PowerBASIC Gazette

We put the POWER in BASIC!

Volume 2 Issue 1

PowerBASIC creates DLLs for Windows!

PowerBASIC announces shipment of the much anticipated PowerBASIC DLL Compiler for Windows! The new PB/DLL lays claim to the title of "World's Fastest PC Compiler". Even though the structure of BASIC requires two pass compilation, this product actually compiles to native code at speeds in excess of 500,000 lines per minute on a fast Pentium CPU. This level of performance may prove to be a lofty goal for other compilers, which typically operate at more modest speeds.

Initial press reviews are scheduled by most of the respected programming trade journals like PC Week, Access/Visual Basic Advisor, Visual Basic Programmer's Journal, Information Week, Visual Basic Tech Journal, and more. In a "First Look" at PB/DLL, J.D. Hildebrand, Editor of Visual Basic Tech Journal, remarked, "You don't have to wait for Microsoft or learn another language to get better performance out of your Visual Basic apps — just identify the bottlenecks in your code and compile the slow procedures as a DLL. It's a snap!".

PB/DLL is a native code compiler which creates industry standard 16-bit DLLs and EXEs using the full power of the 32-bit instruction set. Its target audience is the Visual Basic programmer who needs improved performance from his application. Computational code can simply be excised from a VB project, compiled to a DLL, then called directly by VB or any other Windows language.

According to company president Robert Zale, "We're about to bring some sensibility back to an era of fat, bloated code. Some interpretive languages require a megabyte of code and run-time just to print "Hello World!". The better C compilers do it in 40k to 50k. But PB/DLL needs only about 3k, and with absolutely no run-time requirements! That's a level of performance we can live with." For further proof, he noted "Just look at the debugger included with this product. PBDW was crafted entirely with PB/DLL. A native-code debugger written entirely in BASIC... that must be a first!"

The Sieve of Eratosthenes is a traditional benchmark used to compute prime numbers. Using standard Visual Basic code, 500 iterations took 28.67 seconds. Using PB/DLL, the code code ran in 3.02 seconds, an improvement of 849%. The Long Integer test, first published by BasicPro Magazine (now Visual Basic Programmer's Journal) calculates the speed of standard math operations on 32-bit signed numbers. For one million iterations, Visual Basic took 7.58 seconds, while PB/DLL ran it in just 1.09 seconds. The Array test, which fills and sorts a large numeric array took Visual Basic 927.96 seconds. That same code compiled with PB/DLL took 73.60 seconds, an improvement of 1,161%. Optimizing with PB/DLL's Array/Sort added yet another order of magnitude, to just 0.44 seconds. Tests were performed using a 90-MHz Pentium CPU.

"(If you) find that Visual Basic simply isn't fast enough for your code-intensive routines, you absolutely owe it to yourself to add PB/DLL to your arsenal of tools," commented Karl E. Peterson, contributing

(Continued on page 6)

QuickPak Pro for PowerBASIC

QuickPak Professional, the most popular Basic toolkit ever, is now available for PowerBASIC programmers. While the original QuickPak Pro sells for $199, PowerBASIC, Inc. has made a special purchase and passes the savings along to you. Now you can save $70 off the regular price of $199.

QuickPak Pro is a toolbox of more than 500 Basic and assembly subroutines which help developers improve the quality of their applications and complete them faster and easier. Included are programs for windowing, creating pull-down and Lotus-style menus, user-interface dialogs, accepting data input, accessing DOS and BIOS services, financial calculations, string manipulation and much more. Extensive documentation is provided with tutorials on files, arrays, subprograms, sorting, compiling and linking, plus many other related topics.

"Customers keep asking if there is a version of QuickPak Pro for PowerBASIC," reported Bob Zale, President of PowerBASIC, Inc. "It is an extremely popular package that

(Continued on page 5)

what's inside...

...and don't miss the Power Tips throughout the issue!

Visit PowerBASIC on the Net

Are you surfing the World Wide Web on the Internet? If so, catch a wave and ride on over to **http://www.powerbasic.com** and visit *PowerBASIC on the Net* — an interactive web site where you can find the latest news and information about all of our products.

Browse our *Software Catalog* and place your order online. Or access our *Tech Center* and leave feedback to our technical support engineers.

Find out more about the PB/DLL compiler for Windows. Download a white paper with technical details and specifications. Or download an interactive demo which demostrates how much faster your Visual Basic programs can be when you optimize them with machine code DLLs.

