phuse·connect
connect·share·advance
10th–13th November
RAI Amsterdam
The Netherlands
EU 2019
The Clinical Data Science Conference

Paper SI10

# ADSL – Friend or Foe ?

Jørgen Mangor Iversen, LEO Pharma A/S, Ballerup, Denmark

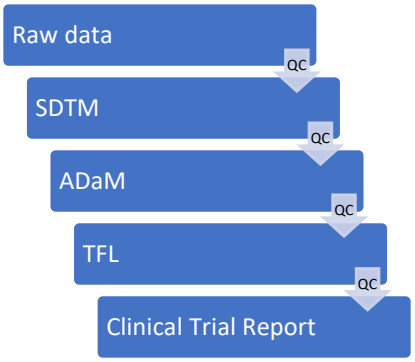
## ABSTRACT

When creating a set of ADaMs for a trial or a submission, one commonly used approach is to create the ADSL dataset first, and then pick variables from ADSL when creating other ADaM datasets. While this approach may seem clever at first, only creating each variable once, it may create other problems downstream, in particular when performing QC of ADaM. To avoid costly and lengthy re-QC, it is advocated to create all variables common to more than one ADaM dataset outside of the set of ADaMs, either as WORK datasets or separate permanent datasets. This approach eliminates dependencies between individual datasets, especially dependencies of ADSL, avoiding the need to re-QC all ADaM dataset when a variable in ADSL, but not in other ADaMs, needs to be updated.

## INTRODUCTION

At LEO, we produce the Clinical Trial Report following a very familiar model, transforming our clinical data from collected raw data via SDTM and AdaM to Tables, Figures, and Listings (TFL) to be made part of the clinical trial report. After each step we perform QC and validation, sometimes quite extensively:

- Raw data are validated according to a Data Validation Plan associated to the eCRF
- SDTM datasets are validated through Pinnacle 21 Enterprise
- ADaM datasets are validated through Pinnacle 21 Enterprise, as well as parallel programming of the most critical parts
- Tables, Figures, and Listing are subject to parallel programming
- Collection of TFL's into the Clinical Trial Report is subject of reviews




*Data flow process*

We aim to always lift the SDTMs to a submission ready state, laying much emphasis on the validation and QC, and maximizing the transparency downstream.

When producing ADaMs, we define two classes of variables across ADaM datasets:

- Common variables, identical across all datasets
- Other variables in one or more datasets.

As an example of this classification we have the following:

| Class | Dataset | Variable | Label | Comment |
|---|---|---|---|---|
| Common | Common variables | USUBJID | Unique Subject Identifier | Needs the same length and label across ADaMs |
| Other | ADSL | DTHDT | Date of Death | May be updated during the trial |
| Other | Occurrence data model (ADAE) | TRTEMFL | Treatment Emergent Analysis Flag | May be updated by Medical Expert |
| Other | Basic Data Structure (ADLB) | CRIT1 | Analysis Criterion 1 | May be updated/created during reporting |

*Variable Classes*

Based on this classification, I will define, 3 very small ADaM dataset excerpts containing only affected variables for clarity. Please imagine that each dataset has additional variables according to the CSISC standards.

| ADSL |
|---|
| USUBJID |
| DTHDT |

| ADAE |
|---|
| USUBJID |
| TRTEMFL |

| ADLB |
|---|
| USUBJID |
| CRIT1 |

*Working example datasets*

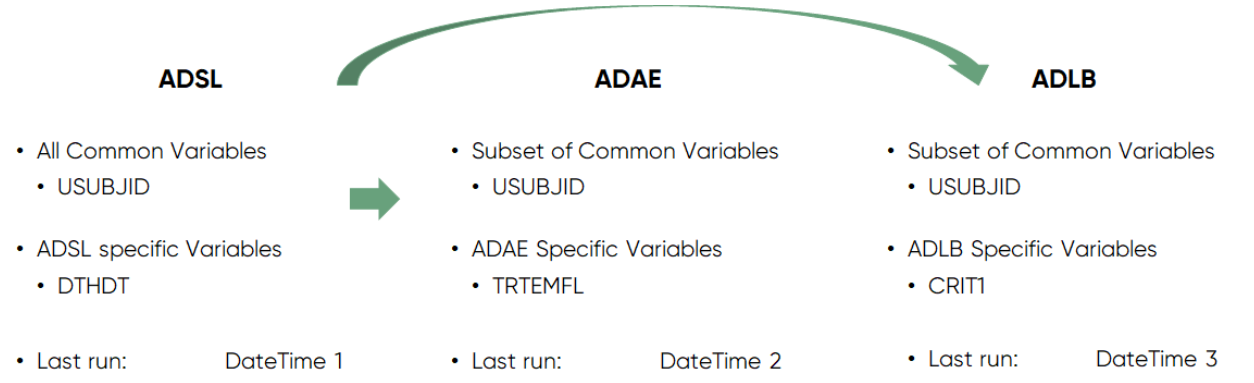
Please notice that USUBJID is common to all 3 datasets, while the other variables are unique to one dataset.

