hard work for us.

Please note: the following code assumes the Codelist column of the VARIABLES tab sheet has correct NCI Codelist names (if NCI mapping is needed).

Step One - Bring in the Variables tab sheet of the specifications

```
proc import out= study_variables(rename=(codelist=ct))
  datafile= "file_path\ddt_name"
  dbms=xlsx replace;
  sheet="Variables";
  getnames=yes;
run;
```

After import, drop any records where CT is null or contains 'EXTERNALCODELIST' or 'ISO8601' since they are not going to have records in the CodeLists tab sheet.

Step Two – Create Macro Variables for Dataset, Variable, CT and Type

```
data _null_;
  set study_variables end=eof;
  length varnm $30;

  varnm = 'ds' || left(put(_n_,5.));
  call symput(strip(varnm), upcase(dataset));

  varnm = 'var' || left(put(_n_,5.));
  call symput(strip(varnm), upcase(variable));

  varnm = 'ct' || left(put(_n_,5.));
  call symput(strip(varnm), upcase(ct));

  varnm = 'type' || left(put(_n_,5.));

  if Data_Type ne 'text' then type_chk=1;
  else type_chk=0;

  call symputx(strip(varnm), type_chk);

  if eof then call symput('variable_total',trim(left(put(_n_,5.))));
run;
```

This is done for each unique record in the study_variables dataset, and at the same time we create a macro variable representing the total number of records and variable type.

Step Three – Determine Variable Pairs for all values in the data

```
%macro get_codelist(ds, dsout, var, type, ct);  
  /* Make a temporary dataset containing only unique values for the variable passed in */  
  proc sort data=&ds out=_temp(keep=&var) nodupkey;  
    by &var;  
  run;  
  
  data temp (keep=variable term ct);  
    length variable term ct $200;  
    set _temp;  
  
    /* Set the Variable name and Codelist name to merge back onto the specifications later */  
    variable=upcase("&var");  
    ct=upcase("&ct");  
  
    /* Set the value of Term using the correct numeric or character format */  
    %if &type %then  
      %do;  
        term=strip(put(&var,best.));  
      %end;  
    %else  
      %do;  
        term=&var;  
      %end;  
  
    if term not in('.' ' ');  
  run;  
  
  /* Append all records together to use after looping is complete */  
  proc append base=&dsout data= _temp force;  
  run;  
%mend;  
  
%macro get_cts();  
  %do i=1 %to &variable_total;  
    %get_codelist(data.&&ds&i, all_ct, &&var&i, &&type&i, &&ct&i );  
  %end;  
%mend get_cts;  
  
%get_cts();
```

Call the macro so that it loops through each record until it reaches the defined number of total possible records (&variable_total). The appended dataset will contain all possible data values for each codelist.

Step Four – 2 Part Process to get the NCI Codelist and Term Codes

For our example, we'll be using the SDTM Terminology 2016-06-24 version, make sure to import the correct SDTM Terminology document needed for your study, you can use the same code as depicted in step one for importing.

NCI STEP ONE – Import, Sort and Keep Specified Variables

```
proc sort data=SDTM_Terminology_file (where=(Codelist_Code = '' )  
      keep=Codelist_Code code extensible CDISC_Submission_Value)  
      out=ig_ct (rename=(CDISC_Submission_Value=_ct));  
  by CDISC_Submission_Value;  
run;
```

We rename CDISC_Submission_Value to CT to merge onto the dataset created in step one. After merging by CT, you'll have the correct NCI CodeList code associated with each record in the specs.

NCI STEP TWO – Merge on the NCI Codelist Term Code

```
proc sort data=ctlib.ct (where=(Codelist_Code ne ''))
    keep=Codelist_Code code CDISC_Submission_Value CDISC_Synonym_s_)
    out=ig_ct (rename=(CDISC_Submission_Value=term));
    by Codelist_Code CDISC_Submission_Value;
run;
```

We rename CDISC_Submission_Value to Term to match the variable name created in step 3, then sort both datasets by Codelist_Code (from step: NCI Step One) All_CT (from Step 3) by Codelist_Code and Term we can merge. Once merged, we'll have both NCI Codelist Codes and Term Codes for all the applicable values.

