Paper 101X-C3F2A4C3H5

Multi-study programming – a best practice outline

Simon Purkess, PHASTAR, London, UK
Matt Metherell, PHASTAR, London, UK

INTRODUCTION

When multiple studies are nearly identical and run in parallel, it can save a lot of effort if we can simply update one set of code that is used by all relevant studies. This can bring its own set of problems to overcome, so in this paper we lay out a number of principles we recommend adhering to.

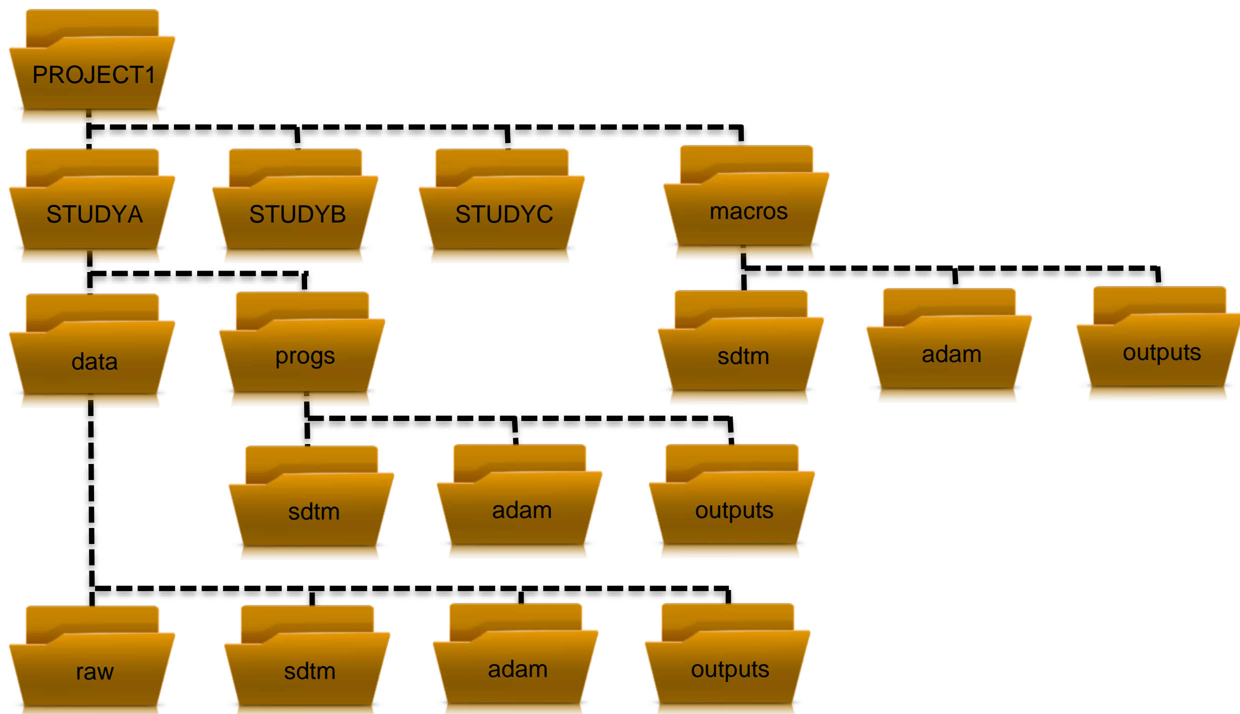
The basic idea is that each study contains shell programs that contain little more than a call to a central macro that contains the main body of code, and even study-specific code will be contained in this macro.

PRACTICALITIES

This section will guide the reader through the process of creating an area for the central macro programs, and separate areas for each study, from where the shell programs can be run, calling these central macros.

FOLDER STRUCTURE

This shows the folder structure we are using for the examples in this paper, where three studies (known as STUDYA, STUDYB and STUDYC) are part of one umbrella project (called PROJECT1). Alternative folder structures can work equally well but consistency between studies is obviously key. For the sake of our examples we are ignoring QC work, but these would most likely follow the same pattern.



We will see later on that we interrogate this folder structure to determine which study a program belongs to, by seeing which directory it is saved in, and creating macro variables accordingly.