Browse the feature list of PowerBASIC for DOS (including a list of those which are not in QuickBASIC). Download a demo of the PowerBASIC 3 compiler that allows you to type in your own code, compile it to an EXE and see for yourself why PC Magazine chose PowerBASIC as the best DOS Basic compiler in their Editors' Choice awards (September 28, 1993).

Download a copy of the new shareware First Basic compiler for DOS and give a copy to a friend or upload it to your favorite web site. First Basic is the only shareware Basic compiler which is fully compatible with your GWBASIC and BASICA source code.

Read the back issues of the *PowerBASIC Gazette*, complete with all of the Power Tips in one convenient location to browse and download.

Visit the *Source Code Center* and get a head start on those difficult programming projects with examples for playing WAV files on your SoundBlaster card, reading the serial numbers from your floppy or hard disks, quickly counting the number of lines in a text file and much more.

We're adding new pages and features all the time, so make *PowerBASIC on the Net* one of your regular stops.

If you're looking for a place on the Internet to find discussions about PowerBASIC for DOS and PBasic, set your Usenet newsgroup reader to *comp.lang.basic.misc* and *alt.lang.basic*. Windows programmers can find PB/DLL discussions by setting your newsgroup reader to *comp.lang.basic.visual.3rdparty*.

We've made it easier for CompuServe members to locate our support forum. Just type **GO POWERBASIC** at any command prompt (!) or from within WinCIM. We're located in section 12, where you can read the latest news, join conversations (or just read them if you're shy) and download from a large collection of frequently asked questions, source code, libraries and other useful files.

If you're not already a CompuServe member, you can join easily by calling **Representative #194** at **1-800-848-8199**. In the United Kingdom, call **0800-289-378**. In Germany, call **0130-37-32**. In the rest of Europe, call **44-272-255-111**. Outside the U. S., Canada, and Europe call **614-457-0802**.

Power Tip

A convenient location for allocating global arrays and pre-setting global values in your DLL is in the *LibMain()* function. Any code contained in *LibMain()* is only called when the DLL is first loaded into memory, not every time the DLL is accessed.

```
FUNCTION LibMain%(BYVAL hInstance%, BYVAL wDataSeg??, _
         BYVAL wHeapSize??, lpszCmdLine AS ASCIIZ PTR) EXPORT

  DIM x(1 to 30) AS GLOBAL INTEGER
  IF ERRCLEAR THEN   'not enough memory
    FUNCTION = 0
    EXIT FUNCTION
  END IF

  FUNCTION = 1       'success!

END FUNCTION
```

The ERRCLEAR keyword is used to test for any errors allocating the array. If there is an error the return value is set to 0 to indicate a memory error and the Function ends. Otherwise a return value of 1 is set to indicate that everything worked smoothly.

You could then add code to your DLL to access the x%() array.

```
FUNCTION GetXvalue%(BYVAL Ele ment%) EXPORT
  FUNCTION = x%(Element%)
END FUNCTION

SUB SetXvalue(BYVAL Element%, BYVAL Value%) EXPORT
  x%(Element%) = Va lue%
END SUB
```

And you can use the built-in ARRAY SORT statement for fast sorting:

```
SUB SortArray() EXPORT
  ARRAY SORT x%()
```

Contact Information:

PowerBASIC, Inc.
316 Mid Valley Center
Carmel, CA 93923

On CompuServe:
GO POWERBASIC, section 12

(800) 780-7707	*Orders*
(831) 659-8000	*Voice*
(831) 659-8008	*Fax*

Word Wide Web:
www.powerbasic.com

Internet Email:

info@powerbasic.com	*Information*
sales@powerbasic.com	*Sales*
support@powerbasic.com	*Support*

PowerBASIC Gazette is a creation of the PowerBASIC, Inc. product development and marketing staffs. It's purpose is to inform customers and prospective customers of new developments and tips for using existing products.

Getting started with PB/DLL

You've got PB/DLL, you've carefully read through the manual and you're ready to try PB/DLL. But where do you begin? You can start by taking a look at the samples located in the EXAMPLES subdirectory of PB/DLL.

The following instructions take you step by step through compiling the VBEX1.BAS example which comes with PB/DLL into a DLL and calling it from an example Visual Basic application.

