SAS® Explorer: Use and Customization

Richard A. DeVenezia, Independent Consultant

ABSTRACT

This paper describes the functioning of the SAS® Explorer window, the details of customization and strategies for managing customizations.

Explorer

The Explorer motif is one with which almost everyone is familiar; a left hand panel contains a tree view, and a right hand panel contains a list view. The tree view contains nodes that can be opened and closed. The list view contains items corresponding to the node actively selected in the tree view. The items can be displayed as icons or as details, and they can be sorted. The tree view panel can be turned off when focus needs to be concentrated on the items.

This motif is the basis of the SAS® Explorer Window. A wide variety of interaction with SAS tables, catalogs, and applications is only a point and click away.

SAS Environment

The root of the SAS Explorer is the *SAS Environment*. This node may contain six components: Libraries, File Shortcuts, Favorite Folders, My Computer, Results and Metadata Servers. The first four are displayed by default.

The Explorer Options dialog is used to select which components are active. The General tab contains a check box for each component.

The dialog is raised by using the Explorer window menu selection **Tools / Options / Explorer**..., or by entering the Explorer window command EXPOPT.

The state of each component's selection is stored in the SAS Registry beneath the path [CORE \ EXPLORER \ INIT]

The SAS Explorer operates in the following two viewing modes:

- Explorer view – both the tree and list are visible.
- Contents Only view – only the list is visible.

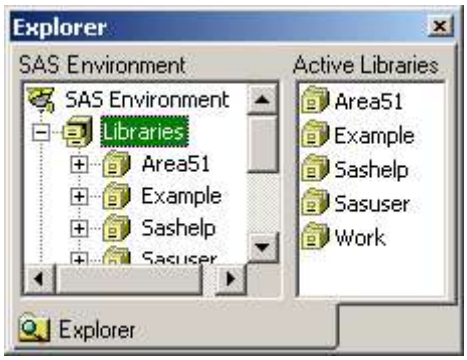
You can switch between these modes by using menu View/Show Tree, by issuing the TREE

command, or by clicking on the Toggle Tree toolbar icon.  The list displays items in one of four modes: Large Icons, Small Icons, List, Details. You can select the mode you want using the View menu, or by issuing the appropriate command: Largeview, Smallview, Details off, Details on

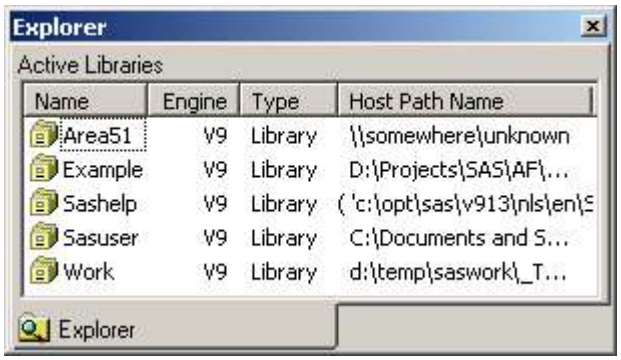
Libraries

The detail pane displays the libnames that are currently assigned. Each item listed corresponds to an assigned library (**libname**). The items are displayed in one of four selectable modes: Large Icons, Small Icons, List, Details. When in details mode, the items can be sorted by Name, Engine, Type or Host Path Name. Sort by clicking on a column header; click again to reverse the sort order.

Tree with node expanded, Details off

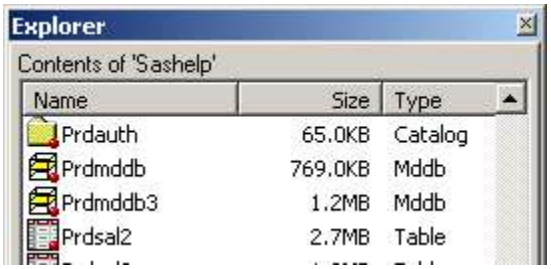


Tree off, Details on



Contents of a library

The detail pane displays the members of a libname. The members shown are items such as tables, views and catalogs. Each item listed corresponds to a two-level name, **libname · memname**. The items are displayed in one of four selectable modes: Large Icons, Small Icons, List, Details. When in details mode, the items can be sorted by Name, Type, Size, Description or Modified date. Sort by clicking on a column header; click again to reverse the sort order.



