

PCKG: Managing SAS Macro Libraries

Magnus Mengelbier, Limelogic Ltd, London, United Kingdom

ABSTRACT

Development of standard SAS® macro libraries is a continuous exercise in feature development, maintenance, logistics and version control that can make periodic release of macro library versions a complex activity. The PCKG macros were developed to enable easy control, administration and use of macro libraries used in the reporting of Clinical Trials. The PCKG features add simple version control allowing multiple active versions, traceability, impact analysis, archiving and rudimentary library configuration.

INTRODUCTION

The SAS macro facility allows for both simple one-off macros for adding basic flexibility and more extensive libraries for complex and standardised integrated tasks. There are a several different approaches to manage multiple versions of macro libraries. The PCKG utility is a suite of SAS macros to simplify the administration and use of SAS macro libraries and macro library versions in a regulated environment.

PCKG was developed to provide an easy mechanism to specify which macro libraries are used and, more specifically, what version. At the same time, we are able to document what macro libraries and associated versions are made available for a program. The entire configuration aspect allows both traceability and simple efficient impact analysis.

The PCKG utility is based on a set of registry data sets and very small set of SAS macros to deploy, administer and use within a SAS project environment. The Open Source license allows anyone who is interested to use, contribute, extend and evolve the PCKG utility to fit their process..

APPROACH MACRO VERSIONS

Different strategies exist for to version individual macros, from a simple version identifier embedded in the macro name to more comprehensive approaches that that rely on a version parameter or flag. The approaches are sufficient for single macros, but do not give simple means to version an entire package of macros.

A common approach to version a library of macros is to include a version identifier somewhere in the path to the macro directory.

VERSION IN THE MACRO NAME

The most simple approach is to embed a version number in the macro name, e.g. %mymacro1(...), %mymacro2(...), etc. This allows for different versions of a macro to co-exist without naming collisions.

The exact naming convention is flexible as you can

- use a version increment, e.g. %mymacro1(...), %mymacro2(...), etc.,
- identify a major and minor version, e.g. %mymacro_1_0(...), %mymacro_1_1(...), %mymacro_2_0(...), etc. or
- any other convention that provides enough process detail to manage versions.

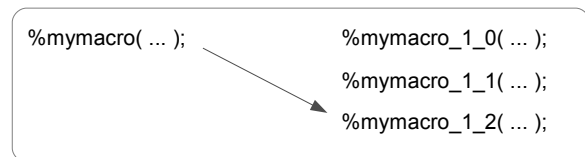


Figure 1. General macro links latest version

Referring to the *latest* version can be as simple as defining a general %mymacro(...) that is updated for each release of a new version for %mymacro() that simply redirects the macro call to the latest version (Figure 1).

VERSION AS A PARAMETER

An extended variant of the above approach and its general %mymacro(...) is to include version as part of the macro parameters, which enables a single macro call to specify the *latest* or a specific macro version.

As above, macro names embed major and minor version details as part of the macro name, e.g. %mymacro_1_0(...), %mymacro_1_1(...), %mymacro_2_0(...), etc.

A generic macro %mymacro(...) is defined that includes all the current and depreciated macro parameter definitions including the reserved special parameter *version*. The parameter *version* is used to explicitly call a macro version. If version is omitted,, the latest version will be used.

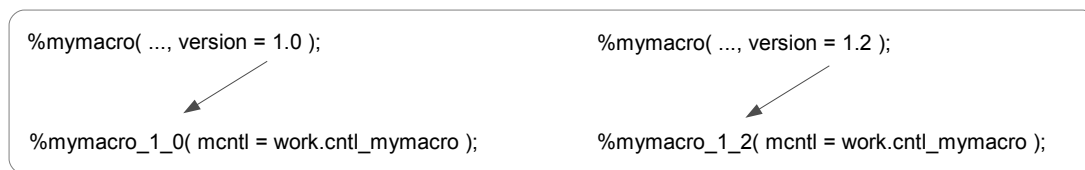


Figure 2. Use of a version parameter

Macro parameter values are passed to the “version” macro using a parameter data set, the *mcntl* parameter, as this provides consistent parameter references for all macro versions, eliminating the potential for nested if-statements when macro parameters are added and depreciated across macro versions.

