

SM06

AUTOCODE: Metadata-Driven Automation for Faster, Traceable SDTM and ADaM Programming

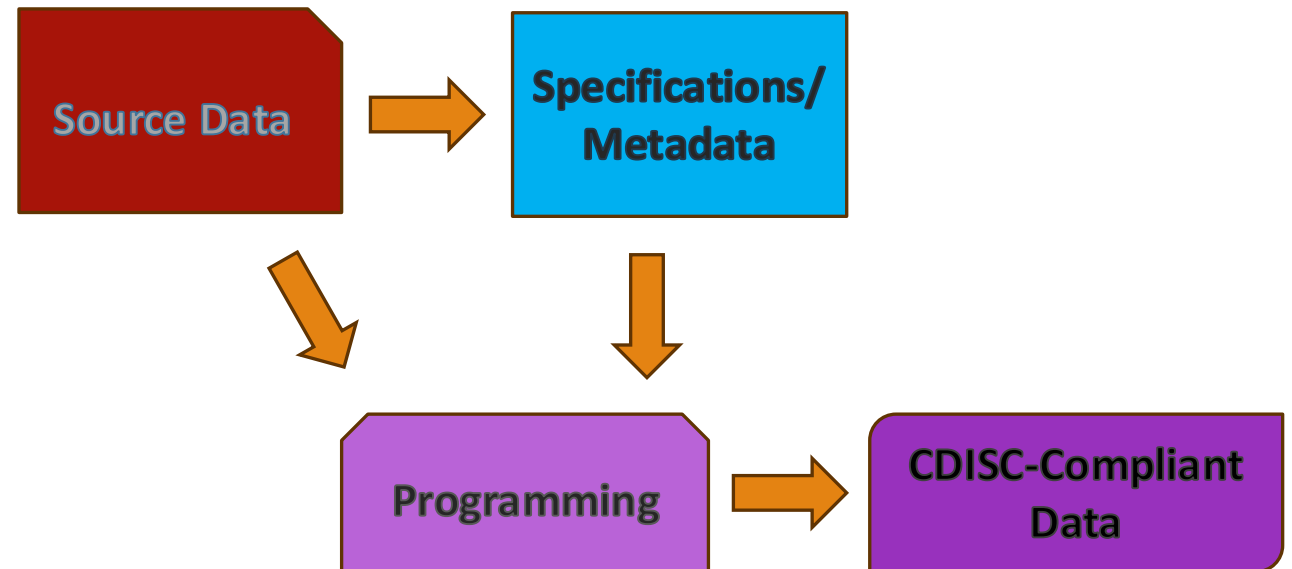
MATTHEW FINNEMEYER, VERTEX PHARMACEUTICALS, INC.,
BOSTON, USA

Matthew Finnemeyer is a veteran SAS programmer for Vertex Pharmaceuticals Inc., focusing on the development of SAS-driven efficiencies and training materials to support departmental activities.

Introduction

When generating SDTM & ADaM datasets, the concept of **traceability** states that a clear link exists between each source “Raw” datapoint and its final form in analysis-ready datasets.

The structure of the source data shapes the dataset-creation specifications and metadata, which guide the specific programming used to generate CDISC-compliant data.



Improving Specification & Programming Pipelines

Generating “template” SDTM & ADaM specifications has become an increasingly common practice; time spent on their development is quickly recouped across repeated use.

Likewise, standardized macros can reduce programming overhead and eliminate “repetitive coding” across studies, but at the cost of reduced “code transparency”; striking a balance between **traceability** and efficiency can be challenging.

Efforts have been made to “automate” the generation of dataset-creation programs by leveraging “pseudocode” in specifications to eliminate repetitive coding without macros; this “pseudocode” may not be acceptable for the DefineXML and may add downstream work.

