

WHITE PAPER

Meeting the SQR Development Challenge

Effective development, testing and maintenance of SQR programs

Kevin Reschenberg
SparkPath Technologies, Inc.



Introduction

Programs written in the SQR programming language form an integral part of many PeopleSoft applications. This language is used to produce data conversions, reports, interfaces, and other processes. It can handle complex logic and formatting tasks that would be impractical in another tool such as Application Engine. For developers familiar with general-purpose procedural programming languages, SQR is a flexible solution for a broad class of business problems in PeopleSoft installations.

Flexibility and generality, however, come at a cost. SQR has evolved over a number of years. This has left it with overlapping functions, somewhat inconsistent syntax, and sets of deprecated statements. There are often several ways to accomplish a task in SQR. Each developer adopts a particular style of coding, which can make it more difficult to understand another's code. One requirement of the language (positioning of database column names) has had, in practice, the unintended consequence of encouraging poorly formatted code. A tool called SQRW is provided for running programs easily, but it is controlled by a large number of command-line flags that may be difficult to remember.

As a result, many PeopleSoft programmers and analysts avoid SQR, thinking that it is inefficient and difficult to write and maintain. This is unfortunate. Given the right tools and techniques, it is none of those and can, in fact, be a highly productive asset to the organization.

This white paper discusses approaches to running and debugging SQR programs (commonly called "SQRs"). The objective is to find a way for the programmer to focus on program logic and spend less time on the mechanics of program execution and debugging.

Executing SQR programs

SQRs are generally integrated with a PeopleSoft system by adding them to a menu within the application. When the user selects this menu item and enters any required run control parameters, the Process Scheduler runs the program and presents its output.

This method requires the developer to create various PeopleSoft objects (component, process definition, and possibly a table, page, and/or PeopleCode), add the program to the menu, and grant security to the menu item, before the program can be run. Then, while it is running, it is out of the immediate ability of the developer to influence its execution. For example, the developer cannot interact with the program through INPUT statements. All input to the

program must be supplied in advance. This solution makes development and unit testing relatively difficult.

Another method is to use SQRW, a small application that runs SQR programs on a client machine using a two-tier database connection. It presents a simple dialog panel. The user fills in various parameters (using command-line flags) and clicks an OK button to dismiss the dialog and run the program. The parameters are not saved and must be entered each time SQRW is run.

To save time, some programmers manually create desktop shortcuts to SQRW. The various flags can be entered into the shortcut, but the programmer still must understand the flags and how they are coded, and the small amount of space provided for this by Windows makes it difficult to enter and maintain the flags. Many programmers are not aware of this option and end up entering long strings of parameters every time they run SQRW.

A more effective option is to use a third-party tool designed for maintaining the parameters and running SQRs. SP Debugger for SQR is an example of this type of tool. It presents a form containing fields for the most-used options, and another field for additional flags. It saves these entries between sessions and redisplayes them automatically at startup. Different sets of parameters can be saved separately and selected later, or attached to Windows shortcuts. For example, one set could be created for the development environment and another for the testing environment. In addition, the panel is not dismissed when the SQR program runs. This means that the SQR can be run repeatedly by clicking a single "start" button. SQRW users will immediately recognize the time savings and the ability to reduce mistakes that could cause the run to fail.

Debugging techniques

SQR developers traditionally have debugged their programs by coding DISPLAY or SHOW statements to print messages and intermediate results to the log file, running the program, and then modifying the DISPLAY/SHOW statements and rerunning as needed. This brute-force method takes time, clutters programs with extraneous statements, exacts a performance penalty, and can introduce the risk of inadvertently changing the flow of control in a program.

SQR improves the situation somewhat by providing the #DEBUGx and #IFxxx directives. When #DEBUG is placed at the beginning of a statement (for example, a SHOW), that statement is executed only if the -DEBUG command-line flag is specified. Other combinations are supported. For example, a line beginning with #DEBUGAB is included only if a -DEBUGA, -DEBUGB, or -DEBUGAB flag is specified, while a #DEBUG line is included if

-DEBUG or any of the other combinations is found. #IFDEF DEBUG can be used to mark a block of code. Other configurations are possible using #DEFINE and #IFDEF, #IFNDEF, or #IF using defined substitution variables and constants.

While very flexible, this feature requires good documentation to be useful (What does -DEBUGABFH mean for a particular program?) and it still leaves the program full of code that exists only for debugging purposes. In addition, writing all of the #DEBUGs in the first place consumes extra time for coding and correction of syntax errors. This method also requires the programmer to predict in advance where the problems are likely to occur, so the debugging code can be added in the appropriate places. If this is left until testing time, programmers will often insert the SHOW statements without the #DEBUG wrappers—and those statements can end up in production.

SQR also provides a SQL “trace” feature through the -S command-line flag. This flag instructs SQR to list the SQL statement for each cursor, along with the number of compiles, executes and rows processed for each. This listing is produced at the end of the run. It can give useful information, especially for optimizing SQL; however, it is rarely of use in debugging logic errors.

An interactive debugger allows us to avoid these problems, while providing detailed, real-time control over program execution. Using SP Debugger for SQR, the programmer can step through a program, one statement at a time, and see the flow of control as it happens. The programmer can also inspect and modify variable values, set breakpoints, set a watch on a variable (instruct the program to pause when the variable’s value changes or meets a specified condition), skip over procedures that are not of interest, and watch as the program steps automatically.

While providing this immediate control, SP Debugger makes it unnecessary to modify your program. A program can be debugged without adding statements and running the risk of damaging the program.

Errors often occur within included source (.SQC) files. SP Debugger shows the entire expanded program (including both the main file and all included files) in one place. When a syntax or run-time error is detected by SQR, SP Debugger shows the approximate location of the error regardless of whether it is located in the main file or an SQC. This expanded view also permits easy text searches to find particular variables, procedures, etc. In addition to the expanded code, the debugger displays a tree-like structure map of the program, showing each procedure name or nested SQC and the name of the file in which it is located.

The -S and -DEBUG features can still be used under SP Debugger and switched on or off as needed. The debugger shows inactive code (such as that wrapped by inactive #DEBUG or

#IFDEF directives) in a light gray font, thereby greatly improving the readability of the program.

Unit testing

A debugger helps to find bugs, but it is also a useful tool for unit testing during initial development.

```
! Clear $PLANTYPES= general plan type
let $Plantypes = $PlantypeChar1 || '%'
begin-sql
  DELETE FROM PS_X_TEMP
  WHERE  EMPLID = $Emplid
  AND    PLAN_TYPE LIKE $Plantypes;
end-sql
if #sql-count = 0
```

Debugging session in progress:

Checking a variable's value enables the programmer to detect a problem before the SQL is executed

A new program rarely works correctly the first time. Rather than running the entire program and then trying to determine what went wrong, it is often better to step through the program using the debugger. In this way, the developer can watch the logic flow, review the design, and spot problems that might not appear during the initial tests. Once a problem has been identified and corrected, the programmer can set a breakpoint after that section, run the program to the breakpoint, and then step through the next section of code to find any additional problems.

Save time, reduce effort, and improve code quality

While not a tool for *building* programs, SP Debugger for SQR helps developers work more efficiently and produce better code. It allows them to focus on program design, coding and testing, while spending less time on “hacking” their programs to see what is happening. By making the development and testing process easier and more productive, it encourages practices that result in a better product for your users.

SP Debugger for SQR is available at www.sparkpath.com/products.php