

In this example if we were connecting from WINDOWS to UNIX this %IF condition would be false, therefore nothing would be sent to the remote host. This is where it helps to know and understand the difference between what is macro generated text and what is compiled code. Since compiled code is not remote submitted inside a macro generated RSUBMIT the %IF is executed on the local host. With the %IF being false the 3 statements following it are never generated by the macro therefore leaving nothing to remote submit.

The good news is we have two great macro options for determining text and compiled code. These are MLOGIC and MPRINT.

MLOGIC will help you determine, instructional code, which will be what executes on the local side. MLOGIC specifies whether the macro processor prints a message whenever the SAS System executes any macro instructional code within a macro. Any statements produced by MLOGIC will happen on the local side and everything else will be executed on the remote host.

MPRINT will help you determine, text, which will be what executes on the remote side. MPRINT displays SAS statements generated by macro execution. Any statements produced by MPRINT that appear between the RSUBMIT/ENDRSUBMIT block will happen on the remote host and everything else will be executed on the local host.

The example below illustrates how useful MLOGIC and MPRINT can be:

```
options mlogic mprint;
%macro test;
  rsubmit;
  data one;
    x=100;
  run;
  %let y=200;
  %put &y;
endrsubmit;
%mend;
%test

/** Results from code above **/
NOTE: Remote signon to HOST complete.
139
140 options mlogic mprint;
141 %macro test;
142   rsubmit;
143   data one;
144     x=100;
145   run;
146   %let y=200;
147   %put &y;
148   endrsubmit;
149 %mend;
150 %test
MLOGIC(TEST): Beginning execution.
MPRINT(TEST): rsubmit
NOTE: Remote submit to HOST commencing.
MPRINT(TEST): ; data one;
MPRINT(TEST): x=100;
```

```

MPRINT(TEST):    run;
MLOGIC(TEST):   %LET (variable name is Y)
MLOGIC(TEST):   %PUT &y
200
1      data one;
2      x=100;
3      run;

```

NOTE: The data set WORK.ONE has 1 observations and 1 variables.
NOTE: DATA statement used:

real time	0.23 seconds
cpu time	0.02 seconds

NOTE: Remote submit to HOST complete.
MPRINT(TEST): endrsubmit;
MLOGIC(TEST): Ending execution.

Notice that MPRINT shows the text which will be pushed to the remote host, it consists of the data step. MLOGIC shows the compiled statements which is what remains on the local host and this consists of the %LET as well as the %PUT.

There is a way to hide some of the macro statements so that they are executed on the remote host. You can do this by using the %NRSTR function.

%NRSTR prevents the macro processor from interpreting a macro statement that is within the RSUBMIT statement in the local session. Instead, the macro statement is interpreted and used in the remote session.
For example: %nrstr(%put abc=&abc one=&one time=&time;)

This example illustrates what would happen without %NRSTR:

```

%macro test;
%put &sysscp;
  rsubmit;
  %let x=100;
  data new;
  put "&x";
  run;
%put &sysscp;
endrsubmit;
%mend test;
%test

```

/** Results from code above **/

```

MLOGIC(TEST):   Beginning execution.
MLOGIC(TEST):   %PUT &sysscp
WIN
MPRINT(TEST):   rsubmit
NOTE: Remote submit to HOST commencing.
MLOGIC(TEST):   %LET (variable name is X)
MPRINT(TEST):   ; data new;
MPRINT(TEST):   put "&x";
MPRINT(TEST):   run;
MLOGIC(TEST):   %PUT &sysscp
WIN
16      data new;
17      put "&x";

```

WARNING: Apparent symbolic reference X not resolved.

```
18      run;
```

```
&x
```

NOTE: The data set WORK.NEW has 1 observations and 0 variables.