Automation at Vertex; “AUTOCODE”

For the past four years, Vertex has leveraged an automated approach to generating SDTM dataset-creation programs, using DefineXML-ready specifications/metadata (i.e. no “pseudocode”), and is currently piloting the approach for ADaM datasets.

Termed “AUTOCODE”, this process parses *plain-text* specifications into SAS code, reducing the need for both macros and repetitive manual coding.

As the SAS code is generated directly from the specifications, without manual modification, **traceability** is maintained while study-specific modifications are easy to implement.

AUTOCODE in Action: SDTM Programming

VARIABLE NAME	VARIABLE "SPECIFICATION" (Programming Notes/Comment)
<i>SDTM.MH.MHSPID</i>	MH.IGSEQ
<i>SDTM.MH.MHTERM</i>	Equal to MH.MHTERM. Upper case the variable and remove any special characters.
<i>SDTM.MH.MHLLT</i>	MH.LLT_NAME
<i>SDTM.MH.MHLLTCD</i>	MH.LLT_CODE
<i>SDTM.MH.MHENRTPT</i>	"ONGOING" when MH.MHONGO_C = "Y". "BEFORE" when MH.MHONGO_C = "N".

```
data %trim(_MH);  
  set MH(rename=( MHTERM = xMHTERM ) );  
  MHSPID = put(IGSEQ,best32.-1);  
  MHTERM = upcase(xMHTERM);  
  MHLLT = LLT_NAME;  
  MHLLTCD = input(LLT_CODE,??best.);  
  if (MHONGO_C in ("Y")) then MHENRTPT = "ONGOING";  
  if (MHONGO_C in ("N")) then MHENRTPT = "BEFORE";  
run;
```

AUTOCODE in Action: ADaM Programming

VARIABLE NAME	VLM	VARIABLE "SPECIFICATION" (Comment)
<i>ADaM.ADLB.PARCAT1</i>	PARAMCD IN ("ASTBILI", "ALTBILI", "ALTAST", "ALTASTBI")	Set to "CHEMISTRY".
<i>ADaM.ADLB.PARCAT1N</i>		Equal to informatted value of PARCAT1.
<i>ADaM.ADLB.EMDESC</i>		Equal to "P" when APHASE = "Screening". Equal to "A" when ADT > TSFENDT. Equal to "T" when ADT <= TSFENDT and APHASE contains "Treatment" or "Follow-up". *For derived parameters, equal to "P" when ABLFL= "Y", otherwise, equal to "T".

```
data ADLB_2postlitvar;  
  set ADLB;  
  if PARAMCD IN ("ASTBILI", "ALTBILI", "ALTAST", "ALTASTBI") then do;  
    PARCAT1 = "CHEMISTRY";  
  end;  
  PARCAT1N = input(PARCAT1, PARCATN_LB.);  
  if (APHASE in ("Screening")) then EMDESC = "P";  
  if (ADT > TSFENDT) then EMDESC = "A";  
  if (ADT<=TSFENDT) and (prxmatch("/Treatment|Follow-up/", APHASE)) then EMDESC = "T";  
run;
```

Parser & Patterns

AUTOCODE parses “plain-text” syntax into SAS code by identifying key words and/or phrases and flagging them for downstream processing.

PERL regular expressions are the most effective means for parsing and are incorporated into a set of “patterns”, which allow AUTOCODE to select the optimal means of parsing the syntax.

Each variable is evaluated and assigned an optimal “pattern” based on a set of rules, including:

1. What is the Origin metadata (e.g. Predecessor, Assigned-Investigator, Derived) for the variable?
2. Are value(s) of the created variable conditioned on the value of one or more variable(s)?
3. Are all relevant variables (see #2) existent in the source datasets (e.g. RAW or SDTM datasets, for SDTM and ADaM creation) or are they generated in the current dataset?
4. Is the created variable generated by a standard macro (based on variable naming structure)?

