Paper AD09

"Beep Beep" A Metada code writer to generate SAS programs

Giuseppe Di Monaco, UCB BioSciences GmbH, Monheim, Germany

ABSTRACT

Datasets and TFL programs are written by many different programmers, each using their own different programming techniques and styles. Often these programs are not built using available metadata and standardized business rules, making it difficult to modify, share, and automate program creation, all at a significant cost in both time and money. To facilitate the sharing of standardized code, we have created the Navigator application. This is a metadata-driven code writer that generates standard, well-structured SAS programs that use validated SAS macros to create datasets and TFL. Specifically, Navigator allows the user to associate ADaM (or TFL) metadata specifications with their respective source data and metadata descriptions. These relationship definitions (mappings) provide the information required for Navigator to write SAS programs. Under certain qualifying conditions, mappings can also be copied across datasets and studies making the entire process of clinical data reporting much faster, therefore achieving substantial cost savings.

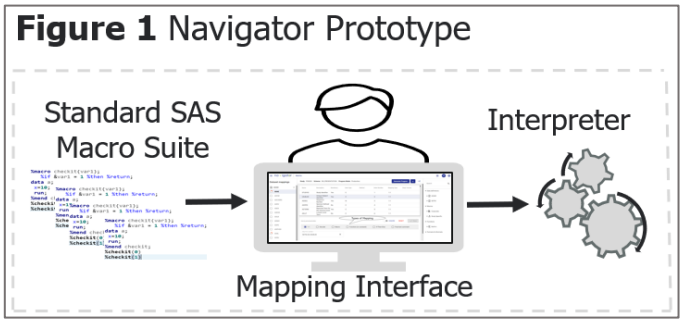
OBJECTIVES

The primary objective of this paper is to show how the application speed up the entire process of clinical data reporting and facilitates the sharing of standardized code. The secondary objective is to show how the application helps standardize the way specifications are built and developed.

INTRODUCTION

The Navigator application was derived from an earlier SAS macro suite, user interface, and interpreter that were initially developed by UCB several years ago. The application consists of three main components: a metadata-driven interface, also referred to as the mapping interface, a SAS macro suite, and an interpreter (see **Figure 1**).

Figure 1 Navigator Prototype



Scaling up the prototype into a corporate web-based application was done in collaboration with the UCB IT department, however, SAS parts remained the responsibility of the statistical programming group.

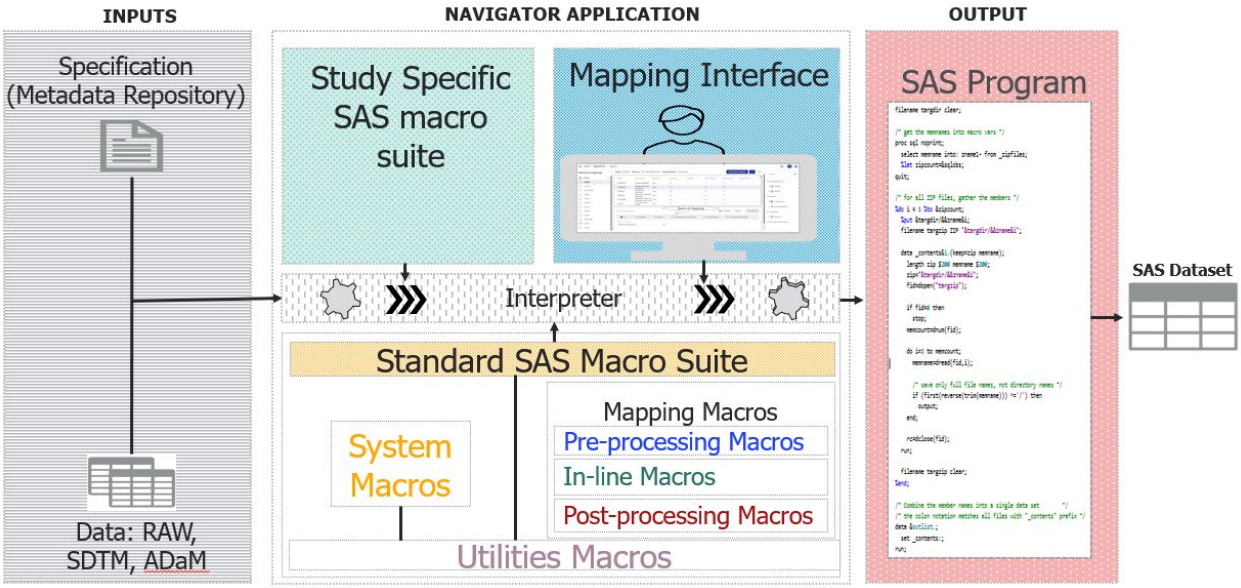
It is important to remember that Navigator is not intended to write programs that build complete ADaM (or SDTM) datasets. In fact, these datasets are built by a sequence of derivations, each derivation adds one or more variables or parameters to the dataset. Each derivation has an associated mapping and a macro or function call. This method makes it easy to adjust code by adding, removing, or modifying derivations.