MACRO VARIABLES

If not already standard practice, this is the perfect time to set up libraries and certain standard macro variables in an autoexec file or similar, including the STUDYID and certain folder paths, which will enable the same code to work across multiple study areas without redeclaring them each time.

This also allows for the user to use %if statements based on STUDYID for the cases where there are small differences between the studies.

In the following example, %setup_program shows how to assign several things:

- Useful macro variables such as the study, to allow the central macros to control the program flow based on the study.
- The libraries and macro folders that the programs can draw upon.
- Any global options you wish to declare.
- Applying format catalogues.

You could set these up explicitly in each area, or dynamically in a central area, but either way this will allow the central code to work across all areas. Of course, we're only focussing on the bits that affect multi-study programming, whereas many other things can be included.
Here, we first interrogate the filepath to extract the name of the study and project and assign them to macro variables for study-specific programming. Then we use these macro variables to set the location of our store of central macros, before finally declaring our libraries.

```
%macro setup_program;

%*Define macro variables based on directory structure;
%global project study;
%let rc = %sysfunc(filename(fr,.));
%let project = %scan(%sysfunc(pathname(&fr)),3,\);
%let study = %scan(%sysfunc(pathname(&fr)),4,\);
%let rc = %sysfunc(filename(fr));

%*Macro library;
filename sdtmmac "\\&project.\macros\sdtm";
options mautosource
        sasautos  = (sdtmmac sasautos);

%*Study libraries;
libname raw      "\\&project.\&study.\data\raw" access=readonly;
libname sdtm     "\\&project.\&study.\data\sdtm";
libname adam     "\\&project.\&study.\data\adam";
libname output   "\\&project.\&study.\data\outputs";

%mend setup_program;
```

UTILITY MACROS

Unfortunately, if we need to make an update for one study, we will often need to re-run the code across all of them, so it is important to make this as easy as possible (and to remember that it is still much easier than having to update the code in all areas as well as run it in all areas). Therefore a macro that can run a program across all studies is really useful, especially if you can configure it to:

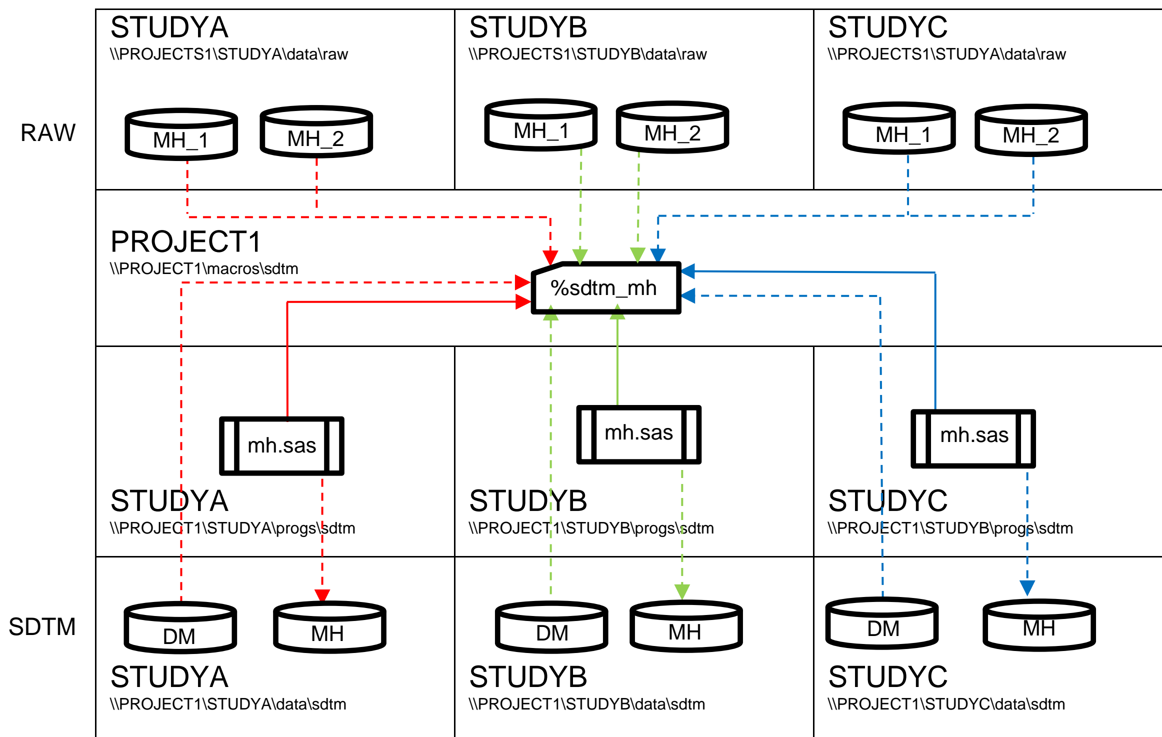
- Run both prod and qc when required
- Check both logs
- Check the compare
- Run all programs in a folder (eg all SDTMs)