Pattern Assignment: Unconditional Assignment

VARIABLE NAME	VARIABLE “SPECIFICATION” (Programming Notes/Comment)
<i>SDTM.AE.DOMAIN</i>	Set to “AE”

DOMAIN has a simple structure, assigning a character value to all records in the dataset.

AUTOCODE uses a “**Literal_Post**” Pattern to identify this form of assignment:

```
(SET TO |EQUAL TO )?((null)|("([\x20-\x21\x23-\x7E]*)")|([\x30-\x39]+)|("&([\x30-\x39\x41-\x5A\x2D\x20]+\.)\?.?"))
```

This pattern can also identify the unconditional assignment of numeric, Null, or macro variable values to a variable, using the above PERL regular expression.

Parsing Syntax to Code: “Literal_Post”

```
(SET TO |EQUAL TO )?((null)|("([\x20-\x21\x23-\x7E]*)")|([\x30-\x39]+)|("&([\x30-\x39\x41-\x5A\x2D\x20]+)\.?.?"))
```

Using the above PERL regular expression, the specification can be parsed into SAS code; we highlight key code snippets below:

After pattern assignment, AUTOCODE can parse if the assigned value is Null, character, numeric, or a macro variable, then use the appropriate assignment when building the SAS code:

```
else if upcase(patternname) in ('LITERAL_POST') then do;
  if prxposn(p_expression,3,linetext) ne '' then constant_null='Y';
  else if prxposn(p_expression,5,linetext) ne '' then constant_char=prxposn(p_expression,5,linetext);
  else if prxposn(p_expression,6,linetext) ne '' then constant_num=prxposn(p_expression,6,linetext);
  else if prxposn(p_expression,8,linetext) ne '' then constant_macro=prxposn(p_expression,8,linetext);
end;
```

Pattern Assignment: Conditional Assignment

VARIABLE NAME	VARIABLE “SPECIFICATION” (Programming Notes/Comment)
<i>SDTM.CM.CMROUTE</i>	Set to "ORAL" when CMTRT = "IBUPROFEN". Set to "INTRAVENOUS" when CMTRT = "FENTANYL CITRATE".

CMROUTE conditions assigning a character value to records based on the CMTRT variable.

AUTOCODE uses a “**LiteralCondAND_Post**” Pattern to identify this form of assignment:

```
(SET TO |EQUAL TO )?((null)|("[\x20-\x21\x23-\x7E]+"))([\x30-\x39]+\.[\x30-\x39]+))( when | if )((( [\x30-\x39\x41-\x5A\x5F]+)(( is not Null| is Null| is numeric| is not numeric)|(( is | is not | = | is not equal to | contains | does not contain | < | <= | <= | > | >= | => )"([\x20-\x21\x23-\x7E]*)"[\x30-\x39]+\.[\x30-\x39]*[\x30-\x39\x41-\x5A\x5F]+))(( or |, )"([\x20-\x21\x23-\x7E]*)"[\x30-\x39]+\.[\x30-\x39]*[\x30-\x39\x41-\x5A\x5F]+))*)+(( and )"([\x30-\x39\x41-\x5A\x5F]+)(( is not Null| is Null| is numeric| is not numeric)|(( is | is not | = | is not equal to | contains | does not contain | < | <= | <= | > | >= | => )"([\x20-\x21\x23-\x7E]*)"[\x30-\x39]+\.[\x30-\x39]*[\x30-\x39\x41-\x5A\x5F]+))(( or |, )"([\x20-\x21\x23-\x7E]*)"[\x30-\x39]+\.[\x30-\x39]*[\x30-\x39\x41-\x5A\x5F]+))*)*).?(( Upper case before comparing\.)?)
```

This pattern can identify multiple conditions, based on variables derived during dataset creation, including exact values of another variable (as above), generalized conditions (e.g. “is Null”, “is not numeric”), and comparisons between one or more variables/constant values (e.g. “is not equal to”, “contains”, “>=”).