STEP FIVE – MERGE ON THE DECODE VALUES

1. Define variables where decode values are needed, for example xxTESTCD and VISITNUM
2. Loop through data to get unique values of variable, such as unique values of xxTEST per xxTESTCD.
3. Merge on decode values.

Once all the merging is done and you have renamed a few variables, you'll have everything you need to populate the Codelist tab. The only manual entry will be any additional codelist values that were collected on the CRF, but are not in the data.

CROSS CHECK EXAMPLE: VERIFYING DATASETS TAB KEY VARIABLES

P21 does not check that all the variables listed in the Key Variables column of the DATASETS tab are in the data. As a study progresses or if a specification template is used, it's possible that some key variables were listed in error, you can manually check all the listed key variables per domain to ensure they are present, or parse the column and compare the values against the data programmatically.

```
/*Import the specification */
proc import out= ddt(where=(dataset ne ''))
    datafile= "file_path\ddt_name"
    dbms=xlsx replace;
    sheet="Datasets";
    getnames=yes;
run;

data ddt (keep=dataset variable);
    length dataset $7 variable $8;
    set ddt;

    /* Determine the number of key variables listed per record by counting the commas */
    num_keys=countw(key_variables, ',');

    do i=1 to num_keys;
        /*split each key var into its own variable and output, one record per dataset per key var */
        variable=strip(scan(key_variables, __i, ', '));
        output;
    end;
run;

/* get a list off all datasets and variable names in the target library*/
proc contents data=data._all_ out=dataset_meta(keep=memname name) noprint;
run;

data dataset_meta (keep=dataset variable);
    length dataset $7 variable $8;
    set dataset_meta;
    /* Map memname to dataset and name to variable to match the structure of the dataset in step 3 */
    dataset=strip(memname);
```

define.xml (noun): Fear Inducing Task for SAS Programmers, continued

```
variable=strip(name);  
run;  
  
/* Sort both datasets by Dataset and Variable*/  
proc sort data=ddt;    by dataset variable; run;  
proc sort data=dataset_meta;    by dataset variable; run;  
  
/* Merge datasets by Dataset and Variable, output if in key variables dataset (step 3) but not step 5 */  
data need_to_delete;  
    merge ddt (in=a) dataset_meta(in=b);  
    by dataset variable;  
    if a and not b;  
run;
```

The same logic can be used to determine if a dataset listed in the DATASETS tab sheet is not in the data. This logic is helpful when a specification template is used as a starting point.

This programmatic approach can also be used to; populate the lengths on the Variables tab sheet, ensure the Origin column values are consistent with supporting attributes, match data types across tabs, confirm defined roles are consistent with CDISC IG roles, and confirm all required methods (per the Variable tab sheet) are present in the Methods tab sheet..

