

SAS / AF "Gets Focus"

By Cheryl Olson, Trilogy Consulting Corporation

ABSTRACT

The ability to detect the exact moment that a widget becomes active or inactive enables SAS/AF® Frame entry developers to incorporate functionality into applications that would otherwise be unattainable. In Microsoft Windows®, these events are also referred to as "get focus" or "lose focus". A new feature provided to SAS/AF developers in release 6.11 enables this capability by way of the `_TAB_IN_` and `_TAB_OUT_` methods.

A widget gets focus or becomes active when the user tabs to the widget or selects the widget with a mouse click. The same widget loses focus or becomes inactive when the user tabs to any other widget or clicks anywhere outside of its region. This paper will suggest how these methods should be used and demonstrate the inability to achieve the same functionality without a `_TAB_IN_` or `_TAB_OUT_` method.

INTRODUCTION TO THE `_TAB_IN_` AND `_TAB_OUT_` METHODS

Before discussing the application of the `_TAB_IN_` and `_TAB_OUT_` methods, an understanding must be established for what the `_TAB_IN_` and `_TAB_OUT_` methods are and when they execute.

A widget's `_TAB_IN_` method executes when the widget "gets focus" or gains control of the application within a frame. Typically, the user can easily identify which widget on a window has focus. For example, when a text entry field has focus a flashing cursor is visible within the text entry field's region. While the text entry has focus, the user's key-strokes will result in the entry of characters into the text entry field. A second way the user can identify which widget has focus is when a dark border appears around the widget. For example, a push button or command button's border will appear bold when it has focus. If the user presses the enter key while a button has focus, the SCL code will be executed for that button.

In a SAS/AF® application, only one widget may have "focus" at any given point in time. As mentioned previously, focus is transferred from one widget to another in one of two ways: the user may use the TAB key to transfer control to the next widget on a frame, or the user may use the mouse to click on the widget to which s/he would like to transfer control.

To contrast the `_TAB_IN_` method, SAS/AF provides the `_TAB_OUT_` method which executes for a widget when that widget "loses focus", and thereby another widget on the same frame has contemporaneously gained focus. For example, if the user wishes to transfer control from widget A to widget B by pressing the TAB key, the `_TAB_OUT_` method will execute for widget A and the `_TAB_IN_` method will execute for widget B.

The availability of the `_TAB_IN_` and `_TAB_OUT_` methods which execute when a widget gains or loses control enables the developer to incorporate into applications functionality that could never before be accomplished. Prior to version 6.11, a developer could detect which widget had control of a frame using the `CURFLD` function within that frame's MAIN section as follows:

```
MAIN:
select (curfld() );
    when ('NAME') link SAMPLE;
    otherwise;
end;
return;
```

In this example, each time MAIN executes `CURFLD` returns the object on which the cursor is located. If the object returned is 'NAME', the code is executed in the label 'SAMPLE'. Although this code will accomplish your objective if you need only to determine which widget has focus, it does not indicate when the widget gained focus.

The ability of a SAS/AF application to wait for an event to occur (such as `_TAB_IN_` or `_TAB_OUT_`) and then proceed based on that event presents to the developer a challenging opportunity to develop modularized, asynchronous code rather than the traditional procedural code which runs from start to finish. Programming within this paradigm is commonly referred to as "event-driven" programming.

