

PHUSE EU Connect 2022, Coding Tips & Tricks

CT11 - Practical implementation of Good Programming Practice and other theories in macro development

Fred Ross



13–16 November
ICC Belfast

EU 2022

The Clinical Data
Science Conference

parexel®

Agenda

- Project, Starting Point & Good Programming Practice – RDM Lab Macro
 - RDM Lab Macro – What was the project?
 - RDM Lab Macro – My Starting Point
 - Good Programming Practices in Context
- Boiling Down the Code – Modularity
 - CRF/NONCRF macros
Early & Late Phase and PDMA
 - Programs – Refining, Combining & Prefixing
- Structuring & Adding Code
 - User Errors and Warnings + Example of a Macro
 - Example code from Final Program
 - Conclusion – How far did we meet the PHUSE Guidance?
- Questions

Project, Starting Point & Good Programming Practice

RDM Lab Macro

RDM Lab Macro – What was the project?

➤ Macro Purpose & Use?

- Originally a suite of macros for taking raw data and turning this into SDTM equivalent data which is then pushed through a LAB ADaM macro to create lab gradings.
- 6 Differing versions & combinations of macros for Early Phase, Late Phase and PDMA with CRF and NONCRF versions of each
- Used across hundreds of different studies
- Used by multiple programmers who aren't SDTM/ADaM programmers
- Maintained by a handful of individuals who have other responsibilities
- Original macros are taking too much time of the maintainers!



RDM Lab Macro – My Starting Point

› Challenges



- › Familiarity – I have not used these macros
- › Understanding – I have not worked on this area of programming before
- › Time – I am not full time on this project

› Start of the Project



- › Lots of meeting and discussions
- › Determining the requirements for new macro
- › Looking at issues the current maintaining team was facing

Good Programming Practices in Context

➤ PHUSE GPP relating to RDM Lab Macro Task

- Write code that is clear and easy to read and review
- Write code that is easy to maintain, modify and debug
- Use robust programming to reduce the need for code maintenance
- Develop code that can be reused or easily adapted
- Write code that uses minimal computer processing resource (efficient code)
- Write code in such a way to reduce logical and syntax errors



Pages / ... / Good Programming Practice Project Team

Good Programming Practice Guidance

Created by , last modified by Paula on Sep 21, 2021



Boiling Down the Code

Modularity

CRF/NONCRF macros

Early & Late Phase and PDMA

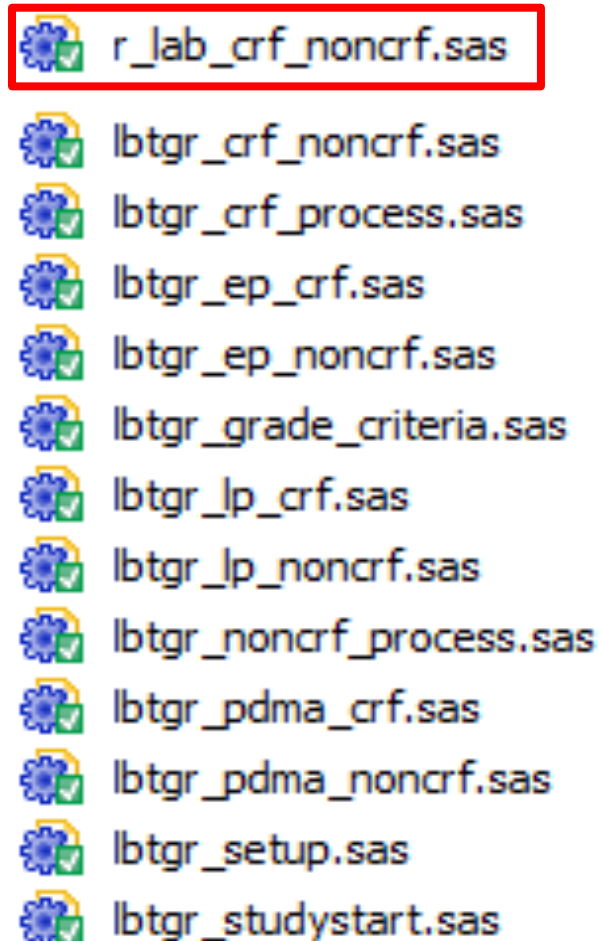
Old Program Folders	New Program Folders
Early_Phase/program	RDM_Templates/program
Late_Phase/program	
PDMA/program	
RDM/program	
Early_Phase/macro	RDM_Templates/macro
Late_Phase/macro	
PDMA/macro	
RDM/macro	