Parsing Syntax to Code: “LiteralCondAND_Post”

```
(SET TO |EQUAL TO )?(null)((["\x20-\x21\x23-\x7E"]+)|(["\x30-\x39"]+\.?["\x30-\x39"]+))(( when | if )(((["\x30-\x39\x41-\x5A\x5F"]+)(( is not Null| is Null| is numeric| is not numeric))((( is | is not | = | is not equal to | contains | does not contain | < | <= | > | >= | => )(["\x20-\x21\x23-\x7E"]+)|["\x30-\x39"]+\.?["\x30-\x39"]*["\x30-\x39\x41-\x5A\x5F"]+))(( or |, )(["\x20-\x21\x23-\x7E"]+)|["\x30-\x39"]+\.?["\x30-\x39"]*["\x30-\x39\x41-\x5A\x5F"]+)))+(( and )(["\x30-\x39\x41-\x5A\x5F"]+)(( is not Null| is Null| is numeric| is not numeric))((( is | is not | = | is not equal to | contains | does not contain | < | <= | > | >= | => )(["\x20-\x21\x23-\x7E"]+)|["\x30-\x39"]+\.?["\x30-\x39"]*["\x30-\x39\x41-\x5A\x5F"]+))(( or |, )(["\x20-\x21\x23-\x7E"]+)|["\x30-\x39"]+\.?["\x30-\x39"]*["\x30-\x39\x41-\x5A\x5F"]+)))*\.(?(( Upper case before comparing\.)?)
```

Using the above PERL regular expression, the specification can be parsed into SAS code; we highlight key code snippets below:

```
if prxposn(p_expression,3,linetext) ne '' then constant_null='Y';
else if prxposn(p_expression,5,linetext) ne '' then constant_char=prxposn(p_expression,5,linetext);
else if prxposn(p_expression,6,linetext) ne '' then constant_num=prxposn(p_expression,6,linetext);
condition=prxposn(p_expression,8,linetext);
if index(upcase(prxposn(p_expression,35,linetext)), 'UPPER CASE BEFORE COMPARING') then _condupcase='Y';
```

For the conditional statements, the syntax must be further parsed into SAS code, often using PRXMATCH:

```
's/(?i)(["\x41-\x5A\x5F"]+["\x30-\x39\x41-\x5A\x5F"]+)(( contains | does (not) contain )(["\x20-\x21\x23-\x7E"]*|["\x30-\x39"]+\.?["\x30-\x39"]*))(( or |, )(["\x20-\x21\x23-\x7E"]*|["\x30-\x39"]+\.?["\x30-\x39"]*)))/$4(prxmatch("\/$5/", $1))/';

's/(?i)(["\x41-\x5A\x5F"]+["\x30-\x39\x41-\x5A\x5F"]+)(( is (not) Null| is Null)(( or |, )(["\x20-\x21\x23-\x7E"]*|["\x30-\x39"]+)))/$3(missing($1) or $1 in ($4))/';
```

Outputting SAS Code

Following syntax parsing, each variable's *code statement* is compiled and added to an output SAS program:

Building LiteralPost Code:

```
if constant_char ne '' then pgm_statement = strip(variable) || ' = "' || strip(constant_char) || '";' ;  
else if constant_num ne '' then pgm_statement = strip(variable) || ' = ' || strip(constant_num) || ';' ;  
else if constant_macro ne '' then pgm_statement = strip(variable) || ' = "&' || strip(constant_macro) || '."' ;  
else if constant_null='Y' then pgm_statement = 'call missing(' || strip(variable) || ');' ;
```

Output Code:

```
DOMAIN = "AE";
```

Building LiteralCondAND_Post Code:

```
if constant_char ne '' then pgm_statement = "if " || strip(condition) || " then " || strip(variable) || ' = "' || strip(constant_char) || '";' ;  
else if constant_null='Y' then pgm_statement = "if " || strip(condition) || " then " || 'call missing(' || strip(variable) || ');' ;  
else pgm_statement = "if " || strip(condition) || " then " || strip(variable) || " = " || strip(constant_num) || ";" ;
```