**PROBLEM**

The core problem described here is, that when you choose to create ADSL as a sort of 'master' ADaM dataset from which variables common to all ADaM datasets are picked, you create unnecessary dependencies between ADaM datasets. It may at first glance seem like a clever idea to create any variable only once, and it certainly is. But to place common variables in ADSL creates another (avoidable) problem as a consequence. Consider this example:

- ADSL is created first and goes through a satisfactory QC process to verify and validate its contents.
- ADAE is created second, picking common variables from ADSL. Those variables need not be QC'd and validates once more, only the new variables specific to ADAE.
- ADLB is created subsequently, still only requiring QC and validation of ADLB specific variables.

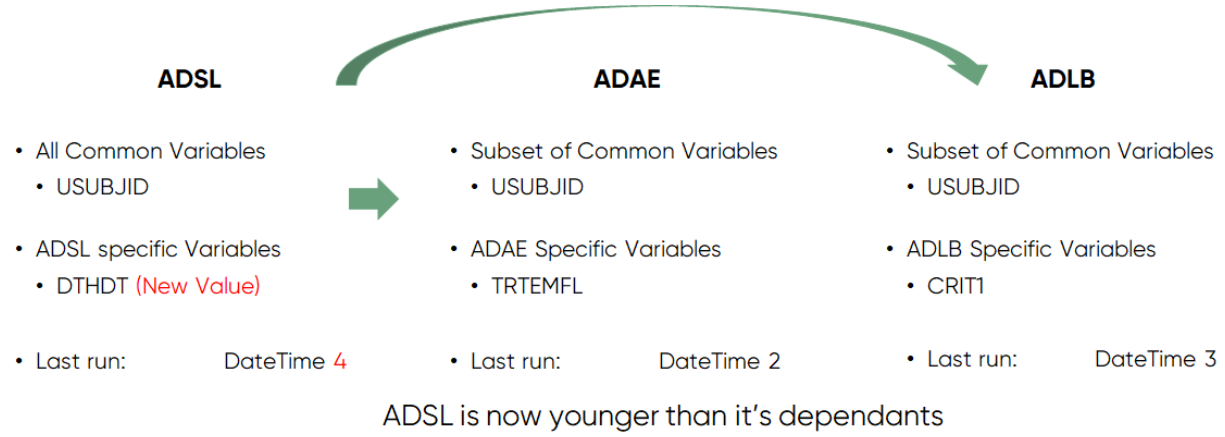
One of many outcomes of this process is, that the date stamps for when each dataset is finalized and QC'd reflects the order in which way the were created.




*Process flow for producingADaM's*

If a change to ADSL is then introduced late in the trial conduct, say for the sake of argument, a subject dies late in the follow-up epoch of the trial, the variable DTHDT, Date of Death, needs to be updated through-out the clinical data,

calling for a rerun of SDTM, affected ADaMs and subsequently affected TFLs. This in turn calls for an equal QC and validation of everything being rerun, mostly unnecessary, as the death of a subject only affects ADSL and a few tables and listings. If you choose only to update ADSL and the necessary tables, you will end up in a situation where the time stamps of finalized ADaMs and TFL, are no longer in sequence.




*Updating ADSL brakes dependants time stamp flow*

Misaligned dates are the first things an inspector looks for, together with version numbers (in case of a versioned clinical repository to hold programs and data) out of sequence, as they are so easy to spot. You may then risk, that trust in all results of the entire trial can be questioned, as you cannot document full traceability from raw data to Clinical Trial Report.

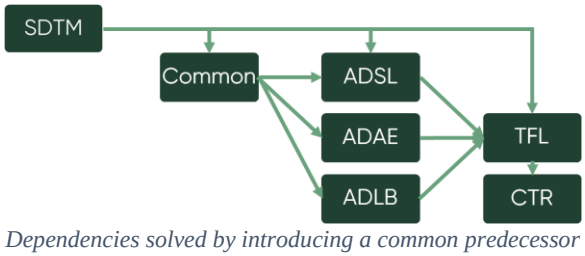
This holds true regardless of which technology is used for holding programs and data. Even a simple disk drive has timestamps of documents and logs, as does indeed the logs themselves. The challenge then becomes to guarantee that any changes to ADSL do not affect any dependent datasets? The only certain option is to rerun all dependent datasets to line up all the time stamps. Then of course you have to rerun all the QC and validation to make sure their time stamps are younger than the sources. You may be lucky not to have to rerun the entire trial, but you are still facing a daunting task.

