



Handling Errors in Drivers

Overview

When making changes to meta-data and programming using the new programming architecture, you may error in any number of driver programs. When this happens, you will frequently be many levels deep in called programs, and it may be difficult to relate the error to any problems you have in either your code or in your meta-data entries.

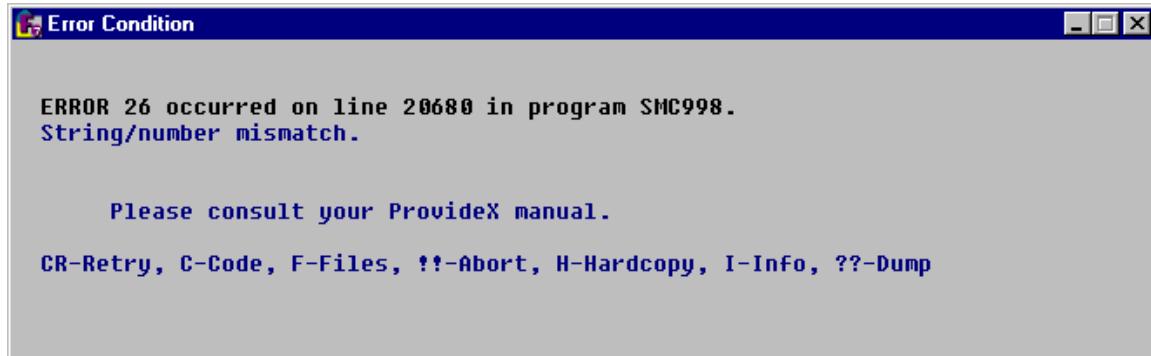
Since the driver program simply react to specific values in the meta-data, the problem is usually in how your meta-data is setup, and there are three specific errors which are fairly easily traced.

► Chapter Outline

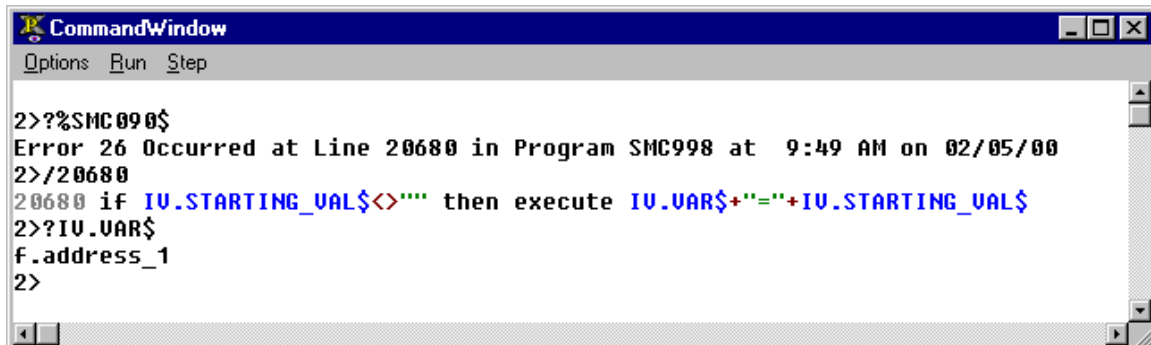
- Overview
- Error 26
- Error 23
- Error 65
- Error 55
- Error 29
- Error 11 in a File Maintenance
- Your File Maintenance or Report Will Not Execute from a Menu
- Troubleshooting Tips

Error 26

In the following example, an error 26 was encountered at line 20680 in program SMC998. The resolution to the problem will be similar for most error 26s.



Using the debugging environment's command mode window, we were able to interact with the program without adversely impacting the screen. We typed "ES" at the error condition, then typed "EXIT" to get out of SMC090. Then we did the following:



%SMC090\$ is a global variable that contains the description of the error message given by SMC090. Looking at the line of code, we see that the program was attempting to set the variable contained in IV.VAR\$ to the value contained in IV.STARTING_VAL\$, but IV.VAR\$ was not a string variable. The Variable field in the SMPRMT record should have a dollar sign at the end of it.

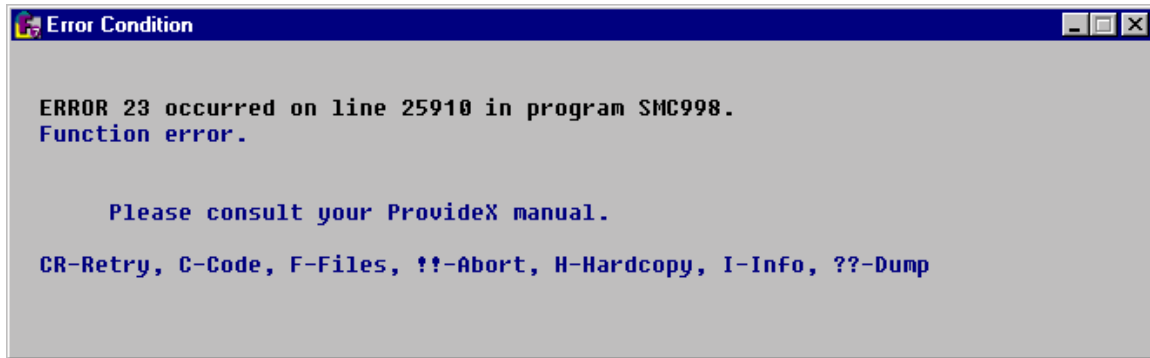
Almost all error 26s will result from variables which should contain valid string expressions having instead invalid expressions.