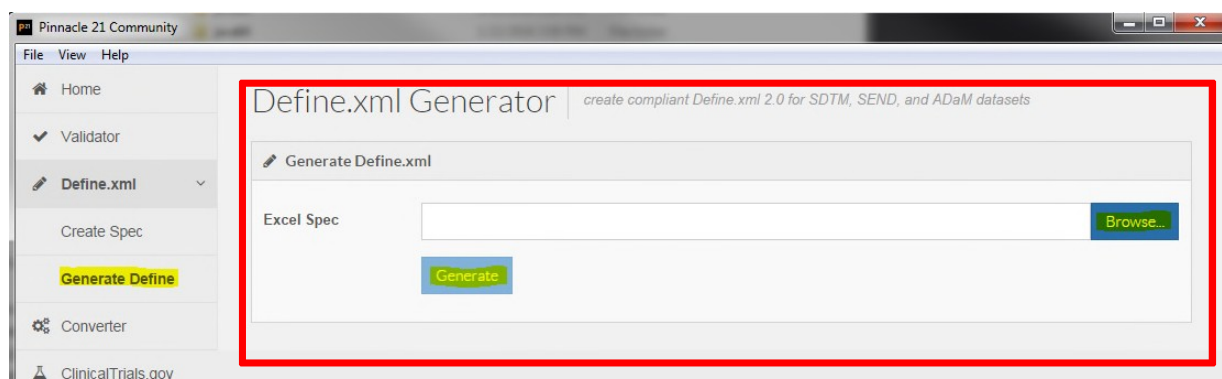
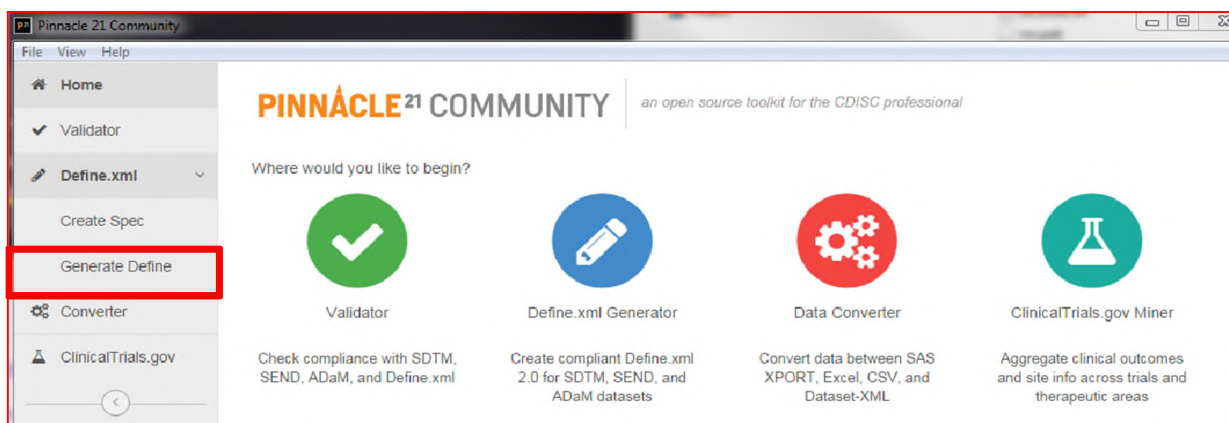
DEFINE GENERATION (VERB): MOVING FROM SCARY TO EASY PEASY

So why have we been afraid all this time? We have a great tool that once you set the specs up makes define generation as easy as 1, 2, 3. We have created a nice wee guide to follow, depicted below, with screenshots, to show you just how simple this whole process can be.

STEP ONE

Open Pinnacle 21 and locate where you placed your specification spreadsheet.

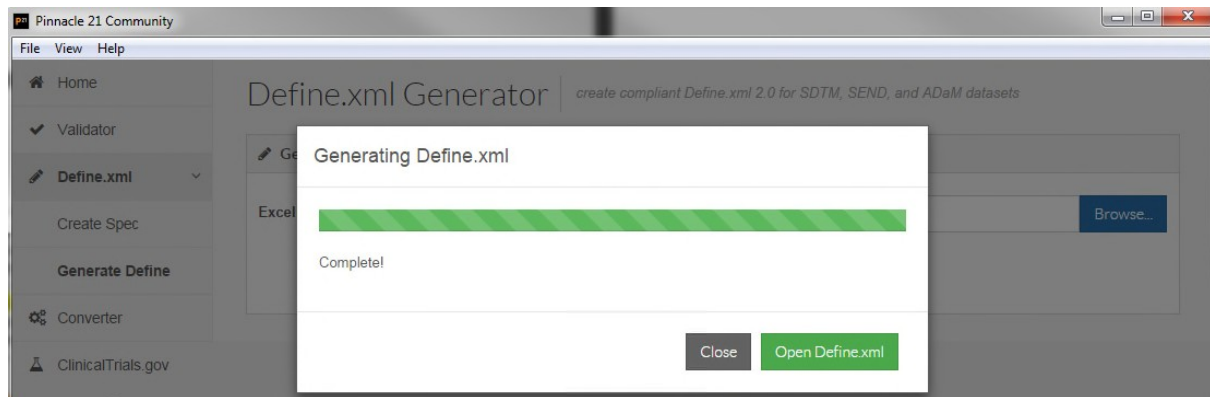
Navigate to the Define.xml section on the left hand side menu, and following the screenshots below select generate.