If the Tree view is visible the expansion of a library node will show only the catalogs.

Contents of catalog

Drill into a catalog by double clicking on its List view icon, or by selecting its Tree view node. The details pane displays the entries of a catalog. A catalog can contain hundreds of items. There are also at least 60 different types of catalog entries. Some common types are source, log, output, frame, scl, class, slist. Each item has a four-level name, **libname · memname · objname · objtype**. The

items are displayed in one of four selectable modes: Large Icons, Small Icons, List, Details. When in details mode, the items can be sorted by Name, Type, Description, Size or Modified date. Sort by clicking on a column header; click again to reverse the sort order.

A catalog node in the Tree view does not expand.

Context menus

When an item in the List view is right clicked, a context menu appears. This means the menu choices presented for an item are based on its type. In the screen shot, the Gng item is of type scl, and it has a menu choice that lets you run the SAS/AF program.

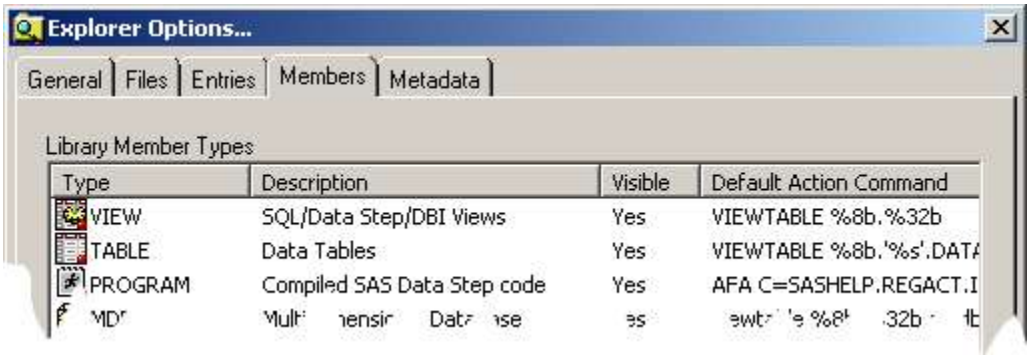
You have now reached base camp for climbing the Everest of extending SAS Explorer.

SAS Explorer Options

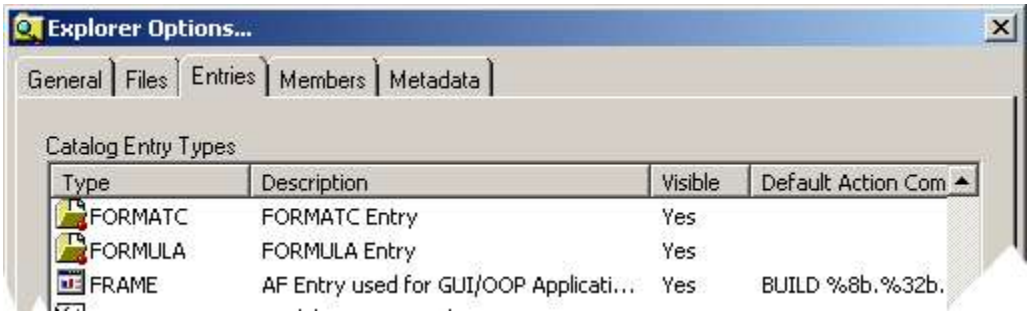
The default configuration of SAS Explorer addresses a wide range of features, meeting almost every demand of the typical SAS user, but you are not typical. The design of Explorer allows for the customization of the context menu choices of every member and entry type. The mouseful way to create customized choices is to use the Explorer window menu **Tools / Options / Explorer...** to raise the Explorer Options... dialog window. The dialog is also opened by the command `EXPOPTS`.

Options for library members

The dialog has tabs for dealing with different hierarchical groups.

