Screen Design For Frame Entries

Don Stanley
Information Power Limited
Wellington
NEW ZEALAND

Overview

"An application interface should be visually, conceptually, and linguistically clear. Visual elements
should be immediately comprehensible, ideally because they relate to real-world analogues, and
should be arranged so that their functions are comprehensible. Conceptual metaphors should be
simple and realistic. Interface text should be clear, unambiguous, and free of jargon”.

The preceding statement from Microsoft is an excellent comment on the ideal GUI interface. To try and
achieve this, this paper presents some ideas for designing screens used with Frame applications. The ideas
presented here are a combination of tried and tested techniques, and my own ideas that I have found work.

Please note that unless stated otherwise, statements made about SAS Software and code presented are up to
the release 6.10 level. This paper is an abridged version of a chapter in my forthcoming book on Frame.
As such, it is essentially incomplete. Please be aware of this as you read the paper. Many more topics will

We know GUI development is different to the old “menu driven, select a number and be branched off
somewhere to fill out a couple of fields” type things, that we used to see. But how appreciative are we that
the GUI is not just a pretty face. It is also often the only thing that a user bases their opinion of a system on.

How much effort do we place on making the GUI look good, feel good and act responsively and
intuitively?

A point to raise at this stage. Who uses our systems? What right have we to dictate to the end user how their system interface will look? When developing GUI applications in an organisation, work to accepted organisation standards and involve users in
deciding how the look and feel of the screen will work.

As developers, few of us are trained to develop screens that are ergonomic. We know that if the user clicks
a certain 'box', certain events must happen and we code those events with relish. But how much thought
goes into the following topics covered in this paper:

- Error message handling
- Data flow when tabbing and automatically moving to following fields
- Placement of action widgets on screen
- Which widget to use
- Popmenus
- Screen sizes & related issues
- Menuing in a GUI environment
Error Messages

The user has a right to expect meaningful error messages. They should be concise. All fields in error should be able to have an error displayed. All errors should be displayed in the same place. Error messages should follow the same format. From 6.11 the _MSG_ area is available irrespective of whether a command line exists. This is the most sensible standard position for display of single error messages.

If your application generates multiple messages, try displaying the messages in an extended table. Display the table in the first two lines of the screen, or the last two near the existing message area. By using the extended table (or a listbox) your user can scroll through messages as necessary. Store the messages in a list for easy manipulation.

This is fine up to and including 6.10. However 6.11 offers a vastly superior way of displaying messages. At least, it offers all the facilities you need to create a vastly superior way of displaying messages. You essentially do the following:

- Switch errors on for each field in error.
- Enable cursor tracking
- When the _CURSOR_TRACKING_ method detects the mouse moving over a field in error, instantiate a custom graphics based widget (extended text entry is a good one) and display it on screen (e.g. in the top right hand corner or right above the region in error). Destroy it when the user either clicks to leave the cursor on a field, or when another field in error is moved onto.

I have implemented this under 6.11 beta. A composite object has been constructed to allow this. I created a graphic in Powerpoint, then embedded an extended text object into it. The graphic pops up as above when a field is in error. The screen looks like the following when the cursor is moved onto the field in error. When the cursor moves off the field in error, the explanation box disappears again.

I am not presenting the code here, mainly because it was written using 6.11 beta and I want to ensure it works optimally in the production release. It will be presented as a fully functional Frame object in my forthcoming BBU book.

Difficulties With Error Processing.

Working with Frame can sometimes be difficult to co-ordinate error messages. Suppose a screen has a number of fields and the user uses the mouse to place the cursor on the 5th field. They enter an incorrect value. Do we display messages for fields 1 to 4 or for field 5. We cannot force processing of the labelled sections for 1 to 4 (up to and including 6.10) so how do we cause processing to occur?

The following are issues we have to consider. There is no hard and fast answer as to how we cope this
these issues. The literature and courses on screen design that I have attended simply don’t address these issues.

1. Suppose a user fills out a number of fields. The 3rd and 5th ones are in error. We switch ERRORON for each of these and display a message for the third one. The user corrects the 5th one but doesn’t change the 3rd. Because the label for the third doesn’t get carried out by default, no message is displayed but the 3rd field is still in error.

