

Building Efficiencies in Standard Macro Library using Polymorphism

Binoy Varghese, MedImmune; Sagar Rana, Chiltern

ABSTRACT

It is common practice in the bio-pharma industry to develop and maintain a library of standard SAS® macros to facilitate expedited data analysis & reporting, directed at ensuring compliance, consistency, quality and reusability. As requirements change over time, new macros are created or existing macros are updated. In both scenarios, there is a tradeoff. If new macros are created, programs calling earlier versions of these macros have to be modified before being used on a new project, thereby affecting the portability of the programs. If existing macros are updated, programs from older projects may cease to execute as originally intended impeding backward compatibility. Both these issues can be addressed by using polymorphism. Polymorphism is an object oriented programming concept that refers to the ability of managing methods bearing the same name but exhibiting different behaviors. From the context of SAS programming, this may be translated as having the capability of calling homonymous macros that may accept identical or different parameters to perform a diverse set of tasks. In this paper, we examine the concept of polymorphism from a SAS macro programming perspective and present an implementation of this technique.

ANATOMY OF A STANDARD MACRO LIBRARY

The objective of a standard macro library is to serve as a central repository for adaptable, dynamic and robust code which can be instantiated at the study level without the need for duplication. These macros are carefully designed, developed and validated to ensure that the underlying algorithms can handle a diverse set of inputs, generate a variety of outputs, and allow a wide range of study specific customization. The benefits of having such a library are many, some of which are: cross-study consistency of code & outputs, reduction in implementation & validation efforts and improvement in overall quality & accuracy. The standard macro library comprises of various types of macros that may be loosely categorized as high-level (end-user) macros and low-level (building-block & utility) macros. The reason behind modularizing repetitive tasks into low-level macros is to ensure that global changes applied to a low-level macro automatically permeate into high-level macros.

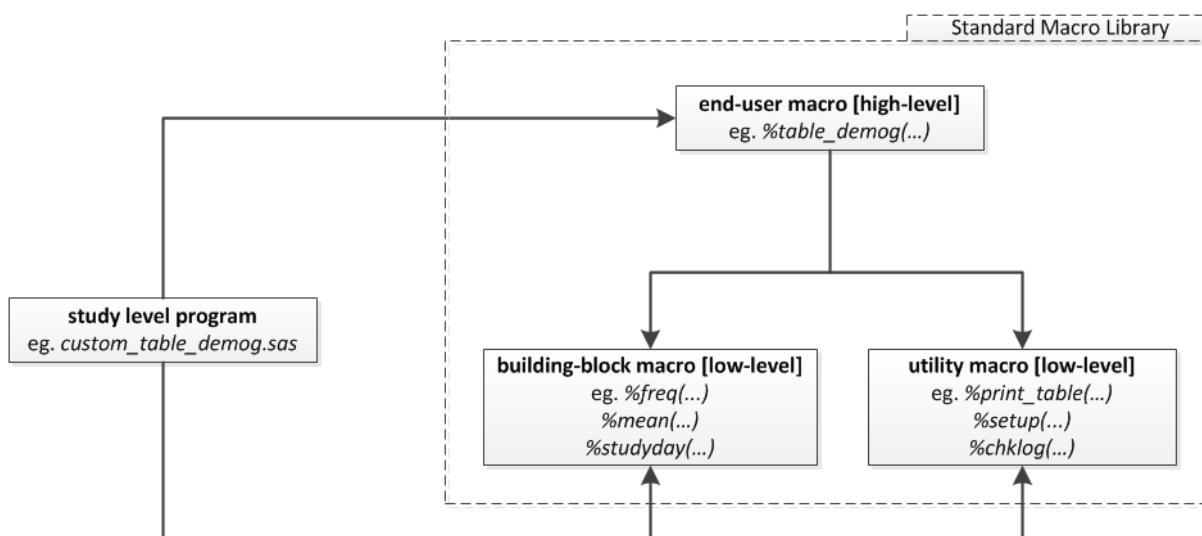


Figure 1 Constituents of a Standard Macro Library

In Figure 1 above, the study level program “custom_table_demog.sas” introduces study specific programming prior to invoking end-user macro “%table_demog” which may, depending on its architecture, invoke various building-block or utility macros.

