PP18

No ADaM is an island…planning study implementation of ADaM programming

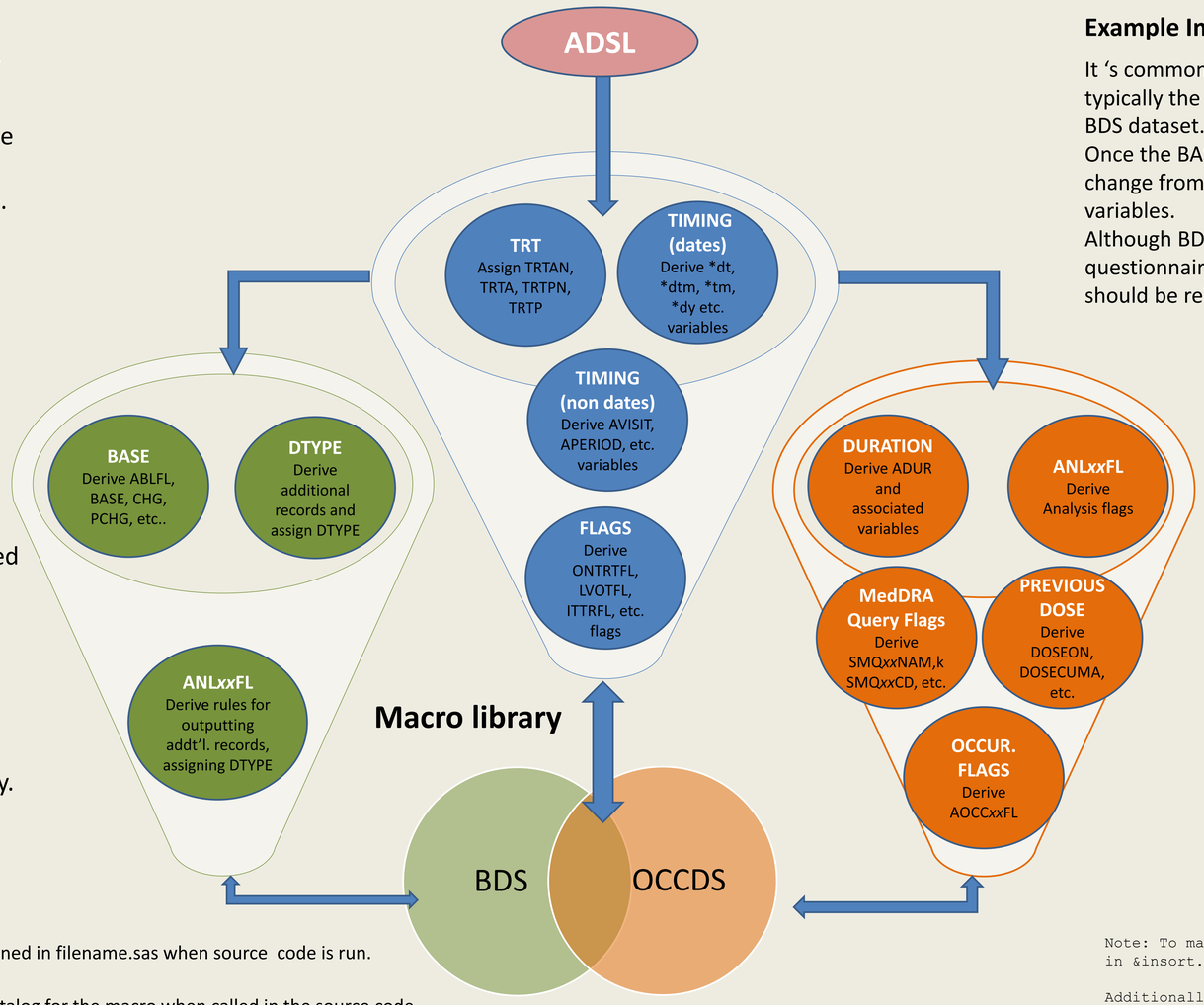
Alice Ehmann & Kirsty Lauderdale
aehmann@csg-inc.com, klauderdale@csg-inc.com