How to create such a program could be a paper in itself, and will vary by system setup, so we don't offer examples, but if you currently have a way to run all programs in a given study, this can no doubt be adapted to incorporate multiple studies. Similarly, any tools you have for checking logs and compares for a study should be extended to work over multiple studies for ease of oversight.

DATA SET PROGRAMMING

In the following diagram we show how all the programs fit together for a simple example of the Medical History (MH) data set. We have three columns representing the three studies, then the top row represents the raw data library, with each study having its own data sets, and similarly the bottom row represents the SDTM library. The second row shows the central macro at the project-level (\\PROJECT1\macros\sdtm) which is why it spans all three columns. The third row has a shell program in each study that does little more than call the central macro with specific macro variables and libraries set up to point it in the right direction.

The solid arrows show the shell program calling the central macro, while the dotted arrows show the flow of data (raw data sets and DM going in, the SDTM data set MH going out).



INSIDE THE PROGRAMS

In the following diagram we look at the code inside these programs. Again we have three columns to represent the three studies, and the top row shows the contents of the three shell programs. The %setup_program macro is the one we talked about earlier to create macro variables, libraries and suchlike. At the end is a %end_program macro to do similar admin things such as check your log and compare data set attribute against CDISC standards , and in the middle we call our central macro, which occupies the second row, spanning all three columns.

STUDYA \\PROJECT1\STUDYA\progs\sdtm	STUDYB \\PROJECT1\STUDYA\progs\sdtm	STUDYC \\PROJECT1\STUDYA\progs\sdtm
`%*Set-up program;` `%setup_program;`	`%*Set-up program;` `%setup_program;`	`%*Set-up program;` `%setup_program;`
`%*Run central macro;` `%sdtm_xx();`	`%*Run central macro;` `%sdtm_xx();`	`%*Run central macro;` `%sdtm_xx();`
`%*Close program;` `%end_program;`	`%*Close program;` `%end_program;`	`%*Close program;` `%end_program;`

PROJECT1
\\PROJECT1\macros\sdtm

```
%*Macro to create XX data set;
%macro sdtm_xx() / minoperator;

%*Read in raw data set and derive variables;
data xx;
   length domain $8 usubjid $20;

   %if &study.=STUDYA %then %do;
      set raw.xx_1;
   %end;
   %else %if &study. in STUDYB STUDYC %then %do;
      set raw.xx_2;
   %end;

   domain    = "XX";
   usubjid   = catx("-", studyid, subjid);
run;

%mend sdtm_xx;
```

