

Developing Applications & Development Tools Using Release 6.11 of the SAS® System

Philip Van Dusen, Stuart B. Levine, and Merry Rabb
Consulting Services Department, SAS Institute Inc.

ABSTRACT

Since Version 6 of the SAS System was introduced, SAS/AF® software has been a powerful device for the development of both custom applications and reusable tools. The latest release of the product in Release 6.11 has augmented that power with the addition of many new widgets and features. These new features add to the list of items that can be developed as reusable tools and stored in a tools repository, or toolbox. This can make a toolbox an even more useful resource for applications developers, but also can add to the challenge of maintaining it. This paper will discuss some of the new features, how they can be used, and some concepts to consider when developing and maintaining a tools repository.

INTRODUCTION

The concept of a tool repository, or toolbox, of reusable screens, programs, and routines is not a new one. The idea of a toolbox has existed for many years. It began as a library or directory containing frequently used pieces of program code that multiple users could copy into their own programs or applications.

With the advent of the windowing environment, such as in Release 6.06 of the SAS System, toolboxes began containing more than just code segments. Developers began storing frequently needed windows used for menus or messages within their applications in repositories that their colleagues could access. As with the code segments, these would be windows that had been thoroughly tested and could be easily placed within an application rather than each developer creating their own versions of the same idea.

Now, using Release 6.11 of the SAS System, the types of members that may reside in a development toolbox can include subclasses of the supplied FRAME widgets, user-written classes, and even composite widgets. A composite widget is a single object that contains multiple objects that interact with each other. Once defined, an applications developer can place it onto a FRAME entry and

have it work without writing any Screen Control Language (SCL). The SCL would already be written as part of the composite by the class or widget developer.

The ability to develop new classes and to develop subclasses of existing classes has been available in SAS software since the first object-based environment was introduced in Release 6.08. However, the facilities for creating these classes have been greatly improved in the latest release, along with many new objects and features that have been introduced in Release 6.11. The combination of these new features and widgets allow a class developer to create very exciting composite widgets. If these composites are developed with reusability in mind, they can become very powerful tools that can enhance the application developer's ability for rapid prototyping or application development.

The list of new objects in Release 6.11 include:

- **DATA TABLE** - The Data Table widget displays part or all of a data set on a FRAME entry in table format.
- **DATA FORM** - The Data Form widget displays a data set in form format, allowing variables to be displayed as various FRAME widgets such as CSFs or list boxes.
- **EXTENDED TEXT ENTRY** - The ETE is similar to the standard text entry object, except it is scrollable to allow a smaller widget to contain a character value that is longer than the displayed widget.

Some of the new features also available in the new release are:

- **DRAG-AND-DROP** - This feature allows the user to mark an item in one widget and drag it to another on the same FRAME entry.
- **EVENT HANDLING** - This allows the developer to establish communication between objects without either object knowing where the event was initiated or where it has been sent.

- **COMPOSITE WIDGETS** - A composite widget is a user-defined widget containing two or more simple widgets that are coded to work together.

This paper describes two examples of composite widgets, one simple composite and another more complex one. Both are part of the tools repository used at the Consulting Services Department at SAS Institute.

COMPOSITE WIDGETS: GENERAL CONSIDERATIONS FOR TOOLBOX

A composite widget is a packaging of a constellation of widgets which work together to accomplish a useful task. This package is defined as an object class which can be added to resource entries and dropped into FRAME entries which are built for custom applications. The methods built into the class control all the functionality within the composite, including interactions of the components (such as cross-validation checks for paired data entry widgets). The availability of composite widgets in Release 6.11 opens an enormous, virtually unlimited vista of possibilities for reuse. A toolbox which includes composite widget tools could increase in size by orders of magnitude, inducing a corresponding increase in the challenges of managing its growth and planning for the consistency of its future direction.

