#4012

Implementing Intelligent Function Keys with Hot Keys in Speedware

Michael W. Spahr

The Andersons, Inc.
480 W. Dussel Dr.
Maumee, OH 43537
(419) 891-6425

Introduction

Speedware offers many advantages to the developer by way of the Designer and 4GL products. The developer must design applications which also have many advantages for the users. I have taken the user interface and expanded the applicability of some of the navigational characteristics. There are two techniques which I have implemented in our Speedware applications which seem to be getting high profile attention from other Speedware developers who design custom applications. The techniques are custom functions known as “intelligent function keys” and “hot keys”.

During this discussion, I will cover some of the underlying reasons why you might want to implement these features in your own Speedware applications. There will also be discussions detailing the methods used in installing these concepts into your Speedware applications. I’ll start by presenting a brief definition of the terms Hot Key and Intelligent Function Key, along with a look from the user’s perspective. I will talk about the advantages of implementing these features, and also review the implementation of the Hot Key and Intelligent Function Keys from a designer’s perspective. I have included some screen samples and the 4GL / Designer code used in our application.

Hot Key

The term “Hot Key” refers to the ability of moving <u>from</u> anywhere in the application <u>to</u> anywhere in the application directly, without the use of intermediary menu systems. Unlike standard menu driven systems, where the user is consistently accessing programs from the same entry point (the main and/or sub menus), using the Hot Key allows users to access the programs from many entry points within the application. For example, in a menu driven application, you would always access the customer profile screen by first selecting Customers from the main menu bar and then selecting Profile from the Customer submenu. Using the Hot Key, however you may access the Customer Profile screen from any of the other screens in the application.

To highlight the benefits of implementing the Hot Key into your application, let’s step through a scenario. You have just called up the customer profile, or basic information (name, address, phone, contact, etc.) and now you must quickly find some other related piece of information such as the status of his current orders. In a typical menu driven system, you would first exit the profile screen, returning you back to the main menu, and then select the appropriate menu selection to view his order information. Once at the order selection screen you would look up the customer by entering his number, name, phone number, etc (just like in the initial profile inquiry) and then get to the point where you will be able to answer his questions regarding his order status. Phew! This is user friendly?

Using the Hot Key concept, if you were at the customer profile screen, you would simply press the Hot Key function key, enter the command CO for (C)ustomer

(O)rders and then press the return key. The next screen to display will be the order selection screen with the previously established customer information already inserted. Add additional selection criteria if you like and then press the return key. That's all there is.

The Hot Key becomes a powerful productivity tool from the user's perspective very quickly, once the users can grasp the concept of not having to exit previous screens, and choosing all the selections from a menu structure. Due to the ability of the user to move swiftly and efficiently throughout the application, the Hot Key navigational capabilities have facilitated the need for the implementation of "intelligent function keys".

Intelligent Function Keys

The term "intelligent function keys" is a term which originated in one of our design teams. It refers to the capability of dynamically assigning labels and processing logic to appropriate function keys. The labels and processing logic are based on the calling Speedware program. This feature has been implemented in all of our Speedware applications and has been well accepted by our end users. It has been proven that using the intelligent function keys with the hot key capabilities has lessened the learning curve to navigating within a new application and supports efficient use of all applications.

Within Speedware, there are many standard functions regarding the structure and features of the designer and 4GL products. One such standard is the use of "Previous Action" on the F8 function key to denote exiting from the current application and returning to the previously executed application. The use of such a generic term as "Previous Action" was not acceptable to the users (design teams). Therefore, I had to develop a method in which we would allow the function key to become more "intelligent" and informational for the end users. What facilitated the desire for intelligent function keys, was that I had committed to providing "hot key" capability within our Speedware application.

As stated earlier, the hot key concept allows the user to navigate throughout the application without the use of the standard menu structure. This adds to the efficient use of the Speedware product, as well as eliminating the need for redundant entry when moving through the application. However, due to the flexibility in accessing the programs, it was necessary for me to develop a method by which the user will always be able to distinguish where he/she came from within the application. This is what is referred to when defining the Intelligent Function Keys.

Activating the Hot Key

Within the Speedware application, when the user wishes to move to another section, he/she simply presses the "Hot Key" function key. To help facilitate quick efficient use of this feature, I have opted to consistently use the F2 function key. This function key will appear on all subsequent screens where the hot key function is available. Once this key is pressed, the user is prompted to enter the desired Hot Key Command. *(figure 1)*

The commands are an abbreviated representation of the destination program within the application. Our approach to defining the commands is to mirror the shortcut commands used within the standard menu system. For example, if you were on the main menu and wanted to choose Files, you could either use the cursor movement keys to locate the cursor on the branch and then press return, or you could enter the unique starting characters of the branch, in this case "F", and press return. These unique characters are what I decided to use to identify the hot key commands. This supports an easy transition from the menu system to the hot key or "expert" mode of operation. More on this when we discuss defining your hot key commands later in this paper.

Once the user enters the desired hot key command, the logic is then executed to call the associated program passing the appropriate parameters. At this point, the intelligent function key really becomes important. The program the user was in when the hot key was activated will become the label on the F8 function key of the called program. This is to provide the user with a point of reference.