NOTE: DATA statement used:

```
      real time          0.02 seconds
      cpu time           0.00 seconds
```

NOTE: Remote submit to HOST complete.

```
MPRINT(TEST):  endrsubmit;
```

```
MLOGIC(TEST):  Ending execution.
```

Let's say this code was submitted on a WINDOWS platform and we have a connection established to an HP platform. The first %PUT executes on the local host therefore printing WIN in the log. THE RSUBMIT takes place, but two of the items within the macro generated RSUBMIT are executed on the local host which are the %LET and %PUT. The data step is pushed to the REMOTE host and executed which generates a warning because the %LET that created the macro variable X executed on the local host not the remote host. Let's look at the same example, but this time use %NRSTR to assist us in what gets pushed to the REMOTE host.

This example illustrates what would happen with %NRSTR:

```
%macro test;
%put &sysscp;
  rsubmit;
  %put &sysscp;
  %nrstr(%let x=100;)
  data new;
  put "&x";
  run;
%nrstr(%put &sysscp);
  endrsubmit;
%mend test;
%test
```

```
/** Results from code above **/
```

```
MLOGIC(TEST):  Beginning execution.
```

```
MLOGIC(TEST):  %PUT &sysscp
```

```
WIN
```

```
MPRINT(TEST):  rsubmit
```

NOTE: Remote submit to HOST commencing.

```
MLOGIC(TEST):  %PUT &sysscp
```

```
WIN
```

```
31      %let x=100;
32      ;
33      data new;
34      put "&x";
35      run;
```

```
100
```

NOTE: The data set WORK.NEW has 1 observations and 0 variables.

NOTE: DATA statement used:

```
      real time          0.02 seconds
      cpu time           0.01 seconds
```

```

36      %put &sysscp;
HP 800
NOTE: Remote submit to HOST complete.
MPRINT(TEST):  ; %let x=100;
MPRINT(TEST):  data new;
MPRINT(TEST):  put "&x";
MPRINT(TEST):  run;
MPRINT(TEST):  %put &sysscp;
MPRINT(TEST):  endrsubmit;
MLOGIC(TEST):  Ending execution.

```

Just as before we will assume this code was submitted on a WINDOWS platform and a connection has been established to an HP platform. The first %PUT (just as before) executes on the local host therefore printing WIN in the log. THE RSUBMIT takes place, but this time everything is executed on the remote host as shown with the MPRINT. Now when the data step is executed on the remote host, x resolves without warning because this time the %NRSTR hid the %LET at compilation and treated it as text so that it was pushed and executed on the remote host. Also notice that the %PUT that was enclosed within the %NRSTR function prints out HP 800 since this time it is executing on the remote host because it was hid during compilation on the local host.

Another problem run into when using SAS/CONNECT and macro is using macro variables. Many times the macro variable will be created on the local host and resolution tries to take place on the remote host or vice versa. We have two statements that can help us get around this; %SYSLPUT and %SYSRPUT. Let's start off by looking at %SYSLPUT.

%SYSLPUT creates a new macro variable or modifies the value of an existing macro variable on a remote host or server. The syntax for %SYSLPUT does vary across releases of SAS.

In Release 6 and 7 %SYSLPUT is a SAS sample program with the syntax of:

```
%SYSLPUT(macro-variable,value,remote=);
```

In Release 8 %SYSLPUT is a macro statement with the syntax of:

```
%SYSLPUT macro-variable=value;
```

In Release 8 there is also a SAS sample program called %LPUT with the syntax of:

```
%LPUT(macro-variable,value,remote=);
```

In Release 9 %SYSLPUT is a macro statement that contains a new option with the syntax of:

```
%SYSLPUT macro-variable=value </remote=server-id>;
```

macro-variable is either the name of a macro variable or a macro expression that produces a macro variable name. The name can refer to a new or existing macro variable on a remote host or server.

value is a string or a macro expression that yields a string. Omitting the value produces a null (0 characters). Leading and trailing blanks are ignored; to make them significant, enclose the value in the %STR function.

To use %SYSLPUT, you must have initiated a link between a local SAS session or client and a remote SAS session or server using the SIGNON command or SIGNON statement.

This is an example in Release 8 using %SYSLPUT to create a macro variable called dir1 on the remote host:

```
%macro test;
  %let dir1=/dept/test;
  %syslput dir1=&dir1;
  rsubmit;
    proc upload infile= eng101
      outfile="&dir1/eng101";
    run;
  endrsubmit;
%mend test;
%test
```

The new option for %SYSLPUT in Release 9 allows you to specify the name of the session that the macro variable will be created in.