To determine the specific field in SMPRMT (or other meta-data file), you can use the NOMADS Dictionary Maintenance utility or the NOMADS Library utility and look at ssutils.en.

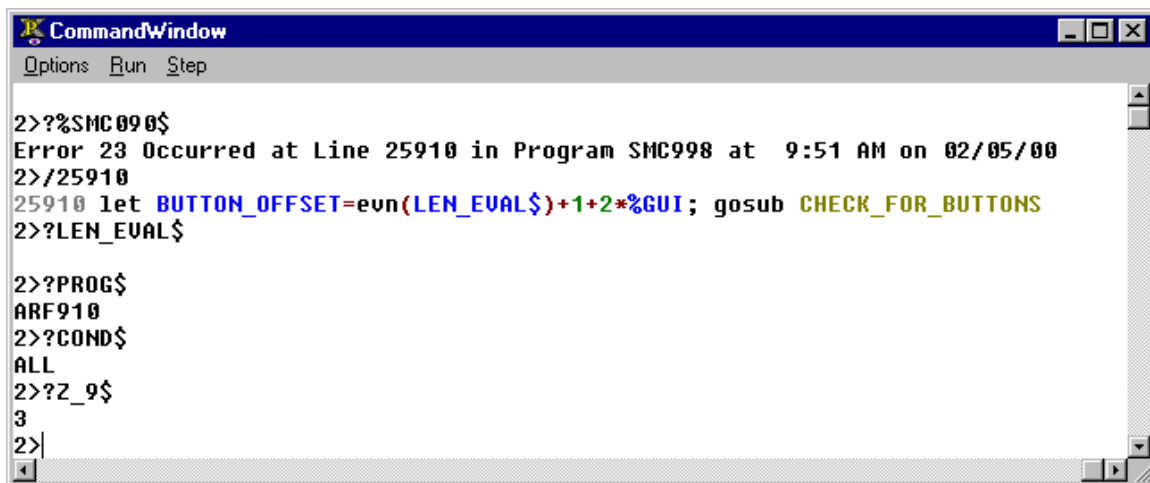
Other error 26s may occur if you've entered a procedure incorrectly, left off an ending quotation mark, or treated an eval field as a literal.

Error 23

Whereas error 26s are typically invalid string expressions, error 23s are typically invalid numeric expressions. In the following example, an error 23 occurred at 25910 in SMC998.



Again, we make use of the Command Mode Window. After ES'ing and EXITting from SMC090, we do



the following:

evn() is the function that evaluates a numeric expression. Similarly, evs() is the function that evaluates a string expression. When you encounter an error 26, check all evs() functions. When you encounter an error 23, check all evn() functions.

In this case, we look at the variable inside the only evn() function, and discover it is blank. This is a case where the Length field in SMPRMT was left blank, and it should be filled in.