The parameters mentioned contain items such as customer number, contract number or any other item which you can globally link to your programs. In my case, I use customer number and, where applicable, I also include contract number in the passed parameters between programs. This supports the capability of keeping the same customer number when branching to another window and not needing to re-enter the customer number again. Of course the user can still override the passed customer number and enter any desired valid value. By passing the key parameters to each called program we can reduce the amount of redundant key strokes by the users.

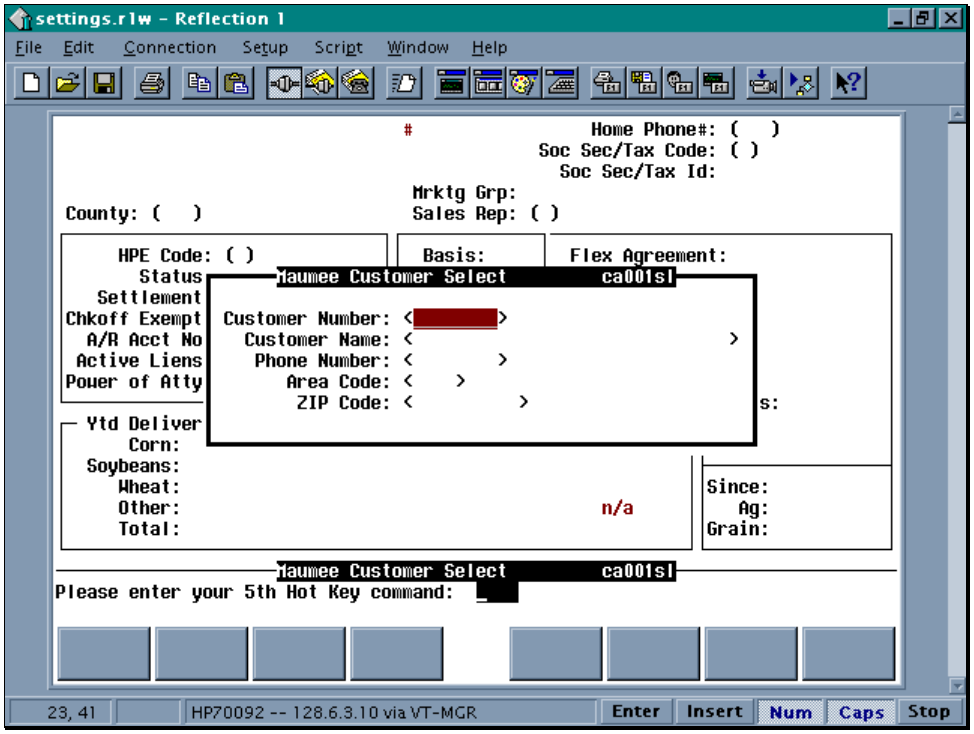

Figure 1 (Hot Key command prompt)

Please note the reference to the number of hot key commands entered in the prompt. This is a valuable tool for the end user and will be used to programatically control the Speedware imposed limit regarding the maximum number of concurrent windows within an application.

When the user is finished with the called program, the hot key may be used again to further navigate through the application. Likewise, the intelligent function key would also follow the user and the F8 function key of the newly called program will be labeled referencing the calling programs. The user may also simply decide to exit the called program which will return him/her directly back to the point in the previous (calling program) where they left off.

I structured the functionality of the hot key around one basic and core idea which is that the previous screens or windows will remain open. That is, the window you were in when you activated the hot key process will be the window you return to when you end the called program to which you hot keyed to. I have incorporated using the hot key in this manner to allow the user to briefly go look up other information and then return right back to where he/she left off without interrupting the current process. An alternative approach to the hot key logic could close the current screen or window once the called program is executed. I felt that to make the hot key a truly valuable tool to the user and provide the maximum navigation flexibility, I would leave the previous screens open.

As you may be able to see, the intelligent function key plays an integral part in the successful implementation of the hot key. If it were not for the ability to dynamically label the function key based on the calling program, the user would not know where they would be returned to when the pressed the F8 "Previous Action" function key, since they could enter the existing program from anywhere in the application.

Implementing the Hot Key

Seeing how the user implements these features, you can probably see that there is a fair amount of setup and process logic which must be defined. The following discussion will preview the setup steps needed to implement the hot key logic in your application along with possible alternative ways of implementation. You will see with these features, as with many other items within Speedware, that there is more than one way to accomplish a stated objective.

Since the existence of the hot key feature is what facilitated the introduction of intelligent function keys, it only makes sense to present them in a related manner. Therefore, these discussions will address both the hot key and intelligent function keys at the same time in order to help explain and establish the relationship between the two features. Let's face it, if it weren't for hot keys, the intelligent function key would not be needed.

We will cover these issues...

- Defining Hot Key Commands.
- Where and How to Store the Commands.
- Dynamically Calling the Proper Program.
- Establishing Parameters Between Programs.
- Hot Key Command On-Line Help.

Defining Hot Key Commands

The strategic approach to defining your hot key commands is that they should be intuitive to the user. They should be short (2-4 characters), uncomplicated and have a stated structure. One thing to keep in mind if you are implementing the hot key features into an existing application, is that the transition for the user must be simple and easy for her to do, otherwise the true benefits of the hot key will never be realized.