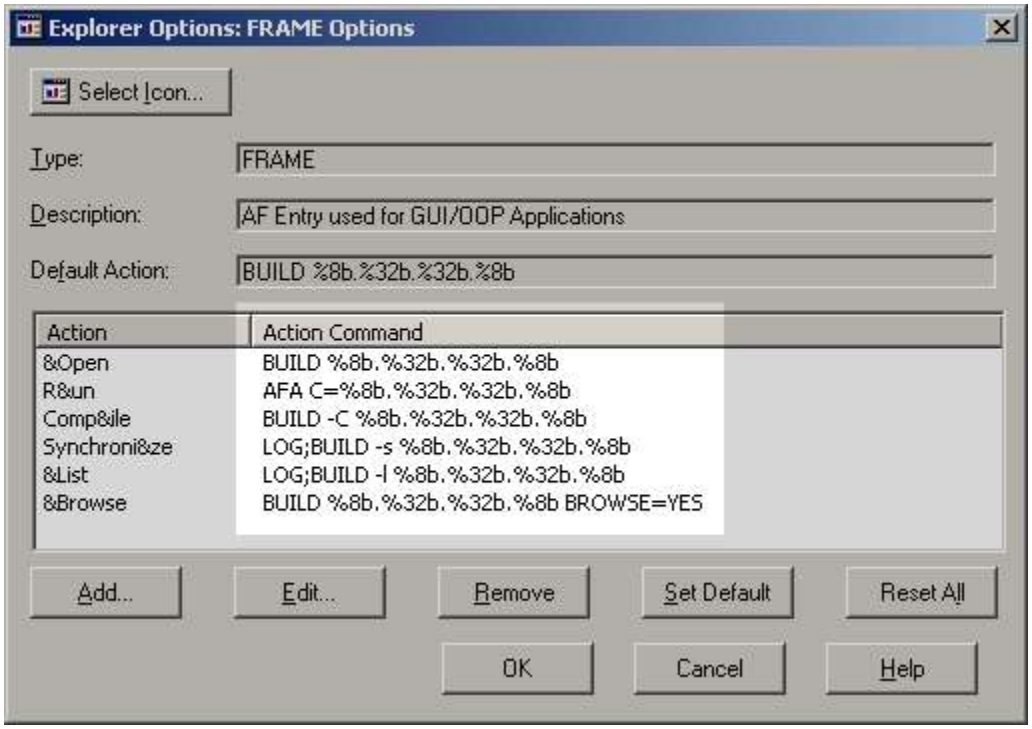


Options for library members



SAS Online Help discusses this dialog. Go to the table of contents; drill down as follows:
SAS Products
  Base SAS
    Using Base SAS Software
      Using the SAS Windowing Environment
        Using the Explorer Window
          Using the Explorer Options Window
          Follow the link to "Changing the Pop-Up Actions and Icon for a Registered Type"

SAS Explorer and the settings of the Explorer Options dialog interact in a manner very similar to *Windows Explorer* and its *Tools/Folder Options* dialog, *File Types* tab. You select a type and then edit it.



A list of actions for the type is displayed; you can add new ones, and edit or delete current ones. Each action has an **Action** and an **Action Command**. The action command is composed of one or more SAS commands, and it may utilize the macro system if desired.

Item name parts

When the context menu of a SAS Explorer item is opened, the parts of the item name are available to the action command. The manner in which they are available is a bit unusual. Each % symbol specified in an action command is replaced with the item's corresponding name part in sequence. You must also specify how many bytes of the namepart you wish to use.

	Items of a Library		***Items of a Catalog***	
	Namepart	***SAS spec***	***Namepart***	***SAS spec***
1st %	libname	%8b	libname	%8b
2nd %	memname	%32b	memname	%32b
3rd %	libname	%8b	objname	%32b
4th %	memname	%32b	objtype	%8b
5th %	n/a		libname	%8b
6th %	n/a		memname	%32b
7th %	n/a		objname	%32b
8th %	n/a		objtype	%8b

The number between a **%** and **b** specifies how many bytes of the name part are used in the replacement. Do not be concerned about somehow crashing SAS with this. When you specify more bytes than a namepart has, SAS will not grab unknown characters from memory storage 'beyond' namepart. However, if you accidentally specify a number of bytes fewer than a namepart has, you will truncate a namepart, which can cause problems for you. Suppose you mistakenly used **%1b** for the **libname** namepart. The action command would only get the first letter of the library name!