Before diving into the architecture and design of the Navigator web-based application, it is anticipated that some variables cannot be mapped in Navigator. At the time this paper is being written Navigator web-based application can write SAS programs to generate datasets and it is being upgraded to accommodate Standard TFLs.

NAVIGATOR – WEB-BASED APPLICATION

The application can write SAS programs that can be used to generate SDTM data from raw data or ADaM data from SDTM data. Specifically, the interface allows the user to associate ADaM (or SDTM) metadata specifications with their respective source metadata descriptions. These relationship definitions (mappings) provide the information required for Navigator to write SAS programs to produce the required ADaM (or SDTM) dataset. Note that the output of Navigator is a SAS program, NOT a SAS dataset. The idea is that interpreter programs inside Navigator are used to build virtually complete programs in text files. Users may subsequently edit this generated code using traditional methods before final submission. This is a design feature, and it means that programmers are empowered to address things that are not currently covered by the available business rules available inside the tool. The programmer will eventually run the program and create the SAS dataset. Let's see in more detail the SAS output of the application (red), the mapping interface (light blue), the Study Specific SAS Macro Suite (teal) and the Standard SAS Macro suite (light yellow). The interpreter and Navigator Inputs will be partially described in this paper.

Figure 2 NAVIGATOR Web-Based Application



NAVIGATOR – OUTPUT

The output of Navigator is a well-structured SAS program that uses SAS macros. Three design principles were used to design the output program (and naturally SAS macros too).

Usability: macros and functions are easy to use, and each macro has a document containing all test cases done during development and validation.

Simplicity: all macros and functions have a clear purpose without trying to do numerous and very different derivations. Macros are not too general, and they do not try to fulfil multiple, complex purposes.

Readability: macros, functions, and output programs have a high degree of consistency and readability. Navigator output programs are all structured the same way. This solution facilitates code sharing in a controlled and standard way. It also ensures all programmers understand the style and techniques of how the "new standard code" is structured and built to facilitate modification.

Finally, since structure and readability are of utmost importance for Navigator (and any other aspect of standardization and automation), let's see an example of the output program. When you generate the

program in Navigator, a message will appear describing the success or failure of the build request. Here is an example of what output programs look like.