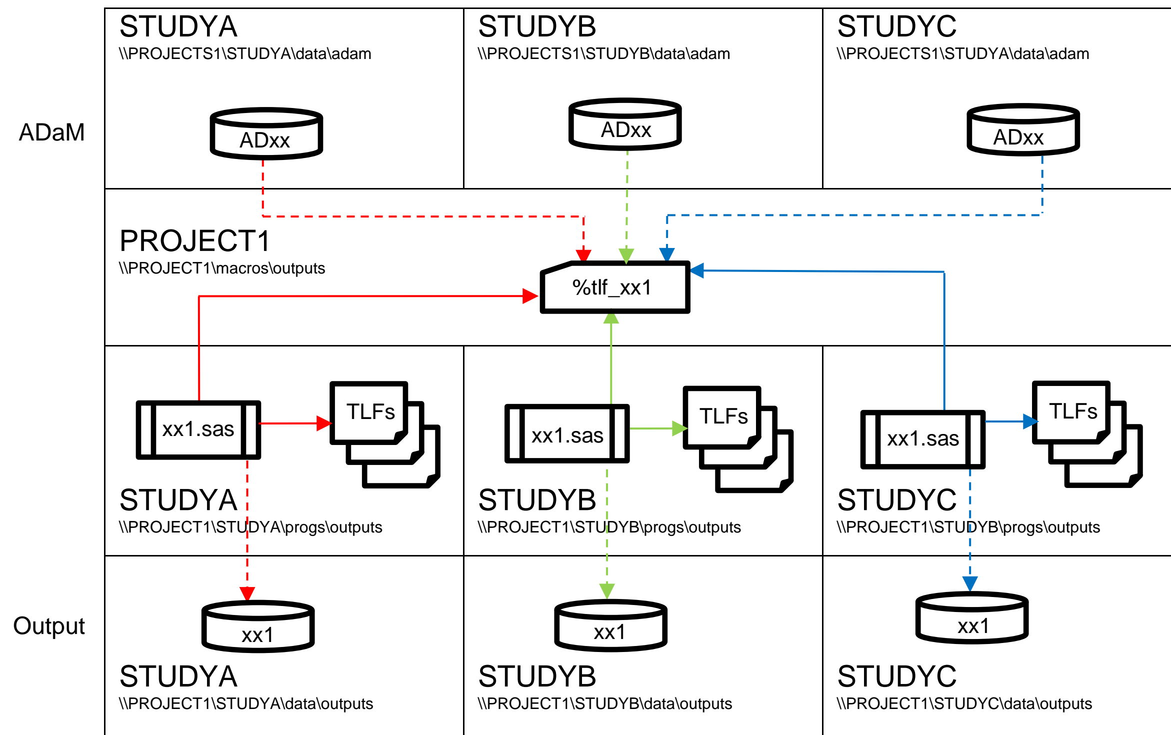
In the central macro we can use our pre-defined macro variables whenever our code varies by study. In the example above, we use &study in a %if block to ensure that STUDYA uses a different raw data set to the one used by STUDYB and STUDYC.

TFL PROGRAMMING

TLF programming follows a very similar pattern, although it's not uncommon to use the same program to create multiple tables or listings if they have the same structure. This doesn't have to be a concern as we can easily continue to do this. We can control the flow of our programs by using the macro parameters we parse in just the same way as we can use the &study macro variable.

In the following diagram we again have three columns representing the three studies, then the top row represents the ADaM data library, with each study having its own data sets, and similarly the bottom row represents the output library. The second row shows the central macro at the project-level (\\PROJECT1\macros\outputs) which is why it spans all three columns. The third row has a shell program in each study that calls the central macro, perhaps multiple times, to create outputs that are stored in the same folder as the program itself.

The solid arrows show the shell program calling the central macro, while the dotted arrows show the flow of data (ADaM data sets going in, TLF data set and output going out).



Perhaps in output programming even moreso than in SDTM or ADaM programming, there is always the chance that our needs will differ between studies. Certain outputs may only be required for certain studies, so we may find that we don't need to follow the central-macro system for every output. For all of its benefits, it can be more difficult to develop and troubleshoot than standard programming, so it may be worth using traditional programming for these. They can always be moved into a central macro later if it turns out that we do need them on multiple studies. On that note, it may often be worth programming the first version outside of a macro, and turned into a macro once it is stable.

INSIDE THE TLF PROGRAMS

In the following diagram we look at the code inside the TLF programs. Again we have three columns to represent the three studies, and the top row shows the contents of the three shell programs. Aside from the %setup_program and %end_program macros, we have multiple calls to our central macro, which occupies the second row, spanning all three columns.