A superficial contemplation of combinatorics illustrates why the possibilities for designing composite widget tools are essentially unlimited. For instance, consider the number of distinct visual classes supplied by the SAS System in Release 6.11 (greater than 30) and imagine that there is a potentially useful composite class for any selection of 5 widget components. A given component widget (such as a list box) can appear more than once in a composite, as one of our example tools will illustrate. Hence, there would be at least $30^{*5} / 5!$ (202,500) distinct combinations for composite classes limited to 5 widget components. Of course, many of these combinations would not make sense in a practical user interface, but it must also be observed that any one of these widget combinations could be used to build a very large number of distinct composite classes because different instance variables and methods may be attached to a given widget combination. In fact, the composite class has been designed so that any FRAME entry could be repackaged as a functionally equivalent composite widget tool. Although it would not make sense to do this in most cases, it is a possibility that is always available.

As with the development of the more traditional AF tools (such as stand-alone FRAME entries, or several AF entries working together), it is natural to expect that the development of new composite-widget tools will be driven in tandem by the creativity of developers and by the needs of users. It is also likely that many "natural" composites will be recognized in existing custom applications that were built prior to Release 6.11.

There are some distinctive reuse advantages conferred by composites. By their nature, composites involve interaction of components and the code behind such interactions often involves more complexity than code behind a single widget. Hence, the development time saved by subsequent reuse of a composite is greater. Also, many composites can comprise most or all of the functionality behind a given custom FRAME entry, sometimes making it unnecessary to develop any FRAME-specific SCL. As a result of these special advantages, the general planning and design considerations for composite widget tools will often be more critical than those for traditional AF tools. Some typical questions/decisions that might be raised during the design phase include:

- Can we design the composite widget tool so that it better lends itself to further subclassing?
- Do we prefer a version of a composite widget tool which consumes less screen real estate and is less user friendly, or vice versa?
- How do we size the components for the most effective user interface?
- If a stand-alone FRAME can be repackaged as a composite widget, do we maintain both versions in the toolbox?

To ensure that these issues are addressed consistently, it may be necessary for an organization to maintain a tools advisory committee that devotes special attention to contributed tools which are packaged as composites. Such a committee may have associated roles, such as deciding what classes of potential composite tools should have the highest priority for implementation. It is quite likely that there will be preference in demand for small, well-defined composite tools which are relatively unrestricted and can be used in many types of applications.

SAMPLE TOOL #1: SLIDER/TEXT ENTRY COMPOSITE

The first example of a composite widget is a simple one. This composite consists of a slider object and a text entry object. Both of these objects were available in Release 6.08. However, the ability to package these two objects together as a single entity significantly increases their usefulness and possibilities for reuse as a tool. The purpose of this composite is to allow the user to move the slider to set a numeric value on the FRAME entry rather than having to type the value.

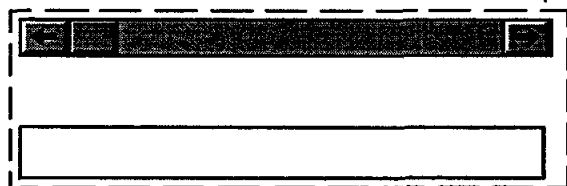


Figure 1 Slider/Text Entry Composite

To follow recommended toolbox naming conventions, the class definition for the composite is stored in a catalog named CLASSES with the CLASS object stored as _STXT.CLASS and the associated method code stored in _STXT.SCL. The test/demo program is stored as _STXT.FRAME in a catalog named CLSDemo. The CLSDemo catalog is also supplied with the RESOURCE entry which is necessary for adding the new tool to a FRAME. For a composite widget, a minimal but adequate demo program consists of a single test FRAME entry on which the composite widget has been placed by itself. A simple TESTAF serves to test the new tool.