## SOLUTION

In classical computer science, the problem of dependencies is normally solved by shifting the originator out of the line of dependencies. Likewise, instead of having ADaMs dependent of each other, I suggest to isolate all common variables into a separate program or dataset, and let *all* ADaMs, including ADSL, be dependent of such a common source. This ensures that:

- ADSL specific variables can be updated independently of any other ADaM dataset containing common and other variables
- Other ADaM specific variables can be updated independently of ADSL
- When a common variable need updating, you have to re-run (and re-QC) everything anyway

You then need to reorganize the structure of your ADaM program and datasets in the following way:

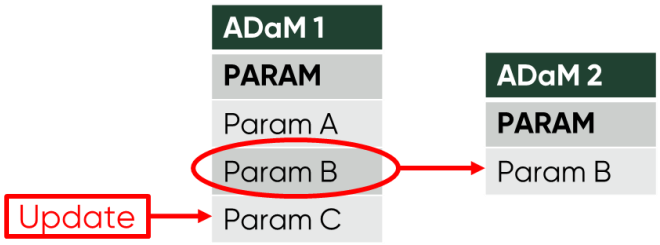



*Dependencies solved by introducing a common predecessor*

Common, in this structure, is a WORK dataset at LEO. There is no reason why this cannot be a permanent dataset, even if WORK datasets have it advantages. The decision on WORK versus permanent datasets only really depends on resources needed to recreate the Common dataset while creating any given ADaM dataset. Having it as a WORK dataset eliminates all dependencies, as there is no fixed originator among ADaM datasets.

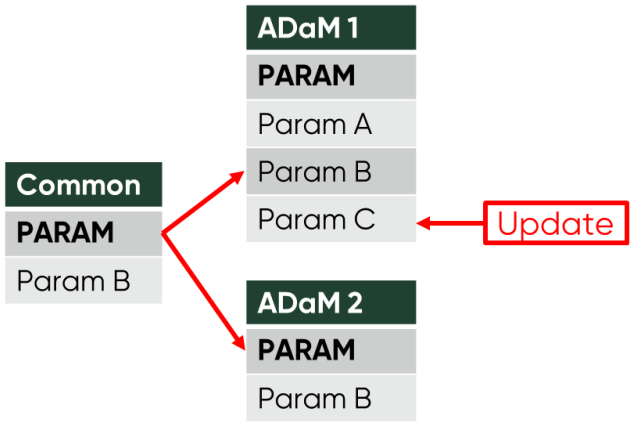
**A SIMMILAR EXAMPLE**

Dependencies can exist between other ADaM dataset apart from ADSL. You may have a collection of Basic Data Structure (BDS) datasets based on PARAM subsets from a global pool of values. PARAM may collect many discreet subsets, where one or more subsets requires further processing. A dependency problem quite similar to the one regarding ADSL arises when one of the subsets need updating.




*Dependencies between PARAM sets*

Say that the ADaM 1 dataset having Param C needs updating, and not Param A nor Param B. As ADaM 1 consists of a simple stacking of logically independent Params, there really is no need to update Param A or Param B when any change only affects Param C. The solution thus becomes to create a common BDS AdaM dataset containing Param B (common to ADaM 1 and ADaM 2), and then generate ADaM 1 and ADaM 2 from the Common ADaM.

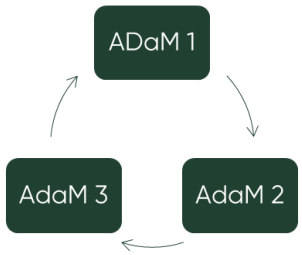



*Common PARAMs*

This way Param C can be updated, QC'd and validated without affecting Param B. Param A will be affected via the recreation of ADaM 1, but this solution reduces the problem of re-QC and validation, even if the problem may not go away entirely. The recommendation here is to use any such dependencies themselves to decide the structure of a common datasets for Param to be constructed.

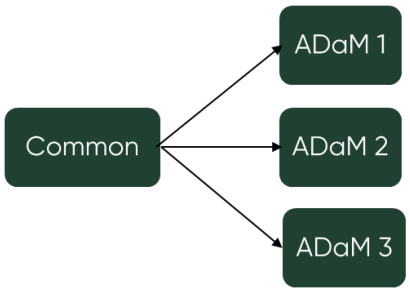
**A CIRCULAR EXAMPLE**

Before writing this paper, I thought that the example below having circular references was purely hypothetical, but I have later learned that this sort of thing happens in the real world. If you have a structure of ADaM datasets where variables are inherited in a circular fashion ADaM1->ADaM2->ADaM3->ADaM1, it will in real life always be random which ADaM is the youngest.