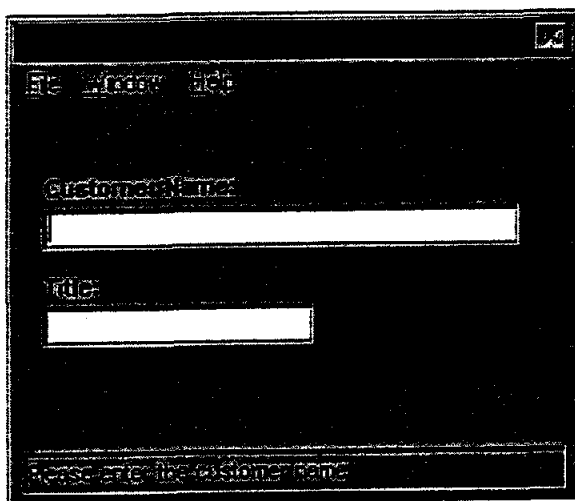


Figure 1

USAGE EXAMPLES

One example of desired functionality which often requires implementation of the `_TAB_IN_` and `_TAB_OUT_` methods is when the developer would like to associate some type of user help with the object that has focus. For example, in Figure 1, the developer would like to display user help in the lower portion of the window (commonly referred to as micro-help) for the object which has focus. When the 'Customer Name' text box has focus, the micro-help should display 'Please enter the customer name'. When the 'Title' text box has focus the micro-help should display 'Please enter the customer's professional title.'

To begin, let's address why the proposed functionality could not be accomplished before version 6.11.

The developer could have followed the previous example and set the micro-help in the MAIN section of SCL as follows:

```
MAIN:
  select (curfId() );
    when('CUSTNAME')link NAMEHLP;
    when ('TITLE') link TITLEHLP;
    otherwise;
  end;
return;

NAMEHLP:
  call notify ('microhlp',
    '_SET_TEXT_', 'Please enter the
    customer name.');
```

```
TITLEHLP:
  call notify ('microhlp', '_SET_TEXT_',
    'Please enter the customer's
    professional title.');
```

```
return;
```

This approach would eventually accomplish our objective although the exact time at which the code executes presents a problem. The MAIN section of scl for the frame executes when the user strikes the ENTER key or a function key. It does not execute when the user strikes the TAB key or clicks a mouse button. Therefore the following sequence of events could occur:

(To begin, the cursor is automatically positioned in the customer name field and the micro-help displays 'Please enter the customer name'.)

Event 1: The user types in the customer name and does not strike the enter key.

Event 2: The user utilizes the mouse to transfer focus to the title field. Note that the MAIN section of scl has not yet run, so the micro-help continues to display 'Please enter the customer name.'.

Event 3: The user enters the customer title.

Event 4: The user utilizes the mouse to make a selection from the window menu to save the information.

In this example the user was never able to view the micro-help that was associated with the customer title entry field and should have displayed the message, 'Please enter the customer's professional title.'. The MAIN label did not execute while the title field had focus. If this were a data sensitive, mission critical application for which the micro-help displayed warnings relative to the data entered, the results could threaten the integrity of the data and the accuracy of the system.

An alternative solution which would enable the developer to achieve all of the desired functionality at the appropriate time, utilizes the `_TAB_IN_` and `_TAB_OUT_` methods. The steps for implementation are as follows:

Step 1: From the SAS® provided extended text entry field, create a subclass to be used for the customer name field and the customer title field.

Step 2: Create a character instance variable string for the class. Call it `is_microhelp`.

Step 3: Create a custom attributes frame for entry of the micro-help text upon creation of an instance at build time. Your custom attributes window should look similar to the frame shown in figure 2.

The SCL for this frame should resemble the following code:

```
entry _widget_ 8;  
length user_val $200;  
  
OK:  
  call notify ("microhlp", "_get_text_",  
              user_val);  
  rc = setnitemc (_widget_, user_val,  
                 "is_microhelp");  
  call execcmd("end;");  
return;
```

