Creating a Local Library
Ed Heaton, Westat, Rockville, MD

Abstract
SAS® provides us with the %local statement to insure that macro variables that are created in our macros do not collide with macro variables that might exist outside the macro. However, SAS provides no such utility for SAS datasets.

This paper presents a macro that will create a SAS library that is local to the macro that creates it. With this utility, the users of the macro do not have to be concerned about whether they have a SAS dataset or catalog with the same name as one created by the macro. This paper shows the use of the macro and discusses elements of sound macro design.

Introduction
If you are a developer who writes macros that are used by others—included with an %include statement or called from either an autocall library or a compiled macro catalog—you will find that your macros often need to create datasets that are only used within the macro that created them.

Suppose, for example, we are writing a macro to change the name of all the variables in a dataset by removing the year from the name. (Re. Example 1.)

Example 1: Macro that changes variable names

```sas
%macro dropYearFromName( year , data=&sysLast , out=_data_ ) ;
    %let data = &data ;
    %local renameClauses ;
    Proc contents data=&data out=vars( keep=Name ) noPrint ;
    Run ;
    Proc sql noPrint ;
        Select cats( Name , '=' , tranWrd( Name , "&year" , '' ) )
            into :renameClauses separated by ' ' 
            from vars ;
    Quit ;
    Data &out ;  Set &data( rename=(&renameClauses) ) ;  Run ;
%mEnd dropYearFromName ;
```

The problem here is the dataset named `work.vars`. The user of this macro might already have a dataset named `work.vars`. As a developer, we can't simply overwrite the user's dataset! How can we avoid such dataset collisions?

- It isn't enough to simply write the macro's local datasets to the `work` library because the user of your macro will likely be writing to the `work` library.
- You can tell the user that your macro creates a dataset in the `work` library, tell them the name, and request that they don't use that name for any of their datasets. But this isn't very user-friendly.
- You can allow the user to pass in a name for all of the datasets that the macro creates. But why should you burden the user with this task?
- You can use a random-number generator to create a name for the dataset that is very unlikely to collide with any the user is likely to have, but these names are very cryptic and the code can be tedious.

None of these solutions is satisfactory. We need to create a whole new library that's only used in the %dropYearFromName() macro. We want the library to be temporary and to disappear from the disk drive when the job is done, just like the `work` library.

To that end, let's create a macro called %localLib(). We will create this macro as a function that will generate a libref and return that name to the calling macro. We want the developer to be able to create the local library in a %let statement. E.g.,

```sas
%let libRef = %localLib() ;
```

The developer can then specify a local dataset something like `&libRef..foo`. The developer does not need to know the name of the library since they only need to refer to it by the macro variable. (Re. Example 2.)

The Barebones Macro
Our macro needs to create a folder for the library. We will put that folder in the folder for the SAS `work` library. We can get the fully-qualified name of the `work` folder using the `pathName()` function. Let's call the new folder `LocalLib`. (Re. Example 3.)

The macro returns eight underscores as the name of the local library. Even though the user probably has not previously created a libref named __________, this isn't a particularly good solution. We will improve on it later.
Example 2: Creating a local library in our macro.

```sas
%macro dropYearFromName( year , data=&sysLast , out=_data_ ) ;
  %let data = &data ;
  %local libRef renameClauses ;
  %let libRef = %localLib() ;
  Proc contents data=&data out=&libRef..vars( keep=Name ) noPrint ;
  Run ;
  Proc sql noPrint ;
      Select catS( Name , '=' , tranWrd( Name , "&year" , '' ) )
          into :renameClauses separated by ' ' 
      from &libRef..vars ;
  Quit ;
  Data &out ;  Set &data( rename=(&renameClauses) ) ;  Run ;
  LibName &libRef clear ;
%mEnd dropYearFromName ;
```

