Macros Invocation Hierarchy: Session Compiled, Autocall, and Compiled Stored Macros
Susan M. O'Connor, SAS Institute Inc., Cary, NC

ABSTRACT
This paper describes the compiled stored macro facility's functionality and syntax. A comparison of positive and negative considerations in selecting the use of session compiled, autocal, and compiled stored macros is made. Such issues as maintenance, performance, and development environments are discussed.

Recommendations for macro-related considerations, such as autocall concatenation versus permanent macro catalogs and macro option defaults, are discussed. For a greater in-depth overview, related issues of hierarchy for these types of macros, of macro variable resolution, and of the interaction of compiled stored macros and the stored program facility are discussed.

The timing and resolution of CALL EXECUTE and the RESOLVE function with compiled stored macros and session stored programs are addressed. An overview of these facilities and the order and timing of them might be helpful for users.

SESSION COMPILED MACROS
The SAS System historically has had macros which are compiled in a SAS session and can be executed many times in that same session. These macros, referred to as session compiled macros, are defined in a SAS job. A macro definition is everything contained between a %MACRO statement and a matching %MEND statement. A macro definition may contain both SAS text and SAS macro statements. The SAS macro facility compiles or translates the macro definition into intermediate code. The macro compiler has two passes; the first pass is completed when the %MEND statement is reached. The second pass is used to resolve such things as the %GOTO, %LABEL and %DO statements. The intermediate code consists of instructions such as "execute a %PUT statement" and "jump to another location to begin execution there". Intermediate code also may contain constant text to emit to the SAS System.

This intermediate code is stored in both pass one and pass two in a SAS catalog. A SAS catalog is a SAS file which is stored in a SAS data library and has the data type of CATALOG. SAS catalogs have two-level names like other SAS files; the first level name, a libref, refers to the SAS data library. In the session compiled macros, the first level name is WORK. The second-level name refers to the catalog itself, which for all macros is SASMACR. So for session compiled macros, the macro catalog is always WORK.SASMACR.

A SAS catalog contains entries which are referred to by two-level names also. The first-level name is the entry name, which for macros is the macro name from the macro definition. The second-level entry name is the entry type, MACRO. Actually, during the first pass the entry name is a temporary reserved macro name and it is at the completion of the second pass that the macro compiler renames the temporary entry name to the macro name. Of course, at the termination of the SAS session, SAS WORK files (such as data sets and catalogs) are erased from the current SAS work data library. Thus, by default, the WORK.SASMCR catalog containing the session compiled macros is deleted at the end of the SAS session.

After the macro has been compiled, whenever the macro is executed during the rest of the entire SAS session, the SAS System will locate the catalog entry for the macro, read the object record by record, and execute the interpreted code. If the macro is redefined again in the SAS session in open code, the macro object in WORK.SASMCR is deleted and the newly compiled macro is written to the catalog object again. The next time the same SAS job is executed in a different SAS session, the macro is compiled again and is written to the WORK.SASMCR catalog again as an object. This object is then read when the macro is invoked during that session.

Rather than interpreting the macro statements' source code directly each time the macro itself is invoked, the macro facility uses intermediate code because it is faster. During compilation, for example, the macro compiler figures out where to jump to if the condition of a %IF statement is false, and it codes that information in an intermediate code Instruction. It would be time consuming to figure out the macro source code each time you executed the macro. The macro interpreted code is very different from the interpreted code generated in the DATA step. In the SAS DATA step, the SAS System continues on with the machine-code generation because there are generally a lot of computations and other operations in the SAS DATA step that can be done very quickly if done directly by machine hardware. This is the part of the SAS System which can be accessed by the user with the stored program facility. The stored program facility stores and accesses machine code generated by the DATA step. In macro we stop at intermediate code rather than continuing on to machine-code manipulation because macro does very little numeric computation. While machine languages can quickly do string manipulations, macro string-handling is unusual and more complicated than most machine languages. Macro predominately does string manipulation which involves macro variable resolution and macro-quoting functions. But what gains a performance improvement in macro is eliminating the overhead of macro compilation each time a macro is executed.

To summarize then, for session compiled macros, each macro is compiled and written to a temporary catalog object during the first pass. The temporary object in the WORK.SASMCR catalog is closed and then reopened. During the processing of the second pass, the intermediate code in the temporary object is read, processed, and then written to a new object with the same macro name. So, there is a write-read-write scenario during the compilation phase before the macro itself is executed. During invocation of the macro, the catalog object is opened, and the intermediate instructions are read and executed. These instructions generate SAS session code which is passed to waiting parts of the SAS session, such as a procedure, DATA step, DMS, or SCL.