CONTROL ALWAYS doesn’t help here. Even though the third field is still in error no message is displayed as the labelled section didn’t run so the message could not be initialised.

Can we get round this? There are ways. We could create a variable in each labelled section that flags when a labelled section is run. In main, check for the existence of these variables and link to any section that doesn’t have the variable set. Then set all the variables to ‘not run’ before going back to the GUI. You must have CONTROL ALWAYS to use this. The variables need to set to ‘not run’ in MAIN because there will be nowhere to set them when processing starts again. This is because we have no section that routinely executes prior to all labelled sections, and we cannot set at the start of each section as we have no guarantee that any given section will execute. This of course causes extra work in MAIN.

2. If multiple sections find fields in error, by default the last one is the one that _MSG_ gets written for. You need to change the logic flow a little to include a check whether _MSG_ contains a value and not assign a value if so. If you don’t do this, the user sees error messages appearing in an illogical order.

3. You can bypass labelled section processing completely and just execute MAIN, linking to each labelled section. This has some distinct advantages in bypassing the problems in (1) and (2) but defeats the purpose of modular programming to some extent, also causing a lot of extra work to be done unnecessarily. It also makes life difficult bypassing sections that should not execute, e.g. hidden fields etc.

Error message layout

You can have several types of message in an application, E.g. Error, Warning, Notes. Errors prevent the application continuing. Warnings are a suggestion to the user that although the data is valid, it may not be what they want to use (E.g. a report covering a period that is very old). Notes are purely informational.

Start each message with the word ERROR, WARNING or NOTE or any other term you use. Make sure the user is well aware from the message what needs doing.

Whatever you do, ensure that the display of messages is logically in sync with the placement of the corresponding fields on screen. And also, ensure that the cursor is set on the field that the error message is displayed for.

Fields that the user hasn’t yet changed or filled in

Arguments exist that such fields should not return errors until actually filled in. The fact is that with Frame the labelled section won’t execute until the field is filled in anyway. A generic message that the field should be filled in should suffice. Issuing error messages that a field value is invalid when the user hasn’t yet put anything in there appeals to the pedant in both users and testers. Speaking from experience!

You may find it easiest to issue such messages during the TERM processing. At that stage you can check for any required fields not being present and issue an appropriate error message. Or you could set the REQUIRED attribute to ensure the field is entered.
Personal preference is to not use the REQUIRED attribute because I usually want to issue a more context specific message than the SCL default.

**Data flow when tabbing, and automatically moving to following fields**

One of the first factors to consider when planning your GUI screen is the logical flow from field to field. Does TAB always take you to a sensible place or are you routed somewhere unusual? Does ENTER cause processing to be carried out that triggers errors on fields not yet filled in? Should pressing ENTER automatically end the window and return to a parent or spawn a child?

There are certain generally accepted rules of screen layout and processing flow. The following are discussed further in this section:

- when you tab to a group of related fields, tabbing should proceed through the related fields which should be on screen before following unrelated fields, i.e. don't cause tabbing to jump up and down the screen
- fields that cannot be tabbed to should be hidden, grayed or protected
- the first data entry field on screen is where the cursor should be placed on entry
- don't do error processing for fields that haven't yet been filled unless the user tries to advance to the next function. However messages requiring the field to be filled in are fine.

A rule that I don't discuss but is often overlooked involves using a data entry screen that enters data written on a pre-formatted paper form. You should design the screen to follow the form.

**Groups of fields**

Frame allows the concept of groups of fields. When the user tabs, by default the cursor moves from left to right, top to bottom around the tabbable fields. Once the user tabs to a grouped set of fields, the tabbing order follows the same rule, but within the group. Tabbing from the last element of the group moves back to the next logical element on screen. Since this may be above the last element of the group, care is needed to ensure that confusion does not occur for the user.

Tabbing can take users to widgets that are not obviously selected. Try to avoid this. For instance, setting the cursor on a graphic (e.g. SAS/GRAPH) region may not render the region as obviously being selected. GUI's should literally scream at the user 'here I am' all the time.