define.xml (noun): Fear Inducing Task for SAS Programmers, continued

STEP TWO

Click to open and view the define 2.0 file, or if you prefer you can navigate to find the .xml file at the following location on your hard drive: pinnacle21-community\components\reports. Please note that the file will be named with the project name.xml – you will need to rename it to “define.xml” for submission.

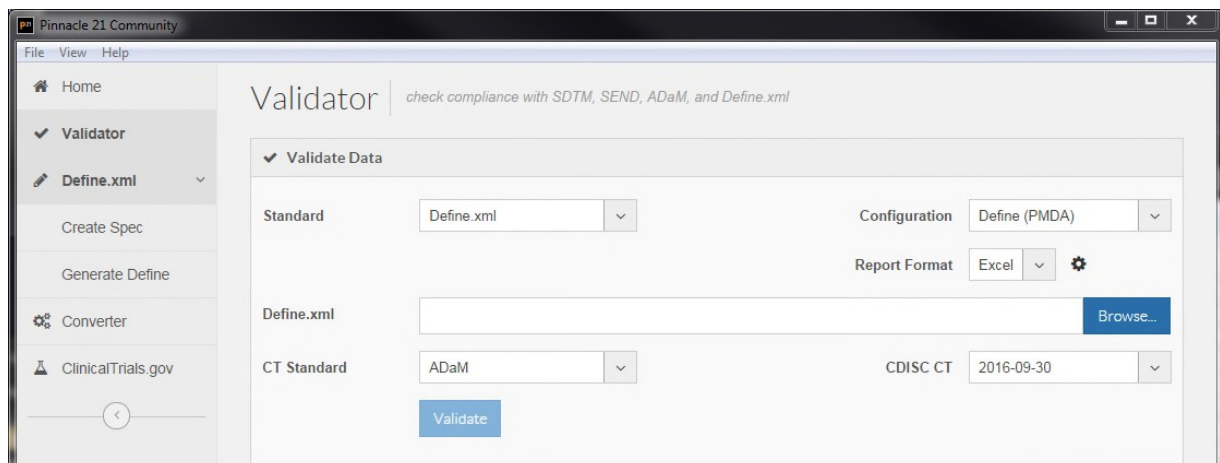


STEP THREE

The last step is to validate your new creation. This is the most involved part after specification creation, but it is not difficult. There are two last parts to follow, and after you complete those you will have a fully compliant define file that your sponsor/team will love, not to mention the FDA really likes these too!

Part One - Standard option pick = “Define.xml”

First we need to validate the define itself, to ensure it is compliant. To do this, navigate to the screen, as shown, below and change any settings as needed.



- Pick Define.xml for the Standard – this will only check the define itself
- ☐ Pick the correct configuration for your study
- ☐ Click browse and locate the define 2.0 generated above
- ☐ Pick the correct CDISC CT version used for your study
- ☐ Pick SDTM or ADaM for CT Standard
- ☐ Click validate

define.xml (noun): Fear Inducing Task for SAS Programmers, continued

Once the P21 validator completes its check, it will generate a report that should look like this. You will need to review this file carefully, and address as the noted findings as much as possible. We will talk about some of the issues to look out for in this report in the next section.

