Paper SM06

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

Matthew Finnemeyer, Vertex Pharmaceuticals, Inc., Boston, USA

ABSTRACT

Statistical programmers increasingly maintain repositories of standardized SDTM and ADaM mapping specifications, sometimes within a metadata repository (MDR). This practice streamlines the creation of study-level specifications and promotes harmonization across studies, enabling future integrated analyses such as ISS and ISE.

We enhance this approach through AUTOCODE, a process that parses human-language specifications into machine-readable syntax to automatically generate SDTM and ADaM dataset-creation programs; this reduces programmer time-on-task, strengthens traceability between specifications, metadata, and programs, and supports consistent implementation across studies.

In this presentation, we demonstrate the AUTOCODE workflow, highlight key programming techniques for embedding automation into existing processes, and share lessons learned from implementation. Attendees will gain practical strategies for adopting metadata-driven automation and insights into how this approach can reshape the efficiency and transparency of statistical programming.

INTRODUCTION

For statistical programmers, it is critical to ensure that the underlying metadata of SDTM and ADaM datasets are accurately represented in programming specifications; likewise, the underlying programming used to generate these datasets must be fully reflective of these metadata, based on the CDISC principle of **"traceability**."

Across the industry, there are a variety of approaches taken by statistical programming groups to ensure traceability in their programming and metadata/specifications, including: 1) Post-hoc development of metadata, following the production of stable dataset programming; 2) Manual generation of dataset programming, including using macros, following development of metadata/specifications; 3) Automated generation of dataset programming, driven by metadata/specifications.

Although prior publications (Cui *et al*, 2024) have outlined "automated generation" approaches, these largely depend on a bespoke suite of macros and "pseudocode" within the specifications to generate the dataset-creation programs; while effective, this approach requires further work to transform the "pseudocode" into acceptable DefineXML metadata and introduces the need for ongoing maintenance and/or modification of the underpinning macros in order to address novel programming challenges; adoption of "pseudocode" also introduces a novel learning curve to the programming pipeline, serving as an additional impact on concurrent timelines and raising the "training bar" for both current and future programmers. As an enhancement to this approach, there have been proposals to leverage "machine-readable" specifications, to generate *de novo* programs directly from specifications (Dsouza, 2024; Sushchenko *et al*, 2024) by parsing their text, but these also heavily rely on "pseudocode" to ensure sufficient conversion to SAS code.

At Vertex Pharmaceuticals, we combine these approaches (bespoke macros, machine-readable specifications, template programs), along with an enhanced "plain-text" parser, allowing for the SDTM/ADaM specifications to be directly used as metadata for the DefineXML, while simultaneously being interpreted and transformed into SAS-ready code that is directly referenced by our dataset-creation programs. In this paper, we will outline the necessary foundations for our "AUTOCODE" system, including the essential "plain-text" parser that further enhances the **traceability** of our programming.

AUTOCODE IN ACTION

To best demonstrate the efficacy of our approach, we provide exemplars of SDTM and ADaM specifications, along with the commensurate SAS code generated by our AUTOCODE system; no manual modifications are made to the code, following its creation, to best preserve **traceability**.

SDTM SPECIFICATION & CODE

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

Code Output*:*

```
data %trim(_MH);
  set MH(rename=( MHTERM = xMHTERM));
  MHSPID = put(IGSEQ,best32.-l);
  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;
```

ADAM SPECIFICATION & CODE

VARIABLE NAME	VLM	VARIABLE "SPECIFICATION" (Comment)
ADaM.ADLB.PARCAT1	PARAMCD IN ("ASTBILI", "ALTBILI", "ALTAST", "ALTASTBI")	Set to "CHEMISTRY".
ADaM.ADLB.PARCAT1N		Equal to informatted value of PARCAT1.
ADaM.ADLB.EMDESC		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".

Code Output*:*

```
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;
```

As can be seen above, the AUTOCODE process captures the essential processing elements from the "plain-text" specification, transforming them programmatically into functional SAS code; this includes key elements such as pre-specified formats, casing, whole or partial-matching to variable values, and VLM.