First Line Output code (NOTE: All programs have the exact same structure)

```
%* 1. Configure the study programming environment;
%include "...\sm_autoexec.sas";
```

```
%* 2. Add in the user macro library;
%sm_localmac(infolder=....);
```

```
%* 3. Allocate the study data libraries;
libname ADAM '...';
libname SDTM '...' access=ReadOnly;
```

```
%* 4. Configure the macro variables for ADSL dataset;
%let domain = ADSL;
```

```
%* 5. Merge on any available SUPP data;
%sm_addsupp(indomain = DM);
```

```
%* 6. Define KEEPVAR macro variable;
%let keepvars = ..................;
%* 7. Define PERMVAR macro variable;
%let permvars = ..................;
%* 8. Define ATTRIBS macro variable;
%let attribs = ..................;
%* 9. Define SORTSTRING macro variable;
%let sortstring = ....;
%let sortstringSql = %sysfunc(tranwrd
(%cmpres(&sortstring.), %str( ), %str(, )));
```

```
%* 10. Call code to build formats for joining on to base data;
%sm_fmt(inds=, idkey=, fmtname=, label=, where=);
```

```
%* 11. Build required codelist formats and informats;
Proc format;run;
```

```
%* 12. Code to build workspace dataset;
proc sql feedback noprint nowarnrecurs;
create table work.&domain. as select
  DM.STUDYID as STUDYID length=200,
   ..................
  %sm_arace(RACE=RACE,RACEOTH=RACEOTH) as ARACE length=200,
  . as ARACEN,
  %sm_ndate(DATEVAR=DM.RFXSTDTC) as TRTSDT,
  %sm_nTime(DATEVAR=DM.RFXSTDTC, hilo=) as TRTSTM,
from (select * from WORK.DM)
order &sortStringSql.;
quit;
```

```
%* 13.Finalize the dataset with sorting,keep/drop and labelling;
%sm_trimvars(inDs = work.&domain.);
%* 14 drop any not-mandatory variables Sort the data and sort;
%sm_dropmiss(inDs = work.&domain.,outds = ADAM.ADSL,keepvars =
&keepvars.,sortvars = &sortstring.,dslabel =
%str(Subject Level Analysis Dataset),attrib = &attribs.,
dropmiss = n,permvars = &permvars.);
```

Standard SAS Macro suite
System Macros

Standard SAS Macro suite
Pre-processing

SAS libraries

Output Dataset Name

Standard SAS Macro suite
Pre-processing

Setting macro variables which will be used at the end of the program

Standard SAS Macro suite
Utility Macros

Mapping Codelist

Standard SAS Macro suite
In-line Macros

Standard SAS Macro suite
Post-processing Macros

NAVIGATOR APPLICATION

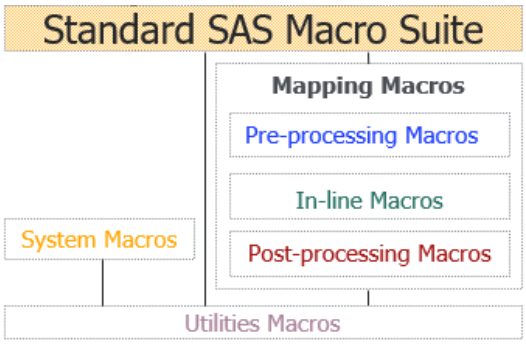
Study Specific SAS Macro Suite

Study-specific macros can be written by any user, and study programmers are strongly encouraged to do so. Study programmers are also encouraged to find mapping solutions (business rules) for target variables that cannot be addressed with simple mappings and currently have no associated macro in the standard SAS macro suite. It is envisaged that any solution written by any user can be made into a generic solution for sharing amongst other programmers through the Navigator Auto-map functionality. These macros can be anything the user wants them to be, but they must be compliant with the specific code-structure rules for macros that enable them to be loaded into Navigator. Once again, sharing code and code components is a key principle of the Navigator and we encourage all users to share best practices that enable the delivery of quality study data faster. Eventually (and where appropriate) these macros can also be promoted to standard macros. Study specific SAS macros are included at the step 2 of the output program (2. Add in the user macro library).

Standard SAS Macro Suite

There are different types of standard macros in the Standard Macro Suite:

- **System Macros**: Set up the environment or interact with metadata.
- **Utility Macros**: Embedded in system or mapping macros, or to perform other common tasks.
- **Mapping Macros**: Derive dataset variables.
    - **Pre-processing Macros**: Update or derive variables in the input dataset.
    - **In-line macros**: Derive variables directly in SQL or SAS data steps. These are used in the main dataset created from the "SET" dataset.
    - **Post-processing Macros**: Update or derive variables in the output dataset.