As mentioned earlier, the stated structure I chose for our hot key command involves the standard Speedware menu program's intrinsic use of what we call "shortcut commands".

Where and How to Store the Commands

When initially setting up the hot key commands, the approach I took was to load them into an internal table using global variable arrays at the startup of the application. This is done via the use of the Include document hot-key-load. (*figure 2)*

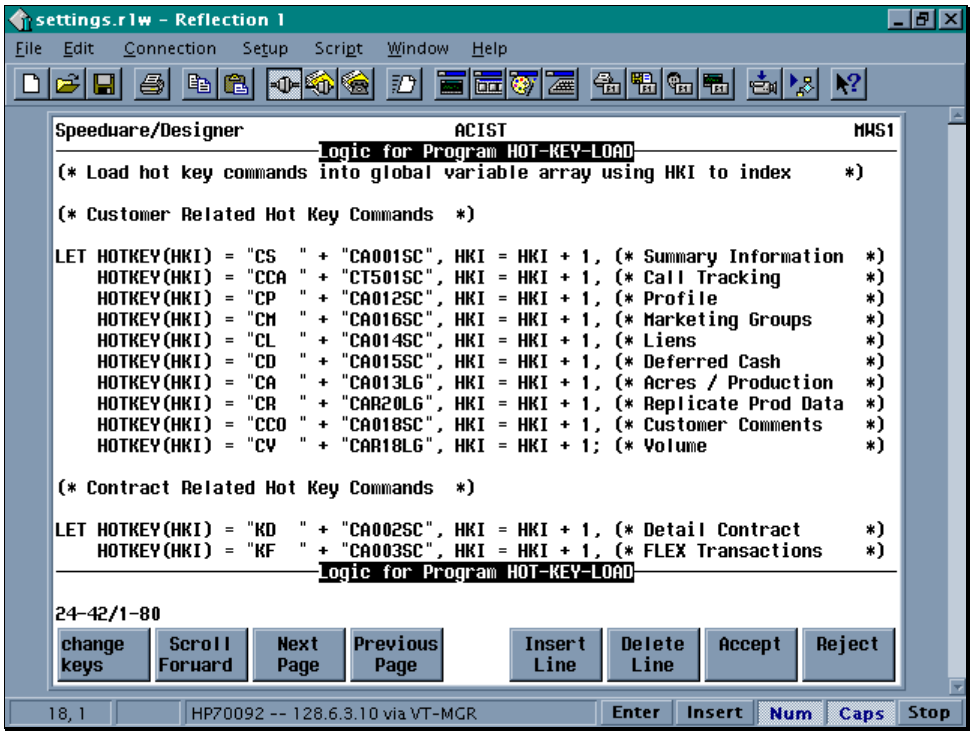

Figure 2 (Loading the Hot Key commands)

The internal table simply consists of the hot key command, limited to a maximum of four characters in order to maintain ease of use, and the program which is called when the hot key command is executed. The internal table is defined as occurs 500 times. I use a hot key command of XXXX to provide a recognition point for the end of entries marker within the table so that the entire table does not need to be searched before we can determine the entered command is invalid..

Our initial implementation of the command load routine accessed this table through the logic in an Include document. After working with this approach for some time, I feel that it would be more flexible and dynamic to store the information in an Image dataset.

The primary application in which the hot key has been implemented was written by only one Speedware developer. Therefore, all additional programs and objects were controlled.
However, now that we have a team of three developers working on this project, it has become more crucial that we allow as much flexibility as possible. Using a dataset to

store the hot key commands will allow additional commands to be added simply by adding a record. This is more dynamic than modifying the Include document and re-compiling the Speedware application.

Dynamically Calling the Proper Program

As discussed in the previous section, the associated program identification is stored with the hot key command. It was my intention to develop a method by which we could use one set of logic to call any associated program. In order to do so, I needed to identify the program type with each hot key command. There could be a separate field within the internal array used to identify the program type, however, I decided to incorporate the program type directly into the program identifier. Not only did this seem to easily support the hot key concepts, but it also provides additional assistance to the developers when referencing the programs within Designer.

We have discussed how to determine the hot key command and how to store the information associated with the command. We are now ready to review the underlying logic used to implement the hot key functionality. *Figure 3* displays the first portion of the logic used to validate the command entered by the end user.