Output Code:

```
if (CMTRT in ("IBUPROFEN")) then CMROUTE = "ORAL";  
if (CMTRT in ("FENTANYL CITRATE")) then CMROUTE = "INTRAVENOUS";
```

Integration of Code into Programs

To maintain independence of AUTOCODE-generated programming from manual programming, we generate up to two domain-specific SAS programs, collectively containing all “automated” variables:

- <**Domain**>-Pre programs contain all variables created from the source (RAW or SDTM) dataset(s).
- <**Domain**>-Post programs contain all variables without source dataset dependency (e.g. SDTM.XX.DOMAIN) or dependent on variables created for the current dataset (e.g. SDTM.CM.CMROUTE dependent on SDTM.CM.CMTRT)

Dataset creation programs can reference these AUTOCODE programs using %INCLUDE statements:

```
/* Pre process data */  
%include "..\autocode\ae-pre.sas";  
  
/* Post process data */  
%include "..\autocode\ae-post.sas";
```

Additional Programming Support

To facilitate the “specification to code” pipeline, AUTOCODE incorporates additional programmatic checks, including:

- Source data cross-checking (e.g. required variables present in source data)
- Metadata cross-checking
- Syntax pre-checking to ensure proper parsing (Error/Warnings, color-coding)
- Color-coding of RTF specifications to document automation

*We have found the color-coding particular helpful, as it provides immediate feedback at time of specification creation and facilitates review of the AUTOCODE outputs.

Color-coded Excel Specification {Pre-AUTOCODE}

ADAE	FUPFL	Equal to "Y" if the adverse event started after the end of the treatment emergent period (i.e. ASTDT > ADSL.TSFENDT), otherwise Null.
ADAE	EMDESC	Equal to "P" when PREFL = "Y". Equal to "A" when FUPFL = "Y". Equal to "T" when TRTEMFL = "Y".
ADAE	AEDSTRFL	Equal to "Y" when AEACNN = 4.
ADAE	AEINTFL	Equal to "Y" when AEACNN = 3.
ADAE	AESER	AE.AESER

Color-coded RTF Metadata {Post-AUTOCODE}

FUPFL	Derived: Equal to "Y" if the adverse event started after the end of the treatment emergent period (i.e. ASTDT > ADSL.TSFENDT), otherwise Null.
EMDESC	Derived: Equal to "P" when PREFL = "Y". Equal to "A" when FUPFL = "Y". Equal to "T" when TRTEMFL = "Y".
AEDSTRFL	Derived: Equal to "Y" when AEACNN = 4.
AEINTFL	Derived: Equal to "Y" when AEACNN = 3.

Conclusions

Since incorporating the AUTOCODE process into our SDTM programming workflows, Vertex has seen a greater than 50% reduction in programming time.

- Basic variable assignment(s) and/or derivation(s) handled by the process-generated programs
- More complex variables managed by standard/adhoc macros.

At time of presenting, there is growing incorporation of the new ADaM-level AUTOCODE process across our compounds; we anticipate equivalent gains.

Developing a “plain-text specification” to SAS code automation pipeline takes significant time-investment for development, implementation, and uptake.

- Improvements to programmer efficiency and enhanced traceability between metadata specifications and dataset creation programs have proven worthwhile

Acknowledgements

My colleague, Rohit Dhanjal, helped to conceive the SDTM-level AUTOCODE process at Vertex several years ago and was the primary developer of the underlying parsing process and its integration into our metadata and programming workflows.

Our colleague, Susheel Arkala, also served as an architect in the development of AUTOCODE and its rollout and maintenance at Vertex; both have been tremendous mentors to me, as I developed and rolled-out the ADaM-level AUTOCODE process at Vertex this past year.