The number between a **%** and **b** may seem extraneous, but it is not. Although not required per se, it is a good idea to have the number so as to remind you which namepart is supposed to be getting placed where. The lack of a number can result in part of your command being 'eaten' by some internal processor. Case in point, `POSTMESSAGE "(%b,%b,%b,%b)"`, in SAS version 8. Go numberless at your own risk.

Simple commands

The first step up from base camp is using simple task orientated commands that accept a SAS named thing as an argument.

Consider this action command for a library member of type data or view:

Action: `FSVIEW`
Action Command: `FSVIEW %8b.%32b`

For item `sashelp.class` the action command gets resolved to `FSVIEW sashelp.class`

Using macro

The macro system can be used in an action command. Recall that a single % is used as a token for replacement with a namepart. But % is also important to macro! Use two consecutive % in an action command to indicate macro processing.

Consider this action command for a catalog entry:

Action: `MPut`
Action Command: `%%put %8b.%32b.%32b.%8b`

For item `sashelp.devices.pscolor.dev` the action command gets resolved to

```
%put sashelp.devices.pscolor.dev
```

Using DATA or Proc steps

The GSUBMIT command is used to submit SAS code snippets.

Consider this action command for a catalog entry:

Action: `DPut`
Action Command: `gsubmit 'data _null_;put "%8b.%32b.%32b.%8b";run;'`

For item `sashelp.devices.pscolor.dev` the code submitted by the resolved command is:

```
data _null_;
put "sashelp.devices.pscolor.dev";
run;
```

Using SAS/AF Frames or SCL

The AFA command launches SAS/AF programs. Parameters are passed to the program using name=value pairs on the command line.

Consider this action command for a catalog entry:

Action: `AFPut`
Action Command: `afa c=sasuser.myhandlers.foo.scl entry=%8b.%32b.%32b.%8b`

For item `sashelp.devices.pscolor.dev` the action command gets resolved to

```
afa c=sasuser.myhandlers.foo.scl entry=sashelp.devices.pscolor.dev
```

The AFA command executor parses **name=value** pairs found on the command line and stores them as named character items in the _CMDLIST_ sublist of the local environment list. This sample SCL demonstrates how to obtain an arguments value.

```
declare char VALUE NAME='entry' ;

VALUE
  = getNitemC
    ( getNitemL
      ( envlist('L'), '_CMDLIST_' )
    , NAME, 1, 1, ''
    );

put NAME= VALUE=;
```

WARNING: The _CMDLIST_ reserves items of name LIBNAME, CATALOG, NAME and TYPE for documenting the active program. Do not use these names as parameter names in an AFA command.

An examination of the SAS registry shows the AFA command is one favored by SAS Institute. These `SASHELP.EXPLORER` SCL entries are invoked by a context menu selection:

- Copy_to_clipboard* - Explorer Action to a Copy a Table to the Clipboard as HTML.
- Excel_table_open* - Explorer Action to open a SAS Table in Excel.
- Save_as_html* - Explorer Action to save a SAS Table as HTML.
- Open_html_entry - Open an HTML entry with WBROWSE.
- Open_sas_file - Open a SAS7BDAT or SAS7BCAT file.

The catalog also contains

- Open_file* - Example Explorer Action for processing a file.
- Open_catalog_entry* - Example Explorer Action for Processing a Catalog Entry.

* Starred entries include source and demonstrate a range of SCL features that can be used in implementing an action.

Using autocall macros

You can use autocall macros to offload the code writing part of the action commands to a text file. This approach is recommended for situations when SAS/AF development is infeasible; perhaps because SAS/AF is not licensed, or the available programming skills are in Base SAS and SAS Macro only.

