Advanced Macro Programming
Lynn Hannah, SAS Institute Inc.

Introduction
This tutorial was presented in order to address the most common problems encountered by macro programmers. The major concepts covered were writing efficient programs and introducing Version 5 features. This paper will cover the key points addressed during the tutorial.

Writing Efficient Macro Programs
The key to writing more efficient programs comes primarily from having a thorough understanding of the programming techniques you are using. Most macro programmers have problems in the same areas. The problem areas we will cover include quoting functions, evaluations, and unquoting. A major emphasis is placed on quoting functions because they tend to make up the common problem area.

Macro Quoting Functions
Quoting functions are an essential part of the macro facility. The SAS® language uses single and double quotes to literally assign text. The macro facility, because it constructs SAS code, also needs a way of literally assigning text and quoting text after resolution.

The importance of quoting can be seen in the SAS language by the simple example

```
Data Test;
X=1+2;
Y='l+2';
Put X= Y=;
```

In the macro language, the same type of situation occurs.

Once the need for quoting has been recognized the second question raised by macro users is, 'Why are there so many?'. It began with one quoting function, %STR, the others were added as their needs arose. The following diagram illustrates the quoting functions that are present in Version 5, in their relative order.

```
<table>
<thead>
<tr>
<th>Macro Quoting Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>%STR</td>
</tr>
<tr>
<td>%SUPERQ</td>
</tr>
</tbody>
</table>
```

%STR removes the significance of all special characters, excluding % and & at compile time. Use this function when you can see the special characters you wish to quote, but you do not want to remove the significance of % and &. %STR removes significance from the special characters that are present at compile time, (the ones you see), it does not affect special characters that occur after a macro variable has been resolved, or a macro has been invoked.

```
%let Checkdat=%Str(Proc Print; Proc Contents);:
```

Using %Str on single and double quotes and unmatched parentheses

```
%let Title=%Str(Sally%39s Garden);:
```

%NRSTR
%NRSTR removes the significance of all special characters, including % and &, at compile time. Use this function when you can see the special characters you wish to quote and you also want to remove the significance of % and &. The only difference in this function and %STR is that %NRSTR also removes significance from & and %.

```
%let Dealer=%Nstr(M&P Auto Sales);
```

%QUOTE
%QUOTE removes the significance of all special characters, excluding % and & at execution time. Use this function when you cannot see the special characters you wish to quote, and you do not want to remove the significance of % and &. %QUOTE removes significance from the special characters that occur in the resolved value of a macro variable, or the result of a macro invocation. It should not be used to quote literal strings, that is what %STR is for. %QUOTE will most always only be used on macro variable references.

%QUOTE is usually needed when the value of a macro variable has not been literally assigned. For instance when a %INPUT statement is used or when CALL SYMPUT is used to create a value from a data step variable. For literal assignments, using %LET to assign a macro variable its value, the value can be quoted using %STR.

```
%Macro Subset;
%if %Quote(&State)=NC %Then %Put GREAT BASKETBALL!!;
%End Subset;
```
%NRQUOTE

%NRQUOTE removes the significance of all special characters, including % and &, at execution time. Use this function when you cannot see the special characters you wish to quote, and you also want to remove the significance of % and &. %QUOTE removes significance from the special characters that occur in the completely resolved value of the macro variable. If the value of the macro variable contains a % or & followed by a non-blank character, we will attempt to resolve the macro variable or invoke the macro. If this is successful, the %value is replaced. If not, a warning message is issued. Once the attempt at resolution has been made, the unresolved & is quoted, and resolution will not be attempted again.

%Let Cust=%nrquote(&Response);
%Put &Cust;

J&K Junk Collectors Inc.

%QUOTE

%QUOTE removes the significance from unanticipated special characters, excluding % and & in the resolved argument. It does not remove the significance from mnemonic operators. Unanticipated special characters (unmatched parentheses and quotes), could not be handled before this function was designed. Unanticipated special characters occur from %INPUT and CALL SYMPUT, cases where the programmer had no control over what the value would be.

Before %QUOTE, the only way a user could enter a value containing an unmatched quote, without causing problems for the programmer, would be if the user entered

%STR(Bailey%'s)

%NRQUOTE

%NRQUOTE removes the significance from unanticipated special characters, including % and &. It does not remove the significance from mnemonic operators. %NRQUOTE removes significance from these special characters in the completely resolved argument. If the value of the macro variable contains % or &, we will attempt to invoke the macro or resolve the macro variable before quoting the special characters.

%Macro Checkit;
%Put What are you looking for?;
%Input;
%Do i=1 to 5;
%If %Index(&&Mac&I,%NRQUOTE(&Sysbuffr)) > 0
%Then
%Put %nrquote(&Sysbuffr) found in &&Mac&I;
%End;
%Mend Checkit;

%SUPERQ

%SUPERQ removes the significance from all unanticipated special characters at execution time. It does not remove the significance from mnemonic operators. The argument to this function is the name of a macro variable, without the &. %SUPERQ uses an external module to bring in the value of the macro variable, it performs no resolution. This means that if the value contains % or & these characters are quoted immediately, no attempt at resolution is made.