If only one session is active, the server-ID can be omitted. For multiple server sessions that are active, omitting this option causes the macro to be created in the most recently accessed server session.

You can find out which server session is current by examining the value assigned to the CONNECTREMOTE system option.

The /REMOTE= option that is specified with the %SYSLPUT macro statement overrides the CONNECTREMOTE= global option.

Due to the addition of the /REMOTE option in the %SYSLPUT statement any value that contains forward slashes should now be quoted with a macro quoting function.

The following example uses the %BQUOTE function to mask forward slashes that are used in a UNIX pathname that is assigned in the %SYSLPUT statement.

```
%let path=/testa/testb;
%syslput path=%bquote(&path);
rsubmit;
%put &path;
endrsubmit;
```

```
/** Results from code above **/
```

```
NOTE: Remote submit to HOST complete.
```

```
917 %let path=/testa/testb;
```

```
918 %syslput path=%bquote(&path);
```

```
919 rsubmit;
```

```
NOTE: Remote submit to HOST commencing.
```

```
5 %put &path;
```

```
/testa/testb
```

```
NOTE: Remote submit to HOST complete.
```

This next example illustrates what will happen if the macro variable begins with a forward slash and a macro quoting function is not used:

```
%let path=/testa/testb;
%syslput path =&path;
  rsubmit;
  %put &path;
endrsubmit;

/** Results from code above **/
NOTE: Remote submit to HOST complete.
```

```
8   %let path=/testa/testb;
9   %syslput path=&path;
ERROR: Unrecognized option to the %SYSLPUT statement.
NOTE: Line generated by the macro variable "PATH".
1   /testa/testb
      -
      180
ERROR 180-322: Statement is not valid or it is used out of
              proper order.
```

```
10  rsubmit;
NOTE: Remote submit to HOST commencing.
2   %put &path;
/testa/testb
NOTE: Remote submit to HOST complete.
```

The error is generated because once &path resolves the first thing we see is the forward slash so we assume the option REMOTE= is coming up next. Since this is not the case an error occurs. This is not an issue in releases prior to Release 9, because the option did not exist.

With so many different versions of %SYSLPUT how do we know which one to use?

local host	remote host	
-----	-----	
Release 6 & 7 -->	Release 8	use %SYSLPUT sample program
Release 8 -->	Release 8 and beyond	use %SYSLPUT macro statement
Release 8 -->	Release 6 & 7	use %LPUT sample program

Note: %LPUT can be used going from V8 to V8 if the REMOTE= option is needed.

Now, let's look at the opposite of %SYSLPUT in the macro statement %SYSRPUT.

%SYSRPUT assigns the value of a macro variable on a remote host to a macro variable on the local host.

The only syntax for %SYSRPUT is:
%SYSRPUT local-macro-variable=value;

local-macro-variable specifies the name of a macro variable on the local host.

value is a macro variable reference or a character string on the remote host that will be assigned to the local-macro-variable.

```
/** Example of using %SYSRPUT **/  
rsubmit;  
  %macro download;  
    proc download data=remote.mydata out=local.mydata;  
      run;  
      %sysrput retcode=&sysinfo;  
    %mend download;  
  %download  
endrsubmit;  
  
%macro checkit;  
  %if &retcode = 0 %then %do;  
    further processing on local host  
  %end;  
%mend checkit;  
%checkit
```

Conclusion

The problems discussed in this paper come from using a macro generated RSUBMIT, where an RSUBMIT resides inside a macro definition. After reading this paper you should now see what happens in regards to compiled code and macro generated text and what actually gets pushed to the remote session. In many cases just moving the RSUBMIT/ENDRSUBMIT outside the macro causes everything to work as desired. By doing this you are causing the macro itself to be compiled on the remote host leaving no room for error as to what should and shouldn't get remote submitted. Just remember MLOGIC and MPRINT as these will help you in your debugging process of determining what is getting remote submitted.

For more information, see SAS/CONNECT User's Guide in SAS OnlineDoc.