AUTOCALL MACRO FACILITY
If you have a macro which you like to use in several SAS jobs, you could always put the macro source definition code for that macro in an external flat file. This file could be shared by many SAS sessions, SAS jobs, and SAS users. One approach is to include a flat external file, using the %INCLUDE statement, containing a macro definition whenever you needed it in a SAS session. This way, you would always have access to the macro definition could be shared and easily maintained. However, you would have the overhead of reading an additional flat file from a disk as well as compiling the macro and storing it in a temporary SAS catalog, which will be deleted at the end of a SAS session. Another approach is to put the macro in an autoexec file and the macro would be compiled at
head of slowing down your session start unnecessarily if you did not go to invoke the macro in the current session.

The SAS autocall facility is another way of keeping macros as external files which can be accessed by the SAS System. The macros are stored in aggregate storage locations or autocall libraries: a partitioned data set under MVS, a maccall under CMS, a directory or VMS text library under VMS, a library under VSE, and a directory on most directory-based machines. The autocall facility is especially useful when you have a large number of macros you might need to execute, but you want to avoid having to compile all of them at the start of each SAS session.

The autocall facility allows you to store macros as members of an autocall library. The macro must have the same name as the member of an autocall library. (On some operating systems this may require a .SAS type extension.)

An autocall library is a storage location referenced by

\begin{verbatim}
OPTION SASAUTOS=filename | physical_name | (filename | physical_name | ...); 
\end{verbatim}

\begin{verbatim}
OPTION SASAUTOS= may reference one autocall library or a concatenated list of autocall libraries.
\end{verbatim}

To invoke an autocall macro, OPTION MAUTOSOURCE must be in effect. When you invoke a macro that has not been defined in the current session, the autocall facility searches the autocall libraries for a member with the same name as the macro being invoked. After finding a member in the autocall library, all the code in that file is included. The macro definition with the same name is compiled and any other contained macro definitions are compiled. Then the macro being invoked is executed automatically.

By default, OPTION NONRECALL causes the autocall facility to search only once for an undefined autocall macro.

If you have a long list of concatenated autocall libraries, the SAS System will attempt to open the autocall libraries and then search them in order, looking for a file with the same name as the macro or, on some systems, for the same name with a .SAS extension. For example, in this hypothetical operating system, if you had:

\begin{verbatim}
filename a 'physical-name-of-directory';
filename b 'directory-across-inaccessible-network';
filename c 'physical-name-of-mds';
filename d 'physical-name-of-macllib';
options nonrecall (a b c d);
\end{verbatim}

Here, A, B, C, and D are filerefs (or physical names in quotes) to aggregate storage locations. If B could not be opened, but at least one aggregate location did open, no message would be issued. However, the SAS System would try to open B for each attempted autocall macro invoked. The overhead of attempting to open a non-existing autocall library could be unnecessary overhead in a SAS session with a large concatenated list of autocall libraries.

An error and warning is only issued if no autocall library at all will open. This decision was made for such operating systems as OS/2 where the concatenated directories may be unavailable to be opened on the first attempt but available on subsequent attempts.

\begin{verbatim}
ERROR: No file referenced by SASAUTOS OPTION can be opened.
WARNING: Source level autocall is not found or cannot be opened. Autocall has been suspended and OPTION MAUTOSOURCE has been set. To use the autocall facility again, set autocall facility again, set OPTION MAUTOSOURCE.
\end{verbatim}

The advantages of autocall macros are that they can be easily maintained in associated aggregate storage locations, accessed by differing applications and users, used to develop complicated macro systems, and concatenated between levels of macros from production to development.

But there are disadvantages also. When you execute an autocall macro, it is essentially including the flat file and executing anything in that file that could be open code as well as compiling the macro definition and executing the macro. You can have several macro definitions in the autocall member, and all macros will be compiled at once. But if you are only after the macro of the same name as the member, you have the overhead of compiling a lot of unused macros. Therefore, when using autocall macros, an important efficiency consideration is the organization of the autocall libraries and the contents of the autocall members. For the convenience of the autocall facility, you also have the risk of attempting to open unavailable autocall macro libraries with no warning or error.