Example 3: Macro `%localLib()` will create a Windows directory and a libref that points to that directory

```sas
%macro localLib() ;
  %local folder ;  %let folder = LocalLib ;
  %local parentDir ;  %let parentDir = %sysFunc( pathName(work) ) ;
  %local libRef ;  %let libRef = ________ ;   /* 8 underscores */
  %local libCreationErrMsg ;
  %let libCreationErrMsg = ERROR: The local library was not created. ;
  %local path ; %let path = %sysFunc( dCreate( &folder , &parentDir ) ) ;
  %if not %sysFunc( libName( &libRef , &path ) )
      %then %do ;  %put &libCreationErrMsg ;  %abort abEnd ;  %end ;
  %else &libRef ;
%mEnd localLib ;
```

Find a Folder Name that Does Not Exist

We need to create a new folder for our local library. But first, we need to make sure the folder we are trying to create doesn't already exist. The `fileExist()` function will return a zero if the folder does not exist. Let's just attach a dot and consecutive numbers to the end of the folder name until we find a name that doesn't exist in this location. (Re. Example 4.) This will create a folder named `LocalLib.1` or `LocalLib.2` or `LocalLib.3`, etc.

Example 4: Code added to `%localLib()` to create a Windows directory that does not previously exist

```sas
%local subDir ;  %let subDir = LocalLib ;
%local i ;  %let i = 0 ;
%local folder ;
%do %until ( not %sysFunc( fileExist("&parentDir\&folder") ) ) ;
    %let i = %eval( &i + 1 ) ;
    %let folder = &folderBase..&i ;
%end ;
```

Find a Library Name that is not Used

We also have a problem with the library name. We need a library name that is not in use. We will use somewhat the same process that we used for naming the Windows directory, but using the `libRef()` function. (Re. Example 5) This will create a library named `_______1` or `_______2` or `_______3` ... or `_______10`, etc.
Example 5: Code added to `%localLib()` to find a libref that does not previously exist

```
/* %local libRef ; %let libRef = ________ ; /* 8 underscores */
   %local libRefBase ; %let libRefBase = ________ ; /* 8 underscores */
%let i = 0 ; /* This was previously declared local to this macro. */
%local libRef ;
%do %until ( %sysFunc( libRef(&libRef) ) ) ;
   %let i = %eval( &i + 1 ) ;
   %let libRef = %substr(
       &libRefBase , 1 , %length(&libRefBase) - %length(&i) )&i ;
%end ;
```

Allowing More Flexibility

This macro works. However, we can add more flexibility.

Allowing the Developer to Choose the Folder Name

Now, we should probably allow the developer to choose a different base for the directory name. (I like to use `%sysMacroName`.) We will make LocalLib the default. Of course we have to do some error checking.

We will use the subDir= parameter to pass in the name of the subdirectory. It has a default value of LocalLib, so, the developer can code `%localLib()` and the value for &subDir will be LocalLib. However, the developer might call the macro something like `%localLib( subDir= ),` which would cause a fatal error since it would remove the default value. So, let's catch this error and write an error message. I like to use the `%length()` macro function to determine if a macro variable is empty. (Re. Example 6.)

The developer that calls the `%localLib()` macro might quote the name of the subdirectory. That's okay; we'll remove the quotes if they do. The `dequote()` function works nicely for this; it removes the quotes if the string is quoted but leaves it alone if it isn't.

Example 6: Code added to `%localLib()` to allow the developer to choose the directory name

```
%macro localLib( subDir=LocalLib ) ;
   %local subDirErrMsg libCreationErrMsg ;
   %let subDirErrMsg = ERROR: The SUBDIR= parameter must pass a value. ;
   %if not %length(&subDir) %then %do ;  %put &subDirErrMsg ;  %abort abEnd ;  %end ;
   %else %let subDir = %sysFunc( dequote( &subDir ) ) ;
%end ;
```

We Can Let the User Keep their Datasets

We're okay so far. But the developers might want to keep their datasets when they are in debugging mode. We can give them that with two new parameters – `persist=` and `parentDir=`. If the user doesn't specify a parent directory, let's use the directory specified by the Windows environment variable named TEMP. (Re. Example 7.)

