

Development Facilities and Tools for SAS® Applications

Philip Van Dusen, SAS Consulting Services Inc.

ABSTRACT

SAS Consulting Services has an application development environment (ADE) that centers around a collection of generic SAS programs we call Toolbox. Components of the Toolbox include SAS/AF subclasses, utility programs, applications, SAS macros, and SAS sample code. The organization of, and access to, Toolbox itself is controlled by an AF program. This paper will describe the infrastructure of our Toolbox using example Toolbox screens and Toolbox entries.

INTRODUCTION

In the effort to achieve higher levels of productivity and quality in the software industry, a great deal of attention has been focussed on the issue of effectively using proven software. The most common vehicle for code reuse is the toolbox, a library of reusable software modules which extend the native capabilities of the underlying programming language or software system. Code reuse has existed in some form since the earliest years of the software industry. A traditional toolbox typically consisted of compiled subroutines in a commonly used language such as COBOL or FORTRAN. In recent years, the shift towards object-oriented design methodologies has facilitated reuse by exploiting the benefits of inheritance and the encapsulation of functionality and data. Hence, many toolboxes today contain object classes.

In the SAS arena, the evolution of user-created toolboxes has paralleled developments in the rest of the industry. A form of SAS toolbox which existed in the early years of SAS was a library of user-written PROCs. With the later development of the macro facility, macro routines became a more flexible way of packaging and customizing reusable SAS code. With the advent of Version 6 and the substantially enhanced capabilities for building user-friendly applications, the well-stocked toolbox expanded to include a variety of commonly-used PROGRAM entries and customized FSEDIT screens.

Over the last several years, the dramatically

improved graphical user interface (GUI) offered by FRAME technology has moved the FRAME entry to the forefront of the leading-edge SAS toolbox. Simultaneously, the OOPS capabilities introduced with FRAME technology allows the subclassing of objects and widgets. These CLASS entries are another rapidly growing component of the leading-edge SAS toolbox and may eventually form the bulk of it.

Regardless of the type or number of entries, the toolbox confers significant advantages beyond the greater productivity attained by reusability. These include:

- **Templates**

Toolbox entries can be used as templates for new, similar utilities.
- **Modularity**

Toolbox entries are cohesive units which allow for easier modularizing of programs in which they are included.
- **Prototyping**

The toolbox is an excellent source for components to build a quick prototype for customer demos.
- **Reliability**

Toolbox entries are thoroughly tested and debugged, enhancing the reliability of applications in which they are included.
- **Standards**

Toolbox entries enforce coding standards and screen design standards.

For an evolving software base as extensive as the SAS System, it is especially important to harness its power effectively by exploiting the recognized advantages of toolboxes. However, achieving a

high level of reuse in practice has proved difficult for many organizations throughout the industry. One important step in overcoming these difficulties is to make the toolbox readily accessible to all members of the organization, regardless of the level of programming experience. To achieve this, it is not sufficient to implement the toolbox simply as an expandable repository from which developers can copy tools when needed. It is necessary to organize the toolbox in a way which makes it easy for a developer to locate a needed tool, readily assess its suitability for a target application, and incorporate the tool into the target application with minimal effort. One way to achieve these objectives is by managing the toolbox with the use of a SAS application which serves as a type of application development environment (ADE). An example of such an ADE, the SAS Consulting Services (SCS) Toolbox, is described in the following sections.

INFRASTRUCTURE OF THE SCS TOOLBOX

The SCS toolbox application provides capabilities for convenient browsing of the tools, demonstration of their capabilities, and copying of the tools into custom applications. This infrastructure and its attendant functionality is now described.

MAIN MENU

The main menu for Toolbox is shown in Figure 1.

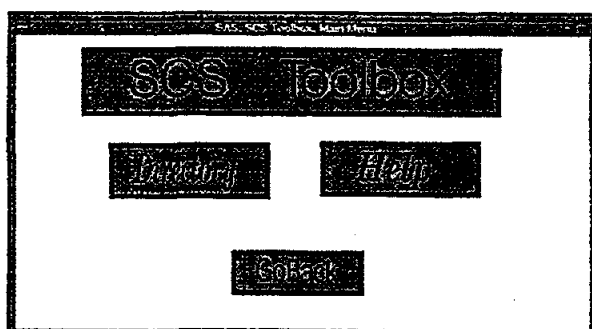


Figure 1

DIRECTORY

The contents of the Toolbox are viewed by selecting the "Directory" button on the main

menu. This displays a listing of all Toolbox entries as shown in Figure 2.