Clinical Solutions Group
BIOPHARMA SERVICES EXCELLENCE

By carefully assessing overlaps and similarities in derivations across ADaMs, we can create building blocks that result in robust, consistent, easily update-able, and well documented code. These building blocks can handle derivations including multiple treatment variables (TRT*x*, TRT*x*N), complex baseline derivations and visit windowing; by moving these derivations out of individual domain codes, we can ensure consistent application across all ADaMs. This poster shows how to identify similar derivations across ADaMs (ODS, BDS and ADSL) and how to implement these when coding to ensure consistency and easy maintenance.

Advantages

- ✓ Consistency in derivations across study/ studies and compounds.
- ✓ Modifying derivations that cross multiple domains done in one location.
- ✓ Easily maintained and transferable code.
- ✓ Fast and efficient replication for new reporting efforts and similar studies.
- ✓ Programming time is reduced because certain derivations are “standardized”.
- ✓ QC time is reduced due to consistent implementation of derivations.
- ✓ Aids in ensuring consistency and high quality cross-checking across deliverables.
- ✓ By not hiding derivations in long ADaM dataset code, they can be easily reviewed across reporting efforts, studies, and compounds.
- ✓ Aids in the ADRG and Define.xml development by keeping similar derivations in one location..
- ✓ Deviations from similar derivations can be found and documented appropriately.



Example Implementation: BASELINE FLAG

It ‘s common to require a baseline flag for the baseline record, and typically the baseline value is stored in the BASE variable within a BDS dataset.
Once the BASE value is present, it’s commonly used to calculate change from baseline, percent change from baseline, and shift variables.
Although BDS datasets may store vastly different data (e.g. questionnaires vs. labs), the methodology for calculating these values should be relatively consistent across BDS datasets.

```
*assumption: baseline=the last obs on/prior to studyday 1 *;

%macro bds_ablfl(inds,
                 outds,
                 insort=usubjid paramcd adt atm,
                 inlast=paramcd);

 data _temp;
   set &inds.;
   if . < ady <=1 then t_ablfl='Y';
 run;

 proc sort data=_temp;
   by &insort.;
 run;

 data &outds. (drop=t_abfl);
   set _temp;
   by t_abfl &insort.;
   if last.&inlast. and t_ablfl='Y' then ablfl='Y';
 run;
%mend bds_ablfl;

* Assign ABLFL for Labs *;
%bds_ablfl(adlb, adlb_ablfl);

* Assigning flags for ADQS *;
%bds_ablfl(adqs, adqs_ablfl);

* What if ADQS did not collect times? *;
%bds_ablfl(adqs, adqs_ablfl, insort=usubjid paramcd adt);
```

Note: To make the macro more robust you can check to see if the vars in &insort. exist in &inds and drop them if they don’t.

Additionally, it may be advantageous to limit the sort and last var imputations and instead use %if %then logic within the macro based on the value of &inds. This ensures baseline derivations are consistent across BDS and becomes a source of documentation for deviations.

Calling Macros

- ➢ **%include “…/filename.sas”**
  File can contain multiple macros, inserts code contained in filename.sas when source code is run.
- ➢ **Compiled Macros**
  Stored in SASMACR SAS catalog, SAS searches the catalog for the macro when called in the source code.
- ➢ AUTOCALL macros :
  Directory & file refs are specified in the SASAUTOS options, SAS searches the directories in the sasautos for the macro when called in the source code.

Conclusion

By carefully planning ADaM programming prior to starting code development, we can gain huge benefits across multiple domains. Using building blocks for common variable derivations, we gain important efficiencies in programming and validation time and in senior review. Macro-tizing ADaM development on both the primary and qc side of programming can ensure all values are updated quickly and consistently, without fear of missing a domain, saving time and ensuring consistent high quality deliverables. Macros can be incremental and do not need to be complex to benefit the entire programming team at the study, compound, and global level.
By centrally aligning these derivations in one location, we also aid in the development of the ARDG and Define documentation, which helps immensely during study close-out.