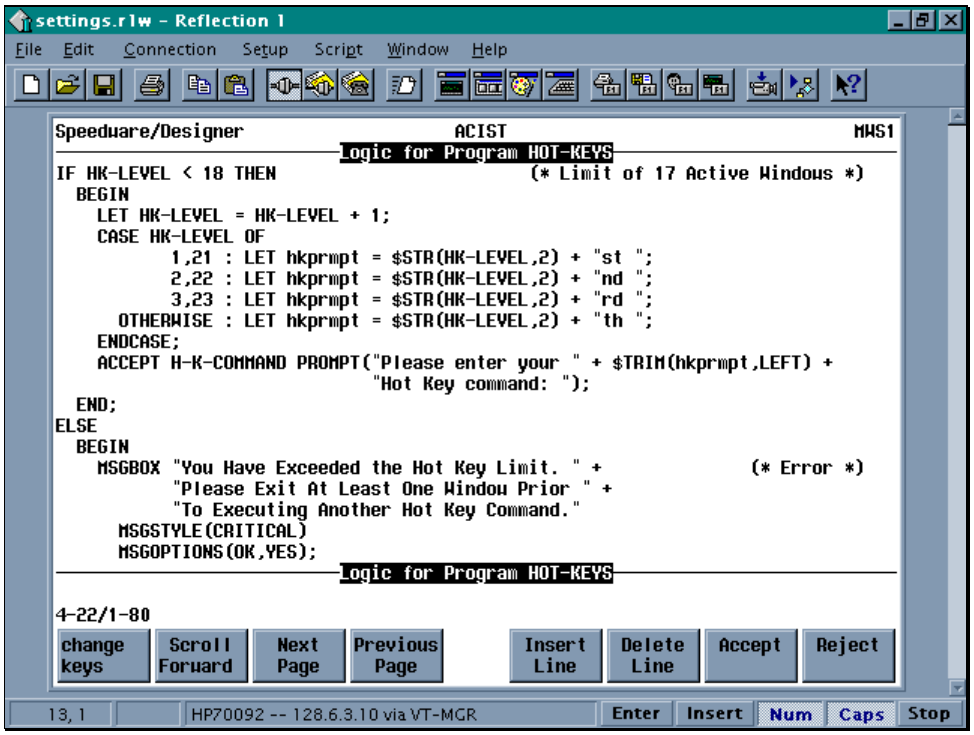

Figure 3 (Evaluating the entered Hot Key command)

First, the user is prompted for the hot key command. Note, that the total number of currently active hot key commands is displayed with the prompt.

Speedware has a limit of supporting approximately 27 concurrent active windows. In an attempt to trap the error condition before Speedware does, we check the number ourselves. If Speedware traps out, the application is terminated. Therefore, we have set the limit less than the Speedware limit and can then display the proper message boxes to the user to notify him/her that they have exceeded the limit and must therefore close at least one active window before another hot key command is allowed. There is one undocumented hot key command which I support, but the end users are unaware of. This command is used to execute the intrinsic command within Speedware to pause the application and exit to the MPE command prompt. This feature is used when troubleshooting issues with the end user. First, we check to see if this command has been entered.

The internal table is then inspected in order to find the entry in the table matching the user entered hot key command. A message is displayed if the entered value is not within the table, otherwise the call logic is executed. *Figures 4-7* represent the hot key call logic.