**Hidden and Grayed fields**

Frame allows hiding and exposure of fields under program control. This masks fields from users until some event occurs. Likewise you can gray fields, which means they exist and can be seen, but can't be tabbed to or selected.

There isn't an excuse for leaving selectable fields on screen if the current status of the screen is that the field is not relevant. Personally, I prefer graying to hiding as it means that the screen doesn't have disjointed chunks missing.

From 6.11 there is a new concept, swapping regions in and out. Swapping is quite similar to the concept of virtual memory. With virtual memory, when an application needs more memory it moves something not being used at the time out of the way. In Frame, you can swap a region out of the way when its not in use and reuse the space with another region.

Swapping has some distinct advantages over hiding or graying. For a start, you are not using screen real estate for a widget that is unusable. Secondly, you can swap a text region out and place another text region in its place. Thirdly, you do not have to overlay regions to achieve the same result as you did in earlier releases. I have one application that displays either a listbox or a graph, depending on user selection. The two regions physically overlap and the development environment looks messy. Swapping will allow one to exist on screen and the other to be swapped in as needed.
Swapping regions also requires that you instantiate at least one of the regions at run time. This permits the apparent overlapping of text based regions.

**Initial Field Placement and insert mode.**

The cursor should be placed on the first field that requires some data entry. Otherwise it is confusing.

A good idea for how to define field positions is to place the field that is most meaningful to the screen at the start. For example, if you code a field that requires a title to be entered and nothing more, make that the first field. If you require to enter some company details, make the most important one (e.g., Company name) the first. Reason -- it's a natural way that we work!

---

**SAS Trap**

A perfect example of poor initial field placement occurs on the REGION ATTRIBUTES -- SET TITLE frame entry prior to release 6.11. The cursor does not sit on the first field (on entry) which is the entry of the text. The bulk of the time, this field is the reason for accessing this window, yet it does not allow the user to just type when the screen is displayed. The cursor is actually defaulted to the OK pushbutton, but no highlighting of any description exists to show that that is where the cursor is placed.

---

One of the most irritating aspects of Frame is the apparent inability under program control to switch insert mode on or off. I nearly always want it off when a screen starts, but SAS gives no obvious override ability if the user has pressed INS earlier in a SAS session. However on some platforms the insert key can be turned off easily by issuing the command

```
call execcmd('winsert oW') ;
```

in the INIT section of the SCL. This is documented in the operating system companion manual. This is probably of interest mostly to SCL developers, so perhaps SAS Institute could look at having the same documentation in their SAS/AF, FRAME or SCL manuals.

You should make use of WINSERT OFF. It seems pointless to allow typists and developers to access a screen, with the cursor positioned ready to start typing, and be inhibited immediately or a few characters into the text. I would really like to see methods (i.e. OOP methods) that allow us to test the status of keys such as CAPS LOCK and INSERT and use that information to add a note, E.g. INSERT IS ON in the status area.

I think that on nearly every occasion that I have entered OBJECT ATTRIBUTES for a new region, the insert mode is on and a few characters into typing, the dreaded alarm occurs as the field hits its end. My belief is that I'm not unique in not pressing insert every time I access a new window, especially when my application software gives me no indication that I need to!

Incidentally, for comparison sake, Delphi, Visual Basic and Microsoft Access all have the ability to query and set the INSERT and CAPS LOCK status.

**Automatic branching on enter**

This is the sort of thing you see in some non SAS windows where a button such as OK or END is highlighted already when you access the window. The cursor on entry is placed in a different field, you can type and press ENTER without having to tab to, or select, the highlighted button. ENTER causes the command associated with the button to occur.

Such a logic flow is so tightly ingrained in Windows and OS/2 GUI screen design and interface that it seems a glaring omission in Frame. The fact is that when you see a window in SAS that does do the described activity above it is an operating system widget called from inside SAS -- not a Frame widget.
Applications Development

SAS Institute say you can simulate the above in Frame, by setting border colours and carrying out extra processing to trigger the command associated with the 'highlit' object. The extra processing includes undoing things when users tab beyond the default fields. I had a try, its awful messy.