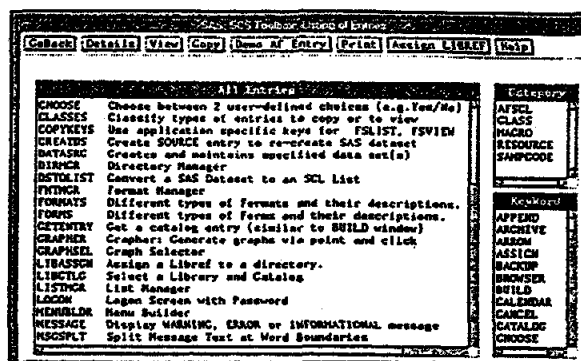


Figure 2

It should be pointed out that while each directory entry is associated with a single tool, a given tool may be comprised of several catalog entries. These entries are grouped together within a single mnemonically named catalog. By using a categorization scheme and keywords a tool can be easily found.

All tools in the SCS Toolbox fall into one of the following five categories:

- AFSCL

The AFSCL category includes FRAME, PROGRAM, and SCL entries. It does not include CLASS objects or widgets. The AFSCL tools constitute the core of the current Toolbox and represent a wide range of complexity. An example of one of the simpler tools is MESSAGE.PROGRAM. This might be used to inform a user of an error or warning condition, something needed in almost every application. A more complex example is a tool such as FORMAT MANAGER which can function as a stand-alone utility.

Each AFSCL tool is found in its own SAS catalog. This catalog also contains any support entries, such as PMENU entries, which might be required to use the tool in an application. Other potential support entry types include SLISTs (saved SCL lists), GRSEGS (for graphs), and SOURCE entries. Each catalog also contains a README.SOURCE entry which supplies any special information you might need to use the tool. Another standard entry is a Demo program

which is supplied to show how the tool would function in an application. The demo programs allow you to try the tool out and test the parameters that might be used when linking the tool into an application.

Example:

Catalog Toolbox.MESSAGE - The entries and descriptions associated with this tool are as follows:

GOBACK.PMENU

Pmenu for MESSAGE.PROGRAM.

MESSAGE.PROGRAM

Display Messages to Users.

DEMO.SCL

Demo program for MESSAGE.PROGRAM.

README.SOURCE

Readme file for MESSAGE.PROGRAM.

Note that the name of the catalog matches the name of the primary functional entry, MESSAGE.PROGRAM.

• **CLASS**

The CLASS category includes all tools based upon user-defined subclasses, usually widget subclasses. As with the AFSCL category, all of the entries necessary to define the tool are packaged in a single catalog. The entries in the catalog include the CLASS entry, the custom attribute FRAME/SCL entries, a README.SOURCE entry, a METHODS.SCL entry containing any custom methods (new or override) defined for the subclass, and any support entries (such as PMENUs) needed for the custom attributes.

Example:

Catalog Toolbox.RITXTENT - This defines the referential integrity text entry widget. The entries and descriptions associated with this tool are as

follows:

RITXTENT.CLASS

Referential Integrity Text Entry Subclass.

ATTRIBUTE.FRAME

Custom Attribute Screen for RITXTENT Subclass.

GOBACK.PMENU

Goback pmenu.

ATTRIBUTE.SCL

Custom Attribute SCL for RITXTENT Subclass.

METHODS.SCL

Referential Integrity Text Entry Methods.

README.SOURCE

Read Me file for Referential Integrity class.

• **MACRO**

The MACRO category contains SAS macro definitions stored in SOURCE entries. Unlike the previous categories, all macro tools are stored together in a single catalog named Toolbox.MACRO, and no demo programs are supplied. To get more information about what the macro does, you must View the entry and read the comment block.

Example:

Entry NOBS.SOURCE - This macro determines the number of observations in a SAS dataset.

• **RESOURCE**

The RESOURCE category contains the BUILD.RESOURCE entry within which all the Toolbox CLASS entry widgets are specified.

• **SAMPCODE**

The SAMPCODE category contains SOURCE entries of sample SAS code, which might consist

of DATA step code, SCL code, PMENU code, Macro code, etc. As with the MACRO category, all Sample Code SOURCE entries are stored in the Toolbox.SAMPCODE catalog. Generally, the entries in this category are intended to be used by cutting and pasting source code rather than by linking into an application. Often they are useful for supplying a model for a solution to a problem in cases where a completely generalized solution is not applicable, does not yet exist, or would be impractical to develop.