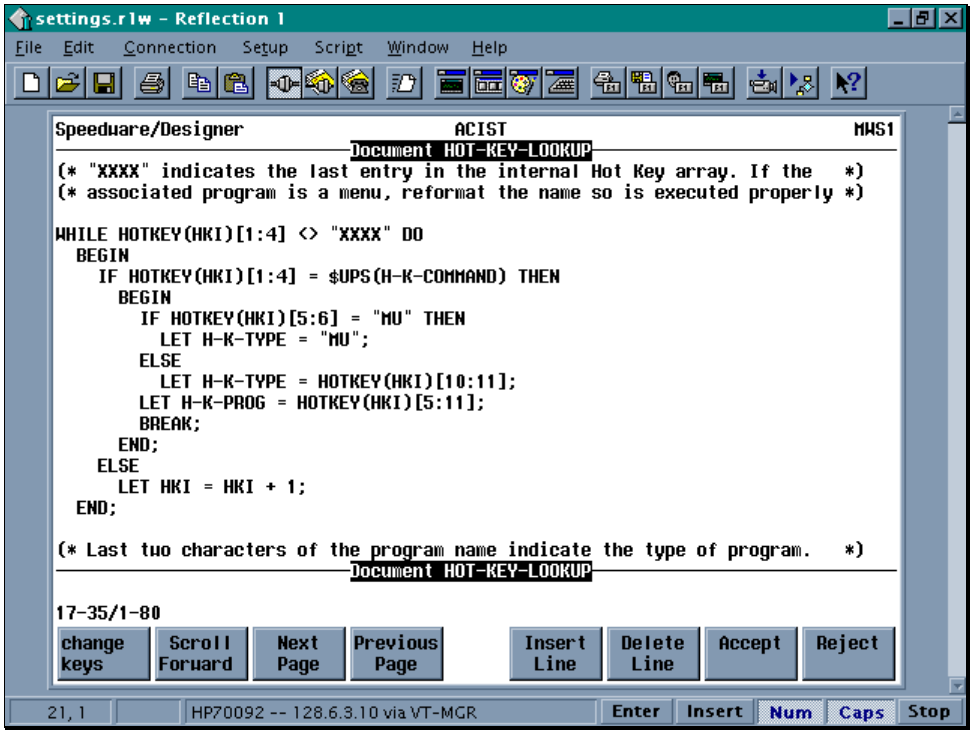

Figure 4 Hot Key Call Logic

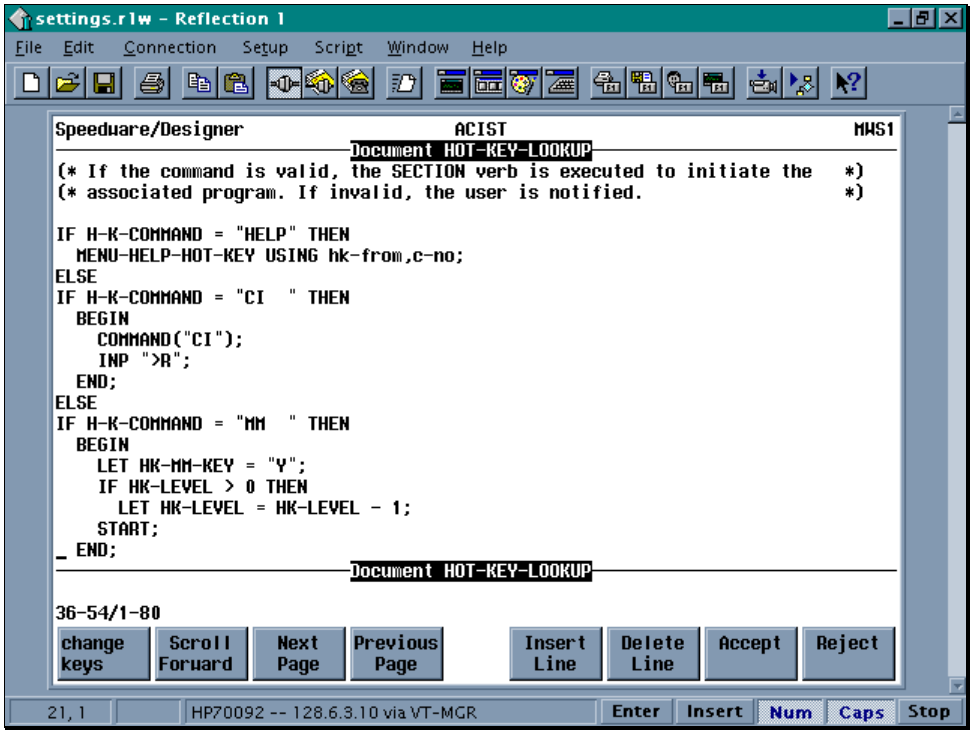

Figure 5 Hot Key Call Logic

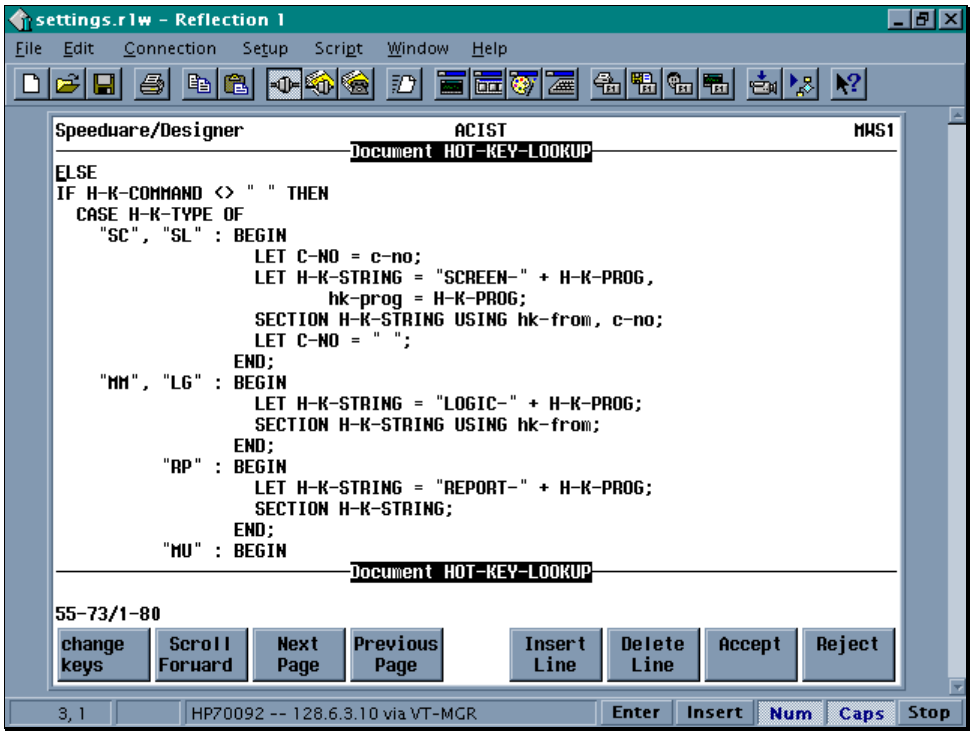

Figure 6 Hot Key Call Logic

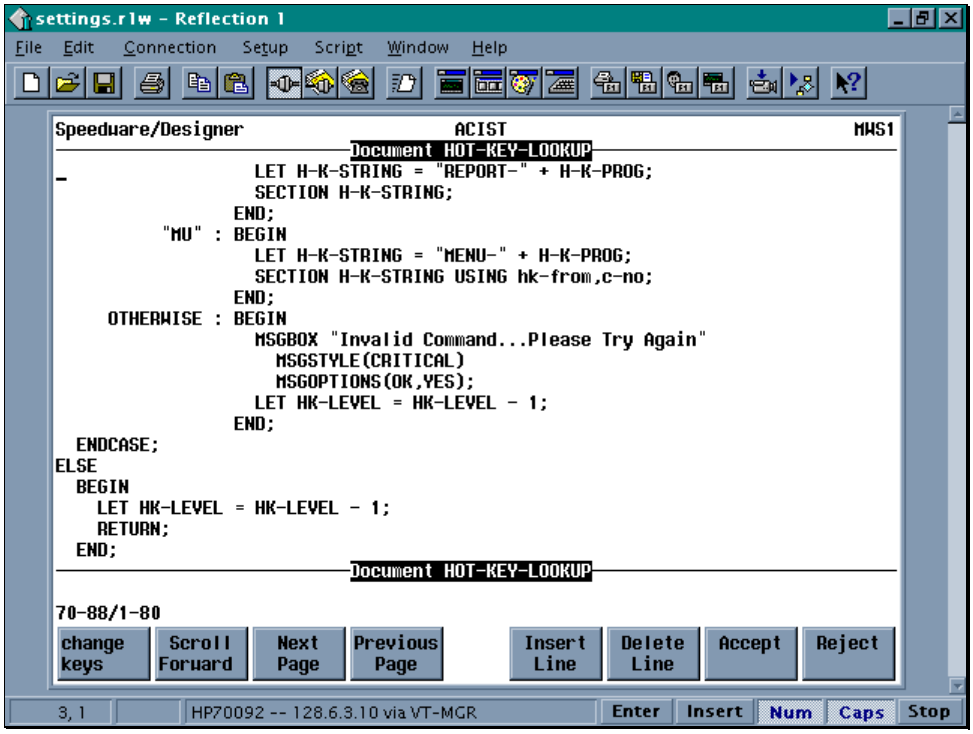

Figure 7 Hot Key Call Logic

The next CASE statement dissects the program name to identify the type of program (screen, select, report, logic, or menu). Based on the value of the program type indicator (HK-PROG[6:7]) the appropriate SECTION verb is executed. This allows the logic program to adapt dynamically to each hot key command entered without requiring additional logic to determine the type of section verb to execute. In these examples, a portion of the program name is used to identify the type of program. Another alternative to implementing the hot key logic with existing programs which may not apply themselves to this method is to create a data file containing the program name and an identifier used to signify the type of program. Of course there is yet another way, and that would be to supply the logic using hard coded program names. You could use IF or CASE statements to physically check each program and then issue the corresponding SECTION verb. Although this approach would work, it is NOT recommended.

Establishing Parameters Between Programs

In addition to allowing the user to quickly navigate through the application, another valuable benefit of the hot key concept is to provide the capability of bringing along previously established parameters with you as you proceed from program to program. Along with eliminating the need for redundant entry, this portion of the hot key logic is where the intelligent function keys are implemented.