STUDYA \\PROJECT1\STUDYA\progs\outputs	STUDYB \\PROJECT1\STUDYA\progs\outputs	STUDYC \\PROJECT1\STUDYA\progs\outputs
`%*Set-up program;` `%setup_program;`	`%*Set-up program;` `%setup_program;`	`%*Set-up program;` `%setup_program;`

`%*Run central macro;`
`%tlf_xx(num=1, pop=saffl);`
`%tlf_xx(num=2, pop=fasfl);`
	`%*Run central macro;`
	`%tlf_xx(num=1, pop=saffl);`
	`%tlf_xx(num=2, pop=fasfl);`
	`%tlf_xx(num=3, pop=ppsfl);`
		`%*Run central macro;`
		`%tlf_xx(num=1, pop=saffl);`
		`%tlf_xx(num=2, pop=fasfl);`

`%*Close program;` `%end_program;`	`%*Close program;` `%end_program;`	`%*Close program;` `%end_program;`

PROJECT1
\\PROJECT1\macros\outputs

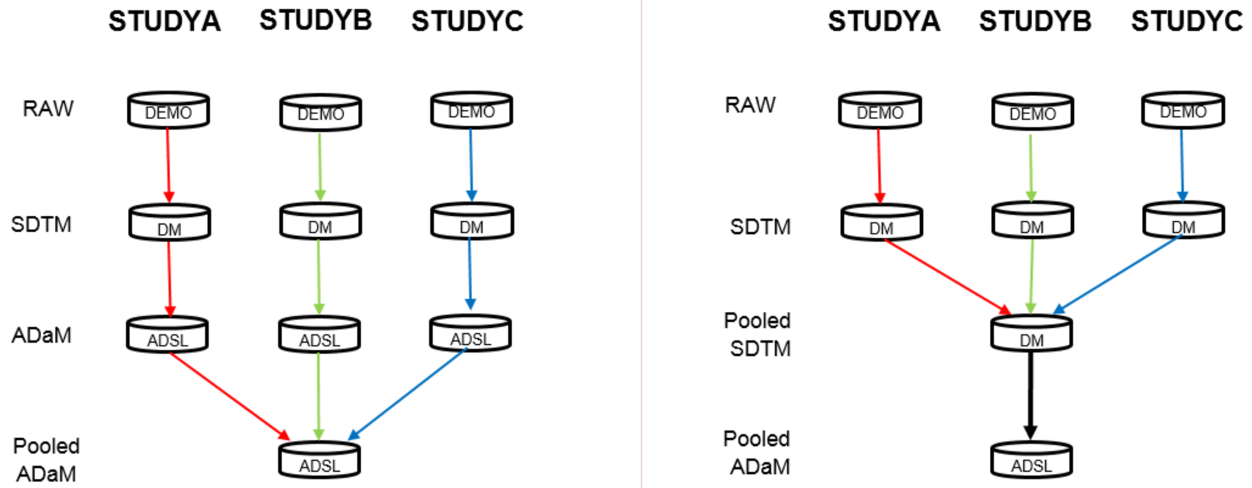
```
%*Macro to create XX output;
%macro tlf_xx(num=, pop=) / minoperator;

%*Create numbered data set for each output;
data output.xx_&num.;

   %if &study. in STUDYA STUDYB  %then %do;
      set adam.adxx;
   %end;
   %else %if &study. = STUDYC %then %do;
      set adam.adyy;
   %end;

   where &pop.= "Y";
run;

%mend tlf_xx;
```

As we saw with the SDTM example, we can use our pre-defined macro variable &study whenever our code varies by study, but we can also use the macro parameters to change the code between calls within any given study.
In the example above, we use &study in a %if block to ensure that STUDYA and STUDYB use a different ADaM data set to the one used by STUDYC, but we filter that data set using the value of &pop from the macro call, and save it as a data set.whose name varies depending on the value of &num, which was also parsed in the macro call.

POOLING STUDIES

When multiple similar studies are run, there is often a desire for a combined analysis such as an ISS/ISE (Integrated Summary of Safety/Integrated Summary of Efficacy) to take advantage of the larger combined study populations. Performing this analysis requires data from individual study data sets to be combined to created pooled project data sets. The more the individual study data sets differ in format, the harder the pooling process becomes, so if possible it's best to design the individual study data sets with pooled analysis in mind. While each study will have some unique features, it's helpful to standardize your programming approach as much as possible, e.g. using the same variables and parameters as much as possible.