Example:

Entry PROJECTR.SOURCE - Optimal Window colors for overhead Projector.

This entry contains a collection of command-style macro definitions consisting of COLOR commands for various SAS windows.

FUNCTIONALITY OF THE TOOLBOX INFRASTRUCTURE

Most of the functionality of the Toolbox infrastructure is managed through the Directory screen. There are three list boxes on this screen as shown in Figure 3:

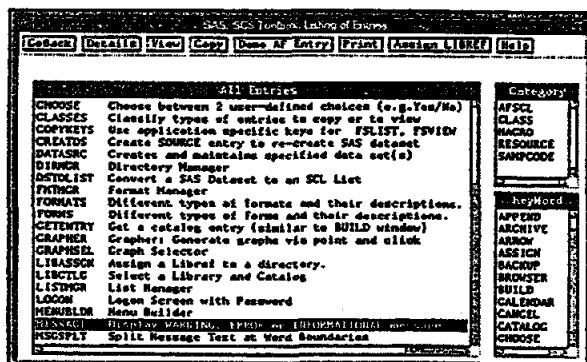


Figure 3

• Category List Box (Upper right)

This is the primary subsetting mechanism for the list box containing the directory listing. The categories listed are those described in the prior section. Clicking on a category subsets the directory listing to the tools from that category. The title of the directory listing changes to match

the selected category. By default, it appears as "All Entries."

• Keyword List Box (Lower right)

This is the secondary subsetting mechanism for the directory listing. The keyword list is populated and maintained by the System Administrator for the Toolbox, using suggestions from the tool developer. One or more keywords may be assigned to each entry in the Toolbox index and a given keyword might apply to more than one tool. Hence, clicking on a keyword subsets the directory listing down to those tools from the currently selected category which also have the selected keyword assigned.

In practice, a selected keyword will often match a word from the description attached to the tool in the directory, although this is not always the case. Because the keyword list serves the same function as the index for a book, it ideally should be just as comprehensive as a good book index.

• Directory List Box (Left side)

The directory list box displays the names and descriptions of all tools which match the active selections within the category and keyword list boxes. Tools displayed in the directory can be selected to be viewed, copied, demoed, or printed using the appropriate PMENU entry. Also, for users with a mouse interface, double clicking on an entry activates the demo program supplied with the tool. In the current Toolbox version, the demoing functionality applies to AFSCCL and CLASS entries only.

The PMENU items function as follows:

• Details

When a directory entry is selected, the Details pmenu item displays a screen that describes the Toolbox entry in more detail. Figure 4 is the Details screen for the MESSAGE.PROGRAM.

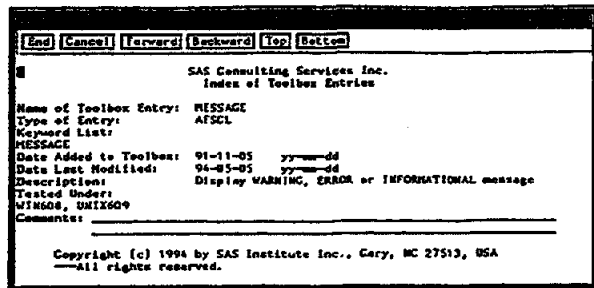


Figure 4

This screen is largely self-explanatory. You will notice that the "Date Added to Toolbox", "Date Last Modified", "Tested Under" (platforms), and "Comments" items expand upon the information that is presented in the directory. Hence, this screen is useful as a first-order look at assessing the suitability of a given tool for a particular target application.

- View

When the View pmenu item is selected and the Toolbox category is AFSCL, RESOURCE, or CLASS, the BUILD window for the corresponding catalog is displayed. Figure 5 shows the BUILD window for the MESSAGE.PROGRAM.

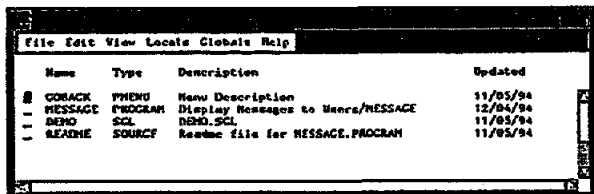


Figure 5

For MACRO and SAMPCODE categories, the PREVIEW window is displayed and contains the source code for the entry.