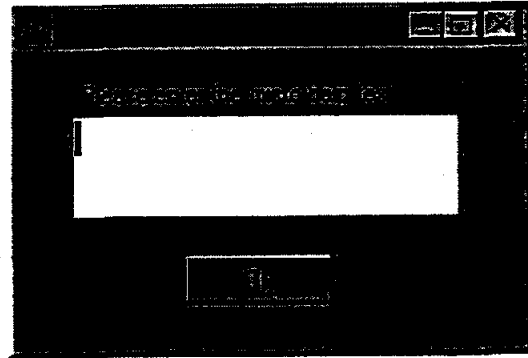


Figure 2

Step 4: Next, override the `_TAB_IN_` and `_TAB_OUT_` methods. Add the following code for each method

```
TAB_IN:  
  method;  
    call super (_self_, '_TAB_IN_');  
    call send (_frame_, '_set_msg_',  
              is_microhelp);  
  endmethod;  
  
TAB_OUT:  
  method;  
    call super (_self_, '_TAB_OUT_');  
    call send (_frame_, '_set_msg_',  
              ' ');  
  endmethod;
```

The suggested code will set the status bar to the string value stored in the instance variable `is_microhelp` when the object gets focus. When the object loses focus, the status bar is set to blank. For each text entry field required by the application we only need to add an instance of our new class. We enter the micro-help text at build time via the custom attribute window; no additional coding is required for adding the microhelp functionality for newly added widgets.

Another use for `_TAB_IN_` and `_TAB_OUT_` would provide field validation in the `_TAB_OUT_` method to ensure that the user-entered information is correct before the user can proceed. Collaborating widgets

could also be updated in the `_TAB_IN_` or `_TAB_OUT_` methods as in figure 3 below.

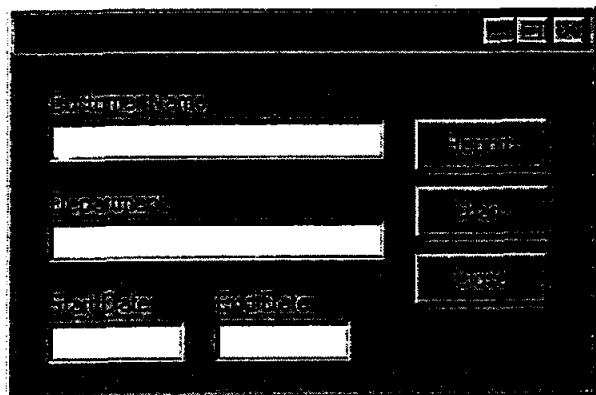


Figure 3

In figure 3, the user must enter a customer name, department name and the date range for which s/he would like to analyze sales information. In order to make the application flexible, these values are additionally stored in a non-visual object. This non-visual object also makes the information easily accessible to other parts of the application.

After the user has specified the required information, s/he may choose to view sales reports for that information, view sales charts or to exit the application (by choosing 'Close'). The 'Reports' button opens a frame which displays a report that is created using the data provided by our non-visual object. The chart button works similarly, only it displays a graphical presentation of the same data. This design presents considerable dependence on the availability of the `_TAB_IN_` and `_TAB_OUT_` methods. From the `_TAB_OUT_` method for each widget, a method on the non-visual object is called to update its value for that widget. As a result, the non-visual object always contains the most recent user-provided values for customer, department, start date and end date.

In version 6.10 we may have placed the same code in the labeled section of SCL for each widget within the frame. As a result, the non-visual object would not update after a mouse click. Therefore the user could change the start date, click the 'Reports' button and the report frame would generate a report for the incorrect date range since the non-visual object had

not been updated and thereby providing the report frame with obsolete information.

CONCLUSION

The addition of the `_TAB_IN_` and `_TAB_OUT_` methods to SAS/AF® release 6.11 further empowers the frames developer to create interfaces that present the look and feel of Microsoft Windows applications. The event-driven nature of such methods results in interfaces that are easy to use and understand, thus are very pleasing to the user.

ACKNOWLEDGMENTS

Thanks to Andy Norton, Lorie Livingston, Sally Goostrey and Mark Olson for reviewing earlier drafts of this paper.

The author may be contacted at:

Cheryl A. Olson
Object Technology Group
Trilogy Consulting Corporation
5278 Lovers Lane
Kalamazoo, MI 49002
(616) 344-4100 (voice)
(616) 344-6849 (fax)
Internet: 102732.11@compuserve.com