[HPBW] Using CSUBS With HP BASIC for Windows

stu beatty

* Like other versions of HP BASIC, HP BASIC for Windows (HPBW) is a powerful tool; but sometimes a user may find that a specialized task can be performed better by a custom-designed C subprogram (CSUB). The E2061A HPBW CSUB Toolkit allows to integrate such C subprograms into your HPBW program.

* The heart of an HP BASIC for Windows CSUB is a 32-bit Dynamic Link Library (DLL). These DLLs can be generated by any compiler and linker combination that generates a Win32 DLL for Intel processors, such as Visual C++ version 2.X; other languages can be used if they produce Win32 DLLs and if you pass your parameters using normal C-language calling conventions.

Win32 DLLs can make use of all the standard C library runtime routines -- as well as some other functions in the Win32 API, such as the "kbdcrt" utility functions that allow CSUBs to access the HP BASIC keyboard and display.

In HPBW, CSUBs are invoked by the CALL statement, executed either in a program or "live" from the keyboard. The parameters passed can be of any HP BASIC data type -- including numeric, complex, string, and arrays. Optional parameters are allowed, and CSUBs can share COM data with the calling HP BASIC program.

Error reporting is handled by the C "return()" function: a "return(0)" indicates no error, while any non-zero return value is reported by HPBW as an error with the numeric value returned by the CSUB (as well as the line number of the CALL that resulted in an error). An exception handler automatically returns control to HPBW from a CSUB on any floating-point exception.

Note that HPBW can still load and run existing HT BASIC CSUBs developed with the DOS extender from Phar Lap; these old CSUBs do not need to be recompiled into DLLs unless they access hardware, and will work alongside the new CSUB scheme.

* An overview of the steps required to create and run an HPBW CSUB follows. There are four steps:

 % Create the CSUB protoytpe file.
 % Create the CSUB DLL.
 % Run the HPCSUBW utility provided with the E2061A CSUB Toolkit.
 % LOAD the CSUB.

* Creating a prototype file is easy. A good programming practice is to start your development with the definition of the CSUB's purpose and parameters; this can be done (in prototype form) by defining an HP BASIC SUB declaration to define all the parameters, their type, and their order.

As a simple example, consider a CSUB to calculate the average of all the numbers in an array. This CSUB has two parameters: first, the array to be

averaged, second, the array to hld the result. The prototype SUB statement is then:

```
10  END
20  SUB Average( REAL Dat(*), Avg )
30  SUBEND
```

After you define this prototype (if you were defining other CSUBs you could add declarations for them as well), execute RUN and then STORE the prototype program as a PROG file:

```
STORE "MY_SUB.PRO"
```

* Next, write the C language program and create the DLL:

```
#include "csubw.h"

int average ( int npar,               /* Number of parameters passed. */
              realptr dat,            /* REAL array elements. */
              dimptr dat_dimen,       /* Dimension pointer for array. */
              realptr avg )           /* REAL result variable. */

{
  int i;                              /* Loop counter. */
  int num_elements = dat_dimen->cae;  /* Get number array elements.  */
  double sum = 0.0;                   /* Clear sum. */

  if( num_elements == 0 )             /* Check for no data. */
  {
    return( 11 );                     /* Error: NUMERIC VALUE REQUIRED. */
  }
  for( i=0; i < num_elements; i++ )   /* Sum all array elements. */
  {
    sum += dat[i];
  }
  *avg = sum / num_elements;          /* Put sum in return variable. */
  return( 0 );                        /* No error. */
}
```

The include file "csubw.h" is provided with the CSUB Toolkit and defines all the types of pointers to HP BASIC variables (the details are provided in the Toolkit's manual). Notice that the first C parameter gives the number of parameters passed and is a part of all HPBW CSUBs; this is useful for handling optional parameters, but we can ignore it here.

Before compiling and linking, we have to declare exported symbols. One way to do this is with:

```
_declspec( dllexport )
```

-- in your source file. Another way is with a minimal DEF file; this should contain at least the name of your DLL and its entry points. For example:

```
LIBRARY   avgsub
EXPORTS   average
```

These source files can be compiled and linked using Visual C++ 2.0. The result is a Win32 DLL named "avgsub.dll". This DLL file can be located by HPBW in a variety of directories -- including the current directory, directories listed in your PATH, HPBW's directory, and the WINDOWS directory or WINDOWS\SYSTEM directory; to minimize confusion, you might want to create a standard place to keep all your CSUB DLL files.