The major issue is that pushbuttons actually have a much larger region than is visible on screen at run time. When you modify a border it is that region border, not the border of the visible pushbutton, that gets altered.

Note that the new 6.11 COMMAND PUSHBUTTON object may be useful here as the border of that has no extra character space surrounding it.

Placement of action widgets on screen

There are some activities that occur repeatedly in applications, for instance EXIT and CANCEL and HELP. Have a look at a few mainstream windows applications and you will see that the standard for placement of these involves drop down menus. EXIT and CANCEL usually are found under the leftmost menu (often a file menu) and HELP is usually the rightmost menu option.

Some products also place these on screen as buttons, but position, text and order tend to be inconsistent. Even in Frame system windows, you will find these at bottom of window or at the right of the window. End (OK) and Cancel are sometimes interchanged position wise. Incidentally, Microsoft's excellent text recommends the bottom of the screen or the right hand of the screen, but internal consistency from window to window and also order of buttons should be maintained.

As developers we do not need to use buttons in our applications. SAS Inst use pushbuttons for END, CANCEL and so on but our end-users don't necessarily use SAS so we don't have to inflict buttons on them if they are used to drop down menus in other applications. However, I believe that they are the best mechanism for triggering action commands -- intuitive, easy to set up and well ingrained in GUI philosophy.

If you use widgets to identify an action such as END be consistent from screen to screen. Don't swap the position of the END and CANCEL buttons in different screens.

I like SAS Institutes usage of the words GOBACK and OK, they imply exiting and further processing respectively and are preferable to using a straight out 'END' or words like 'PROCESS'. If you can use words like this that always retain the same meaning then do so.

Overall, although people often talk about 'Windows compliant' and applications looking like an extension of the underlying operating shell, I find that few people can actually attempt to define just what this means. Don't try to redefine a completely different layout, but follow acceptable guidelines as defined by other applications.

Explaining what options do.

My favourite GUI feature in many Windows products is the ability to display some text (a "hint") that gives a quick description of what a widget will do if you click on it. For instance, in Microsoft Word, move the cursor onto a tool bar picture and hold it there. A short piece of text appears telling us what will happen if the widget is clicked. These are often called TOOL TIPS.

You can do this from release 6.11 of SAS. This should cause a rethink into how we design GUI applications because it will remove the necessity to use ICONS when we want to mix graphics and text. I find ICONS often take up to much space and its to difficult to create your own. Having said that though, they are always very easy to visualise, and sometimes small graphic images are difficult to work out.

Using the CURSOR TRACKING method you can create these short text descriptions that appear as you move over the object. The 6.11 Frame documentation contains a lengthy and necessarily complex discussion (with code) to implement this. A much simpler method (about 15 lines of code) will appear in
my frame book, utilising the message area rather than a widget approach (WordPerfect has a similar hints system).

Incidentally, neither Windows nor OS/2 use this sort of explanation box in the operating system to any extent. Windows '95 does and it makes for a more professional easier to follow GUI interface. I like the fact that you can maximise information and minimise screen real estate.

**Which widget to use**

**Pushbuttons.** They take a bit of extra white space but are a fairly standard way of presenting any sort of clickable selection. They can look at bit grotty when you have different sized ones together -- consequently it's not unusual to see the buttons standardise on size but have more white space within. Note that pushbuttons require an extra character on all sides of the visible button. Don't use them to display non clickable information as they aren't intended for that.

**ICONS.** These are more graphically pleasing as they include a picture which many people consider a primary attribute of GUI (I don't necessarily -- the primary attribute is using resources to make the easiest possible system for users). But you are limited in your selection of ICONS. You can define your own, however its a lengthy task and defeats the point of RAD to some extent. ICONS demand more screen real estate. Observations had an article a while back on icon construction (Second Quarter 1994). All this article did for me was resolve not to create my own icons! You need quite a bit of extra software and time. The article leads you through the process and it should be fairly simple with the appropriate software (all detailed for WINDOWS, OS/2, UNIX, VMS, WIN-NT in the article) but you will need some C or assembler knowledge.

