Moving From SabreTalk to C

Part 1: How SabreTalk "Procedures" Become "C" Functions

by Jeff Robinson

Now is the best possible time for SabreTalk programmers to consider moving to the TPF/C language. Not only is “C” one of the hot languages of the computing industry, but it also have so many similarities to SabreTalk that the transition should be relatively painless for most SabreTalk programmers.

So, with that in mind, beginning with this issue and continuing in the next two, I’ll explore the basic concepts needed to help SabreTalk’ers get started learning “C”. In this first lesson, I’ll begin by comparing the basic programming formats used by SabreTalk and C programmers. Now, without further delay, let’s begin!

The Basic SabreTalk And C Coding Segments

When you learned to write SabreTalk programs, you wrote each program using a format similar to the following basic structure:

```
prog1: PROC;
    DCL statements

    /*procedure source code goes here*/

    END prog1;
```

The preceding sample procedure shows the basic components needed to compile a program cleanly in a SabreTalk environment.

In TPF/C, the basic program segment is called a ‘function’. A function in TPF/C will contain the necessary code to carry out a specific set of actions that the programmer wishes to have done. If we took the preceding SabreTalk sample program and moved it to TPF/C, it might look like this:

```
#include <stdio.h>
#include <tpfapi.h>
void prog1()
{
/*Data declarations statements*/

/* C source code goest here  */
}
```

Here’s an explanation for each part of the preceding “C” function:

1. #include <stdio.h>
   #include <tpfapi.h>

The “#include” statements in these two lines are similar to the “%INCLUDEAF” statements used by SabreTalk in that they cause the insertion of previously writted “DECLARE” statements to be entered the program file. In “C”, the two files used, stdio.h and tpfapi.h, are known as header files. However, in “C”, header files are not pre-compiled and may also include additional non-executable items needed by the compiler to resolve labels used by the programmer.

2. void prog1()

This statement is equivalent to the PROC; statement in a SabreTalk program. Every “C” function must begin with a statement similar to this one.

The first word, void, is a “C” keyword (or reserved word). The purpose of this keyword is to indicate the type of data that the function will return. In this instance, the “void” keyword here indicates that this function will NOT return a value to the process (usually another function) that invoked or called it.

The second word, prog1, is the function’s name — which is the same as the “prog1:” label in the SabreTalk program. A function name can be made up of any number of alphanumeric characters, underscores, and a limited number of other special symbols. Most importantly, it’s always best to use function names that are easy to remember for now and in the future.

The left and right parentheses “()” that appear adjacent to “prog1” is a standard “C” convention used to indicate that a named item is a function rather than a variable. Thus, we know that prog1 is a function because it written as “prog1()”.

3. The left curly brace “{“ following the function name is used throughout “C” language to indicate the beginning of a block of code. Here, the left curly brace indicates the start of the function’s block of code.

4. /* <C source code goest here > */

This is a standard “C” comment. Like with SabreTalk, “C” comments begin with the “/*” composite and terminates with the “*/” composite. Comments are used not only as programmer documentation but as well as a means of source code “beautification”.

The comment and anything appearing with the comment parameter are IGNORED during the compilation process. So, you can place a comment on a line by itself as done in our

example or place them adjacent to instructions on the same line. "C" even allows comments to be imbedded within the flow of an executable statement! Either way, the entire comment will be ignored.

5. Finally, the right curly brace "}" is used to indicate the end of the block of code within the function. Thus, it is equivalent to the "End prog1;" statement in the SabreTalk program. No other statements for that function can be coded after this right curly brace.

Furthermore, without this ending right curly brace, the compiler would not know where the function block ended and would most likely generate an error during compilation. Omitting the right curly brace would result is similar to leaving off the "End prog1; statement in the SabreTalk program.

What Goes Into A Function?
Now that you've seen how an "empty" TPF/C function looks, you may be wondering how a function looks that actually does something.

Usually, functions contain the processing statements to accomplish a specified unit of work. This includes calling other functions as well as doing whatever else the programmer wants to do. Following is a more complete example of the preceding SabreTalk procedure along with its corresponding "C" program; however, we'll save the explanation of what all the new "C" code means until later.

```
prog1: PROC;
                    DCL wt BIN(31);
                    DCL msg CHAR(20);
                    DCL rwfactor BIN;
                    DCL clear PIC '999';
                    DCL clearance BIN;
                    DCL decr BIN;
                    DCL obsthgt BIN;
                    DCL i BIN;
                    decr = 100;

loop:               DO i = 1 TO 10;
                      clearance = wt * rwfactor - obsthgt;
                      IF clearance > 0 THEN GOTO fin;
                      wt = wt - decr;
                    END;
                    decr = decr + 500;
                    GOTO loop;

fin:                clear = clearance;
                    msg = 'clearance = ' || clear || ' feet' ;
                    END prog1;
```

The "C" Equivalent:

```
void prog1()
{
  long wt;
  char msg[20];
  short rwfactor,clear,clearance,decr,obsthgt,i;

  decr=100;

loop:
  for(i=1;i<=10;i++)
  {
    clearance = wt * rwfactor - obsthgt;
    if (clearance>0)
      goto fin;
    wt -= decr;
  }
  goto loop;

fin: clear = clearance;
     sprintf(msg,"clearance = %d feet\n");
}
```

In the Next Issue…
In the next installment, we'll explore the similarities and differences in data concepts/manipulation you'll face as you move from SabreTalk to C programming.

Finally, you can find the complete lesson series upon which these articles are based by going online to www.robisoft.com and visiting the "TPF Education" section. In addition to the lessons, you'll find quizzes and tests related to each topic.