Let's see in more detail each type of mapping.

System Macros

An example is the autoexec which set up the entire environment. This macro is called at step 1 in the "First Line Output code". Due to space restrictions in this paper, the code cannot be shown.

Utilities Macros

For instance, the sm_fmt macro is used to join data coming from different source datasets. Following is an example of how this macro is called at step 10 in the "First Line Output code".

```
%sm_fmt(inds=SDTM.VS,idkey=USUBJID,fmtname=VS_VSSTRESN_HEIGHT_U,
label=VS.VSSTRESN, where=VSTESTCD='HEIGHT' and VSBLFL='Y');
```

This macro creates a format (VS_VSSTRESN_HEIGHT_U) which later can be used in a proc sql to derive the variable height.

```
proc sql feedback noprint nowarnrecurs;
  create table work.&domain. as select
  DM.USUBJID as USUBJID length=200,
  input(USUBJID, ?VS_VSSTRESN_HEIGHT_U.) as HEIGHT,
  from (select * from WORK.DM)
  order &sortStringSql.;
quit
```

Mapping Macros

Small and re-usable SAS macros (and functions) used to derive variables and to create the structure of the dataset. These standard macros can be considered as codes snippets and can be grouped into pre-processing, inline, and post-processing (e.g.: trimming, supplementary, baseline values, AE occurrence, date imputations, etc.…). Please refer to "First Line Output code"**.**

Navigator directly supports three kinds of Standard Study Mapping Macros:

- **Pre-processing Macros** (Called before step 12 in the "First Line Output code")
- **In-line Macros** (Called in step 12 in the "First Line Output code")
- **Post-processing Macros** (Called after step 12 in the "First Line Output code")

Note that Pre-processing Macros and Post-processing Macros are called automatically by the interpreter when it writes the output SAS program. In other words, there is no explicit action from the user that triggers the Pre-processing and Post-processing Macros to be written in the output SAS program. On the contrary, In-line Macros are the result of user interaction (mapping) with the application interface. Following is an example of an In-line Macro for the treatment-emergent flag, sm_trtemf. (Note that due to space limitations in this paper, the code for Pre-processing Macros and Post-processing Macros won't be shown).

In-line Macros

```
%macro sm_trtemf(EVSTDT= /* Numeric event start datetime variable */,
                 TRTSDT= /* Numeric datetime variable for first dose date */,
                 TRTEDT= /* Numeric datetime variable for last dose date */,
                 WINDOW=84 /* Window for days past last dose that count */);
  %* ********************************************;
  %* if missing TRTSDT then set no flag;
  ifc(missing(&TRTSDT.)," ",
    %* ****************************************;
    %* if missing EVSTDT set no flag;
    ifc(missing(&EVSTDT.)," ",
      %* ************************************************************;
      %* if EVSTDT on or equal to TRTSDT and missing TRTEDT set to Y;
      ifc(&EVSTDT. ge &TRTSDT. and missing(&TRTEDT.),"Y",
      %* ************************************************************;
      %* if EVSTDT is less than TRTEDT + window set to Y;
      ifc((&EVSTDT. ge &TRTSDT.) and ((&EVSTDT. le ((&TRTEDT. +
      dhms(&window.,0,0,0))))),"Y",
        %* ******************;
        %* else set to missing;
        " "),
      " "),
    " ")
  )
  %* ************************************************************;
  %* note deliberately no trailing semicolon so code works in sql;
%mend;
```