**Text entries.** Any command associated with the widget won't execute -- that's by design. You will need code to trigger a command. They aren't really intended for command driving, however, you can associate a command (perhaps the attribute should be disabled in Frame?). You can make them look like buttons which I find quite visually pleasing. I usually use a border around them. as the region is not obvious otherwise. This object is irritating with its insistence on not permitting use of the first and last character position. In 6.10 you are really forced to use this for text entry, from 6.11 there are excellent alternatives. Note that text entries are character, not graphics based.

**Graphics text widgets** can be used just like text labels. But of course you can change the type font to make them look nicer. You can also cram different lengths of text into the same sized boxes. Try to avoid this as it can look awful! I use graphics text sparingly, usually just one to a frame for this reason.

**SAS/GRAFHP & IMAGE** widgets. Put a picture instead of the text. But note that the more pictures you use in this manner the longer it will take for your application window to open and redraw. With these widgets you can use as small a region as you like as Frame scales the picture to fit. You can get down to an unintelligible picture though -- something created at 640*480 and displayed at about 40*40 can't maintain all the information it started with. See the attached image of Einstein for an example! So keep them simple. You can use region attributes to make them resemble a button. Image objects seem to maintain the picture better.
than GRSEG's, however the image will not necessarily fill the whole region if you use the 'KEEP ASPECT RATIO' attribute. Not using that attribute causes the image to fill the region, but it is distorted.

From 6.11 you can use some new objects. The Image ICON is an image with text attached. Toolboxes are a collection of images or text all under one region. Extended text entries are a graphical form of text entry. They only accept character input and can use operating system fonts rather than SAS. This object is great in its ability to allow the entry of a character to be an event that drives a method called _FEEDBACK_. You can have code running as the user types and put some real intelligence in the object. There are others that I haven't explored as yet.

How many widgets exist on a screen

We're limited by screen real estate and imagination. But cramming in too many widgets, even character based ones such as text entries, will slow down the application as the background work increases (this is the work that frame itself does to support our development, for instance the creation of the lists that define each widget on screen).

6.11 allows us to add extra widgets at runtime by creating dynamic class instances using the CLASS class _NEW_ method. 6.11’s ability to swap regions in and out will also allow additional widgets to be viewed in an application, without them having to be on screen (hidden, grayed or visible). I have experimented with _NEW_ in beta and found it quite slow, so now I instantiate most widgets at startup time so the actual application performs better.

So where do we stop? Nothing's changed over the last 20 years in the way we look at screens. We still need to be able to read them, understand and interpret them. We can't visualise cluttered screens with too much information on them easily.

Evidently, as GUI designers we need to maximise visual impact as well as provide users with a screen that suits their work habit and methodology. If a user requires many fields on a screen and regularly works with all that data then we maximise the users ability to carry their job out by having all those fields on screen. If they regularly work with a few fields that are physically divorced from an aspect of their job that uses other fields then putting all the fields together may be confusing.

There is only one hard and fast rule. Get the user to help design (or completely design) the GUI. They know their job, we are just here to make it a bit easier for them, not to impose our interpretations of how we think they work on them. At very least, prototype and have the user (or someone who knows the area that the application is addressing) review. As a secondary rule, screens can get cluttered quite easily, and it is easier to view a screen with white space then one that is very cluttered.

Frame makes it easy to forget that we can call up other frame entries to do additional tasks. Its easy to see some spare space and put another field in it. Question whether the overall impact of the screen is improved by doing that or whether calling another entry would be better. My experience is that I often get carried away with the ability to get everything in one place and have to go back and re-examine the screen when I test it. However, I'm rapidly adjusting my design habits under 6.11 by making liberal use of composite widgets and swapping.

Prompting Widgets

This refers to messages like 'Please Enter Your Name'.

A tip here is that the title area of a widget can be used to provide prompting if the widget is wide enough. You don't need a separate widget that says 'Enter Name:' to the left of the field that name is entered into, just use the entry field with the title 'Enter Name' at the top. I suggest that you use an offset of 3 to position the title as 0 and 1 seem to cause the widget itself to slice the bottom few pixels off the title. By not using that prompt widget, you gain a lot more usable screen real estate which may be important when developing
under 640*480 resolution size constraints.