**COMPILED STORED MACRO FACILITY**

The compiled stored macro facility allows you to compile and store SAS macros as objects in a permanent SASMACR catalog. Later, you can access the compiled stored macro code when you execute the macro in a different SAS program. This facility will save macro-compilation overhead in any SAS program which accesses the compiled stored macro.

When using the compiled stored macro facility, the macro code is compiled and then stored in a permanent catalog named SASMACR in an assigned SAS data library. This compilation will occur only once. The overhead of the attendant write-read-write of the two passes for the macro compilation will be gone. Later, when a different SAS session invokes that macro, the overhead of compilation will not be included in resource usage.

In order to use the compiled stored macro facility, macro code should be production macros, which will be accessed by user programs to eliminate macro compile-time overhead before macro invocation. The macro source code itself should be well-documented and saved, in addition to saving the compiled stored macro. This will be essential for maintenance.

**COMPILED STORED MACRO SYNTAX**

There are two SAS system options which must be set to use compiled stored macro:

\begin{verbatim}
MSTORED allows the SAS System to use the compiled stored macro facility. The default is NOSTORED.
\end{verbatim}

\begin{verbatim}
SASMSTORE= specifies libref for SAS data library containing the SAS compiled stored macro catalog SASMACR. The default is blank. This libref should be a valid libref.
\end{verbatim}

There are also two new options in the %MACRO statement that relate to storing compiled stored macros. The expanded %MACRO statement options include:

\begin{verbatim}
%MACRO name <parameters> <COMMANDS> <PARMBUFF | PARM> <STORE> <DESCRIPTION> <text-string> >;
\end{verbatim}

For stored compiled macros, the STORE option specifies that the compiled macro will be stored as an entry in a SAS macro catalog in a permanent SAS data library. The MSTORED system option must be in effect to use compiled-stored invocations.

You are not allowed to use the libref WORK for compiled stored macros. A dummy macro will be compiled if you use OPTIONS
SASMSTORE=WORK. A dummy macro is a macro that the macro processor compiles but does not store. It is recommended that compiled stored macros be name-style macros for the greatest performance gains. Both statement-style and command-style macros are a bit less efficient than name-style macros, and the purpose of using compiled stored macros is to increase performance by eliminating the overhead of macro compilations. When using compiled stored macros, as when using autocall macros, you should use %LOCAL for macro variables, which will be used only inside the compiled stored macro. Otherwise, a macro variable value in a different environment might be altered.

It will be essential that programmers using compiled stored macros maintain a copy of the macro source code used to compile and store the macro. One suggestion would be to maintain a copy of the macro source code in an autocall library.

The SAS System does not support the CPORT and IMPORT procedures for macro catalogs because of the complicity of delta characters and upcoming KANJI development on your operating system. However, you could use the following to obtain the contents of a macro catalog containing compiled stored macros:

```
proc catalog libref.aassmacr <entrytype>='macro';
   contents;
   run;
```

You will get the following headings:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
</table>

When OPTION MLOGIC is on to trace the execution of a macro for compiled stored macros, after the beginning execution message you will receive the following message indicating the libref and date of compilation:

```
MLOGIC(ONE): Beginning execution.
MLOGIC(ONE): Compiled stored macro in libref THERE compiled on 08/01/xx.
```

Also another new %MACRO statement option, the DES= option, is available to keep track of a macro description. The syntax for the DES= option is

```
DES="text-string"
```

This option specifies a description or title for the macro entry in the macro catalog. The text string is limited to a length of 40 characters inside a quoted string. Strings longer than 40 characters will be truncated to 40 characters. The macro catalog-directory field will contain the description you provided with this %MACRO statement option for the macro entry. When you are in the DIRECTORY window, this description will appear when displaying the contents of the catalog containing the compiled stored macros. Since the WORK macro catalog is deleted for session compiled macros, there is no purpose for adding descriptions unless you are using the compiled stored macro facility.

**COMPILED STORED MACRO CONSIDERATIONS**

Compiled stored macros have been developed with the idea of performance, which means that design considerations were made for efficiency. For Release 6.07 of the SAS System, there is no concatenation of compiled stored macro DATA libraries. Compiled stored macros do not apply themselves to a macro development environment. The burden of maintaining the macro source code rests with the macro application developer. The macro performance gain is noticed especially for large macros, which would take a long time to compile. Since a macro is compiled once during a session, the overhead is only noticed the first time a compiled stored macro is executed and not for subsequent executions. Since autocall macros may contain several related macros, the additional overhead of compiling several macros at once is eliminated.