Consider this scheme; create a text file `xactmac.sas`. This file resides in a library specified in the SASAUTOS System Option. The file is constructed as such:

```
%macro xactmac (type,action,lib,mem,obj,typ);
  /* %xactmac will cause this file to be automatically included once,
   * and by side effect all the action_* macros will get compiled
   */

  /* dispatch typed action handler */
  %action_&type._&action (&lib,&mem,&obj,&typ);
%mend;

%macro action_data_fsview (lib,mem,obj,typ);
  fsview &lib..&mem;
%mend;
```

`xactmac.sas` might contains several if not dozens of macro declarations. For consistency, each macro would be named according to the construct **action_{type}_{action}**

Consider this action command for a data library member:

Action: `FSView`
Action Command: `%%xactmac(data,fsview,%8b,%32b,.,.)`

For item `sashelp.class` the action command resolves to `%xactmac (data,fsview,.,.)` which in turn dispatches `%action_data_fsview (sashelp,class,.,.)` which in final issues the command `fsview sashelp.class`.

This dispatch methodology allows you to centralize the implementation details of action commands outside of the registry.

Programmatic customization

The discussion up to this point only covers adding an action using the point and click Explorer Options dialog. Using the dialog is fine for a few actions, but what happens if you have dozens of actions to configure? What if you want to reuse your custom actions in new projects or share them with coworkers or a programming community?

The SAS Explorer Options dialog is a front end for adding or updating keys in the SAS Registry. The keys that get altered by the dialog are under these branches:

```
CORE\EXPLORER\MENUS\MEMBERS
CORE\EXPLORER\MENUS\ENTRIES
CORE\EXPLORER\MENUS\FILES
CORE\EXPLORER\MENUS\METAEXPLORE\TypeFilters
```

There are three ways to alter the SAS Registry.

- Explorer Options dialog
- `Proc REGISTRY`
- SAS/AF class `sashelp.fsp.registry.class`

Proc REGISTRY

A moderate amount of information about Proc REGISTRY is available in the Online Help. Read it. You will come to understand that values are loaded into the registry by creating a file that is constructed as follows:

```
[key]
"value-name"="value-content"
```

The keys of interest for customizing actions are

```
[CORE\EXPLORER\MENUS\MEMBERS\MEMBERTYPE]
[CORE\EXPLORER\MENUS\ENTRIES\ENTRYTYPE]
[CORE\EXPLORER\MENUS\FILES\FILETYPE]
```

plus

```
[CORE\EXPLORER\MENUS\MEMBERS\ROOT]
[CORE\EXPLORER\MENUS\MEMBERS\LIBRARIES]
```

value-name

The construct of the value-name is important (and partly undocumented.)

- An ampersand, &, can precede your desired single letter mnemonic (aka shortcut key.)
- The form {A};{B} has several interpretations. {A} is a positional value, {B} is the menu text.
  If {A} starts with a minus sign, the item will be grayed.
  If {B} starts with a minus sign, the item will not appear in the menu (and may cause other items to disappear from the context menu for the entry or member type)
- The form {A};{B};{C} has an additional interpretation.
  {C} can specify a classifier. The classifier determines which icon is displayed next to the menu choice.
  The list of classifiers are found at key `[CORE\CLASSIFIERS]`
  The icon appears only if {B} has a mnemonic

value-name examples

The actions in this sample all do the same thing, examine the data of a table using ViewTable. The difference is in the actions is the name-value, the part to the left of the equals sign.

c:\temp\foo-actions.txt

```
[CORE\EXPLORER\MENUS\MEMBERS\TABLE]
"Foo 1"="VT %8b.%32b"
"&Foo 2"="VT %8b.%32b"
"10;&Foo 3"="VT %8b.%32b"
"09;&Foo 4;100"="VT %8b.%32b"
```

SAS Session

```
Proc REGISTRY IMPORT="c:\temp\foo-actions.txt";
run;
```

Foo 4 appears farther down the context menu than Foo 3 because 09 < 10. The icon next to Foo 4 corresponds to classifier #100.