Run Time Instantiation

6.11 allows instances of widgets to be created at runtime. You can decide where the widget is to be placed and exactly what the attributes of it are in code.

This will allow us to add to the GUI by using pop up interaction widgets that don't need to be hidden. Sounds useful, particularly for the sort of situation where we want a verification window to appear. This feature will allow us extra control over resources, we can create and destroy as required, but I have found it sluggish in the creation phase (perhaps I should do as Intel say and scrap my 486/75 machine for a Pentium -- the point is that these features will appear sluggish as we go back down the hardware spectrum, so again, be aware of what hardware your application will run on).

Objects From Other Applications

OCX and OLE allow Frame entries to use widgets that are external to the SAS software base class. OLE was in 6.10, OCX exists from 6.11. You could thus utilise an existing object stored in a Visual Basic object library as if it were part of the SAS tool set. This means a reduction in re-inventing the wheel, but probably has some difficulties for migrating code -- more than just the SAS application will need to be tracked and moved.

Image processing

Note that image processing supplies a number of SCL functions that can allow an image to be defined at runtime and popped up on the current window. You do not always need to predefine space for an image, although predefining space in an image widget does make things simpler -- you don't have to worry about working out how to avoid overwriting useful parts of the screen and how to destroy the image.

Popmenus

Not strictly widgets, but related to the general area of things that appear and disappear (when is The X-Files going to have an episode on OOP?) are Popmenus. I have written applications that allow a wpopup type operation that is context sensitive, it's quite a nice alternative to using drop down menus at the top. I am currently working on an application that has two distinct types of operation -- working on individual records and working on groups of records. Each individual record is displayed via an extended table which includes an image widget. Clicking on the image pops up a popmenu of options specific to the individual. Options for the overall file or administration options are accessed via drop down menus.

You need to consider placement of Popmenus, particularly where a selection leads to another popmenu. I always code the popmenu start position to ensure that I'm not to near the screen bottom, and also to ensure that follow on Popmenus appear in a logical place.

Note that you can jazz Popmenus up a little by judicious use of the SETLATTR function to gray items. This allows information items to appear in the popmenu. For example, I had a popmenu that could be selected by clicking on a file name in an extended table field. The popmenu was to provide the options 'ADD' 'EDIT' 'DELETE'. I added a header 'SELECT AN OPTION' followed by a blank item, both of these being set to INACTIVE in SETLATTR. At the end I added a blank and a 'DO NOTHING' option, the blank being INACTIVE.

The 'DO NOTHING' option exists to work around Popmenus having no obvious exit without carrying out a selected task. I often have users ask how to exit a popmenu without selecting an option. It's far from obvious to many users that the popmenu will disappear just by clicking somewhere off the popmenu, and also use of Escape is not obvious to users, even though it is fairly standard in Windows.
Applications Development

Also, clicking off the popmenu can lead to all sorts of hassles if another widget is accidentally pressed. In the interests of GUI ease of use, my preference is always to provide an out option on the popmenu.

In SCL, you can use code like ...

```
SELECT (popmenu(<listid>)) ;
  when (1) ...
  when (2) ...
  otherwise;
END ;
```

The OTHERWISE which does nothing is what is executed when the 'DO NOTHING' type item is selected.

**Screen Sizes & Related Issues**

The size of our screens.

Which screen resolution should we develop in? Use the lowest common denominator. I.e. if your site has predominantly 640*480 screens with 16 colours don’t do your development at 1024*780 with 16 million colours, you can guarantee something won’t look right at the end. I suggest that 640*480 with 256 colours is a good size to stick to. Be aware of and guided by your client base hardware.

I don’t use attachments much as I find them quite difficult to set up under 6.10. But 6.11 can feasibly develop an application where all widgets are graphic and attachments work best in that situation. They can be useful for resizing screens at runtime and may allow frames to run in a lower resolution than what they were developed in.