VERSION IN THE DIRECTORY PATH

The above version strategies all focus on a single macro and not entire macro suites. Approaches to version entire macro packages or macro suites vary, but using a version identifier or equivalent reference somewhere in the directory path is most often sufficient and a popular approach.

For example, an organisation may have developed a set of reporting macros over a period in time with multiple releases of major and minor updates (Figure 3). It is common that ongoing studies, and even entire projects, continue to use an older version as a migration is difficult to justify, especially with significant updates to critical tools and utilities.

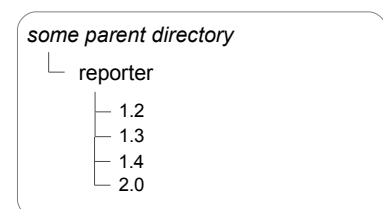


Figure 3. Version in the directory path

The macro library version as part of the directory path also allows for retaining version context for individual macros as these are kept independent.

PCKG UTILITY

The PCKG utility is designed to simplify and assert greater control for a business process where different versions of the same macro suites are allowed to coexist, whether a business requirement or simply backward compatibility.

There are many approaches to include macros in SAS programs, but the SAS autocall facility with the SASAUTOS system option, simple %include statements or both are very common process standards. Many organizations rely on a SAS autoexec or similar configuration files that are either global, project specific or combination in order to point SAS to the correct macro location. For each approach, there is an effort of administration associated that incorporates manually configuring current and pointing to updates to the latest or specific version of a macro library.

A systematic approach is to introduce a standard interface, such as an initialize or standard macro, to manage references. The %initsystems(...) macro [1] allows for standardizing references to specific versions of a macro library or system.

```
%initsystems ( <systemname> = <version>, ... , <systemname> = <version>);
%initsystems ( pharmabox = 1, toolbox = 2);
```

Figure 4. Example of an %initsystems() macro call

The PCKG utility extends the concept of the %initsystems(...) macro to allow for central administration and greater flexibility. PCKG relies on a central registry that maintains a master list of libraries and their respective versions. By default, the latest package version is selected but the tool allows for referring, or linking, to a specific version, which benefit is highlighted by the example from the previous section.

SIMPLE DESIGN

PCKG is designed to be a small utility with very simple functions, which relies on two registry data sets and a core of required process macros (Figure 5). Additional utility macros are available that simplify administration and other tasks, but not required for basic function.

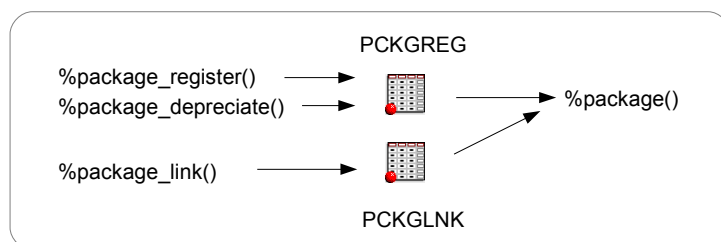


Figure 5. Registry data sets and required macros

The registry includes the master list (PCKGREG) and the link (PCKGLNK) data sets. The master list contains details for each package and version, including any specified dependencies.

The link data set maintains the list of a project's link to a specific package version. It is assumed that if a link is not defined, the latest version is to be used.

The master list, links and associated details are maintained through a set of PCKG macros that allow all common administration tasks.

PCKG MASTER LIST

The PCKG master list is a simple SAS data set that contains all the meta-data required to correctly point to the macro library location and any other relevant associations and qualifying attributes. The data set structure (Table 1) is kept simple and easily readable to simplify debugging and assist in any Quality Control, both peer reviewed and programmatic compare.

NAME	Reference name of package
VERSION	Package version
SEQUENCE	Order of application
PARAMETER	Configuration parameter
VALUE	Parameter value
DEFAULT	Package version default flag
DEPRECIATED	Depreciated package flag

Table 1 Structure of the PCKG master list data set

A package is always referred to by the reference name (NAME) and version (VERSION), which is a general convention. If version is omitted, the version assigned as default will be used. The special package *System* refers to the PCKG utility which allows for PCKG configurations attributes to be stored, which essentially implies that PCKG is able to define libraries, dependencies and attributes for itself (see macro %package_init() macro).