**WHICH COMES FIRST: SESSION COMPILED, COMPILED STORED, OR AUTOCALL MACROS**

In the hierarchy of macro execution, assuming that all appropriate macro options have been set for the SAS session when a macro is invoked, the SAS System first looks for a session compiled macro, then for a compiled stored macro, and finally, for autocall macros. Of course, it would be less efficient to have this type of situation where every type of macro option is set and the search for the macro invoked is for all types of macros. If a macro is used only once by one user in one program, it might be just as well to have it session compiled. However, if it is a large macro, significant compilation time can be saved if it is compiled and stored. This compilation time would be reflected in CPU time, I/O, and elapsed time. Perhaps if the major consideration is for ease of maintenance and for organizing a lot of macros, the autocall facility is the best choice.

**THE STORED PROGRAM FACILITY AND THE COMPILED STORED FACILITY**

There are considerable differences between the compiled stored macro facility and the stored program facility. The stored program facility is for DATA step applications only, and only if those applications do not contain global statements (such as FILENAME, LIBNAME, FOOTNOTE, OPTION, and TITLE statements) or host-specific options for the data set (such as FILE or INFILE statements). Since the compiled stored macro facility is a text-generating facility, it has no program boundaries but can be used to generate SAS source code to execute across DATA steps, procedures, global SAS statements, and host interfaces. Both facilities store their code as intermediate code, but when the stored program facility executes, its unique intermediate code generates the executable machine code for the host environment. The macro intermediate code does not use a code generator to execute the macro statements. The compiled stored programs appear in a SAS data library with the member type of PROGRAM. The compiled macros are stored in SAS catalogs in a SAS data library. You cannot move stored programs to another host with incompatible machine architecture. You must compile and store macros on each host by porting the macro source code and compiling it on the target host.

Release 6.07 contains the CALL EXECUTE routine and the RESOLVE function. Both of these DATA step interface routines, as well as SYMGET and CALL SYMPUT, are routines which are called during execution of the DATA step. As with other DATA step functions, these macro interface routines are called when the SAS DATA step is executing. In the compiled stored macro code, they are just stored as part of constant text.

**ONE CASE STUDY**

At SAS Institute, the Quality Assurance department (QA) uses macros heavily to establish test environments across our operating systems and to have an organized macro system set up to test SAS code in all environments. This system has been in existence for several years. Originally, these start-up macros were in SAS autoexec files, which were executed every time a test was run. There was always a danger that a job might be using a different autoexec file in the user's environment which might override the official one being used.
Then QA began to use the autocall facility. This had the advantage of referencing a concatenated list of autocall libraries, which was established with the SASAUTOS= option at start-up with an autoexec or a SAS configuration file. These macros assured the testing level, the host machine, and autocall macro libraries. Any number of users could simultaneously use the same macros from one location. For efficiency, the autocall macros often contained several related macros, all of which would be compiled in one autocall macro when the invoked autocall macro was compiled and executed. The autocall macros could be shared and maintained in shared locations and ported to all hosts.

This year, with Release 6.07, these start-up macros were converted to compiled stored macros. The actual editing of the 14 flat autocall macro files and the contained 3 or 4 inner macros along with the port to three hosts took a half day of one programmer. These were established production macros which were rather large, but very well used and tested over the prior two-year period. The range in performance gain between using the autocall facility and the compiled stored facility for this test case was 10%-35%, depending on the host. The biggest difference was PC-related. However, significant differences are reported under MVS, CMS, VM, and UNIX systems for this conversion to compiled stored macros. Losing source is not a problem at SAS Institute because of rigorous source management for the quality assurance test environment. This was an optimal conversion because the autocall macros had been production macros for over two years. Each macro was executed only once, and indeed, it is the first execution in which you would see the gain. The programmer doing the conversion reported ease of conversion, and he felt that the lack of concatenation for the compiled stored macro was not a problem for this application.

But this was an optimal example. The macros were large and had been autocall macros for several years. They were all start-up macros for a system and were used just once, which would be optimal in the performance statistics. Misuse varied in the performance statistics. This difference reflects file access on the individual operating system and the SAS System defaults on the operating system. They even reflect whether all of the autocall libraries were successfully opened in the concatenated autocall list.