View is suitable for obtaining the most precise information about a tool, namely the complete source code. However, in many cases, you will use View simply as a way to look at the comment block.

- Copy

Given the structure of the Toolbox ADE, it is not normally necessary to copy a tool. (See the later

section USING TOOLS IN CUSTOM APPLICATIONS for details). Nevertheless, there may be occasions when you want to copy selected tools to another library. For AFSCL, CLASS, and RESOURCE entries, the Copy selection brings up the screen shown in Figure 6 using the MESSAGE tool as an example:

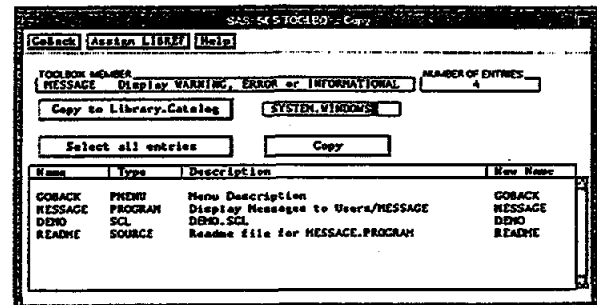


Figure 6

For the MACRO and SAMPCODE tool categories, the Copy selection brings up the screen shown in Figure 7 using the SYSLPUT macro as an example:

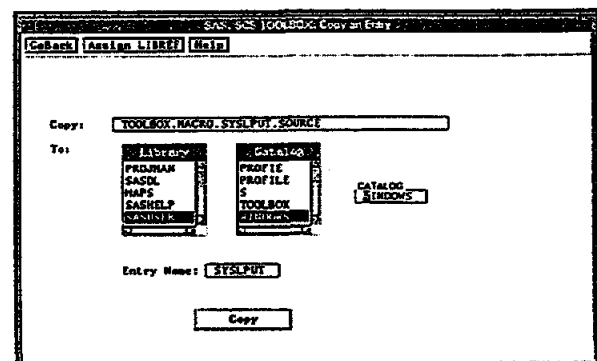


Figure 7

For both types of entries, the general copy mechanism is the same: You specify a destination catalog and press the "Copy" pushbutton if it's OK to proceed. The operational difference is that the MACRO and SAMPCODE entries only involve copying of one source entry, whereas the other categories involve copying a catalog of entries or a specified subset of these entries. Either type of copy allows assignment of a LIBREF for the destination catalog if it doesn't already exist.

- Demo AF Entry

This functionality provides an easy way for you to assess the suitability of a tool for incorporation into an application. In the current version of the SCS Toolbox, this feature is only provided for AFSCAL and CLASS entries. When a tool is selected in the directory list box and the Demo AF Entry pmenu item is selected (double clicking on the directory entry accomplishes the same thing), a runtime demonstration of the tool is provided. The demo program is not part of the tool itself, but is packaged with the tool (in the same catalog) to launch it and show off its capabilities. Where applicable, the demo program allows you to test entry parameters that might be used when calling the tool from a custom application. Also where applicable, the demo programs execute source code to define and populate sample data sets required by the tool. Figure 8 shows the demo program screen for the DATEPAD CLASS tool:

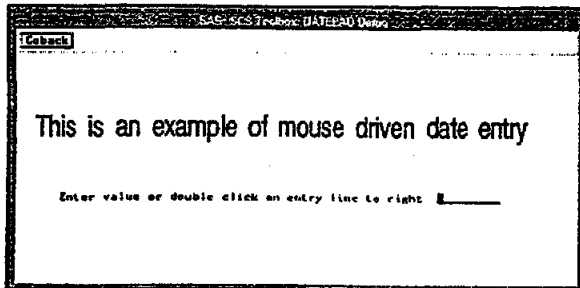


Figure 8

Double-clicking on the data-entry widget displays the tool of interest as shown in Figure 9:

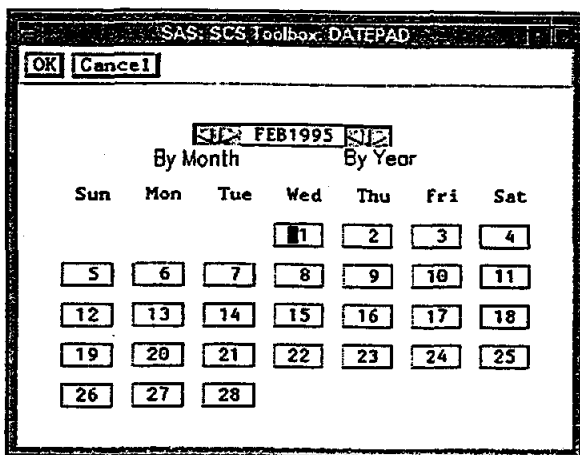


Figure 9

For purposes of assessing suitability, there can be

no doubt that this approach is quickest and best. Especially for FRAME entries, viewing a screen in BUILD mode does not necessarily show how the widgets work together at runtime. On the other hand, a well-constructed runtime demo can show how a screen is used when the functionality might not be so intuitive otherwise. Also, a demo program makes it easy to compare and distinguish tools which may appear very similar based upon their descriptions. For example, one type of MESSAGE tool may allow you to specify four lines of message which the calling program must format. Another type of MESSAGE tool may handle the word-wrapping across lines and allow a longer message text. The demo programs help to show these differences quickly.

Finally, the demo program immediately tests the "look and feel" of the tool at runtime. Even if the functionality of a tool satisfies the requirements, you might reject the tool because the "look and feel" is unsuitable for the end users of the application.

- Print

This item allows you to select one of several print options. You may print the directory listing, details for any or all directory entries (identical to the screen presented by the "View" PMENU item), and source code for any of the tools. Output from the print commands goes to the OUTPUT window or the FSLIST window, from where it can be sent to a standard printer using a selected form.

SYSTEM ADMINISTRATOR'S Toolbox

The SCS Toolbox System Administrator has the responsibility for adding new tools to the production Toolbox as well as enforcing the QA procedures that are required for progression through Development and Test versions. As part of this process, the System Administrator must add a new index entry to the dataset Toolbox.INDEX for each tool that is added. The INDEX entry contains all of the information needed to locate and use the tool through the Toolbox ADE. This includes the entry name, entry type, description, comments, keywords, the 2-level name of the startup program (used to generate the 4-level call when the tool is used), platforms on which the tool has been tested, date added to the Toolbox, and date last modified.

To support the additional responsibility of maintaining the index, the SCS Toolbox application provides a startup path for the System Administrator which is different from that provided for other users. As seen in Figure 10, this path presents a different Main Menu screen from that shown in Figure 1:

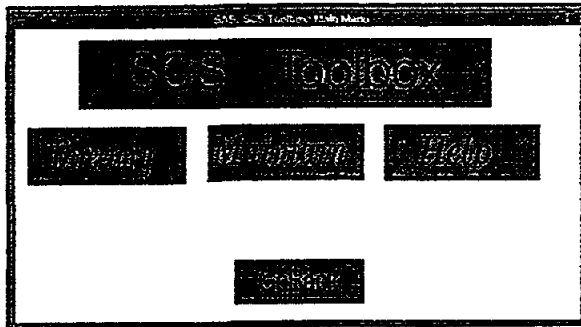


Figure 10

The System Administrator maintains the INDEX dataset by clicking on the "Maintain" icon. This displays the screen shown in Figure 11.

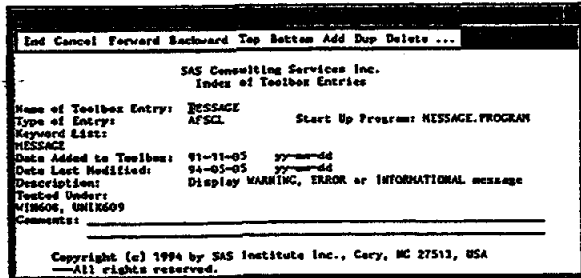


Figure 11

This screen is almost identical to the screen which displays when the "Details" pmenu item is selected from the Directory screen (see Figure 4). The difference is pmenu choices; namely, in Figure 11 note the "Add", "Dup(licate)", and "Delete" choices.

USING TOOLS IN CUSTOM APPLICATIONS

The Toolbox ADE is designed to allow transparent use of AFSCL and CLASS tools from custom applications without having to specify a 2-level name or 4-level name in a CALL

DISPLAY.

As previously stated, each AFSCL and CLASS Toolbox entry is stored in its own SAS catalog. This has the advantage of easy maintainability and eliminates the need for duplicating Toolbox entries. This last point is significant as there are several commonly called Toolbox entries, such as MESSAGE.FRAME, a FRAME tool which is functionally equivalent to the previously discussed MESSAGE.PROGRAM.