A package can have different attributes (Table 2) assigned, which allow for libraries, dependencies and other constraints to be maintained. A parameter name is case insensitive and always stored as lower case.

library	Path to the macro library directory
dependency	Macro library dependency
required	Available by default or on-demand
preload	Compile and store macro library in SAS session (reserved)
default	Macro library version is default to latest version

Table 2 Reserved package attributes

The attribute *library* defines the path to the macro library directory. More than one path can be defined and assigned in ascending order according to the specified *Sequence*.

The attribute *dependency* defines one or more a macro library dependencies. Each dependency is listed as separate record.

The attribute *required* is a feature which will allow you to specify whether a macro library is available by default or only loaded when explicitly requested using the macro %package().

The attribute *preload* is an attribute reserved for future use, which will enable a macro library to be compiled and stored within the current SAS session (WORK library SASMACR catalog). The aim is to decrease performance impact while retaining the flexibility that PCKG provides.

Beyond the reserved attribute names (Table 2), any other attribute name is considered a custom configuration attribute for the macro library and can be obtained through the utility macro %package_attribute().

The DEFAULT flag, which is also reserved as a attribute name for future use, is set for a specified package name and version that is considered the latest version, which does not necessarily have to be the last macro library version in the master list. This approach allows for very flexible scheduled, controlled or partial roll-outs into a production environment for testing, project migration and any other activities. The flag also allows for version to be a non-numeric value, include more than one period and include characters or words like 'a', 'b', 'rc1', etc. without having to resort to natural sorts and more complex sorting routines.

The DEPRECIATED flag is used to mark a package and version as depreciated and can no longer have a link assigned or loaded as the latest version when called. The latter rule is to disallow the use of the last version of a discontinued macro library.

PCKG LINKS

A link is a mechanism that allows a project to refer to a specific version of a package. A link simply consists of a project reference, the package name and version. Only packages and version that are not depreciated can have a link assigned.

NAME	Reference name of package
VERSION	Package version
XPATH	Project reference
INHERIT	Allow inheritance

Table 3 Structure of the PCKG link data set

Identical to the PCKG master list, all references to a specific package and version will use the package name and version.

A project is represented by a path reference (XPATH) that follows well-used directory structure conventions. The path reference is case insensitive with the forward slash (Unix/Linux convention) and backward slash (Microsoft Windows convention) interchangeable and Windows drive letter can be ignored to retain a sense of commonality between platforms. This allows for one single reference for different platforms.

The INHERIT flag is reserved for future functionality where package links can be inherited from a parent directory, if a link for the current path is not defined. For example, a project may include several studies that should all use a specific version. Under the current PCKG version, one link entry is required for each study.

PCKG MACROS

INSTALLING AND INITIALIZATION

The PCKG utility requires to be installed as the installation process creates the required registry data sets and initial configuration. The install process does not automatically update SAS configuration files, specifically SAS autoexec and configuration files such as SASV9.CFG as the configuration is most often dependent on organisation requirements, policies, standards and eventual change control.

The PCKG utility contains a pre-defined directory structure that allows for adaptation to specific requirements.

library	Core PCKG macros
extras	Additional utility macros and any custom macros
data	PCKG registry data sets
test	Test macros and programs
depreciated	Depreciated versions of macros that are still supported

Table 4 PCKG default directory structure

The use of the *data* directory is default but optional since user or change permission in the macro library location may be restricted for deployments.

package_install

The PCKG utility is installed with a single macro call, %package_install(), that creates the two standard data sets and adds the PCKG library path to the master list of libraries.

```
%package_install( path = /this/is/my/pckg/1.0/path,
                  data = /this/is/the/pckg/data,
                  library = static ) ;
```

The *path* macro parameter defines the location of the main PCKG macro library directory, which includes all subdirectories and files.

The *data* macro parameter specifies the location for the master and link data sets. The default location for the data sets is a subdirectory within the directory specified as the path parameter. This allows flexibility where the registry data sets are created and maintained.

The *library* macro parameter specifies if the SAS library pointing to the location of the master and link data sets should be a temporary or permanent library assignment. A temporary library (*library = temporary*) is only created, used and cleared when required by one of the PCKG macros, while a static library is assigned on the first call to a PCKG macro and will remain assigned for the duration of the entire SAS session. If a value other than *temporary* is specified for the *library* parameter, the PCKG utility will attempt to use it as the static library name. Currently, the default static library name is PCKG.