%Macro Helpme;
%Put What problems are you having?;
%Input;
%Let Problem=%Superq(Sysbuffr); %Put %str(Don't call us, we'll call you); %Mend Helpme;

Unquoting

Unquoting, restoring significance back to special characters, is done automatically by the SAS wordscanner when receiving a token from the macro facility. It can also be done explicitly in the macro facility by using the %UNQUOTE function.

Using %UNQUOTE to restore the significance to special characters

%Macro Test;
%Put Enter your expression;
%Input;
%Let Resp=%Quote(&Sysbuffr);
%If Resp NE %Then
%Put &Resp=%Eval(%Unquote(&Resp)); %Mend;
%Test

A special case of using %UNQUOTE, also occurs when significance has been removed from a quoted string. For instance, if %STR has been used to remove the significance from the quotes, their significance is automatically restored when they are passed back to the wordscanner. Because they are seen individually, the expression is not seen as a single string, it is seen as multiple tokens. The %UNQUOTE function allows significance to be restored before the string is passed to the wordscanner, therefore it comes into the wordscanner as a single token.

The following example illustrates this concept. Note that in Version 5 double quotes could have been used instead of the single quotes, and no functions would be necessary.

Data;
Data=%Unquote(%str(%'&SYSdate%'D»;)

Efficient Techniques

Hopefully at this point, you have a better understanding of when quoting functions, and %UNQUOTE are necessary. These are the ONLY times when they should be used. Many users
find it 'SAFE' to quote all assignments and expressions. This serves no purpose, makes
the output harder to understand, sets a bad example for others, and is inefficient.

Other than quoting functions, another function that is more often misused and
overused, is %EVAL. This function should only be used when an explicit evaluation is
necessary. How do you avoid overusing it? You should know the places where an implicit
evaluation, using the same routine, is done for you. These places are:

* In numeric arguments to macro functions
* On the left and right side of the operator
  on a %IF statement.
* In the operands of the %DO statement

Use %EVAL when you need an evaluation that would not automatically be done.

Combining functions

Because the %QUOTE, %BNRQUOTE, and %SUPERQ functions do not quote mnemonics, it
is often necessary to nest more than one function. Be sure that you do this only when
one function does not treat all the characters you need to handle.

This macro shows an example of writing a macro that looks like a function to quote all
characters.

```sas
%Macro Shield (Var);
%Quote(%Superq(Var))
%Mend;
```

Defining Macros within Macros

The inner macro is recompiled over time the outer macro is executed. In most all cases,
this is unnecessary and inefficient. Invoking a macro within another macro, is extremely useful
and efficient. The environment structure will be the same as though the macro was actually
defined and invoked inside.

There are actually some instances when defining a macro inside another is essential, it
is fine in these cases. If it is necessary, then you actually want the macro to be recompiled
based on what is happening in the outer macro.

New Features for Version 5

The Autocall Facility

The autocall facility allows the automatic compilation of macros on a demand basis. Using
autocall allows you to set up macro libraries for different groups and levels of users. It
requires the compilation of only macros that are actually used, this is especially useful for menu
driven systems. Autocall is more efficient than using the %INCLUDE statement. Users may not
even know they are working with macros.

Store the source code for your macro
definitions in PDS members, or in maclibs under
CMS, using the same name for the member and
the macro. Associate the library with the
library name defined by the SASAUTOS= system
option. Be sure the MAUTOSOURCE option is
in effect. Invoke the SAS system and use your
macro invocations. The macros will
automatically be compiled and then executed.

CALL EXECUTE

Resolves an argument representing a macro
call, or SAS statements during data step
execution. The argument may be

* A character string
* The name of a data step character variable
* An expression

RESOLVE

Resolves the resolved value of the argument
after the argument has been processed by the
macro facility. The argument may be

* The name of a data step variable
  whose value is a macro call, or
  macro variable reference
* A macro call, or macro variable
  reference enclosed in quotes
* An expression producing a macro
call, or macro variable
  reference

New Macro Variables

SYSJOBID

SYSJOBID returns the name of the currently
executing batch job, or the current userid.

SYSPBUFF

SYSPBUFF receives all text supplied as
macro parameter values, including parentheses
and comma delimiters.

The PARMBUFF or PBUFF option must be
used on the %MACRO statement or order to use
this variable.

Conclusion

The intent of this paper was to explain the
categories behind the problem areas and to
introduce the new features in Version 5. For
documentation, refer to the following
publications

The SAS USER'S GUIDE: BASICS,
Version 5 Edition

Technical Report P:153,
Enhancements to the Macro Facility