Once you have the PROG and DLL files, you can then create the actual CSUB using the HPBCSUB utility; the resulting CSUB file will have the same name as the prototype file, but with a ".CSB" suffix instead of a ".PRO" suffix.

Our example can be created by executing the following command in DOS or a DOS window:

```
HPCSUBW MY_SUB AVGSUB.DLL
```

This creates the PROG file MY_SUB.CSB that contains the CSUB "Average".

* The last step is to LOAD the new CSUB into an existing HP BASIC program and CALL it. There are several HP BASIC statements available for loading (and deleting) CSUBs; for example:

```
LOADSUB ALL FROM "MY_SUB.CSB"
```

After writing a few lines of test code in HP BASIC and executing a LOADSUB statement, the HPBW BASIC program for our example might look like this:

```
10  INTEGER I
20  REAL Dat(0:9),Avg
30  FOR I=0 TO 9
40    Dat(I) = I
50  NEXT I
60  CALL Average( Dat(*), Avg )
70  PRINT "Average is ";Avg
80  END
90  CSUB Average( REAL Dat(*), REAL Avg )
```

If you use STORE to file this program to disk, the CSUB will be filed along with the rest of the program in the PROG file. A subsequent LOAD will yield a fully functioning program (as long as HPBW can find the DLL at run time). If you SAVE a program containing a CSUB, however, the CSUB is not saved, and you have to do a LOADSUB to bring it in again after you GET the program.

* The CSUB Toolkit provides utility functions for communicating with HPBW's keyboard and display. For example, "kbdcrt_readkbd()" returns the contents of the KBD$ buffer, and "kbdcrt_printstr()" writes a string to HPBW's "Output Area". Routines are available to access the KDB and CRT status and control registers, scroll the display, and other functions; the use of these routines

is explained in the CSUB documentation.

Standard C routines, like "fopen()" and "fprintf()", can be used for file-I/O; note, however, that these functions have no intrinsic knowledge of the unique file formats associated with HPBW (such as PROG or ASCII), and normal DOS file types are recommended for use with CSUBs.

As far as interface I/O is concerned, users may be interested in using CSUBs to improve handling of interfaces that HPBW already supports, such as HPIB -- or gaining access to interfaces that HPBW does not support, such as the VXI backplane on an embedded controller.

Trying to use a CSUB to get to an interface that is already supported by HPBW is not recommended; it can lead to conflicts with the HPBW driver, and most performance issues can be addressed by a better use of HPBW's extensive I/O formatting capabilities.

If you want to access an interface that HPBW does not support, CSUBs are the only avenue; such interfaces are often provided with DLLs and a CSUB could make use of these DLLs -- except for one problem: most of the DLLs out there right now are 16-bit DLLs and, as noted, a CSUB is based on a Win32 DLL.

A 32-bit DLL cannot interchange pointers with a 16-bit DLL without going through a process called "thunking". Thunking introduces several confusing layers between you and your goal, and forces you to be extremely careful in your handling of data types; it also forces you to produce an extra 16-bit DLL beyond the CSUB and the target 16-bit DLL. A good thunking example would be too long for this document; consult your Windows documentation or the "\MSVC32S\UT\SAMPLES" directory.

Some special considerations apply if your CSUB uses thunking. Thunking is enambled by a "UTRegister()" call at DLL load time; this happens when the DLL is attached to your process, not when you execute LOADSUB. Most CSUB examples (including the one in this document) contain only the entry points needed by CALL; to perform the correct "UTRegister()" sequence, you must add an initialization routine to your source file of the form:

```
BOOL APIENTRY
Dllnit( HANDLE hinst, DWORD fdwReason, LPVOID lpReserved )
{
   /* Perform proper UT registration sequence. */
   return( TRUE );
}
```

When creating such a CSUB, you must provide the linker with the name of the DLL entry point. This can be done with a link option, such as:

```
/ENTRY:"Dllnit"
```

* A few sophisticated users have developed CSUBs that directly access CRT hardware on HP Series 200/300 HP BASIC/WS systems, an impressive accomplishment -- but one that cannot be ported to the PC environment as

the PC display systems are entirely different.

Similarly, CSUBs developed for RMB-UX that access interface hardware are useless on the PC. Any old CSUB that required intimate knowledge of specific hardware must be abandoned when you port to the PC/Windows platform.

[<>]