In addition, a two files are created, a README file with instructions on configuring PCKG and a test file that verifies the configuration once the manual installation and configuration is completed. The two files are created in the directory specified to retain the registry data sets as this is assumed most likely to be writeable.

After installation, all directories can be set to be read-only in order to secure the installation from inadvertent changes. In fact, only change permission is required for the registry data sets.

- Only administrators allowed to register a macro library will require write or update access to the PCKG master list data set.
- Only administrators or users allowed to create links will require write or update access to the PCKG link data set.

All other users only require read-only access.

package_init

The standard macro %package_init() is used to initialise the PCKG environment and is most often called from within other PCKG macros to ensure that the PCKG environment has been configured, initialised and a library is defined to access the PCKG registry data sets.

The PCKG utility employs configuration variables to define the location of the PCKG directories, the location of the master list and link data sets and options.

PCKG	Location of the PCKG root directory
PCKGDATA	Location of PCKG registry data sets (optional)
PCKGOPT	PCKG options (optional)

Table 5 PCKG configuration variables

The PCKG configuration variables can be implemented as operating system or SAS environmental variables as well as SAS global macro variables, depending on your environmental set-up and change control requirements. The use of SAS global macro variables is considered the least secure, as their value can easily be changed in a program.

Given the flexibility of configuration variables, the %package_init() macro searches first operating system or SAS environmental variables followed by global macro variables. A search of directories specified in the SASAUTOS system option is being investigated for a future release.

The PCKGDATA configuration variable is optional, and, if not defined, will use the *data* subdirectory in the path specified by the PCKG configuration variable.

PCKGOPT is a simple approach to define static PCKG configuration options, such as to use a static or temporary library. If PCKGOPT is not defined or empty, the initialisation macro will use the registry data set to look for system options.

MANAGE PACKAGES

All administration of the PCKG registry data sets can be performed using PCKG macros even though it may be tempting to open a data set and edit it manually or through DATA steps.

package_register

The macro %package_register() is used to register attributes for a macro library in the master list PCKGREG. The macro adds one attribute at a time to the master list registry data set.

```
%package_register( package = reporter,
                  version = 1.4,
                  library = /this/is/my/reporter/1.4/path,
                  append = N,
                  default = Y ) ;

%package_register( package = reporter,
                  version = 1.4,
                  library = /this/is/my/reporter/1.4/path/extras,
                  append = Y ) ;
```

The above macro calls will register the two specified paths as associated with the *reporter* macro library version 1.4. The first call with *append = N* will clear any library associations prior to associating the specified path. It is worth noting that this only clears the library paths associated with a package while retaining any dependencies, attributes, etc.

Note that *default = Y* is only used in the first assignment. The default specification is applied to a package and version and not the library, so default can be assigned together with libraries, dependencies, attributes or on its own.

Dependencies are added in a similar manner.

```
%package_register( package = reporter,
                  version = 1.4,
                  dependency = ( package = utilities, version = 2.8),
                  append = N ) ;
```

Append has an identical function as with libraries. If you want to add additional dependencies, use *append = Y*.

The PCKG utility uses the exact same approach to assign custom attributes.

```
%package_register( package = reporter,
                  version = 1.4,
                  attribute = ( parameter = orientation, value = landscape ) ) ;
```

Append is not valid for assigning attributes, but the concept is reserved for future implementation.

package_depreciate

The %package_depreciate() macro flags a package and version to be depreciated and can no longer have links assigned or loaded as the latest version when called, which implies that it can no longer be used.

```
%package_depreciate( package = reporter,
                   version = 1.4,
                   depreciate = Y ) ;
```

The *depreciate* parameter is to allow for the depreciated flag to be removed from a package and version. This does eliminate some hacking of the PCKGREG registry data set as that would be the other “undo” alternative.

package_link

The %package_link() macro defines a link from a project to a specific version of the package.

```
%package_link( xpath = /project/therapeutic_area/study ,
               package = reporter,
               version = 1.3 ) ;

%package_link( xpath = /project/therapeutic_area/study ,
               package = reporter,
               version = 1.3,
               drop = Y ) ;
```

The two macro calls above creates and drops a link for a specific project to the *reporter* package version 1.3. As discussed previously, the package and version cannot be depreciated for a link to be created.