Given this architecture, it would seem necessary to either hard code all CALL DISPLAYs to the MESSAGE tool main entry using a 4-level name; or use the SEARCHPATH function to specify a catalog search order.

Using search paths means that all tools must be declared in the catalog search path. As the number of tools grows, the search path specification will become long and performance becomes an issue.

There are two key setup requirements for the Toolbox ADE which allow calls to be handled transparently. The first requirement is that the TOOLBOX libref is assigned to the directory in which the Toolbox resides. Essentially, this is equivalent to treating the TOOLBOX libref as an extension of the SAS System, much like the SASUSER and SASHELP librefs which are always available.

The second requirement is the existence of the %TBOXPATH macro, which defines the default 4-level name for every AF tool. When an AF tool is referenced from within an application, it is coded using the %TBOXPATH macro which resolves to the correct 4-level name at compile time. For example:

```
/* Invoke AFSCL Message tool */
call display(%TBOXPATH("MESSAGE"));
```

would normally resolve to:

```
call display("TOOLBOX.MESSAGE.
MESSAGE.FRAME");
```

when the AF entry is compiled.

The setup for the TBOXPATH macro is handled

by the TBOXPATH.SCL entry, part of the SCS Toolbox application. This initializes a series of macro variables that are required by the TBOXPATH macro, using the STARTUP variable in the Toolbox.INDEX data set. The macro variables are of the form:

```
%let name=startup;
```

where "name" is the name of the catalog entry.

For example, the MESSAGE catalog has a STARTUP value of MESSAGE.FRAME. Hence, the following macro variable will be set:

```
%let message=MESSAGE.FRAME;
```

The SCL behind Toolbox.INDEX ensures that the seven letter root will be unique. When the TBOXPATH macro is invoked, it obtains the name of the entry point for the specified tool by accessing the macro variable set earlier by the TBOXPATH.SCL program.

A developer can optionally override these specifications by compiling a macro named TBOXOVRD which contains %LETs that do the overriding. The TBOXOVRD macro is a dummy macro that is invoked by the TBOXPATH macro. It is only useful when it is overridden by the developer's own TBOXOVRD macro. The developer will typically have specific macro assignment statements that have the effect of overriding the AF entry name of the Toolbox program to execute. For example, the following statement in the TBOXOVRD macro:

```
%let message=
SYSTEM.WINDOWS.MESSAGE.FRAME;
```

results in the execution of SYSTEM.WINDOWS.MESSAGE.FRAME when %TBOXPATH(MESSAGE) is specified in the calling program.

A developer can also override the library that is used by specifying the following in the TBOXOVRD macro:

```
%let libref=MYLIB;
```

If the developer specifies a null LIBREF:

```
%let libref=;
```

all entries will resolve to a 2-level name and therefore will reside in the "home" catalog.

The developer also can override the catalog that is used by specifying the following in the TBOXOVRD macro:

```
%let catalog=MYCAT;
```

If overrides are assigned, the developer will need to recompile all entries that invoke the TBOXPATH macro. This is a more manageable effort than that which might be required if the TBOXPATH feature did not exist.

Toolbox Playpen

When developing new Toolbox entries, a developer will normally like to be able to reference existing Toolbox tools which are in a read-only library by default. However, the catalog in which development occurs must be write-access. This concept of having a development area is sometimes referred to as having a "playpen".

The requirement to have the development and production areas essentially treated as one library can be implemented using library concatenation. This is done in SAS as follows:

Example:

```
libname TOOLBOX
    ("User's playpen SAS data library",
    "SCS Toolbox SAS data library");
```

This allows the Toolbox developer to test new or modified Toolbox entries while having all the other entries in Toolbox also available.

CONCLUSION

The critical importance of software reuse makes it imperative to manage toolboxes with a well-designed application. The SCS Toolbox application is an ADE which utilizes the special features and structure of the SAS System to make a large base of custom tools readily accessible to a wide range of developers. Through its use, the effective development of rapid prototypes and custom applications is significantly enhanced.

REFERENCES

Staff of SAS Consulting Services Inc., Rockville, MD (1994), "A Development Environment for Applications based on SAS Software". *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, 19, 1346-1349.

ACKNOWLEDGMENTS

The author thanks Michael Gilman, Greg Smith, Norm Shusterman, Don Henderson, Martha Johnson, David Septoff, and Smita Shukla for their contributions in the preparation and review of this paper. The author also gratefully acknowledges the staff of SAS Consulting Services for their many contributions to the SAS Consulting Services Toolbox, including those which appear as examples in the text.