Pinnacle 21 Validator Report					
Configuration: C:\U\ [REDACTED]					
Define.xml: C:\U\ [REDACTED] 4.xml					
Generated: 2018-02-06T15:11:23					
Software Version: 2.2.0					
Issue Summary					
Source	Pinnacle 21 ID	Publisher ID	Message	Severity	Found
DEFINE					
	DD0003		'LF.Data Guide' is not a valid value for 'NCName'.	Reject	2
	DD0003		Missing required 'ID' value for 'def:leaf'	Reject	1
	DD0003		Missing required 'leafID' value for 'def:DocumentRef'	Reject	1
	DD0015		Referenced Document is missing	Error	3
	DD0016		Referenced Method is missing	Error	397
	DD0042		Missing Method reference	Error	5
	DD0054		Missing Class value	Error	2
	DD0068		Invalid use of Length	Error	10
	DD0072		Missing Origin	Error	1
	DD0084		Referenced File is missing	Error	16
	DD0048		Referenced Codelist is missing	Error	33
	DD0074		Invalid Mandatory value	Error	303
	DD0079		Duplicate Term in Codelist 'CL.ADSV_PARAMCD'	Error	6
	DD0079		Duplicate Term in Codelist 'CL.ADUPRS_QSSTRESN'	Error	85
	DD0080		Method 'MT.MT.ADAE.AOCCMIL' is not referenced	Warning	1
	DD0080		Method 'MT.MT.ADAE.AOCCSIL' is not referenced	Warning	1
	DD0080		Method 'MT.MT.ADPAM13.AVALC' is not referenced	Warning	1
	DD0082		Codelist 'CL.ADPAM13_DTYPE' is not referenced	Warning	1
	DD0082		Codelist 'CL.BMIGR1' is not referenced	Warning	1
	DD0082		Codelist 'CL.FREQ' is not referenced	Warning	1
	DD0082		Codelist 'CL.MEDDRA' is not referenced	Warning	1
	DD0082		Codelist 'CL.NRIND' is not referenced	Warning	1
	DD0082		Codelist 'CL.PPROTFL' is not referenced	Warning	1
	DD0082		Codelist 'CL.WHO DRUG' is not referenced	Warning	1
					875

Part Two - Standard option pick = “SDTM”/”ADaM”

Secondly, we want to validate the define along with the data. This step will cross check the two and tell us if we missed anything vital. To do this navigate to the screenshot below and change any settings as fit.

define.xml (noun): Fear Inducing Task for SAS Programmers, continued

- ☐ Pick either SDTM or ADaM for the Standard
- ☐ Pick the correct configuration for your study
- ☐ For the Source Data option, click browse and navigate to the datasets that were used for this define and select them all.
- ☐ For the Define.xml option, click browse and locate the define 2.0 generated above
- ☐ Pick the correct CDISC CT version used for your study
- ☐ Click validate

The resulting report should look very familiar to you, it is the same report you get when you validate the data using P21. You will want to review it in the same way you do when just checking the data, but make sure you pay special attention to any issues that reference the define.xml. These are the issues that tell you something doesn't jive between the data and the define.

We will talk about some of the issues to look out for in this report in the next section.

ISSUES (NOUN): AREAS NEEDING A LITTLE MORE ATTENTION AND/OR EXPLANATION

It does have to be mentioned here, that P21 compliance reports are often not the most straight-forward or easy to understand or reports, and, at times, the compliance messages just leave us scratching our heads. In order to get you moving in the right direction, we have summarized some of the most common messages the report will throw at you and given you some pointers to keep you on track.

This is, by no means, a complete list but in our experience these are the that ones seem to pop up most often. For those of you who want to review all the compliance messages you could ever receive, there is a full list on the Pinnacle 21 website.