EVOLUTION OF A STANDARD MACRO LIBRARY

It is natural for the standard macro library to evolve over time as strategies within the organization advance in regard to tables, figures & listings (TFL), dataset structures & code lists, derivation rules and internal processes. Examples of some such changes are: modifications originating from new releases of CDISC® guidelines, updates to reporting style manual and amendments to standard TFL templates. As the library adapts to these organizational decisions, new

versions of standard macros come into existence. The original macro has to be retained in order to safeguard older study level programs from any unintended changes, especially if the study or analysis is locked.

IMPACT OF EVOLUTION

The existence of multiple versions of the same macro can have a debilitating effect on the portability and upkeep of macros & programs at both the study level and within the macro library.

PORTABILITY AT STUDY LEVEL

On many occasions, programs customized for one study can be replicated for another and re-used with minimal changes. However if such a program were to depend on the version of the invoked macro, then that would inhibit its portability, especially as the standard macro library continues to evolve. From a compliance perspective, newer studies will have to use the latest version of macros.

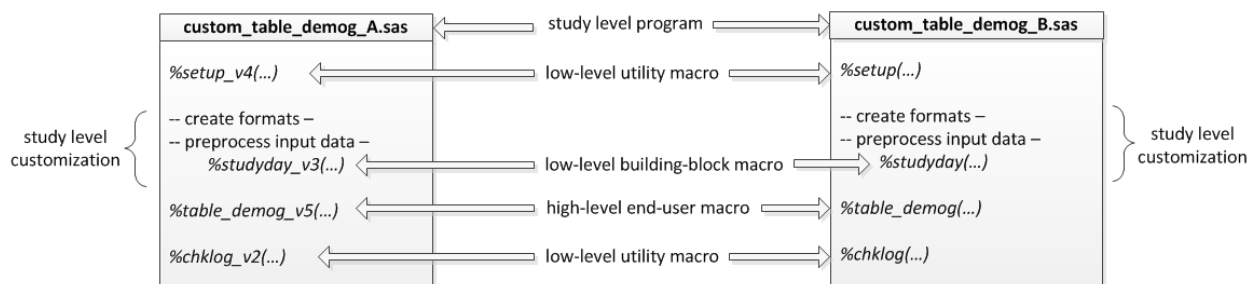


Figure 2 Study level portability

In figure 2 above, “custom_table_demog_A.sas” invokes different versions of low-level and end-user macros. If this program were to be duplicated in another study, the programmer will have to re-visit the macro library documentation to ensure that the latest version of the applicable macros are being invoked. This process can be labor intensive and prone to human errors. Compare this with “custom_table_demog_B.sas” which is version neutral and hence can be readily ported without requiring any modifications to underlying macro calls.

UPKEEP WITHIN STANDARD MACRO LIBRARY

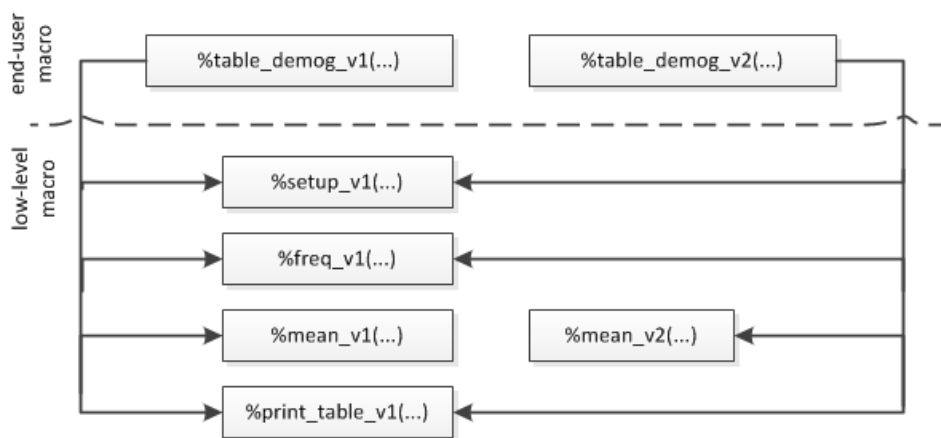


Figure 3 End-user dependency

Examine the scenario presented in Figure 3. A standard end-user macro (%table_demog_v1) is calling multiple low-level macros. An update to the low-level macro (%mean_v1) that results in the release of a new version (%mean_v2) forces the end-user macro to undergo a version release of its own (%table_demog_v2).