Example 7: Code added to `%localLib()` to keep their local datasets

```
%if &persist
   %then %do ;
      %if not %length(&parentDir) %then %let parentDir = %sysGet(TEMP) ;
   %end ;  /* %if &persist */
%else %let parentDir = %sysFunc( pathName(work) ) ;
```

Error Checking

We need to make sure the values passed in by the `persist=` and `parentDir=` parameters are valid. The `persist=` parameter must pass a number. In SAS, zero and missing are false and all other numbers are true. We can easily check to see if the value is a non-negative integer with the `notDigit()` function. If the value fails the test, we will tell the developer to pass a 0 or a 1. This is a little more restrictive than necessary, but we'll do it anyway since it's so easy to code. We'll still accept any non-negative integer. This might be considered bad form, but we can let it slide for this paper. (Re. Example 8.)
Example 8: Code added to `%localLib()` to check the `persist=` parameter

```sas
%local subDirErrMsg persistErrMsg libCreationErrMsg ;
%let persistErrMsg = ERROR: The PERSIST= parameter must pass 0 or 1. ;
%if not %length(&persist)
   %then %do ;
      %put &persistErrMsg ;
      %abort abEnd ;
   %end ;
%else %if %sysFunc( notDigit(&persist) ) %then %do ;
      %put &persistErrMsg ;
      %abort abEnd ;
   %end ;
%
We also need to check the value passed by the `parentDir=` parameter to make sure it is a valid directory. The `fileExist()` function works well for this. Let's unquote the value if the developer passed a quoted string. Also, let's strip off a trailing backslash if the user included one. Windows machines are very forgiving of doubled backslashes in path names, but let's strip it off anyway. (Re. Example 9.)

Example 9: Code added to `%localLib()` to check the `parentDir=` parameter

```sas
%local subDirErrMsg persistErrMsg parentDirErrMsg libCreationErrMsg ;
%let parentDirErrMsg = ERROR: The PARENTDIR= directory does not exist. ;
/* %if not %length(&parentDir)
   %then %let parentDir = %sysGet(TEMP) ; */
%if %length(&parentDir)
   %then %do ;
      %if not %sysFunc( fileExist(&parentDir) ) %then %do ;
         %put &parentDirErrMsg ;
         %abort abEnd ;
      %end ;
      %let parentDir = %sysFunc( dequote( &parentDir ) ) ;
      %if ( %substr( &parentDir , %length(&parentDir) ) eq \ ) %then %let parentDir = %substr( &parentDir , 1 , %length(&parentDir) - 1 ) ;
      %end ; /* %if %length(&parentDir) */
   %end ; /* %if &persist */
%else %let parentDir = %sysGet(TEMP) ;
%
Dropping Our Local Libraries

If we write our local tables to a subdirectory under the SAS `work` library, the SAS `work` directory will not be deleted by SAS unless we first delete all the files in that directory. We can do this with `Proc datasets` and the `kill` option. Let's create another short macro to do this.

We can write this macro in the same file as the `%localLib()` macro because the developer will call `%localLib()` first and SAS will compile both macros in the file at once. Let's call the new macro `%dropLocalLib()`. (Re. Example 10.)

Example 10: The `%dropLocalLib()` macro will clear the `libref` and delete the SAS files

```sas
%macro dropLocalLib( libRef , persist=0 ) ;
/* SAS will not remove a subdirectory of the work directory if it is not empty. So, remove all of the files and drop the libRef. */
%if not &persist %then %do ;
   Proc datasets library=&libRef kill noList ; Quit ;
%end ;
LibName &libRef clear ;
%mend dropLocalLib ;
Using our Macro

We can make our %dropYearFromName() macro more "black box" by calling the %localLib() macro. We will add a parameter called debugging= and use it to turn on debugging features. In this case, we will pass persist=0 if debugging=0 and persist=1 if debugging=1.

I like to use the name of the macro that creates the local library as the name of the library folder. That name is stored in an automatic macro variable called &sysMacroName. However, we can't simply code

\[
\%localLib( subDir=&sysMacroName ) 
\]

because SAS will not resolve the macro variable in the %localLib() parameter list while it is processing %dropYearFromName(). So, first we need to write the value from &sysMacroName to a local macro variable and then pass the value to %localLib(). (Re. Example 11.)

Example 11: The %dropYearFromName() macro demonstrates the use of the %localLib() macro