DEFINE.XML- ALL BY IT'S LONESOME

P21 Message	Resolution
Missing required <attribute> value for <object>	You are missing a required attribute in the define.
Attribute <attribute> is not allowed to appear in element <element>	You specified an attribute that is not allowed, for example - char/character rather than text
Element in wrong position within Define.xml	A variable is out of order - this might only be in the specs/define, but you should check both specs/define and data to be sure
Element occurs more than once	You have duplicated a variable within a domain/dataset, this is likely an issue in the specs/define
Duplicate Document ID	Typically, this means you duplicated a variable in the define
Duplicate Method OID	Typically, this means you duplicated a variable in the define
Duplicate ValueListDef OID	Typically, this means you duplicated a variable in the define
Referenced Document is missing	You referenced a document but it is not in the same location as the define
Referenced Method is missing	You created a method on the VARIABLES/VALUELEVEL tab but did not define it on the METHODS tab
Referenced Value Level metadata is missing	Typically, means that something does not line up between the VALUELEVEL tab and the WHERECLAUSES tab
Invalid Term in Codelist <codelist>	This is applicable primarily to non-extensible code lists. You likely added a code list when you are not allowed to.
Invalid MedDRA Version <version>	MedDRA version must have a decimal representation
Term/NCI Code mismatch in Codelist <codelist>	You probably entered the wrong Term/NCI Code on the CODELIST tab

P21 Message	Resolution
Missing NCI Code for Codelist <codelist>	You did not enter a NCI Code for a code list that is present in CDISC CT
Unknown NCI Code value for Codelist <codelist>	You probably entered the wrong Term/NCI Code on the CODELIST tab
Missing Pages value	You specified the origin as "CRF" but did not enter the page number(s)
Missing Key Variables value	You did not specify the Key variables on the DATASETS tab
Duplicate KeySequence	You specified the same variable more than once in your key variables
Missing Method reference	You created a method on the METHODS tab sheet but have not referenced it anywhere on the VARIABLE/VALUELEVEL tab sheet
Invalid use of Length	The Length attribute must be empty when DataType is not "integer", "float", or "text".
Invalid use of Significant Digits	The Significant Digits attribute must be empty when DataType is not set to "float".
Missing Origin	You did not specify an Origin for a variable
Invalid Origin Type value	The Origin Type attribute must have a value of 'CRF', 'Derived', 'Assigned', 'Protocol', 'eDT', or 'Predecessor'.
Codelist <codelist> is not referenced	You specified a Code list on the CODELIST tab but did not reference it on the VARIABLE/VALUELEVEL tab
Missing Define XSL	Stylesheet (".xsl") file referenced in Define.xml must exist in the same location as the define file

DEFINE.XML ALONG WITH THE DATA

P21 Message	Resolution
Value for variable not found in user-defined codelist	Your Code list is missing a value that is found in the data
Variable in define.xml is not present in the dataset	You have a variable in your specs that is not in the resulting data
Variable in dataset is not present in define.xml	The opposite of the item above.
Define.xml/dataset variable type mismatch	Your data has the type as one thing but the define says it is something else, for example - in the data the field is a text but in define it is an integer Typically the data is correct.
Domain referenced in define.xml but dataset is missing	You have a domain/dataset in the define but no dataset - this often occurs when you don't have data for something that is on the CRF (i.e. IE), in this case do not include it in the define
Dataset is not present in define.xml	You have a domain/dataset but nothing in the define to explain it. You need to add the domain/dataset to your specs and the define.

CONCLUSION (NOUN): WHY WE THINK P21 IS A GREAT TOOL, MAKING DEFINE.XML GENERATION SOMETHING TO EMBRACE RATHER THAN DREAD

P21 gives us a great tool to create our define.xml and define.pdf files, while making the process straight-forward and relatively pain-free. You can quickly and accurately create compliant files, and reduce the tension headaches/late nights that frequently occur at final deliverable stages.

Our recommendation is to use the functionality of P21 to do your study specifications, and garner those efficiencies through the entire study. If you have already created your specifications, you can retro-fit them, and still use the P21 tool to complete your required files.

In summation, P21 is free, its efficient and effective, it gives you a compliant file, what's not to love?

CONTACT INFORMATION

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

Kjersten Offenbecker
Director, Clinical Analytics
Covance, Inc.
Kjersten.Offenbecker@Chiltern.com
Covance.com

Kirsty Lauderdale
Senior Manager, Statistical Programming
Covance, Inc.
Kirsty.Lauderdale@Chiltern.com
Covance.com

Tony Cardozo
Director of Biometrics
Spaulding Clinical
Antonio.Cardozo@SpauldingClinical.com
SpauldingClinical.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.