The method code used to make this composite work is:

```
/* These statements are used to eliminate
  COMPIL time messages. */
_self=_self;
slider=slider;
text=text;

/* The name of this labeled section is defined
  in the CLASS definition. It will be used to
  override the _OBJECT_LABEL_ method */
OBJECT_LABEL: /* _OBJECT_LABEL_ override */

method;

/* This statement is used to inherit the
  characteristics of the default method. */
call super(_self, '_OBJECT_LABEL_');

/* These two statements capture the numeric
  position of the slider and use it to set the
  value of the text entry widget. */
call send(slider, '_GET_VALUE_', value);
call send(text, '_SET_TEXT_', put(value, 3.));

endmethod;
```

The behavior of the composite slider/text entry is straightforward and intuitive:

- Initially, the slider is set at its left-most position and the text entry object is set to the slider's minimum value.
- As the user moves the slider by either clicking on the right or left arrow or moving the thumb, the numeric value of the slider is reflected in the text entry widget.

SAMPLE TOOL #2: COMPOSITE IMAGE VIEWER

This section describes a composite widget tool which incorporates an image viewer, one of the widgets supplied by the SAS System which is new in Release 6.11. This tool will be referred to subsequently as the Composite Image Viewer (see Figure 2). It consists of the following components:

- CATALOG - A list box which presents a list of available SAS catalogs
- ENTRIES - A list box which presents a list of entries in a selected SAS catalog
- CHECK BOX - A check box which subsets the catalog list to those containing images
- UP - A control widget used as a scroller
- DOWN - A control widget used as a scroller
- IMAGE - An image viewer widget (displays bit-mapped images)

Notice that all of the object components described above, except for the image viewer component, were available in Release 6.08. By itself, the image viewer widget allows the user to select an image (catalog or external file) at build time, with methods available to change the image at run time. The composite image viewer makes it easy to browse a list of image entries and select any one for display at run time.

In staying with the recommended toolbox naming conventions mentioned in the previous section, the class definition for the composite is stored in the CLASSES catalog as _IMGV.CLASS, and the associated method code is stored in _IMGV.SCL. The test/demo program is stored as _IMGV.FRAME in a catalog named CLSDemo. This class also uses the RESOURCE entry used by the previous

example. A simple TESTAF on the demo program serves to test the new tool.

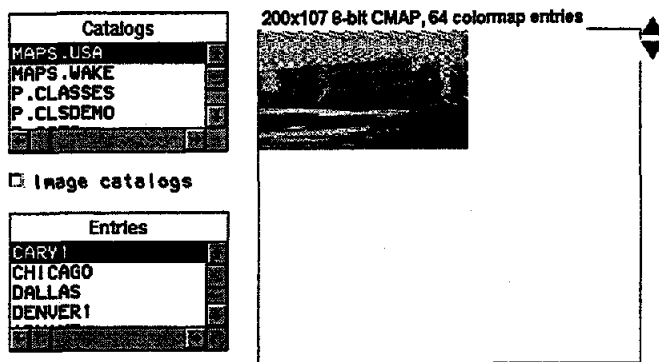


Figure 2 Composite Image Viewer

The behavior of the composite image viewer is straightforward and intuitive:

- Initially, the CATALOG list box is populated with a list of all available SAS catalogs from the assigned SAS libraries. The ENTRIES list box contains a list of all image entries in the first listed SAS catalog; or it may be empty if there are no image entries present. The image viewer widget displays the first image entry in the ENTRIES box, if there are any.

- If you select the CHECK BOX, the CATALOG list box is subset to those catalogs which contain at least one image. (There is a short delay while this processing is performed.) The ENTRIES list box shows the images present in the first catalog in the CATALOG list box.

If you change the catalog selection, the ENTRIES list box displays the image entries contained in it and the image viewer widget displays the first image.

- If you change the selection in the ENTRIES list box, the displayed image changes to match the selection.
- If you select the DOWN (or UP) scrolling widget, the selected entry in the ENTRIES list box moves to the next (or previous) entry and the corresponding image is displayed in the viewer. If the scrolling reaches the last (or first) entry in the ENTRIES list box, the DOWN (or UP) widget is grayed until a different selection is made.

- When an image is displayed in the image viewer, image attribute information is displayed along the top border.
- When an image is displayed and you click inside the image viewer widget, a pop-up menu is displayed to allow you to print the image or copy it to the clipboard.