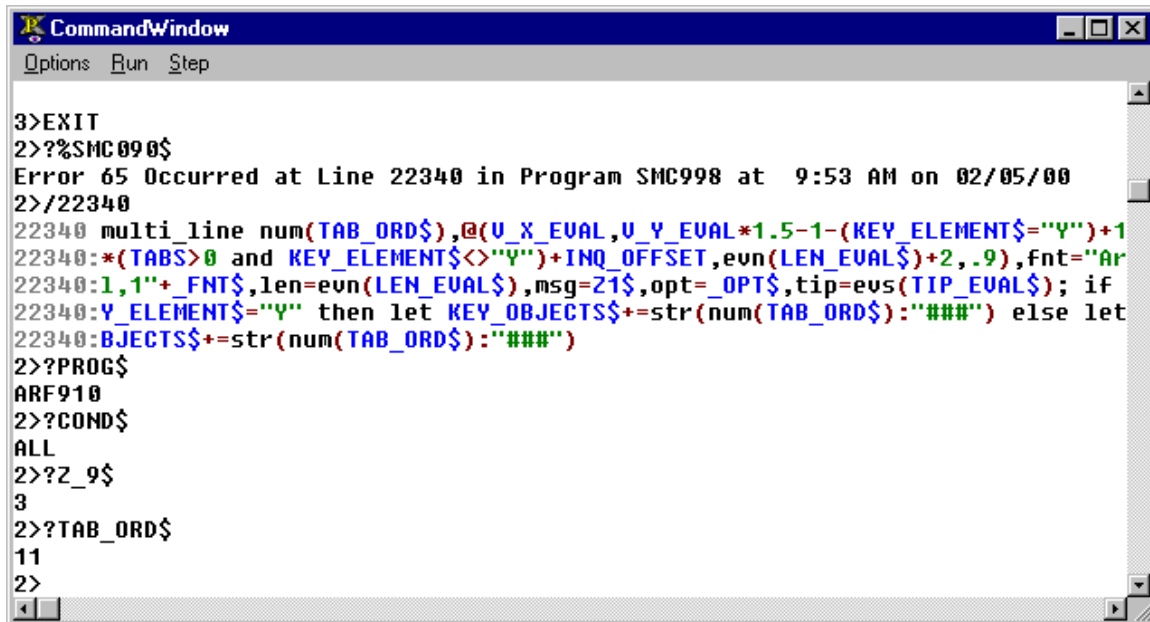
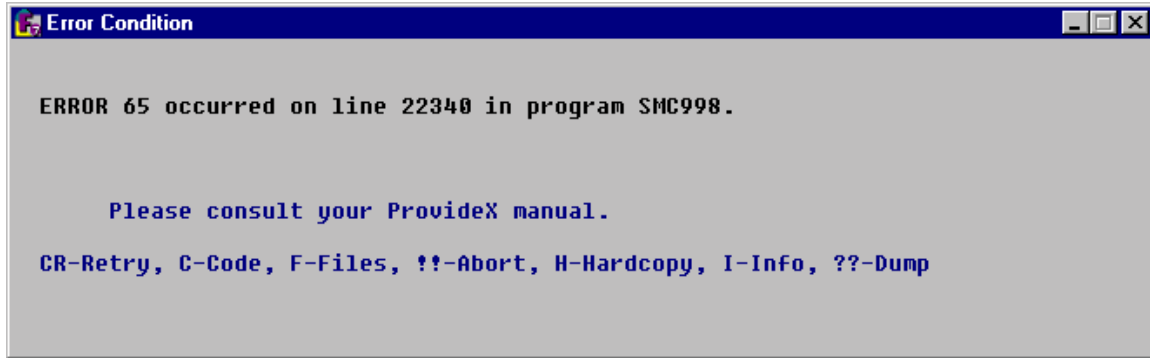
To determine which SMPRMT record needs to be fixed, we printed PROG\$ (the program), Z_9\$ (the Z[9] value), and COND\$ (the runtime condition). These variable should always be available when the problem was with an SMPRMT entry in a driver program.

Other error 23s may if you put a string variable in a numeric eval or enter an invalid calculation.

Error 65

Error 65s are almost always a result of a duplicate Tab Order in SMPRMT. The ProvideX description of error 65, which can be seen by printing MSG(65), is Window element does not exist or already exists.

The following example occurred because two fields had the Tab Order of 11. Again, we use the debuggin environment.



The easiest way to fix and prevent this error from occurring is to use the utility program SMF997, which sequences Tab Orders.

Error 55

Error 55s are GOSUBs or PERFORMs to line labels that do not exist. These may be encountered by a driver if you enter an invalid procedure. However, most of the driver programs have been error trapped around invalid performs. If you determine that your code is not being performed when you think it should be, double check that you entered the procedure correctly in the meta-data – do not assume that the driver will generate a hard error in that case.

Error 29

Error 29s usually indicate that the graphical control being created is outside the boundaries of the window it's being created in. Either determine the SMPRMT record that is bad by printing Z_9\$ and change its location or enlarge the size of the window.

Error 11 in a File Maintenance

This usually indicates that you forgot to set the company field and other non-entered key elements in the New Record procedure.

Your File Maintenance or Report Will Not Execute from the Menu

When you select your program from the menu and nothing happens, it is usually the result of a bad initialization procedure, invalid file name, or invalid template procedure. The drivers must be able to successfully perform these two procedures and open the file in order to do anything at all.

Troubleshooting Tips

Using the ProvideX debugging environment is very helpful, and in particular the trace window can help you figure out what's happening in a program. However, if you've used the trace window while the drivers are running, you have discovered the volumes of code that must be searched through. To help you trace your code, simply put a password on all the driver programs you don't want to see in the trace window. Use a password that is simple and easy to remember, like "password".

When you initiate the trace, only the non-protected code will show up.

Using the break window can also be a great help. If you want the program to stop each time it enters your code, simply set a break for your program name and leave the line number blank.

Combine the break window with the watch window to easily check the value of variables. While in GUI, the watch window works very well for tracking almost any variable you're interested in. In CUI, it doesn't work quite as well because each time the program entered SMC010, the values of the variables are lost (because of the call/enter list).

The break window can also be setup to stop when a condition becomes true, like `f.customer$=" C100"`, or when a variable changes value. Breaking when a variable changes value is again less effective in CUI because it will break each time you enter or exit a called program.

Debugging in WindX is much more difficult, but not impossible. Since you don't have access to the debugging environment in WindX, you have to put real ESCAPEs in the code. Since you can't do a command window, type W at the command line to get a clean window to work in, but be sure to pop the window before allowing the program to continue execution.

You can also use >+ and >- to add and remove logical ESCAPEs in the program.