There is an emphasis placed on the customer identification within our application. Almost the entire database is related to specific customers. Therefore, when you are in the process of looking up multiple pieces of information for the customer, the customer number is used to retrieve the desired information. As the hot key is implemented, in order to reduce the amount of redundant entry, the customer number previously established is automatically passed to the called program. Once the use has used the hot key to go to the next screen, the customer he/she had previously selected from the previous screen is automatically inserted into the desired selection criteria.

In order to implement the intelligent function key, the new function key label is also passed to the called program. The function key label identifies the calling program, and will be displayed on the F8 function key of the called program. Therefore, the user will always be able to identify where he/she entered the program. *Figure 8* shows the 4GL code executed when the use has pressed the hot key and has entered a valid hot key command. This code is what triggers the chain of events necessary to effectively implement the hot key process.

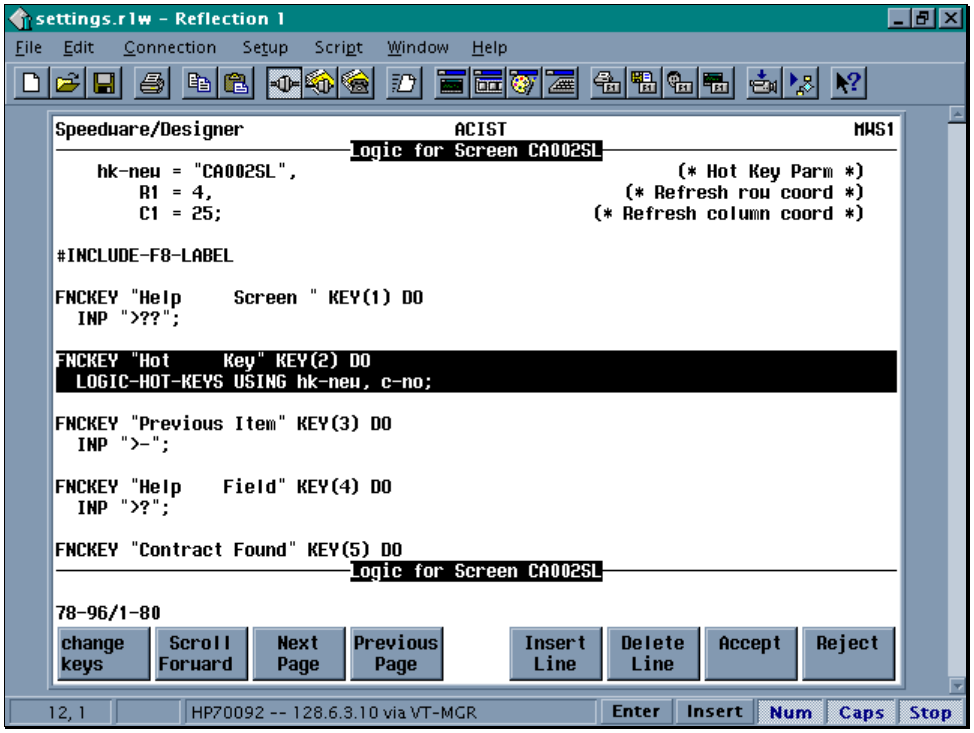

Figure 8 Implementing the Hot Key

The logic program HOT-KEYS is actually the driving program. It executes the logic discussed in the previous sections. It prompts the user for the hot key command and then executes the appropriate section verb passing the newly established parameters.