- Original versions stem from similar versions but are different, sometimes subtly.
- Reviewing all versions and funneling these into one program with liberal use of %if %then %do statements
- Further reduction in code used code

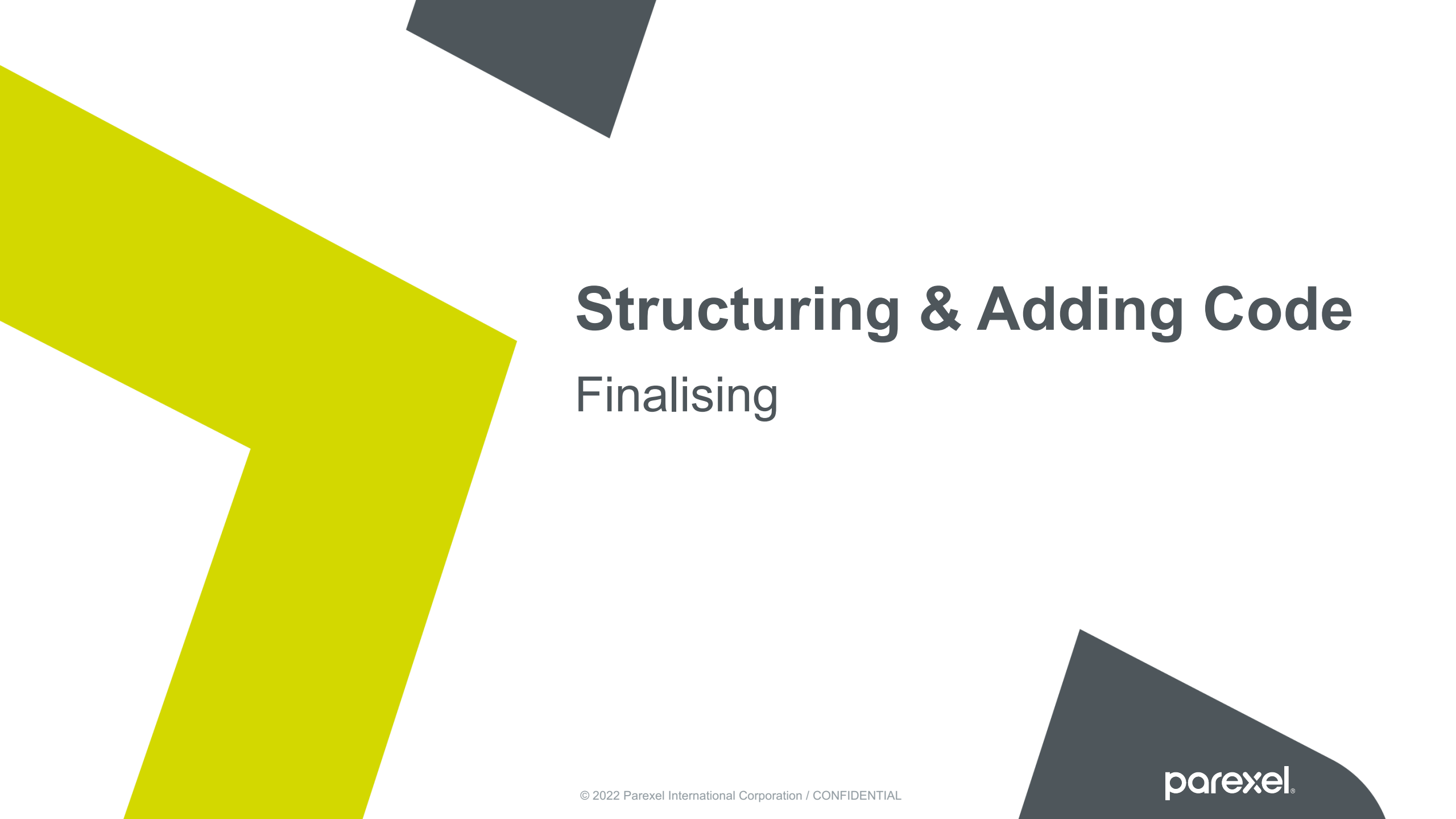
Some of the Early Phase Macros:

-  rdm_crf_process.sas
-  rdm_noncrf_process.sas
-  r_lab_crf.sas
-  r_lab_noncrf.sas
-  rdm_final_processing.sas

Programs – Refining, Combining & Prefixing



- Refined and defined clear purpose for each program
 - *Write code that is clear and easy to read and review*
- Combined some bits of code into logical groupings
 - *Write code that is easy to maintain, modify and debug*
- Prefixes depending on program type
 - *Write code that is clear and easy to read and review*



Structuring & Adding Code

Finalising

User Errors and Warnings + Example of a Macro

- › Adding easy user errors and warnings
 - › Was a valid option selected etc.
 - › Does not account for every conceivable incorrect way of using the code
- › Option to add automatic start date detection code later on.
- › Hiding this code from main users/non-maintaining programmers

Program Header

```
23 %macro lbtgr_studystart(date=);  
24 %***Warnings;  
25 %if %qsubstr(%str(&date.),5,1) ne %str(-)  
26    or %qsubstr(%str(&date.),8,1) ne %str(-)  
27    or %length(&date.) ne 10  
28    %then %do;  
29        %put %str(ERR)OR: Study start date inputed is not of correct format YYYY-MM-DD. Aborting program.;  
30        %abort;  
31    %end;  
32  
33 %*** Set _studystart;  
34 %global _studystart;  
35 %if &date. ne %then %let _studystart = &date.;  
36  
37 %else %do;  
38  
39 %* Automatically detect study start date code here;  
40 %put %str(ERR)OR: Study start date is empty and requires it to be populated by user. Aborting program.;  
41 %abort;  
42  
43 %end;  
44  
45 %mend lbtgr_studystart;
```

Example code from Final Program

Program Header

```
31  %lbtgr_setup(studyphase= /* EP LP or PDMA */ ,
32              crf_or_noncrf= /* CRF or NONCRF or BOTH*/,
33              debug= /*Y or N*/);
34  %* When debug is set to Y diagnostic datasets will be created in outdata;
35
36
37  %*** Update study start in macro lbtgr_studystart to pick up correct LB metadata;
38  %*** Date must be in ISO format YYYY-MM-DD;
39  %lbtgr_studystart(date=);
40
41
42  %*** Put lab grading criteria from the list below into lbtgr_grade_criteria;
43  %*** _EXT_NCICTCV4_ : for NCICTCAE version 4;
44  %*** _NCICTCV4_ : for strict NCICTCAE version 4 (Potential use of Lab Interface only);
45  %*** _NCICTCV5_ : for NCICTCAE version 5;
46  %*** _WHOV1_ : for WHO grading version 1;
47  %lbtgr_grade_criteria(criteria=);
```

Additional Macro Calls and Pre-processing Options

```
75  %*** Call to macros lbtgr_(EP/LP/PDMA)_(CRF/NONCRF) depending on phase and crf option selected in lbtgr_setup;
76  %lbtgr_crf_noncrf(crf_dsin_lab = staging._lab /* Name of input lab dataset for CRF */,
77                  noncrf_dsin_lab = staging._lab_nc /* Name of input lab dataset for NONCRF */,
78
79                  crf_dsout = outdata.r_lb_rdm_vad /* Name of output CRF dataset */,
80                  noncrf_dsout = outdata.r_lb_ncrf_rdm_vad /* Name of output NONCRF dataset */);
81
```

Post-processing Options

Conclusion – How far did we meet the PHUSE Guidance?

› Good Programming Practices should (PHUSE Wiki) :

- › Ensure the clarity of the code and facilitate code review;
- › Save time in case of maintenance, and ease the transfer of code among programmers or companies;
- › Minimize the need for code maintenance by robust programming;
- › Minimize the development effort by development and re-use of standard code and by use of dynamic (easily adaptable) code;
- › Minimize the resources needed at execution time (improve the efficiency of the code);
- › Reduce the risk of logical errors;





Thank you & any questions?

parexel[®]