The final product of the pooling process is generally a set of pooled ADaM data sets, which can then be used for TLF programming. There are two routes to this:



POOLING AT ADAM LEVEL

In this approach, individual ADaM data sets are created for each study, and then these are combined into the pooled ADaM data sets. The pooling step may be as simple as setting the individual data sets on top of each other, but there may also be a need to create additional variables/parameters if the pooled analysis is investigating different endpoints compared to the individual studies.

POOLING AT SDTM LEVEL

The alternative approach is to pool at SDTM level, and then to use these to create a single set of pooled ADaM data sets. This approach may be preferable if individual ADaM data sets are not required for the project as only one program will be needed per ADaM data set in the project, though these programs may end up being more complex than those for individual studies.

MANAGING THE PROJECT

Multi-study programming can provide significant efficiency gains if implemented correctly, but it is important to keep a good handle on the process to achieve this. Here are some of our tips to help manage the project effectively:

ASSESS SUITABLILTY

The ideal candidates for this programming model would be a set of near-identical studies running in parallel timescales. Unfortunately the reality is often messier, with differing study designs and timescales within a project. These issues aren't insurmountable, but the greater the differences the smaller the gains in efficiency, so if studies have too many differences it may be worth considering a hybrid approach where only parts of studies are programmed with project-level macros.

It is common for the scope of a project to change over its lifetime, for example a new study may be commissioned or an existing one may be altered as it progresses. An existing codebase can provide a useful starting point to start programming a new study within a project, but if the design is significantly different to existing studies then changes may need to be made to the existing macros, which can add new complexity to the programming.

KEEPING TRACK

As much as possible, maintain combined project-level documents rather than multiple single-study versions to allow you to track both project-level elements (e.g. macros) and study-level elements (e.g. study data sets) together. If one

macro is being used across several studies, any update to the macro will affect all studies so it is useful to track progress on all studies side-by-side.

			STUDYA			STUDYB			STUDYC		
Program Name	Production Programmer	QC Programmer	Completion Date	Comments	Status	Completion Date	Comments	Status	Completion Date	Comments	Status
DM	MM	SP	2021-07-01		Completed	2021-07-01		Completed		Data not yet available	Not Started
SE	MM	SP	2021-07-04		Completed	2021-07-05		Completed		Data not yet available	Not Started
DS	MM	SP	2021-07-06		Completed		DSDECOD not matching	Started		Data not yet available	Not Started
EX	MM	SP			Not Started			Not Started		Data not yet available	Not Started