By calling the logic program, the current window(s) within Speedware are still active in the background. That is, the windows open at the time the hot key is pressed will remain open in order to allow the user to return back to that point when the called program is complete. This allows the user to briefly interrupt the current process, branch to another related or unrelated process, and then quickly and efficiently return the where he/she left off. The alternative to this practice would be to close the current window prior to calling the next one. Doing so would eliminate the need to track the current number of hot key windows open, but it would eliminate the ability for the user to pop into another screen and then return directly back to where he/she was.

Hot Key Command On-Line Help

I have supplied an extensive on-line help support system to assist the user when executing the hot key commands. First if the hot key is initiated and the command is left blank, the hot key implementation is ignored and the user is returned directly to the current screen. If the user needs help identifying which command to use, simply by typing HELP, a menu approach to a listing of all currently support hot key commands will appear. At this point, the use may either enter the desired command or position the cursor on the command and press return. *Figure 9* presents an example of this on-line help screen.

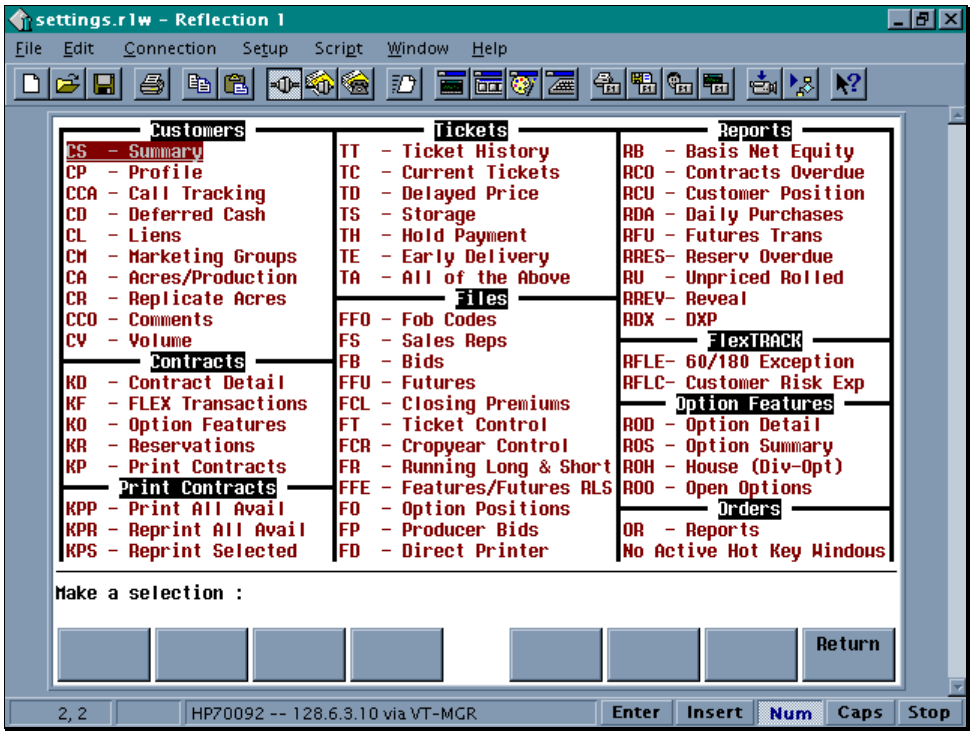

Figure 9 Hot Key Command On-Line Help

In addition to the on-line help, the users have each been given a "Quick Reference Guide" as part of their training. This is basically a hard copy of the on-line help. Currently, this help screen is a MENU program which simply lists each command as a prompt for the menu branch. At the logic of the menu branch, the INP command is used to pass the desired command back to the hot key program. This approach requires a high degree of maintenance, in that each command must be manually added to the menu. It is our desire to add each hot key command and description to an Image dataset so that this help program may then be converted into a simple screen program using the Speedware provided LIST SELECTED event to display and allow user selection of the desired command.

Remote Windows Implementation

This discussion has been based on a standard character based implementation of Speedware. If you are currently developing, or are planning on developing applications in the client/server "Remote Windows" environment, there are other approaches you may take in implementing the features discussed in this paper. Every aspect addressed here is applicable in the client/server environment. The only possible inconsistency with that approach would be in the use of the function keys.