VBEX1.BAS contains a Function called AddOne() which will add one to any single-precision floating point value passed to it:

```
Function AddOne!(ByVal x!) Export _
                           RetPrm
  Addone! = x! + 1
End Function
```

The *Export* keyword tells PB/DLL that you want to call the Function from Visual Basic. The *RetPrm* keyword is required because VB uses a hidden parameter to return floating point values from a DLL.

Compile the VBEX1.BAS file into a DLL using either the command line compiler (PBDLL.EXE), or by clicking on the PB SHELL icon in the PB/DLL group in Windows. Now copy VBEX1.DLL into your VB directory so that your VB program can find it.

Start Visual Basic by double clicking on its icon. Then click on the "File" menu in VB and select "New Project".

Now we've got a blank form in front of us. Let's add a button to it in the middle of the form. Then press F4 to get the properties window for the button and set the Caption to "OK".

Double click on the button in the form and you will get a window to write your code in. Type in the following code inside the Sub Command1_Click ():

```
For i% = 1 TO 10
  x! = AddOne!(x!)
  Debug.Print x!
Next i%
```

In the "Object" combo boxes select "(general)" and "(declarations)". Type in the following Declare statement all on a single line:

```
Declare Function AddOne! Lib
        "VBEX1.DLL" (ByVal x!)
```

Save your project as VBEX1 and then click on the Run button in the VB toolbar.

Click on the OK button in the form and watch the Debug window. You'll see the output of *x!* as *AddOne* operates adds one to *x!* each time through the loop.

Congratulations! Now your ready to tackle the rest of the sample code in the EXAMPLE subdirectory of PB/DLL, or start identifying the bottlenecks in your Visual Basic code and moving your Subs and Functions into DLLs.

Good candidates for converting VB code into DLLs are computational code (such as calculating tax tables), binary, sequential or random access file I/O, string manipulation (such as string parsing), array sorting and searching, and any code which relies heavily on direct calls to the Windows API.

PB/DLL also allows you to directly access memory through PEEK and POKE or data pointers. If you need to access a hardware port directly (for interfacing with a data gathering device such as a bar-code reader) you can use the INP function and OUT statement.

The Introduction chapter in the PB/DLL documentation is a good place to find out all of the many features available in PB/DLL. Be sure to carefully read chapter 2 on creating DLLs as well as Appendix B which provides

Dave's World...

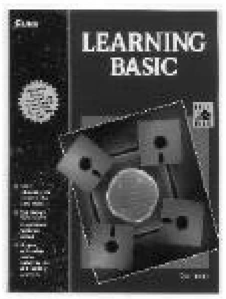
I'm happy to report that the Christmas party went very well. And since I made up all the questions I won the trivia contest. Although, secretly I think a few people might think that I somehow cheated. That's okay, their still not getting my PowerBASIC pocket protector back.

Ever since the first beta version of PB/DLL arrived on my desk I have been extremely excited. I've tried Windows programming in C/C++ and Visual Basic, but I missed too many features from PowerBASIC, and I always gave it up. Now I'm having fun converting all the utilities I've written for DOS into Windows programs. Who can say no to multi-megabyte arrays? Or protected mode assembler programming? Certainly not me.

Speaking of utilities, I'd like to give a hardy thanks to Lance Edmonds, John Pearson and several other users on CompuServe. They used the PowerBASIC forum as a meeting place to write a library for sending faxes from a PB program using a fax modem. And after all their hard work they gave away the source code to the public domain. You can download PBFAX from CompuServe or the PowerBASIC on the Next web page.

Oh, did you notice who won the Super Bowl this year? <grin> ■

Learn Basic Programming for just $29.95



Fast and easy access to the Power of Basic programming! Learn to program with simple, step-by-step instructions and examples for the beginner to intermediate level programmer. Over 600 pages of information... and a full copy of PowerBASIC 2.1 is included! PowerBASIC offers an integrated programming environment with compiler, text editor, debugger, online help system and much more. Need a special program? Learn to write it yourself with this powerful programming tool. Use the order form on page 7 to place your order.

System Requirements: IBM or compatible, PC, XT or better, 512k RAM, DOS 2.0 or later, hard drive or 2 floppy drives.

Working with Visual Basic Strings in PB/DLL

One of the most powerful features of the Basic language is the ability to create, modify and remove dynamic strings as your program executes. Like Visual Basic, PB/DLL contains a very powerful string engine.