USING PACKAGES

The effort to use a macro library or suite after the package and version has been registered and any required links created is quite small.

```
%package( ) ;

%package( xpath = /project/therapeutic_area/study ) ;
```

If you recall, a package and version can be registered as required, which implies that the macro library is always referred to in SASAUTOS or pre-loaded. The above two macro calls all load packages that are defined as required when a package is registered.

The second macro call specifies a value for the *xpath* parameter, which is used to look for a link pointing to a specific version. If a link is defined for the required packages, then the linked version is referred to rather than just the latest version.

The approach to load one or more specific packages is equally straightforward, using a space delimited list of package names. Note that there is no version information, which is entirely taken care of the pre-defined links.

```
%package( xpath = /project/therapeutic_area/study ,
          packages = reporter util this that other ) ;
```

The package locations are added to SASAUTOS in the order as specified in the *packages* parameter. As each package is added, any multiple reference to libraries, dependencies, etc. are resolved and added as well. Note that the role of the *xpath* parameter is identical to previous examples.

PCKG UTILITY MACROS

There are several utility macros that are provided as part of the PCKG utility.

package_link_clone	Clone all of the links for a project and assign a new project reference
package_attribute	Retrieve one or all attributes defined for a package
package_list	List available packages
package_audit_changes	Identify changes in the PCKG registry

Table 6 PCKG utility macros

The utility macros are not required for the basic PCKG functionality, but simplify or add additional features.

The macro %package_audit_changes() is a simple audit report mechanism to document changes in the PCKG registry.

```
%package_audit_changes( ) ;  
  
/* perform any changes to the PCKG registry here */  
  
%package_audit_changes( path = /this/is/my/audit/log/path ,  
                        commit = Y ) ;
```

The first call to the %package_audit_changes() macro creates a snapshot of the registry data sets, which is then used to identify and report changes in the registry data sets during the second macro call with the *commit* parameter equal to Yes.

The current version of the %package_audit_changes() macro produces a PDF report in the location specified by the *path* parameter.

THE PCKG PROJECT

The PCKG utility is an Open Source project hosted on SourceForge under an Apache License. The project currently includes source code repository, documentation, issue tracking, releases and distributions.

The project Wiki is used to document the system requirements, macros, programming conventions, utility roadmap and other references and will be extended to include a Frequently Asked Questions (FAQ) and examples as experience with the utility grows.

Distributions are periodically made available through the SourceForge project site under file downloads. A distribution is a major or minor release of the PCKG utility, which includes end-user documentation. A major release includes new features and re-designs while a minor release is used to resolve and distribute bug fixes.

Issues are tracked using the SourceForge project issue tracker.

The latest bleeding edge development macro code is available directly from the Subversion repository also hosted on SourceForge. For stable and tested code, the project refers to one of the distributions.

There are currently three distribution lists; pckg-announce, pckg-users and pckg-developers that are available for PCKG end-users or developers.

The PCKG project is available at SourceForge at <http://pckg.sourceforge.net>.

CONCLUSION

The Open Source PCKG utility is an attempt to simplify and assert greater control for a business process where different versions of the same macro suites are allowed to coexist, whether a business requirement or simply backward compatibility.

PCKG was developed to provide an easy mechanism to specify which macro libraries are used and, more specifically, what version. At the same time, we are able to document what macro libraries and associated versions are made available for a program. The entire configuration aspect allows both traceability and simple efficient impact analysis.

The PCKG utility uses a very small set of SAS macros to deploy, administer and use within an analytics environment. The Open Source license allows anyone who is interested to use, contribute, extend and evolve the PCKG utility to fit their process..

REFERENCES

- [1] Katja Glaß [2009], "User friendly management of continuously improving standard macro systems", PhUSE 2009 Paper CS01
- [2] Ronald Fehd [2005], "A SASautos Companion: Reusing Macros", SUGI 30 Paper 267-30
<http://www2.sas.com/proceedings/sugi30/267-30.pdf>

CONTACT INFORMATION

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

Magnus Mengelbier
Limelogic Ltd
London, United Kingdom

E-mail: papers@limelogic.com
Web: www.limelogic.com

More on PCKG is available at www.limelogic.com/open_source/pckg or at the project site pckg.sourceforge.net.

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.