```sas
%macro dropYearFromName( year , data=&sysLast , out=_data_ , debug=0 ) ;
  %let data = &data ;
  %local this ;  %let this = &sysMacroName ;
  %local libRef ;  %let libRef = %localLib( subDir=&this , persist=&debug ) ;
  LibName &libRef list ;
  %local renameClauses ;
  Proc contents
    data=&data
    out=&libRef..vars( keep=Name )
    noPrint
    ;
  Run ;
  Proc sql noPrint ;
    Select
      catS( Name , '=' , tranWrd( Name , "&year" , '' ) )
      into :renameClauses separated by ' ' 
      from &libRef..vars
    ;
  Quit ;
  Data &out ;  Set &data( rename=(&renameClauses) ) ;  Run ;
  %dropLocalLib( &libRef , persist=&debug )
%mEnd dropYearFromName ;
```

We added the %dropLocalLib() macro to clean up after ourselves. Now we have a more robust, user-friendly %dropYearFromName() macro. Example 12 is a very simple example of its use.

Example 12: Code to demonstrate the %dropLocalLib() macro

```sas
Data vars ;
   Foo1951 = 55 ;
   Reason1951 = 'because' ;
Run ;
%dropYearFromName( 1951 , data=vars , out=vars , debugging=0 )
Proc contents data=vars short ;
Run ;
```

In the output dataset from Example 12, the years have been removed from the variable names. (Re. Example 13.)

Example 13: Output from our demo