This example is derived from a tool that was actually contributed in the form of a standalone FRAME entry, then was repackaged as a composite. This repackaging makes it easy for the composite to be used in conjunction with other widgets on a common FRAME for a specialized application. For instance, the composite could be used on a FRAME in which the viewer was used to make selections from a set of photographs.

It should be understood that the implementation of this tool as presented above is by no means the only alternative available. For instance, there are other mechanisms for selecting a catalog entry which consume less screen real estate than two list boxes. Also, it may be decided that the usability of the tool would be enhanced by allowing display of more than one image at a time. Or it might be desirable to apply some initial subsetting to the images that are displayed. Alternatively, it might be decided that the implementation of the tool is better achieved as a standalone FRAME.

Regardless of the eventual form of these tools, it is certain that every subsequent reuse will repay the organization the equivalent of their development time, which in this case could easily add up to more than a week, when R&D, programming, and formal QA time are taken into account.

TOOLBOXES

When creating an applications development toolbox, there are a number of issues you will want to address before you start. For example, what will the contents of the toolbox be? Will it consist entirely of SAS/AF classes, or will you also have other SAS/AF and SCL code modules, custom EIS objects, SAS macros, other SAS source code modules, KEYS or FORM entries, and so on. You may even decide that your toolbox needs to contain some related items other than SAS code that are specific to your computing environment such as UNIX scripts, or files of MVS Job Control Language (JCL).

The next step is to decide how to implement storing these items. Ever since Release 6.06 of the SAS System, we have chosen to store our Consulting

toolbox items in catalogs. This allows us to store all modules in SAS files rather than in operating system files, and thus, allows us to do all the file maintenance within SAS and makes it easier to transport the files between operating systems. Originally we had a single catalog for all macros, another catalog for all other code modules, and so on. For each SAS/AF or SAS/EIS® tool there is one catalog with a unique name that corresponds to the name of the tool. The reason for having a separate catalog for each of these tools is that a tool of this type could consist of more than one catalog entry, plus this structure allows for the storage of a README.SOURCE entry and demo entries along with the tool components.

Storage issues become somewhat more complex when you begin including SAS/AF classes in the toolbox. It is still possible to have one uniquely named catalog for each class, but it is very important that there be naming conventions that support making the name of every individual entry unique. The reason for this is that once a class has been developed there is no simple way to globally change the name and the references to an individual entry that is part of the class. One possible approach is to give each tool a unique name that is five characters or less long. Each entry that is part of the class would have a name that includes this class name, still leaving 3 or more characters to use for prefix and suffix characters. Considerable thought should be given to naming and storage convention issues.

There are some other concepts to keep in mind when beginning a tools repository:

- **TESTING, TESTING, TESTING** - Any item included in a toolbox, no matter how small or simple, must be tested thoroughly. If not, the risk of including a problem window or class in multiple applications exists.
- Any item placed into a toolbox should be either very generic or provide the means to pass parameters to it. A toolbox item does not help in rapid application development if it can only be used in specialized circumstances.
- Members in a tools repository should include thorough documentation, both technical and general. Technical documentation should be provided in case modifications have to be made to the member. General documentation provides the application developer with a description of the member and how to use it.

TOOLBOX APPLICATIONS

A toolbox should also be more than just an organized and tested collection of tools. An application to manage the toolbox entries and make them easily accessible to applications developers is needed. At minimum, the toolbox manager application should provide a directory of tools, a way to find out more about the tool, and a facility for allowing the tool to be copied into the user's own catalog. Ideally, a lot more is possible. For example, a front end to the toolbox could be provided that would help the developer find tools applicable to his or her project by providing not only a directory listing the tools but a way to categorize the tools by type of tool or by basic function, keyword searching, and so on. A toolbox can become quite large and if an application developer has to spend a lot of time searching through many unfamiliar tools to see if any might be useful, they are likely to become frustrated and decide not to bother.