Considering that a standard macro library has many end-user macros and countless low-level macros, managing different versions in order to accommodate dependencies can be challenging.

POLYMORPHISM

Polymorphism refers to the ability of managing methods bearing the same name but exhibiting different behaviors. From the context of SAS programming, this may be translated as having the capability of calling homonymous macros that may accept identical or different parameters to perform a diverse set of tasks. Although this is not a feature native to SAS macro programming, it can be introduced using a parent child interface and a selection mechanism. Such an interface provides independence between the invoking macro and the invoked macro.

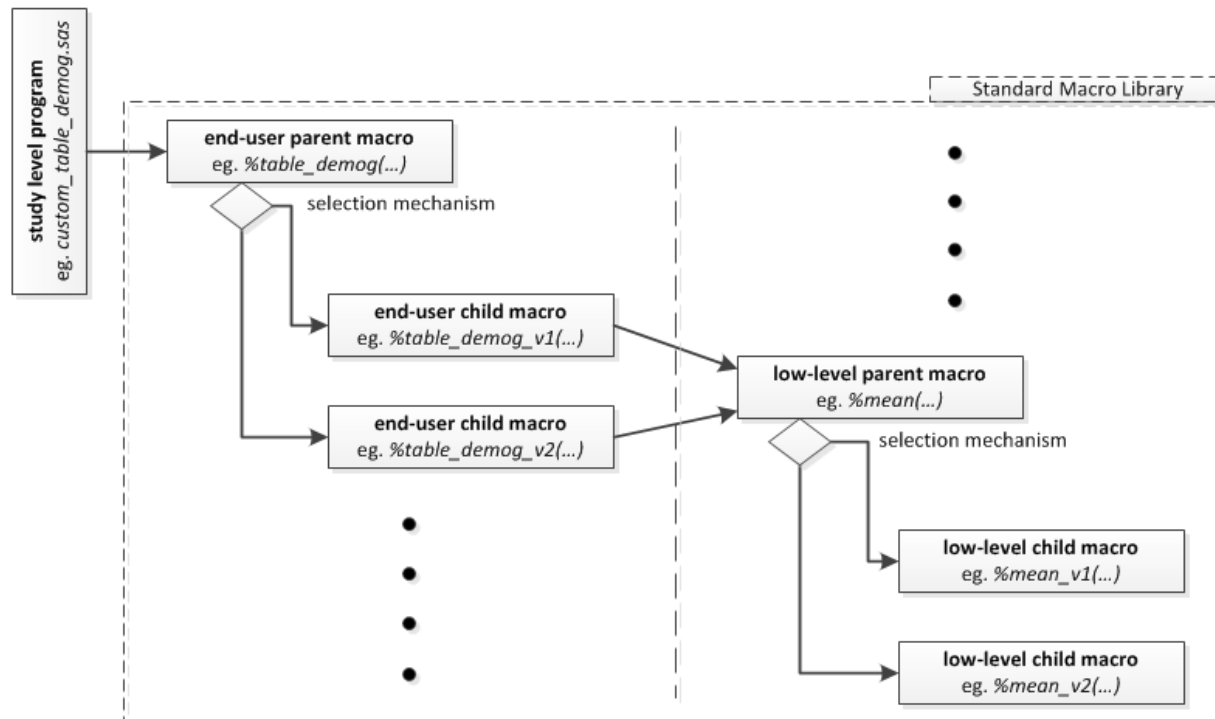


Figure 4 Parent-child interface

In Figure 4 above, “custom_table_demog.sas” calls “%table_demog” which in conjunction with the selection mechanism and incoming parameter values determines whether to delegate the task at hand to “%table_demog_v1” or “%table_demog_v2”. The delegated end-user child macro in turn invokes appropriate low-level parent macros.

The selection mechanism can be individually customized for each parent-child cluster or a broader apparatus can be implemented.