PARSING SPECIFICATIONS INTO CODE LOGIC

To preserve the "plain-text" structure of the specifications/metadata, the AUTOCODE process must be able to parse complete sentences into SAS code. Additionally, the parsing must be able to translate variable assignment/derivation across a range of complexity: from a simple global assignment of a value (e.g. DOMAIN, STUDYID), to conditional logic (e.g. EMDESC, TRTEMFL), to transformations using formats/codelists (e.g. SEXN, VISITNUM); in our experience, PERL regular expressions are the most effective means to convert specifications to the code, as they permit a wide array of user-specified text, provided it is encapsulated in key words and / or phrases.

To account for the different parsing methodologies that are required, we created an array of PERL regular expressions or "patterns" that allow the AUTOCODE process to identify the correct parsing technique(s) for each specified variable; the process selects the optimal pattern by evaluating the syntax of each variable using a set of rules:

1. What is the implicit 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 variables?
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 ***bespoke*** macro (based on variable naming structure)?

PARSING THE SYNTAX WITH PERL

As the underlying PERL regular expressions are rather complex, we will also "demonstrate" their operation narratively, using a series of examples from prior work:

1) *Unconditional assigned value*

VARIABLE NAME	VARIABLE "SPECIFICATION" (Programming Notes/Comment)
SDTM.AE.DOMAIN	Set to "AE"

For SDTM.AE.DOMAIN, the syntax is relatively straightforward to parse, with "Set to" and a quoted character string indicating an Assigned variable, with no conditional values based on other variables. Given the relative simplicity of the assignment, the DOMAIN variable does not have a **bespoke** macro for its creation.

This syntax would be attributed to the **Literal_Post** pattern by AUTOCODE, based on the following PERL regular expression:

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

2) *Conditional assigned value*

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

For SDTM.CM.CMROUTE, the syntax again includes "Set to" and a quoted character string (Assigned), but conditions the assignment on the value of SDTM.CM.CMTRT (the dataset is implied due to the presence of only the variable name). The CMROUTE variable does not have a **bespoke** macro for its creation.

This syntax would be attributed to the **LiteralCondAND_Post** pattern by AUTOCODE, based on the following PERL regular expression:

```
(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\.)?)
```

Note that the pattern allows for conditioned values based on exact values of another variable (as above), as well as more generalized conditions (e.g. "is Null", "is not numeric") and comparisons between one or more variables and or constant values (e.g. "is not equal to", "contains", ">=").

3) *Created by a **bespoke** macro*

VARIABLE NAME	VARIABLE "SPECIFICATION" (Programming Notes/Comment)
SDTM.DM.RFXSTDTC	Select the minimum EX.EXDTC for each subject.

For SDTM.DM.RFXSTDTC, the complexity of the derivation does not allow AUTOCODE to parse the syntax into SAS code. Instead, this is one of the variables created by a **bespoke** macro, which we identify based on the variable name.

TRANSFORMING SYNTAX INTO CODE

Once AUTOCODE has correctly identified the exclusive pattern for each variable, the *specification* syntax can then be transformed into SAS code. This is accomplished within a SAS program, where the *specification* syntax for each variable is read in as SAS record/variable, and PRX commands are used to transform the "plain-text" specification into the relevant code.

The programming logic used for each variable is conditional on the optimal pattern identified by the parser (see above), with pre-specified PRXPARSE/PRXCHANGE clauses used to shape the "plain-text" into function SAS code. For example, when the syntax dictates that a variable value is conditioned on the value of a variable (e.g. Null, non-Null, or one or more specific values), the following PRX *clause* is applied to the syntax, creating a "not missing(**<Variable>**), "missing(**<Variable>**)", or "<**Variable>** in (<value1>, <value2>, etc.)" code output, respectively:

PERL Regular expression:

```
'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))/';
```

VARIABLE NAME	VARIABLE "SPECIFICATION" (Programming Notes/Comment)
SDTM.LB.LBSTAT	Set to "NOT DONE" when LBORRES is null.
SDTM.LB.LBREASND	Set to "NOT COLLECTED" if LBSTAT = "NOT DONE".