RECOMMENDATIONS

It might be worth investigating the time investigating whether a macro system used in your environment would be worth converting to compiled stored macros. A conversion would not show optimal results if the macros are small and used a lot. This compiled stored macro facility is not designed for security. It would not be recommended, either, if the macro source code is ever in danger of being lost in your environment because debugging a compiled stored macro would be impossible without the macro source code.

Catalogs are made up of a variety of components stored in logical units called pages. The host layer of each operating system maps units into physical units, blocks, or on the disks. The physical transfer of data between disk and memory during a catalog read or write transfer of data can be a page, a section of pages, or a block of pages. Each host determines the optimal page size for a catalog, based on the characteristics of the mass storage on which the catalog will reside. The default SAS library block size and page size for catalogs will vary from host to host. For catalog writes and reads performance is system-dependent. However, other conversion tests at the Institute by core testers indicate performance differences of 10%-45% using compiled stored macros versus autocall macros. These differences were reflected in CPU time, elapsed time, and I/O, and they are attributed to many factors. Measurement of performance gains are also difficult.

OTHER NEW MACRO OPTIONS

Now to the SAS System in Release 6.07 are two options related to macro variables which can affect system performance. The MSYMTABMAX= system option specifies the maximum amount of memory available to the macro symbol table. The MVARszIZE= system option specifies the maximum size for in-memory macro variables. In both cases, once this value is reached, additional macro variables are written out to disk. The value specified can range from 0 to the largest nonnegative integer representable on your operating system. The default values for Release 6.07 are identical to the values used in Release 6.06 on your operating system. Since these options can affect SAS System performance, before you alter the defaults for production jobs, you should run tests to determine the optimum value. In general, the larger the amount of memory on a machine, the higher these values can be. For hosts such as the PC with little available memory, leave the defaults as they are. Adjusting the MSYMTABMAX and MVARszIZE values optimally could lead to a reduction of CPU and less I/O.

MACRO INVOCATION AND PERFORMANCE

Three SAS macro invocation types are now available in Release 6.07: name-style, statement-style, and command-style. Name-style invocation is the traditional way to call a macro to execute using a percent sign and the macro name. It is always the most efficient choice for performance of the macro.

Statement-style invocation allows you to call a macro which looks like SAS statements. When the macro is defined with the STMT option in the %MACRO statement and when the IMPLMAC system option is in effect, the first word of any SAS statement being processed by the tokenizer is considered a possible macro invocation. There is a search for the session compiled macro, which will take place always for start of statement tokens. This search would be less complex if only session compiled macros were available, increasing in depth as compiled stored and then autocall macros, were also available in the session.

Command-style invocation is now available compatibility with Version 5 of the SAS System. When a macro is defined with the CMD option in the %MACRO statement and when the CMDMAC system option is set, command-style macros are invoked as if they were display manager or full-screen procedure commands on the command line. If a session compiled macro is command-style or if compiled stored macros are available or if autocall macros are available, at each command-line command, a search will be made in respective locations for a command-style macro.

While statement-style and command-style macros may enhance an application, they will also impact performance slightly. If optimal performance is required of an application name-style invocation might be the first consideration.

CONCLUSION

There are many choices available in Release 6.07 for macro users which will affect the performance and ease of maintenance of macro applications. Macro developers may consider session compiled macros, the autocall macro facility, and compiled stored macros for their applications. Each choice has advantages. Performance improvements with compiled stored macros should be reflected in CPU time, I/O, and elapsed time, but results will vary with operating systems. The new SAS system options MSYMTABMAX= and MVARszIZE= may improve CPU and I/O performance. As with Ver-
Section 5 of the SAS System, name-style invocations are more efficient than statement-style or command-style invocations. In an optimal situation, macro application developers would run tests to determine the best choices for their environment.

REFERENCES


ACKNOWLEDGEMENTS

I would like to thank my managers in the Core Shell Development Group, Jeff Potzin and Pat Herbert, for their support in the preparation of this paper. I would like to thank my colleague Bruce Tindall for his seminars and notes which led to issues addressed in this paper. I would especially like to thank Olta Coleman in the Core Testing Group for her performance statistics and Richard Ragland and David Home, both of Quality Assurance, for information and discussions about their conversion experiences with compiled stored macros for their start-up testing environment.

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. OS/2 is a registered trademark of International Business Machines Corporation.

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