PROOF OF CONCEPT

Although this concept can be expanded to incorporate complex variations, we are demonstrating a simple implementation for the sake of brevity. Few ground rules are established to devise this specific case. They are:

- Folder structure as illustrated in Figure 5.
- Selection mechanism that assumes:
 - a. Naming convention for a child macro is comprised of <parent_macro_name>_<date>, where date is expressed as YYYYMMDD and is used to track the version.
eg. if a parent macro is named “%m_helloworld” then the corresponding child macros may be named as “%m_helloworld_20151025”.
 - b. All child macros will be located in a folder with the same name as the parent macro.
 - c. Existence of a global macro variable “cutoff” in the study level autoexec file that is assigned a date value.
eg. %let cutoff=2016-02-17

FOLDER STRUCTURE

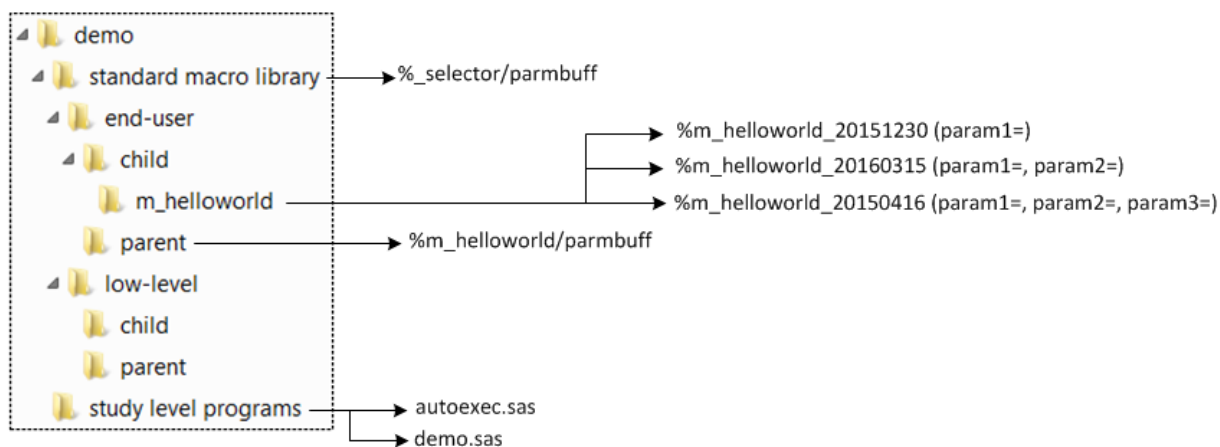


Figure 5 Directory hierarchy

CODE INTERACTION

The selection mechanism in this specific implementation uses the “cutoff” macro variable to determine the most recent version of the macro prior to the specified date. The parent macro utilizes this information and binds the appropriate child macro dynamically to the study level program in real time. Although all three child versions of “m_helloworld” take different number of input parameters, the example presented does not take this factor into consideration for identifying the target child macro.