An application doesn’t need to fill a whole screen. Sometimes the application looks and feels better by utilising just a small part of the screen. In general, PC users are confident with maximising windows and seeing multiple windows from multiple products on screen. My personal feeling is why prevent them seeing and using other windows if I don’t need the whole screen myself.

Related here is use of AFA to start up a separate AF stream. This causes multiple AF applications to be running and allows swapping between the two. I use this a lot, to allow people to open multiple files for editing different things. Just be careful that you don’t allow files that are dependent on each other to be opened.

Logos and logo placement

Unfortunately the mechanism for replacing the default SAS logo requires extra software to imbed bitmaps into DLL’s. I don’t think its a good startup procedure to tell the user all about SAS when they aren’t using SAS -- they’re using XXX Financial Application or whatever. I suggest switching the logo option off in production and start things up with a Frame entry that displays a logo for a few seconds. This of course means a bit extra APPARENT time getting the application started.

If you have a logo, be it on screen during the whole application or just at startup, place it somewhere that it doesn’t interfere with normal operation of the application. Making it non-selectable and non-tabbable is usual.

Wave 2 of release 6.11 will have an invocation option to define which bitmap to use for the startup logo. Brilliant!

**What is a menu in a GUI context**

A menu in the traditional terms is the listing of a set of alternatives from which one can be chosen for
further processing. Usually this was displayed literally as a list, often with a number or letter to be entered on a command line to choose which item to process.

GUI doesn't take away this definition of menu. It adds to it. It adds to it by allowing different ways of looking at selection of choices, by allowing us to logically divide screens up so that previous menu items are all (or several at once) on screen, and by allowing us to display the mechanisms for selection in an intuitive manner.

A frame entry can be strictly for the traditional purpose of selecting an option and branching elsewhere to do some processing. The choices for displaying those options are enormous -- icons, pushbuttons, images, graphs, block menus, traditional list menus, pictures with hot spots, check boxes and radiobuttons, choice boxes and so on.

Alternatively, some applications may utilise a familiar screen so often that the screen displays immediately and the menu into other options is just a set of icons or choices on that screen.

As an example of a number of different types of menu look at SAS/EIS. You start with a block menu which branches to different types of menus depending on what you wish to do. This is a very traditional form of menuing, just prettied up for the GUI environment. Once in the later levels of processing, popmenus are used to select operations to carry out on objects such as datasets. Selection of variables for further processing is done by line commands. Line commands such as S, E, B and so on are themselves a form of menuing.

I talked about EIS because it is a nice template for ideas for menuing. However, that doesn't mean your users will like the methods they use. Once again, company standards and what users are used to and comfortable with are the main criteria. Prototyping should give both developer and user a chance to try out different menus and choose just what is appropriate for the given situation.

Actually, one of the bonus features of frame is how quickly a screen can be modified. Once code is written and working, widgets can be changed with little code change provided names are retained. Of course, there are situations where methods are widget dependent, but in general you can quite quickly alter method calls.

Summary

What I have tried to do here is provide a few ideas about how to approach the whole GUI development paradigm from a screen design perspective. GUI is quite new to many SAS programmers. We need to modify our thinking so that in addition to resolving the issues that the application addresses, we include how to let the user interact with that application.

My forthcoming book on Frame will have more detail on screen design, including discussions of fonts, development methodologies, parameter entry and selection boxes as well as complete descriptions of some of the objects I use to assist with development.

A final word on GUI. If you want to see literally hundreds of different ideas on GUI screen design check out some world wide web home pages on Internet. You will rapidly come to appreciate consistency in other applications. WWW is like anarchy at the best of times, and the home pages show many forms of human interaction with applications.

Credits

My thanks to Susann Ryan and Mark Berger for reviewing this paper and making many valuable suggestions. Some of the background for this paper arose from a 2 day session on developing GUI applications held by SAS Institute and Quality Partners in New Zealand, and the input of the people involved in that is acknowledged here.

There are many frame related papers in Observations and SUGI proceedings that give useful further reading. The text denoted as (I) in the body of this paper is a very good Microsoft text "The Windows Interface - An Application Design Guide" printed by Microsoft Press. Although it is based around Windows 3.1, many of the ideas are relevant to other platforms.