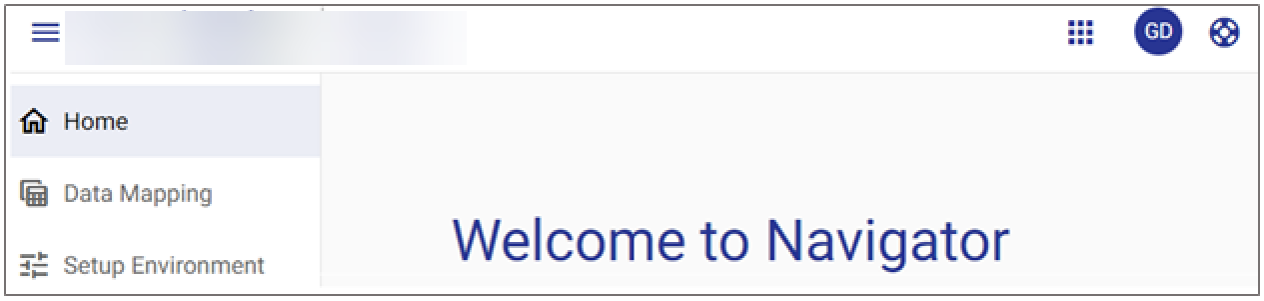
Interpreter

The Interpreter component (also called code builder) is one of the most important components of the project. Unfortunately, due to limited space for this paper, it cannot be described in detail. The reader of this paper should remember that the code builder follows a well-defined algorithm of how target data is achieved from the available source data collection and the methods to translate the mappings into executable code as text files.

Mapping Interface

When the user opens Navigator, the following is displayed.

Figure 3 NAVIGATOR Homepage



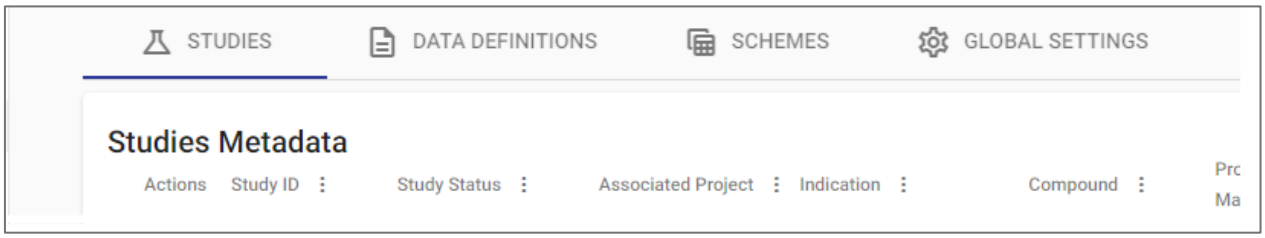
By clicking on the appropriate buttons on the left-hand side of this page you can open the Data Mapping tab and the Setup Environment Tab.

Setup Environment

The Setup Environment includes four tabs: Studies, Data Definitions, Schemes and Global Settings.

All apart from the Global Settings can be customized by the study team. Due to limited space in this paper, screenshots of the different tabs will not be shown. However, a brief description of each of them is given in tabs.

Figure 4 NAVIGATOR Setup Environment Tabs



Studies

This tab includes study specific information such as study macro location, study code list, statistical analysis plan and annotated CRF (Case Report Form).

Data Definitions

A Data definition is a means of allocating data resources to a study. These become the SAS Libraries in the program as well as the resources available in the mapping session. A resource could be a folder of SAS datasets, a CSV file containing data for the study (e.g., unblinding data), a single or multi-page Excel workbook (e.g. an XLSX spreadsheet containing pages of laboratory data), a specification for ADaM and a specification for SDTM.

Schemes

A scheme is the grouping of resources that are made available in the mapping window for a given target library (see the left pane of Figure 5). Ultimately, the scheme becomes a collection of mappings for building programs for a study. Each scheme has a type (either validation or production), a defined target data definition (either SDTM or ADAM), a source data definition a mapping window for you to build the mappings for the scheme, and finally an output destination. This is the location where any generated programs are written to.