*Circular references*

Furthermore, if variables are picked from each ADaM to the next going round the full circle, the question then becomes; how many times do you have to run all the ADaM programs (and in which order) to be certain that any changes have been propagated to all its dependencies? Again, create a common origin independent of all the ADaMs and generate them from this common predecessor.



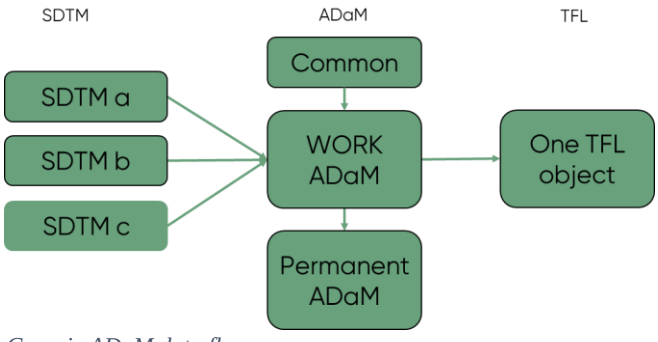

*Common predecessor*

## PERSPECTIVE

Based on the principle of generating ADaM datasets as independent WORK datasets without any interdependencies, I thus propose a generic structure for building ADaM datasets.

### PROPOSED ADAM ARCHITECTURE

The general idea is that the structure of programs to create ADaM datasets and derivates can be made quite uniform, following a simple scheme.



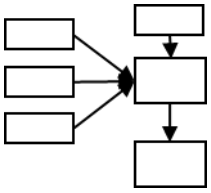

*Generic ADaM data flow*

Even if the Common dataset is also created from SDTM, it is show as an independent entity for simplicity alone.

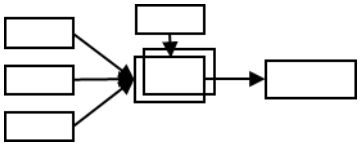
There will always be a number of SDTM datasets carrying details of a specific ADaM dataset. A Common dataset will hold the set of common variables. The WORK ADaM dataset combines data from relevant SDTM and Common datasets into a temporary dataset having all variables and values ready. This WORK ADaM dataset may more or less comply to the formal requirement for ADaM, but all data processing should be done when creating the Common WORK dataset. The program creating the permanent ADaM dataset may then adjust any remaining formalities,

guided by metadata. Finally, any TFL's may be constructed from the WORK ADaM without going through all the formalities of a final ADaM dataset, but without jeopardizing the traceability of datapoint from data collection to submission.

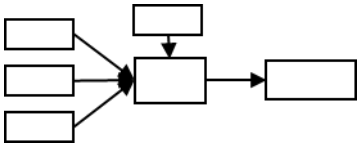
**EXAMPLES OF UTILIZING THE GENERAL STRUCTURE**



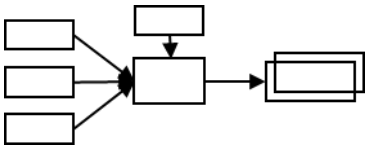
For each permanent ADaM dataset, the data flow will be a variation, producing only a permanent ADaM dataset from Common and relevant SDTM datasets, without producing any TFL



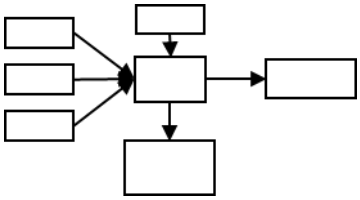
In some cases, i.e. the example earlier of splitting parameters of BDS structured ADaM dataset into separate sub-ADaM's each handling a subset of parameters, it is possible to have several temporary WORK ADaM's for producing a (number of) TFL. A permanent ADaM dataset can also be produced in a similar way.



In case of an AdaM dataset having only one TFL produced from it, it may make little sense to have a permanent dataset at all, as it may just make a submission more complicated. However, the production of such a solitude TFL may be much simplified, and code reuse may be much higher, by just creating a temporary WORK ADaM, and create the TFL from that.



A scenario often seen for figures, is one program creating a series of figures, each only differing from the others by a single parameter. I this case multiple TFL's may be created form a single temporary WORK ADaM dataset.