Native VB Strings

Internally, Visual Basic strings are 4-byte handles which help VB to locate the string data in memory. PB/DLL also uses 4-byte handles to identify its dynamic strings, but because it is a separate compiler, it must use its own string handling engine to access the data. For this reason, string handles from the two string engines are not interchangable.

Visual Basic does provide an Application Programming Interface (API) which allows other languages and compilers to access Visual Basic strings through a handle. Using this API, code can be written which accesses Visual Basic strings. The file VBAPI.INC, which can be found in the EXAMPLE subdirectory where PB/DLL was installed, contains code to access the VB API.

To pass a Visual Basic string to a DLL, use the following syntax in your VB code:

```
Declare Sub VbString Lib “VBS.DLL”
                  (Text As String)
```

When you pass a dynamic string from VB by reference to a DLL, VB actually passes a data pointer to the string handle. When you call the VB API to access the string, the pointer to the handle is required, not the actual handle itsef. Therefore, in our DLL we would use the following syntax to get the data pointer:

```
SUB VbString(BYVAL VbHandPtr _
                AS DWORD) EXPORT
```

The *BYVAL* keyword allows us to directly retrieve the 4-bytes placed on the stack by Visual Basic, which is a pointer.

Once you have the data pointer, a call to the VB API function *VbDerefHlstrLen()* will return the location in memory and length of the VB string. The following code will copy the data from a Visual Basic string into a PB/DLL string:

```
  DIM StPtr AS DWORD
  DIM Length AS INTEGER
  StPtr = VbDerefHlstrLen _
         (VbHandPtr, Length)
  PbStr$ = PEEK$(StPtr, Length)
```

StPtr receives the location in memory where the string data is stored and *Length* receives the number of bytes in the VB string. PEEK$ is then used to copy the data into *PbStr$* where your code can then access it as it would any other string. A function called *VbStringToPb* is located in the VBAPI.INC file to do does this low-level work for you.

If you need to change the data in the VB string itself, the VB API function *VbSetHlstr()* is used. Using it, you can even change the length of the string. The following code will assign a new value from a PB/DLL string to the VB string:

```
StPtr = STRPTR(PbStr$)
Length = LEN(PbStr$)
VbSetHlstr VbHandPtr, StPtr, Length
```

Using the *VbHandPtr* variable which is passed by your VB code, we assign the string a new value. *StPtr* points to the data in our PB/DLL string while *Length* holds the number of bytes in the string. A call to *VbSetHlstr()* sets the new value. If *Length* is zero, the VB string is erased. If *Length* is one then *StPtr* is assumed to point at another VB string handle.

Finally, if you need to return a VB string from a DLL function, you return a string handle pointer as a DWORD. The PB/DLL syntax is:

```
FUNCTION VbStr() EXPORT AS DWORD
```

To create a Visual Basic String from a PB/DLL string you use the VBAPI function *VbCreateTempHlstr* from within your DLL:

```
PbStr$ = “Hello”
Length = LEN(PbStr$)
StPtr = STRPTR(PbStr$)
FUNCTION = VbCreateTempHlstr _
                (StPtr, Length)
```

VbCreateTempHlstr returns a long integer which holds a temporary string handle. Assigning it to the keyword FUNCTION will return that handle to the calling VB program. You’ll find the *PbStringToVb* function in the VBAPI.INC file which already does this low-level work for you.

ASCIIZ Strings

Frankly, all of this is a lot of work and it also impacts your performance. Additionally, your DLL is only useable with Visual Basic applications.

A much easier method is to pass an ASCIIZ string (pronounced “askee z”). An ASCIIZ string is a series of bytes (your string data) followed by a Null byte (a CHR$(0)). VB does not natively support the ASCIIZ string type. However, when you pass a VB string to a DLL using the *ByVal* keyword, VB will convert it into an ASCIIZ string before passing it (in reality, VB strings are stored as ASCIIZ strings internally so no conversion is actually done then, it simply places a 32-bit pointer to the actual string data on the stack). To access the string from your DLL use the native ASCIIZ variable type. All the API calls are eliminated!