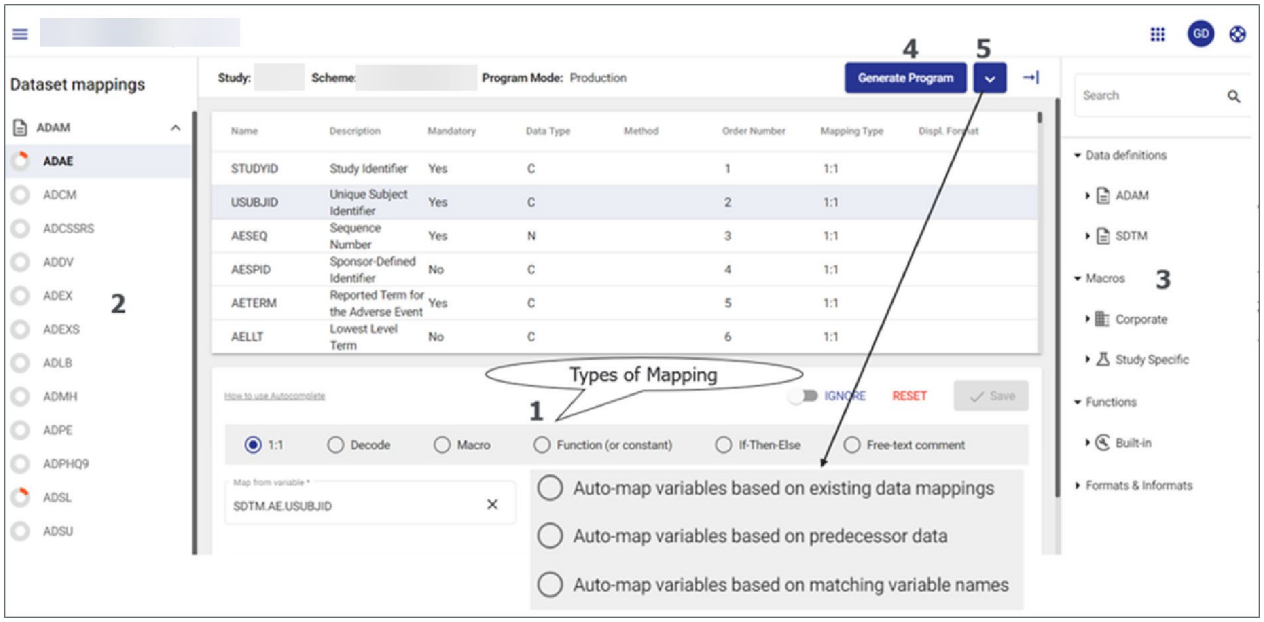
Global Settings

Since the Global settings hold information about corporate standards, they can only be changed by the standard team. This tab includes a version number for the standard SAS macro suite, their respective production and validation locations, and the corporate code list version and location.

Data Mapping

Navigator interface is designed so that it is intuitive, straightforward, and logical, and does not separate tasks into different windows and long paths of clicks. It is responsive; transactions are quick enough so that they do not interfere with the user as the tool is being used. It is also ergonomic; components that are required to perform the job are accessible from inside the tool. Finally, it can identify common metadata and copy relevant mappings between studies at the study-domain level.

Figure 5 NAVIGATOR Data Mapping



Given the limited space in this paper, the entire functionality of the Navigator interface cannot be described in detail. Therefore, let's highlight some of the interfaces' most important parts (see **Figure 5**):

- In the middle part there are radio buttons for selecting the type of mapping (**1**).
- On the left panel (**2**) there are all the target datasets, in the central panel is the dataset (and variable metadata) that is currently being mapped.
- On the right panel (**3**) there are all source datasets, standard macros, study specific macros/functions, formats and informats.
- On the top part (**4**) in blue is the button to generate the output program
- Besides the button to generate (**5**) there is a down arrow for selecting the Auto-map type and other information such as paths to study folders and documentation.

Type of Mapping

For Navigator to produce the output it needs to perform the mapping. The user can choose which type of mapping to assign to each variable. There are different types of mapping in Navigator, as **Figure 5** shows.