The full standard structure can also be used for creating one or more TFL's and permanent ADaM datasets in one go. However, it is not recommended to use this strategy for creating all TFL's and all permanent ADaM datasets at once. Afterall, clarity and readability should always be paramount for producing code, not flashy showing off.

Other constructs beyond those shown above may be devised.

**PROPPOSED TFL CODE ARCHITECTURE**

The organization of the ADaM data flow into standardized modules as just proposed enables a high level of standardization and reuse of code. The central idea is that the Common program creating the Common ADaM dataset as a temporary WORK dataset, can be encapsulated into a single piece of code being rerun whenever the Common WORK dataset is needed. This may lead to some redundancy of code runs, why some effort to optimize this code will be in place. However, for typical LEO trials of 500-1500 subjects this is not an interesting argument as dependencies are avoided, thus the common code only needs to be validated and QC'd once. Likewise, code to produce WORK ADaM's from Common and SDTMs will have the benefit of operating on uniform and well-known inputs, leading to still more code reuse.

Downstream, code to produce TFL's will also benefit from a uniform set of inputs, here being a complete temporary ADaM (or small set of ADaM's) to generate output in a standardized and uniform way. An example of the TFL code structure in this framework may look something like below, using the SAS™ language as an example. Other languages may of course be organized in quite a similar way.

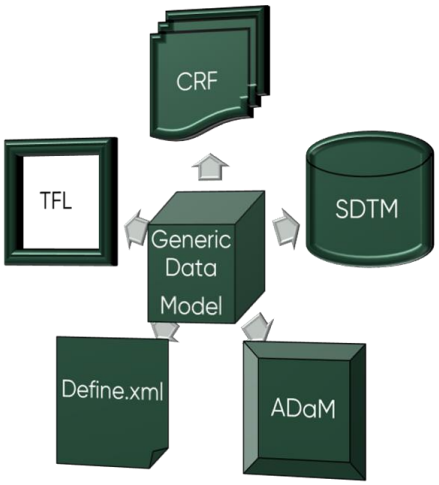
```
%Common;
%WorkADSL;
%WorkADAE;

%TableShell(shell="...");
%TableGet(tableno=...);
%TableTitle(byvars="...");
%TableFootnote(addendum="...");
%TableHeaders(continued=...);
%TableRows(box="...");
%TableStats(pval=..., cl=...);
%TableODS(destination="...");
%TableOutput(colwidths=...);
%TableCleanUp;
```

This is a hypothetical example, showing how code may be organized into segments creating the Common ADaM and creating temporary WORK ADaM's. Then assemble a table from various parts such as shells (table template), table number, title, footnote, column headers, rows, and statistical sections below the table, possibly including p-values and confidence limits. To complete the program, this is followed by pieces of code to direct and format the output, ending in an optional cleanup preparing for the next table.

**THE NEXT GENERATION**

The thought here can be extended even further. All clinical data, not only ADaM, can be standardized in a highly uniform way. One may imagine all clinical data being stored only once in a central data store, having a generic data model capable of supporting many different views of the data.




*Model-viewer organization of data*

This even more hypothetical example illustrates a central generic (linked?) data model, where CRF, SDTM, ADaM, define.xml and TFL are ways of viewing the data though a filter or view, organizing the data according to one of the formalities defined by the views. This way, any change to the data, either at time of data collection through the CRF, statistical analysis, or completely other ways, is immediately reflected in all views simultaneously. The key question then becomes to QC and validate the data and the views. Given the generic data model at the center is highly standardized, the views can be equally standardized. Ensuring the validity of the data may then be a supported by new views for that specific purpose.

## CONCLUSION

By applying a little structured organization to AdaM programs up front, dependencies between ADaM dataset and programs can be avoided. The proposed method is to first identify all variables common to the majority of ADaM datasets and put them in a temporary WORK dataset. Then pick variables from this Common dataset when constructing all ADaM datasets, completely avoiding to pick any variables from ADSL. This allows for ADaM dataset variables to be updated independently. The gain here is to avoid re-QC and validation of ADaM datasets and their derivates when they are not affected by the update. Whenever a common variable needs updating, everything must to be re-QC'd and validated anyway. The proposed structure can be extended to other areas of ADaM datasets and form a basis for a very generic approach to structuring ADaM programs, lending itself to a metadata driven generation of ADaM datasets based on uniform segments of standard code.

## CONTACT INFORMATION

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

Jørgen Mangor Iversen
LEO Pharma A/S
Industriparken 55
DK2750 Ballerup
Denmark
Work Phone: +45 20 60 61 76
Email: jmidk@leo-pharma.com
Web: https://www.leo-pharma.com/

Brand and product names are trademarks of their respective companies.