KEEPING RECORDS

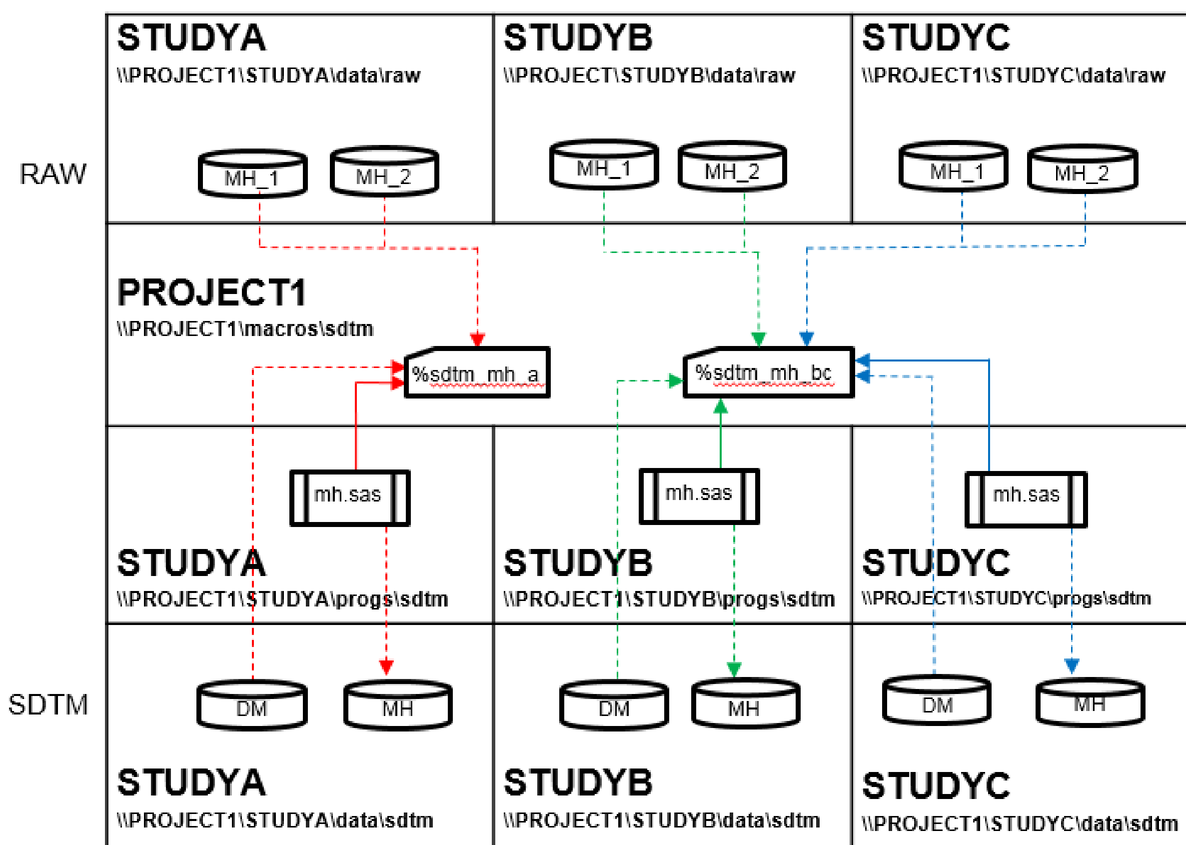
Often different studies within a project will progress at different rates, with some studies delivering while others are dormant, meaning that project macros are liable to change between deliveries of a study. To ensure an accurate programming record, it's crucial at the time of each study delivery to archive not just study-level programs but also any project macros; this means you can retain a snapshot of the project as it was on the day of the study delivery, for audit purposes and to refer back to if project updates for an active study affect a dormant one.

RESOURCING

Bear in mind that while time savings should occur over the project lifetime, the initial set-up required means that the early stages of the first study will take a little longer, as project macros are written and project-level planning is done. You may therefore need to commit extra resource at the start of the project, then it can be significantly reduced once the project-level codebase is in place. For this reason, the budget should be viewed at a project level rather than individual studies, as work done on earlier studies will benefit those that come later.

SPLITTING OFF

If studies diverge over time, central macros can become harder to maintain efficiently. Use of %if blocks for study-specific code may help, but there may still reach a point where it makes sense to split project macros into study-specific versions. This could be either at an individual program level, or for whole studies.



In the example above, the %sdtm_mh macro is being split due to a divergence in STUDYA. One copy of the macro is saved as %sdtm_mh_a with code added to reflect the changes for STUDYA, while another copy is saved as %sdtm_mh_bc for STUDYB and STUDYC to continue using. The mh.sas programs for each study need to be edited to call the correct macro, but there is no need to split other SDTM macros.

Splitting macros off in this way can avoid the need to write complex macro logic which can make future updates harder to execute. However it means that the project-wide efficiencies are no longer available for split sections, and any future project-level updates will need to be made individually for each study.

CONCLUSION

By following the guidelines of this paper, the reader can reduce the time needed for programming, running and troubleshooting, while also being more confident of having incorporated all requested updates. Given the overall trend towards increasing standardization and automation, the time has never been better to explore the possibilities afforded by multi-study programming.

CONTACT INFORMATION

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

Simon Purkess / Matt Metherell
PHASTAR
2D Bollo Lane
London W4 5LE
Email: simon.purkess@phastar.com / matthew.metherell@phastar.com
Web: www.phastar.com

Brand and product names are trademarks of their respective companies.