Factory actions missing

Look carefully and note that the 'default factory' menu choices for the TABLE member type are not seen! Why? Customizations made via the Options dialog or imported via Proc REGISTRY are stored in the SASUSER registry. A SASUSER registry branch automatically overrides a SASHELP registry branch. The SASHELP [...\TABLE] branch contains the default factory settings. To ensure the factory settings are present you need to copy them from SASHELP to your SASUSER.

```
Proc REGISTRY
   STARTAT='CORE\EXPLORER\MENUS\MEMBERS\TABLE'
   USESASHELP EXPORT='c:\temp\factory-table-actions.sasreg';
Proc REGISTRY IMPORT='c:\temp\factory-table-actions.sasreg';
run;
```

Restore factory actions

The default actions of a type can be restored by deleting all the SASUSER customizations made for it. A simple program can export the current settings and then uninstall them.

```
Proc REGISTRY STARTAT='CORE\EXPLORER\MENUS\MEMBERS\TABLE'
                 EXPORT='c:\temp\sasuser-table-actions.sasreg';
Proc REGISTRY UNINSTALL='c:\temp\sasuser-table-actions.sasreg';
run;
```

New

Advanced customizers might also use key

```
[CORE\EXPLORER\NEW\ENTRY\ENTRYTYPE]
[CORE\EXPLORER\NEW\MEMBER\ENTRYTYPE]
```

to alter the list of entry types shown in the New Entry dialog raised by selecting New from the context menu displayed when SAS Explorer is displaying the members of a library or contents of a catalog.

More on Macros

There are various strategies that can be employed when developing macros for use as SAS Explorer action commands.

Macros - Part II.A

Suppose you named your action_* macros to exactly match the key path under which the command action is supposed to be installed. Further suppose the macros description corresponds to the menu choice you want displayed.

```
%macro action_members_table_A1 (lib,mem,pos3,pos4) / des='FS Browse';
  /* browse a table using the likenamed screen stored in sasuser.profile */
  fsbrowse &lib..&mem sasuser.profile&lib..&mem..screen;
%mend;
```

Such a construct has all the information needed for a self realizing scheme. A program can read the macro catalog and write a file that Proc REGISTRY can import.

```
/ *
  * Register SAS Explorer actions based on macro name and description
  */

proc sql;
  create view actions as
  select objname, objdesc
  from dictionary.catalogs
  where libname = 'WORK'
    and memname = 'SASMACR'
    and objname like 'ACTION_%'
    and objtype = 'MACRO'
  ;
quit;

filename xact catalog 'work.explorer.actions.source';

data _null_;
  set actions;

  branch = scan (objname,2,'_');
  type   = scan (objname,3,'_');

  namevalue = quote(trim(objdesc));

  file xact;

  put '[CORE\EXPLORER\MENUS\' branch +(-1) '\' type +(-1) ']';
  put namevalue +(-1) '="%%' objname '(%8b,%32b,%32b,%8b)"';
run;

proc registry
import=xact; run;
```

VIEWTABLE: Work.Actions

	Object Name	Object Description
1	ACTION_MEMBERS_TABLE_A1	FS Browse

FSLIST: WORK.EXPLORER.ACTIONS.SOURCE

```
[CORE\EXPLORER\MENUS\MEMBERS\TABLE]
"FS Browse"="%%ACTION_MEMBERS_TABLE_A1 (%8b,%32b,%32b,%8b)"
                                *** END OF FILE ***
```

Macros - Part II.B

A slightly different construct would use shorter macro names and place all action context info in the description. The macro description is the branch, type and name-value delimited by the pipe character (|):

```
%macro xact_1 (lib, mem) / des='M|TABLE|10;FS View';
  /* view a table using the default formula stored in sasuser.profile */
  fsview &lib..&mem; formula
%mend;
```

The core of this technique relies on extracting information from the macro description.