```
The CONTENTS Procedure

Alphabetic List of Variables for WORK.VARS

   Foo   Reason
```
If we look at the log, we see that we created and used a libref named _______1 that pointed to C:\DOCUME~1\Heaton_e\LOCALS~1\Temp\SAS Temporary Files\TD3468\DROPYEARFROMNAME.1. (Re. Example 14.)

Example 14: Log from the demo that calls %localLib() with persist=0

<table>
<thead>
<tr>
<th>NOTE: Libref=   _______1</th>
<th>Scope=    DMS Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine= V9</td>
<td>Physical Name= C:\DOCUME<del>1\Heaton_e\LOCALS</del>1\Temp\SAS Temporary Files\TD3468\DROPYEARFROMNAME.1</td>
</tr>
<tr>
<td>File Name= C:\DOCUME<del>1\Heaton_e\LOCALS</del>1\Temp\SAS Temporary Files\TD3468\DROPYEARFROMNAME.1</td>
<td></td>
</tr>
</tbody>
</table>

Windows Explorer® shows that we created the folder under the WORK directory. (Re. Figure 1.)

Figure 1: Our Temporary Directory under WORK

The DROPYEARFROMNAME.1 folder is empty because we called the %localLib() macro specifying persist=0. Let's change the call to %dropYearFromName() to pass debugging=1 so that the macro then passes persist=1 to the %localLib() macro. Example 15 shows the note that we get in our SAS log.

Example 15: Log from the demo that calls %localLib() with persist=1

<table>
<thead>
<tr>
<th>NOTE: Libref=   _______1</th>
<th>Scope=    DMS Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine= V9</td>
<td>Physical Name= C:\DOCUME<del>1\Heaton_e\LOCALS</del>1\Temp\DROPYEARFROMNAME.2</td>
</tr>
<tr>
<td>File Name= C:\DOCUME<del>1\Heaton_e\LOCALS</del>1\Temp\DROPYEARFROMNAME.2</td>
<td></td>
</tr>
</tbody>
</table>

Now we see that the libref (still _______1) points to C:\Documents and Settings\Heaton_e\Local Settings\Temp\DROPYEARFROMNAME.2. If we call the macro again it will create another directory for the _______1 libref.

Example 16: Log from the demo that calls %localLib() a second time with persist=1

<table>
<thead>
<tr>
<th>NOTE: Libref=   _______1</th>
<th>Scope=    DMS Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine= V9</td>
<td>Physical Name= C:\DOCUME<del>1\Heaton_e\LOCALS</del>1\Temp\DROPYEARFROMNAME.3</td>
</tr>
<tr>
<td>File Name= C:\DOCUME<del>1\Heaton_e\LOCALS</del>1\Temp\DROPYEARFROMNAME.3</td>
<td></td>
</tr>
</tbody>
</table>

Because the libref was unassigned at the end of the macro, %localLib() used the same libref for all three Windows directories. But the directories remain and so do the datasets.
Conclusion

The %localLib() macro and its accompanying %dropLocalLib() macro are serious tools that can be used by developers who write macros or %include files that will be used by other SAS programmers. If you are such a developer, these macros can make your modules more robust and user-friendly.

Consider adding these macros to a corporate-wide autocall library or compiling them in a macro catalog. Either of these can be automatically accessible with a few statements in the users autoexec.sas.

Acknowledgments

I want to thank all the wonderful and insightful contributors to SAS-L for their selfless contributions. They have proven to be my most valuable aid as I learn how to be a SAS programmer. I also wish to thank Westat for its efforts to support my career growth.

SAS® is a registered trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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

Disclaimer

The content of this paper is the work of the author and does not necessarily represent the opinions, recommendations, or practices of Westat.

Contact Information

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

Edward Heaton
Westat
1650 Research Boulevard
Rockville, MD 20850
Work Phone: (301) 610-4818
Fax: (301) 294-3992
Email: EdHeaton@Westat.com
Appendix: The Complete Source Code

MACRO: %localLib()

OBJECTIVE:
This macro will create a Windows directory and a libRef that points to that Windows directory.

VALID:
%localLib() is valid anywhere. %dropLocalLib must be called between program steps.

USAGE: %localLib( subDir= , persist= , parentDir= )

PROGRAMMER:
Edward Heaton, Senior Systems Analyst, Westat (An Employee-Owned Research Corporation),
1650 Research Boulevard, RW-4541, Rockville, MD 20850-3195
Voice: (301) 610-4818  Fax: (301) 294-3879
mailto:EdHeaton@Westat.com  http://www.Westat.com

DETAILS:
This macro will create a temporary libRef that can be used to write data without concern that it might overwrite other data. It does so by creating a Windows directory exclusively for this libRef.