Once a candidate tool is located, the developer should be able to look at index or detail information about the tool such as a brief description of function, notes on what releases and/or operating systems the tool has been tested for, other dependencies the tool may have, and whom to contact for technical support of the tool. The developer should be able to access the catalog(s) to view the source code, run a demo program that illustrates the use and the look and feel of the tool, and have automated features to copy or print the tool's modules.

Other things that could be part of a toolbox manager application include an online facility for people to contribute tools they have developed or request the development of new tools; a facility for people to report bugs or suggest enhancements to existing tools; a facility for creating a copy of or a subset of the toolbox for temporary use, for example on a laptop when traveling to another site.

USING A TOOLBOX

Sometimes organizations invest time and energy to create toolboxes and still find that it is not used anywhere near as much by the applications developers as the organizers think it should be. If this happens, take a look at the reasons people sometimes give for not using the toolbox:

- "it's too hard to find out what's in it or to find tools applicable to my situation". This can be at least partially addressed by having a really flexible search or keywords mechanism built into the toolbox manager application. The issue

should also be addressed by educating people on the contents of the toolbox, announcing new tools as they become available, having periodic demos and "news bulletins" to make sure the toolbox and its contents are very visible.

- "It takes too long to figure out how a tool works or how to integrate it into my application". Make sure all tools are thoroughly and consistently documented. Make sure the people who support development of the tools are available to help people implement the more complex tools, if possible.
- "the last time I used something from toolbox it didn't work". As we said before: TESTING TESTING TESTING.
- "I just don't have any reason to use it, I can develop things myself". Actually, most applications developers don't come right out and say this, but sometimes people are not using centrally located tools, or not making efforts to reuse code, and there is no apparent reason. The developers are just developing everything from scratch, even though using a toolbox improves employee productivity and turn-around time for the end product. This is the most frustrating toolbox-related problem for managers who want the efficiency and reliability of reusable tools but find some staff members just aren't using them. One way to overcome this is to somehow reward code reuse, possibly by making participation (using a toolbox, contributing to a toolbox) part of your employee evaluation criteria.

CONCLUSION

It soon becomes clear that creating a central toolbox involves a lot more than just pulling together a bunch of programs and tools and storing them in one place. Plus it is somewhat unrealistic to assume that the original developer can take the time to make their tool completely generic, make changes to correspond to any toolbox naming or coding conventions, and fully test and document the modified version. Creating, maintaining and promoting the use of a toolbox in an organization can be a lot of work. We have found that toolboxes are most effective if there can be staff members assigned to spend at least part of their time soliciting, organizing, modifying and testing the tools.

As software technology continues to evolve and new capabilities and features are added to the SAS

System, it becomes possible to develop more powerful classes and other tools. A repository of reusable code modules is even more useful when the tools are SAS/AF classes that are well tested and documented and that require little or no additional SCL coding when added to an application. Extra effort is required to put together such a toolbox, and we think the payoff in saved application development time and increased reliability makes it well worth the effort.

SAS, SAS/AF and SAS/EIS are registered trademarks or trademarks 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.

REFERENCES

Van Dusen, Philip, SAS Consulting Services Inc. (1995), "Development Facilities and Tools for SAS Applications", *Proceedings of the Twentieth Annual SAS Users Group International Conference*.

ACKNOWLEDGMENTS

The authors would like to thank Dave Horne for supplying the Image Viewer example. Thanks to Linda Fineman, Hari Gupta, Patti Brideson, Pam Katchuk and Jull Staub Perry for supplying examples which could not be presented due to space limitations. Thanks also to Martha Johnson and John Patterson and others for help in reviewing the paper.

The authors may be contacted at:

Philip Van Dusen
SAS Institute Inc.
1700 Rockville Pike
Suite 600
Rockville MD 20852
(301) 881-8840
SASPHD@MVS.SAS.COM

Stuart B. Levine
SAS Institute Inc.
1700 Rockville Pike
Suite 600
Rockville MD 20852
(301) 881-8840
SASSZL@UNIX.SAS.COM

Merry Rabb
SAS Institute Inc.
SAS Campus Drive
Cary NC 27513
(919) 677-8000
SASMGR@MVS.SAS.COM

•