branch	`scan(objdesc,1,'|')`
type	`scan(objdesc,2,'|')`
name-value	`scan(objdesc,3,'|')`

```
/*
 * Register SAS Explorer actions based on macro name and description
 */

proc sql;
  create view actions as
  select objname, objdesc
  from dictionary.catalogs
  where libname = 'WORK'
    and memname = 'SASMACR'
    and index (objname,'XACT_') = 1
    and objtype = 'MACRO'
  ;
quit;

filename xact catalog 'work.explorer.actions.source';

data _null_;
  set actions;

  branch  = scan (objdesc,1,'|');
  type    = scan (objdesc,2,'|');
  choice  = scan (objdesc,3,'|');

  if branch = 'M' then branch = 'MEMBERS'; else
  if branch = 'E' then branch = 'ENTRIES';

  select;
    when (branch =: 'M') command='%%'||trim(objname)||'(%8b,%32b)';
    when (branch =: 'E') command='%%'||trim(objname)||'(%8b,%32b,%32b,%8b)';
    otherwise delete;
  end;

  choice  = quote(trim(choice));
  command = quote(trim(command));

  file xact;
  put '[CORE\EXPLORER\MENUS\' branch +(-1) '\' type +(-1) ']';
  put choice +(-1) '=' command;

run;

proc registry import=xact;
run;
```

VIEWTABLE: Work.Actions

	Object Name	Object Description
1	XACT_1	M|TABLE|10;FS View

FSLIST: WORK.EXPLORER.ACTIONS.SOURCE

```
[CORE\EXPLORER\MENUS\MEMBERS\TABLE]
"10;FS View"="%%XACT_1(%8b,%32b)"
                *** END OF FILE ***
```

Macros - Part II.C

A third variant would store a minimum of meta information in name or description. The

minimum is the form of the macros invocation. The remainder of the meta information of the macros use context would be divulged upon request.

```
%macro xact_1 (lib, mem, divulge=) / des='%8b,%32b';
  %let divulge = %upcase(&divulge);
  %if &divulge = BRANCHES %then %do;
    MEMBERS\TABLE
    MEMBERS\VIEW
  %end;
  %else
  %if &divulge = NAME %then %do;
    %str(10;FS View)
  %end;
  %else %do;
    fsview &lib..&mem; formula
  %end;
%mend;
```

The benefit of this approach is that a single macro can indicate its utilization in more than context menu. The program to utilize divulgent macros whose names start with XACT is left as an exercise for the reader. The program would have to loop through the BRANCHES divulged and create a registry key entry for each. (The two earlier approaches allow a macro to specify its applicability to only one branch.)

Bonus section

Comments

Lines that start with the # symbol can be used to comment and document your registry import files.

```
# Purpose:     List the value mappings that comprise a numeric format
# Note:        The format must reside in a catalog listed
#              in the FMTSEARCH system option
# Contributor: Richard A. DeVenezia
# Date:        23Jan2005

[CORE\EXPLORER\MENUS\ENTRIES\FORMAT]
...
```

Name part specifiers

Experimentation shows the %*n*b tokens used for replacement are in fact format specifiers as used in the C function **sprintf**. The **b** Type is not standard; it is speculated to be a custom type implemented by SAS which stops output at width, or the first zero byte or blank. If you are curious, try using a specifier such as %x or %d.

SAS Explorer maintains 2 (or 4) pointers that reference the memory locations where the nameparts are stored. When it is time to resolve the action command, this happens internally:

```
...obtain pointers to nameparts
...obtain action command from regigstry
sprintf ( lpstrCommand
        , lpstrActionCommand
        , lpstrLibname
        , lpstrMemname
        , lpstrObjname
        , lpstrObjtype);
...present lpstrCommand to command executor...
```

Conclusion

SAS Explorer is an essential component for efficient interactive work within a SAS session. It is founded on design principles utilized in a wide variety of other software systems, making the interface easily accessible to both new and veteran SAS users. Repetitive tasks can be simplified by creating custom actions.

About the Author

Richard A. DeVenezia is an independent consultant who has worked extensively with SAS products for over fifteen years. He has presented at previous SUGI, NESUG and SESUG conferences. Richard is interested in learning and applying new technologies. He is a SAS-L Hall of Famer and remains an active contributor to SAS-L.

Downloadable actions

A variety of actions can be downloaded and installed from the author's website. Visit http://www.devenezia.com/downloads/sas/actions/main.php

Acknowledgments

The author is indebted to Dr. Ann DeVenezia for her superb editing.