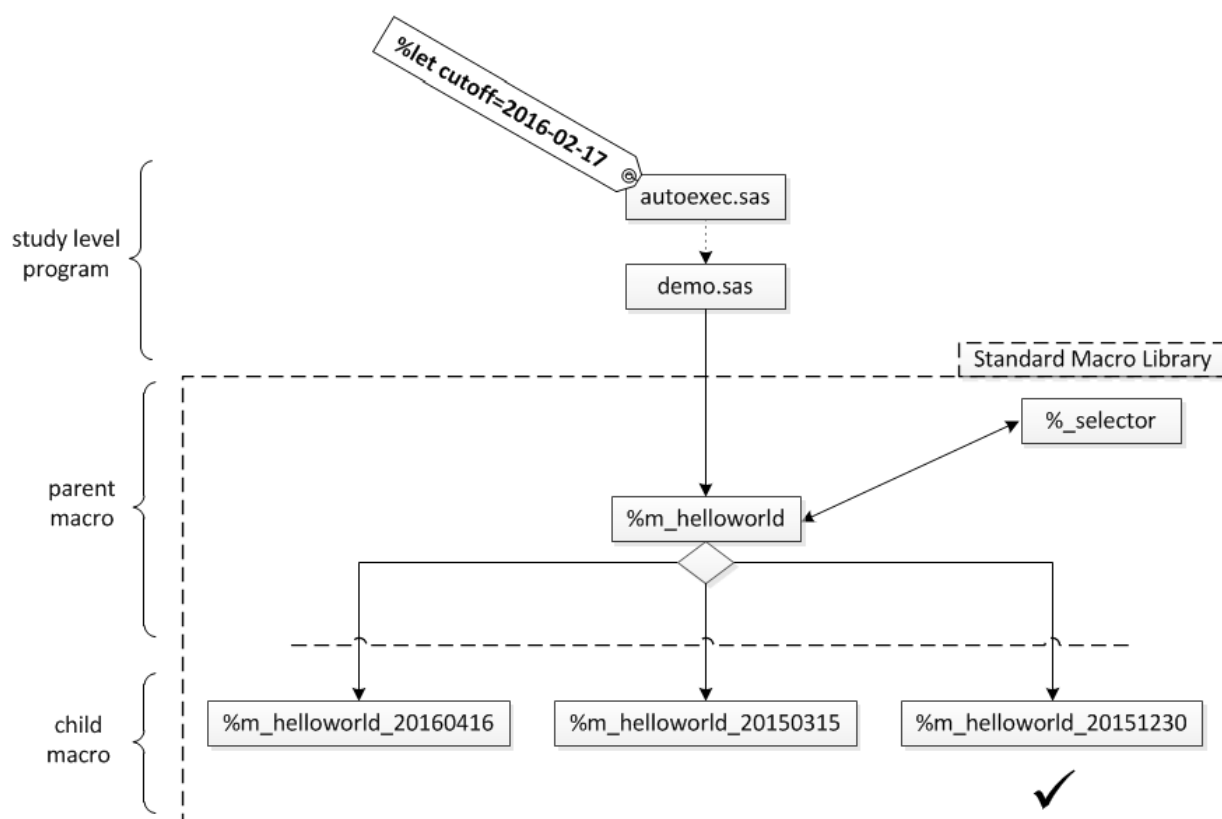


Figure 6 Control flow diagram

SOURCE CODE

autoexec.sas

```
%let cutoff=2016-02-17;

options mautosource sasautos = ("..\standard macro library\end-user\parent"
"..\standard macro library" SASAUTOS);
```

demo.sas

```
%m_helloworld(param1=This is a demo of polymorphism);
```

m_helloworld.sas

```
%macro m_helloworld/parmbuff;

    %let name=&sysmacroname;
    %put &name;
    options insert=(sasautos=("..\standard macro library\end-
user\child\%cmpres(&name)"));
    %put %sysfunc(getoption(sasautos));
    %_selector(name=&name,%substr(&syspbuff,2);

%mend m_helloworld;
```

_selector.sas

```
%macro _selector/parmbuff;

    data _null_;
        length parm $32000;
        parm="%syspbuff";
        parm=tranwrd(tranwrd(parm,scan(parm,1),""), "( ,", "(");
        call symput('parm',parm);
    run;

    filename filelist "..\standard macro library\end-user\child\%cmpres(&name)";

    data mdata;
        length mname $200;
        did=dopen("filelist");
        do i=1 to dnum(did);
            mname=compress(upcase(dread(did,i)));
            mname=tranwrd(mname, '.SAS', '');
            if mname="%cmpres(%upcase(&name))" then do;
mdate=input(cats(tranwrd(mname,"%cmpres(%upcase(&name))"||"_",""), ND8601DA.);
                if mdate <= input("%cmpres(&cutoff)",IS8601DA.) then
                    output;
            end;
        end;
        format mdate is8601da.;
        keep mname mdate;
    run;

    proc sql noprint;
        select mname into: worker
            from mdata having mdate=max(mdate);
    quit;

    proc datasets lib=work noprint;
        delete mdata;
        quit;
    run;
```

```
filename filelist;

%&&worker&parm;

%mend _selector;

m_helloworld_20151230.sas

%macro m_helloworld_20151230(param1=);
    %put &sysmacroname &param1;
%mend;

m_helloworld_20160315.sas

%macro m_helloworld_20160315(param1=,param2=);
    %put &sysmacroname &param1 &param2 ;
%mend;

m_helloworld_20160416.sas

%macro m_helloworld_20160416(param1=,param2=,param3=);
    %put &sysmacroname &param1 &param2 &param3;
%mend;
```

CONCLUSION

Polymorphism in a Standard Macro Library framework provides a high level of abstraction and independence between the caller and the callee. The capability to delegate using a parent-child interface allows study level programs and end-user macros to be version neutral of the underlying macro calls, in effect reducing maintenance efforts and enhancing portability.

The Proof of Concept can be reproduced using the the folder structure presented in Figure 5 and the source code. It has been successfully tested in batch mode using SAS version 9.3 on Windows® 7 professional.

CONTACT INFORMATION

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

Binoy Varghese
MedImmune
vargheseb@medimmune.com

Sagar Rana
Chiltern
sagar.rana@chiltern.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.