Since function keys are not a standard user interface tool in the windows environment, this may seem a bit out of place in your application. However, one could also argue that all other Speedware applications utilize the function keys, so therefore, the client/server applications should be no different. I am not going to justify and/or support either approach. You weigh the alternatives and you decide which is best for your development staff and end users. The biggest advantage in using the Remote Windows product, besides the look and feel of a real windows application, is that we can now implement real pull-down menus with cascading features. This is possible via the PULLDOWNMENU verb. Although this was available with version 7.02.xx, the true functionality is realized through using Remote Windows. I will not get into the details of how to use this 4GL verb in your application, but I will discuss how to utilize this verb in order to continue with the hot key and intelligent function key concepts when using it.

Speedware recommends that the underlying "triggers" for the menu selections (AT MENUITEM OF...) be coded in the Application Options portion of Designer as Free Options. This works great, except that by placing the logic at this point in the code, all local variables have not yet been defined. This causes a problem when you would like to call a program from the AT MENUITEM OF event passing it pre-established variables. There are two solutions to getting around this dilemma when also wanting to code the hot key functions (since the hot key depends on passed variables to launch the next program or establish function key labels).

The same include documents and logic programs may be used for the hot key as in the character based applications, the only difference being that you may use global variables to store the passed program name and function key labels, or you may code the AT MENUITEM OF event in each program you wish to utilize the hot key with. The latter solution will create allot of additional duplicate coding, so if you take this approach, I highly recommend the use of Include documents. Figure 10 references the code associated with supporting the hot key and intelligent function keys when continuing with the same logic as stated earlier, using local variables, and placing the associated code in each program in your application which requires the use of the hot key.

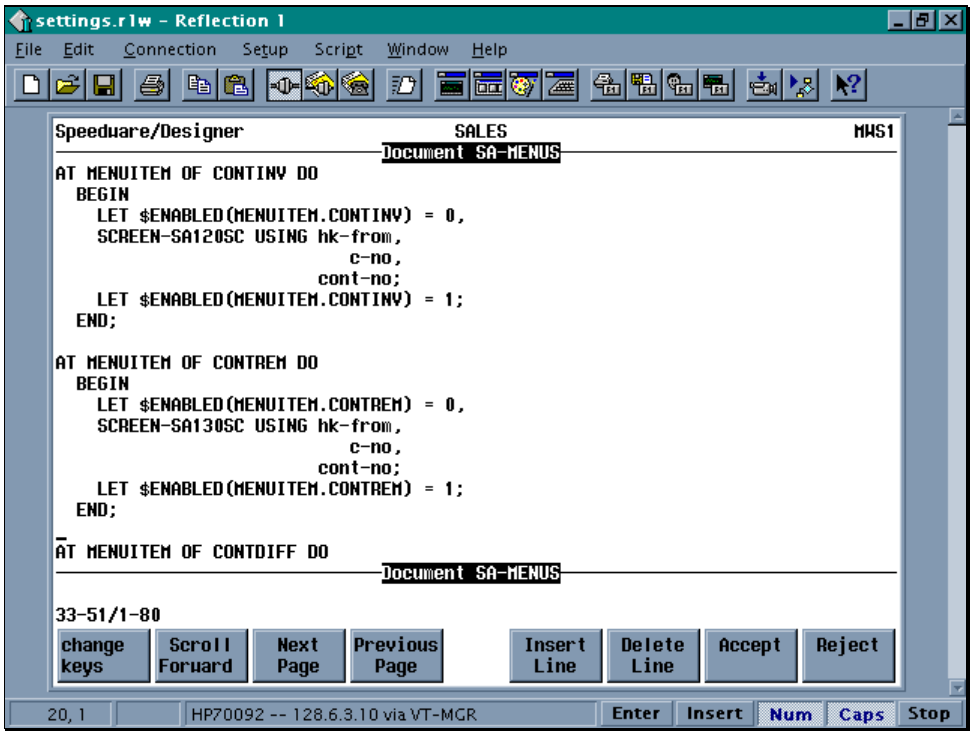

Figure 10 (AT MENUITEM OF Logic to Support the Hot Key)

Note: If you do not take the approach of including this logic in each program requiring access to the hot key (using an Include document is recommended), then you will need to load global variables with the desired values prior to calling the next program. Once the program is loaded, the global variables are loaded into local variables for that program so that if additional hot keys are accessed, the original values will remain with the appropriate program and continue to provide the feature of returning directly back to where you left off when you press the hot key.

In Conclusion...

This discussion has explained what the hot key and intelligent function keys are, how they impact your users, and how they may be implemented in your Speedware application. We have reviewed examples of 4GL code, and have seen, at least, how I decided to implement them. Please realize that there are many instances that you may choose to take a slight or totally different approach to implementing it into your application. That is one of the many things that makes Speedware as dynamic a development tool as it is. The concepts are consistent, but how you introduce them into your application may be as different as the users using your application.

The hot key concept has been implemented within our Speedware application and has been widely accepted by the end users as a way to make the application a more viable "tool" for accomplishing the objectives. As indicated by the associated examples, our approach has been developed along with the implementation of our first production quality application. Therefore, we have structured much of the infrastructure of our design (programs, variables, naming conventions, etc.) to directly support the implementation of the hot key. It is extremely important to realize that the process outlined in this discussion is not the only way to implement these features within your Speedware application. Each section of the process, be it the storing of hot key commands, calling the associated programs, or passing the variables and establishing the intelligent function key labels may be designed according to your specifications and process flow. This discussion, I hope, has provided you with valuable information which you can take back to your application and use.