If the PERSIST= parameter is not specified or if it passes 0, then the Windows directory will be created under the SAS WORK directory and will be removed when SAS closes cleanly. (The %dropLocalLib() must be used to drop the library or the WORK directory will remain.) If PERSIST=1 is passed, the libRef will point to a Windows directory under the directory specified with the PARENTDIR= parameter. If the PARENTDIR= parameter is not specified, the libRef will point to a Windows directory under the directory referenced by the Windows environment variable TEMP. Either way, with PERSIST=1 our directory and the files will remain after SAS ends.

PARAMETERS:
subDir= is the new folder that is referenced by the libRef.
persist= is set to 1 to create a directory that persists. PERSIST=1 will cause the PARENTDIR= parameter to be used, otherwise the parent directory is the Windows directory referenced by the WORK libRef.
parentDir= is the parent directory to the new directory that is referenced by the local libRef. If not specified, the system uses the directory referenced by the Windows environment variable TEMP (the parent of the SAS Temporary Files
OUTPUT: a Windows directory and a libRef that points to that directory.

AUDIT TRAIL:
  20060920 EH Developed the macro.
  20060921 EH Added a macro to clean up the files in the folder.
***************************************************************************/
%macro localLib( subDir=LocalLib , persist=0 , parentDir= ) ;
  %local libRefBase ;  %let libRefBase = ________ ;  /* 8 underscores */
  %local subDirErrMsg persistErrMsg parentDirErrMsg libCreationErrMsg ;
  %let subDirErrMsg = ERROR: The SUBDIR= parameter must pass a value. ;
  %let persistErrMsg = ERROR: The PERSIST= parameter must pass 0 or 1. ;
  %let parentDirErrMsg = ERROR: The PARENTDIR= directory does not exist. ;
  %let libCreationErrMsg = ERROR: The local library was not created. ;
  /* Check the value passed by the SUBDIR= parameter. */
  %if not %length(&subDir)
    %then %do ;
      %put &subDirErrMsg ;
      %abort abEnd ;
    %end ;
  %else %let subDir = %sysFunc( dequote( &subDir ) ) ;
  /* Check the value passed by the PERSIST= parameter. */
  %if not %length(&persist)
    %then %do ;
      %put &persistErrMsg ;
      %abort abEnd ;
    %end ;
  %else %if %sysFunc( notDigit(&persist) ) %then %do ;
    %put &persistErrMsg ;
    %abort abEnd ;
  %end ;
  /* Check the value passed by PARENTDIR= and adjust as needed. */
  %if &persist
    %then %do ;
      %if %length(&parentDir)
        %then %do ;
          /* Make sure any value passed by the PARENTDIR= parameter is a valid directory. */
          %if not %sysFunc( fileExist(&parentDir) ) %then %do ;
            %put &parentDirErrMsg ;
            %abort abEnd ;
          %end ;
          /* Unquote &PARENTDIR if it is quoted. */
          %let parentDir = %sysFunc( dequote( &parentDir ) ) ;
          /* If the user passed a trailing backslash with the PARENTDIR= parameter, remove that trailing backslash. */
        %end ;
      %end ;
    %end ;
%end ;
%if (  
    %substr( &parentDir , %length(&parentDir) ) eq \  
) %then %let parentDir = %substr(  
    &parentDir , 1 , %length(&parentDir) - 1  
) ;  
%end ; /* %if %length(&parentDir) */  
%else %let parentDir = %sysGet(TEMP) ;  
%end ; /* %if &persist */  
%else %let parentDir = %sysFunc( pathName(work) ) ;  
/* Create the folder for the library and store the path in a macro variable  
called &PATH. */  
%local i ; %let i = 0 ;  
%local folder ;  
%do %until ( not %sysFunc( fileExist("&parentDir\&folder") ) ) ;  
    %let i = %eval( &i + 1 ) ;  
    %let folder = &subDir..&i ;  
%end ;  
%local path ; %let path = %sysFunc( dCreate( &folder , &parentDir ) ) ;  
/* Find a libRef that does not yet exist. */  
%let i = 0 ;  
%local libRef ;  
%do %until ( %sysFunc( libRef(&libRef) ) ) ;  
    %let i = %eval( &i + 1 ) ;  
    %let libRef = %substr(  
        &libRefBase , 1 , %length(&libRefBase) - %length(&i)  
    )&i ;  
%end ;  
/* Assign the libRef and return the name. */  
%if %sysFunc( libName( &libRef , &path ) )  
%then %do ;  
    %put &libCreationErrMsg ;  
    %abort abEnd ;  
%end ;  
%else &libRef ;  
%mEnd localLib ;  
**************************************************************************
%macro dropLocalLib( libRef , persist=0 ) ;  
/* SAS will not remove a subdirectory of the work directory if it is not  
empty. So, remove all of the files and drop the libRef. */  
%if not &persist %then %do ;  
    Proc datasets library=&libRef kill noList ;  
    Quit ;  
%end ;  
    LibName &libRef clear ;  
%mend dropLocalLib ;  
**************************************************************************