Code output:

if (missing(LBORRES)) then LBSTAT = "NOT DONE";
if (LBSTAT in ("NOT DONE")) then LBREASND = "NOT COLLECTED";

Through our AUTOCODE process, we use similar PRX clauses to control casing of values, formatting, and the creation of multiple lines of conditional assignment or derivation of variables. Once all eligible variables have been parsed, the process continues with the generation of SAS programs that can be integrated into our dataset creation workflows.

INTEGRATION INTO DATASET PROGRAMS

To maintain the traceability of our AUTOCODE-generated SAS code, and to ensure its independence from manual programming, our process is to generate up to two SAS programs per dataset domain (e.g. ADAE or LB-SUPPLB); these datasets separate the coding between the variables generated from the source data (e.g. Raw or SDTM) and those dependent on variables created for the relevant domain dataset. We chose to name these programs **<DOMAIN>-PRE** and **<DOMAIN>-POST**, and reference them using %INCLUDE statements within the relevant dataset creation programs:

```
%include "./autocode/ae-pre.sas";
{interstitial code and macros}
%include "./autocode/ae-post.sas";
```

ADDITIONAL PROCESS SUPPORTS

Although this paper outlines the critical elements for a "plain-text" to code parsing system, it is recommended that additional programmatic supports are built into the process, such as metadata and source data cross-checking (e.g. ensuring the source and/or conditional variables are present in the source data), error or warning checks for incorrect or mis-specified syntax, and even specification-level utilities that facilitate the adoption and implementation of the necessary syntax for the parser; at Vertex, we incorporate a color-coding system to provide real time feedback on the ability of a variable to be autocoded, within our Excel specifications, and post-execution "issue capture" for user review, within the RTF specifications generated by 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 Excel Specification

FUPFL	Follow-up Flag	Char(1)	NY_Y	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	Description of Treatment Emergent Status	Char(10)	EMDESC	Derived: Equal to "P" when PREFL = "Y". Equal to "A" when FUPFL = "Y". Equal to "T" when TRTEMFL = "Y".
AEDSTRFL	AE Leading to Drug Withdrawn Flag	Char(1)	NY_Y	Derived: Equal to "Y" when AEACNN = 4.
AEINTFL	AE Leading to Drug Interrupt Flag	Char(1)	NY_Y	Derived: Equal to "Y" when AEACNN = 3.

Color-coded RTF Metadata

CONCLUSION

Since incorporating the AUTOCODE process into our SDTM programming workflows, we have seen a greater than 50% reduction in programming time, as basic variable assignment(s) and/or derivation(s) are handled by the process-generated programs and more complex variables are managed by the **bespoke** macros integrated into our dataset creation programs. At time of writing, AUTOCODE is used at the SDTM level by over 90% of our studies, and we are seeing the growing incorporation of the newly-developed ADaM-level AUTOCODE process across our compounds and anticipate equivalent gains.

Although developing a “plain-text” to SAS code parser takes a significant time-investment for development, implementation, and uptake, the improvements to programmer efficiency and enhanced **traceability** between metadata specifications and actual programmer are demonstrable. In the future, we believe the lessons learned and infrastructure we’ve developed will serve as a foundation for additional efficiency improvements, especially in leveraging AI-driven utilities to improve our specification-writing process.

REFERENCES

Cui, B.; Chen, M.; Wang, J. (2024). SM10: A Practical Approach to Automating SDTM Using a Metadata-Driven Method That Leverages CRF Specifications and SDTM Standards. *PHUSE US 2024.*

Dsouza, A. (2024). SM03: SDTM/ADaM Automation – Challenges and Issues from a Programmer’s Perspective While Exploring an Automation Attempt to Address Them. *PHUSE US 2024.*

Sushchenko, L.; Icihashi, R.; (2024). SI02: Automating ADaM Creation in Clinical Trials Part 2. *PHUSE US 2024.*

ACKNOWLEDGMENTS

Full credit goes to my colleague, Rohit Dhanjal, who helped to conceive the SDTM-level AUTCODE 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, these past two years.

CONTACT INFORMATION

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

Author Name: Matthew Finnemeyer
Company: Vertex Pharmaceuticals Inc.
Address: 50 Northern Ave, Boston MA.
Work Phone: --
Email: matthew_finnemeyer@vrtx.com