NOTE that as outlined earlier, the scope of Navigator is to make these mappings sharable so that each subsequent study takes less time to map than the last.

Auto-map

One of the most powerful functionalities of Navigator is "Auto-map". Through this functionality, Navigator ensures programmers make use of what went before, without "reinventing the wheel" and can reuse a mapping, which means a large part of the output SAS program is automatically generated! Navigator will not overwrite an existing mapping with an auto-map, and users are free to modify or remove any auto-mapped variable mappings as they wish.

There are three different types of auto-mapping:

- **Auto-map based on existing data mapping**
  Navigator will use mappings from another study to map variables on a given dataset where the metadata between the studies is complete. Different studies and different schemes can be used. For each unmapped target variable, the process checks the metadata between the studies (dataset names, variable names, variable types, code names, macro names etc…) and if there is a match, the mapping for that target variable is copied across. If there is no match, no copying is done.
- **From Predecessor Variables**
  Navigator uses information from the specification document to map directly from variables defined as a predecessor. At the time this paper is being written; this only works for 1:1 mapping.
- **Based on matching variable names**
  Navigator uses information from the specification document to map directly from variables that have the same name as those on the SET dataset**.** This again only applies to 1:1 mapping.

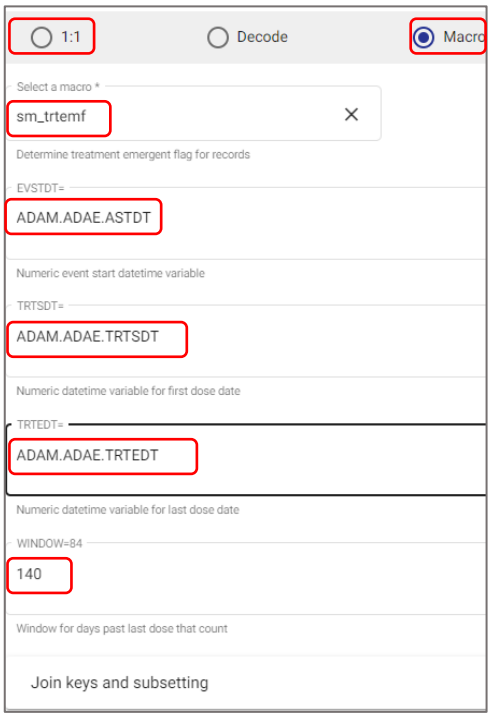
Mapping - Macro Mapping Example

Macro mapping uses the Study Specific SAS Macro suite and the Standard SAS Macro Suite described earlier in this paper. This is probably the most powerful and versatile type of mapping that can be performed in Navigator, and it readily demonstrates how business rules can be encapsulated into code snippets. Some of these code snippets are very simple, others can be extremely complicated. Breaking the methods that are available inside macro mapping into a sub-collection of validated code snippet macros can facilitate quicker code building of more complex macros.

The variable TRTEMFL (Treatment Emergent Analysis Flag) of the ADAE dataset is used as an example. This is mapped by using macro mapping, see **Figure 6.** The figure shows that the user has selected a macro called **sm_trtemf** (also described earlier in the In-line macro section of this paper) and entered the following macro parameters:
EVSTDT=ADAM.ADAE.**ADSTDT**, TRTSDT=ADAM.ADAE.**TRTSDT**,
TRTEDT=ADAM.ADAE.**TRTEDT**,
WINDOW= **140**

Figure 6 NAVIGATOR Macro Mapping

(In-Line Mapping)



Note that macro parameter descriptions that appear in the window (**Figure 6**) are soft coded from the contents of the macro program. See Figure 6 and the **sm_trtemf** macro code in the "In-line Macros" section.

The Interpreter uses the values entered in the above interface form and translates them into a macro call (sm_trtemf) when writing the output SAS program file. The interpreter intelligently recognizes that **sm_trtemf** is an in-line macro and should be embedded into the PROC SQL statement that constructs the workspace dataset ADAE. See **Figure 7**.