SAS, SAS/AF, and SAS Consulting Services are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

ABOUT THE AUTHOR

Philip Van Dusen is a Software Applications Consultant with SAS Consulting Services in the Rockville office. He can be reached at 301-881-8840, extension 3329.

APPENDIX 1

Current partial listing of SAS Consulting Services tools:

CATEGORY

CHOOSE	Choose between 2 user-defined choices (e.g. Yes/No)	AFSCL
COPYKEYS	Use application specific keys for FSLIST, FSVIEW	AFSCL
CREATDS	Create SOURCE entry to re-create SAS data set	AFSCL
DATASRC	Creates and maintains specified data set(s)	AFSCL
DIRMGR	Directory Manager	AFSCL
DSTOLIST	Convert a SAS Dataset to an SCL List	AFSCL
FMTMGR	Format Manager	AFSCL
GETENTRY	Get a catalog entry (similar to BUILD window)	AFSCL
GRAPHER	Grapher: Generate graphs via point and click	AFSCL
GRAPHSEL	Graph Selector	AFSCL
LIBASSGN	Assign a Libref to a directory.	AFSCL
LIBCTLG	Select a Library and Catalog	AFSCL
LISTMGR	List Manager	AFSCL
LOGON	Logon Screen with Password	AFSCL
MENUBLDR	Menu Builder	AFSCL
MESSAGE	Display WARNING, ERROR or INFORMATIONAL message	AFSCL
MSGSPLT	Split Message Text at Word Boundaries	AFSCL
NOTYET	Inform User that selected items not yet available	AFSCL
PARMLIST	Table-Driven Parameter File Maintenance	AFSCL
PMENUMGR	PMENU Manager	AFSCL
RENCCLASS	Rename the Library.Catalogs of Methods in a CLASS	AFSCL
SAVEOPT	Save and Restore SAS System Options	AFSCL
SHELL	Standard Header Comment Block for All SCS programs	AFSCL
SRCMAINT	Display list of SOURCE entries to Run and/or Edit	AFSCL
VALIDATC	Validates entered char. value using dataset var	AFSCL
VALIDATN	Validates entered num value using dataset var	AFSCL
WALKER	Walking Stick Figure	AFSCL
WHEREBLD	Where Clause Builder Using Data Dictionary	AFSCL
WHERECLS	SAS/ASSIST Where Clause Builder	AFSCL
DATACTL	Data Control Object	CLASS
DATEPAD	Mouse driven keypad/Calendar to enter date values	CLASS
KEYPAD	Displays a KEYPAD for user to enter numeric values	CLASS
LNKDARRW	Custom Attr for the Linked Control Arrow Subclass	CLASS
RITXTENT	Referential Integrity Class	CLASS
COPYRITE	Copyright notice that follows the header block	MACRO
DRVPTFED	Print AF screens and SCL source code.	MACRO
EXPIRE	Compares today's date with the code's expire date	MACRO
GETUID	Get USERID for MVS, CMS, VMS, UNIX	MACRO
MAKEFMT	Drvr macro that calls a macro & create a fmt/infmt	MACRO
MOVEDTA	Move or archive records from a dataset to another	MACRO
MVARPRT	List of Global/Local variables and their values	MACRO
NOBS	Returns the number of observations in a dataset	MACRO
PGMTOSCL	Convert a PROGRAM entry to an SCL entry	MACRO
POSTSCRIP	Converts a flat file to a Postscript-Ready file	MACRO
PRINTCAT	Print all source entries in a catalog to a file	MACRO
PRTFSED	Prints source code and display screen for FSEDIT	MACRO
RSTROPT	Restores the system option specified as a parameter	MACRO
SAVEOPT	Saves current system option values to a macro	MACRO
SHELL	Sample comment block for a macro definition	MACRO
SPLTSDS	Split SAS dataset using values of a variable	MACRO
SYSLPUT	Puts local macro variable on remote host	MACRO
TPRINT	Print specified SAS dataset	MACRO
OOPSDEMO	BUILD.RESOURCE for all Toolbox Classes	RESOURCE
DMKEYS	Change and Restore the DMKEYS in a Session.	SAMPCODE
PROJECTR	Optimal Window Colors For Overhead Projector	SAMPCODE
SHELL	Sample comment block for sample code	SAMPCODE