Figure 7 NAVIGATOR Macro Mapping (sm_trtemf)

```
%* 12. Code to build workspace dataset;
proc sql feedback noprint nowarnrecurs;
create table work.&domain. as select
  ……,
  input(USUBJID,?ADAE_ASTDT.) as ASTDT,
  input(USUBJID, ?ADSL_TRTSDT_TRTSDT.) as TRTSDT,
  input(USUBJID, ?ADSL_TRTEDT_TRTEDT.) as TRTEDT,
  %sm_trtemf(EVSTDT=calculated ASTDT,
      TRTSDT=calculated TRTSDT,
      TRTEDT=calculated TRTEDT,
      WINDOW=140) as TRTEMFL length=200,
  from (select * from WORK.AE)
  order &sortStringSql.;
quit;
```

IMPROVEMENTS, INTEGRATION WITH OTHER TOOLS AND LANGUAGE AGNOSTIC

Improvements

Navigator web-based application can write SAS programs to generate datasets and we are planning to release the first version of Standard TFLs component by end of this year.

Integration with other tools.

Navigator can easily be integrated with many different technologies within any statistical computing environment. Following are some of the initiatives that the Navigator team are working on:

- Integration with the SAS environment embedded directly inside Navigator is being evaluated by UCB IT department. This solution would increase usability of the application and improve the user experience and satisfaction. Instead of producing the SAS output file, the application would be able to open the file in SAS automatically, enabling the user to debug their program without switching environments and applications.

- Connect Navigator application directly to the Company Metadata Repository Database. This would increase the speed and make the entire process of clinical programming much more secure and controlled.

Language Agnostic

The team is working on finding a solution for Navigator to write output files in multiple statistical programming languages. In particular, the team is very interested in making Navigator able to write the output program in the R language. This would allow us to utilize existing open-source modular toolboxes, like the admiral [1] package, and actively participate in initiatives such as pharmaverse[2].

[1] https://pharmaverse.github.io/admiral/cran-release/index.html
[2] https://pharmaverse.org/

CONCLUSION

Dataset programs are commonly written by many different programmers who use diverse techniques and styles. These programs are frequently developed without utilizing available metadata, and standardized business rules are rarely established. This approach makes it challenging to share code between related studies and prevents automation, leading to significant time and financial costs. Finding a solution just for the dataset programs wouldn't have had a big positive impact, there at UCB we approached these issues from a holistic view. Our functionality has been created using a set of small, standardized SAS macros. Each macro is designed to perform a limited number of tasks and can be accessed via a user-friendly and straightforward interface. This approach offers several advantages, such as deployability, reliability, scalability, modifiability, and agility. We also took the opportunity to improve and standardize other processes and tools that are used throughout the process. An example is the dataset's specifications, where we are identifying business rules and developing best-practice methods for deriving variables.

What next? We are working on developing a Navigator component for standard Tables, Listings and Figures. After that, we can also focus on Adverse Event Narratives, Integrated Safety Studies…all from the "new standard" concept.

We strongly believe that adopting this solution presents a massive opportunity to reduce costs, enhance efficiency, and share the most effective mapping techniques.

ACKNOWLEDGMENTS

First, a very big thank goes to Jim Elder, the person who ideated Navigator and provided continuous support for its development.
A special thanks goes also to Nate Bennet, Alberto Montironi, Trevor Clulow, Dean Woodhouse, Pierre Dostie, Jonas Maselis, and all the I.T. team. Without their continued efforts and support we would not have been able to bring this automated process to successful completion. A big thank goes also to all colleagues at UCB Biosciences for their support and comments.

CONTACT INFORMATION

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

Giuseppe Di Monaco
UCB BIOSCIENCES GmbH
Alfred-Nobel-Straße 10
40789 Monheim, Germany
Tel: +49.2173.48.2091
Fax: +49.2173.48.1947
Email: Giuseppe.DiMonaco@ucb.com
Web: www.ucb.com