In Visual Basic you would declare your Sub using the following syntax:

```
Declare Sub CapFirst Lib “CAP.DLL”
          (ByVal Text As String)
```

The *ByVal* keyword tells VB that the DLL expects an ASCIIZ string to be passed to it. In your PB/DLL code you would then use the following syntax to access it:

```
SUB CapFirst(Text AS ASCIIZ) EXPORT
```

Even though you use the *ByVal* keyword in your VB code, VB does not pass a copy of the string to the DLL. It passes a pointer to the bytes in the string, so any modifications you make to the string in your PB/DLL code will be returned back to your VB code. If you don’t want to modify the VB string in your DLL, copy the passed string into a local variable and make your changes to that:

```
SUB CapFirst(Text AS ASCIIZ) EXPORT
  LOCAL Temp AS STRING
  Temp = Text
  ...
END SUB
```

If you do modify the incoming string, it is very important that you not exceed the original string’s length. If the original string is not long enough to hold the new value, a General Protection Fault can occur. To avoid this problem, pad your VB string with enough spaces before you pass it to the DLL:

```
x$ = “hello there” + Space$(255)
Call CapFirst x$
```

If CapFirst needs to lengthen the string for any reason, it has an additional 255 bytes to work with. On return from the DLL, the length of the above string will still be 266 bytes regardless of how the DLL code modifies it. You’ll need to trim off the unnecessary bytes. Use the INSTR function to find the trailing CHR$(0) byte and trim off everything from that point on:

```
  n  = Instr(x$, Chr $(0))
  x$ = Left$(x$, n - 1)
```

Using ASCIIZ strings are efficient because no separate calls are necessary to find them in memory or to modify them. The address of their data location is passed directly. The only down-side is that a CHR$(0) cannot be embeded into an ASCIIZ string used to pass data between your program and your DLL.

Conclusion

Passing strings from VB to PB/DLL *ByVal* and using ASCIIZ strings in your DLL code is the most efficient method. Simply put, native VB strings are less efficient than PB/DLL strings and ASCIIZ strings. However, if you really do need to work with them, the VBAPI is there to help you.

(Continued from QuickPak Pro on page 1)

we feel many of our customers could take advantage of in their coding."

All of the QuickPak routines are extremely easy to use. The number of parameters required is kept to an absolute minimum, and complete instructions are included for each program.

QuickPak Pro contains a comprehensive set of scientific and financial functions, including those offered in most commercial spreadsheet programs. Many programs are provided for sophisticated window handling. The window manager accommodates ten levels of window nesting, and may be easily expanded to handle even more if needed. Even use high-level dialog boxes you can just "drop-in" to your own code.

Included are a full screen editor with word wrap, block operations and mouse support, a pop-up calculator, a calendar, an ASCII chart, a file manager, a mathematical equation solver, and a browse program that handles text files of nearly any size. All with source code.

The DOS services permit operations that are sometimes difficult to do using Basic code alone. Developers may obtain a list of filenames from disk, set or get a file's time, date, and attributes, determine the total and free space remaining, read and change the default drive, and read and write disk sectors and volume labels directly. A complete set of file and printer routines are provided to eliminate the need for ON ERROR trapping.

Other low level routines provide string and file encryption, file and array searching and sorting, date and time calculations, access to mouse services, EMS memory for data storage, determine hardware information, saving text and graphics images to disk and extensive string manipulation. A sophisticated screen dump program is included that operates in all Basic supported graphics modes, and works with any printer that recognizes the IBM/Epson or HP LaserJet control codes.

"I give this package an unqualified recommendation on the basis of its value and utility," Bruce Tonkin, Dr. Dobb's Journal.

All of the assembler routines that process strings are provided in an alternate version that ignores capitalization. Video routines operate on any screen page, and automatically support the 43 and 50-line EGA/VGA text modes.

"I can heartily recommend QuickPak Professional to both the beginner (for its extensive tutorials) and the professional (for its breadth and depth of utilities)," M. Steve Baker, Programmer's Journal.

QuickPak Pro is compatible with the QuickBasic version, making it easy to move those QB programs over to PowerBASIC.

QuickPak Pro for PowerBASIC requires PowerBASIC 3.0 or later and can be purchased

New! First BASIC compiler for DOS

Traditionally, PC operating systems like MS-DOS and PC-DOS have included a Basic interpreter so that users could learn about programming. However, newer operating systems (Windows 95, OS/2, PC-DOS) no longer include Basic.

PowerBASIC, Inc. introduces the First BASIC compiler for DOS. First BASIC is a low-cost Basic compiler which takes your Basic source code and creates executables. Gone are 64k string limitations and slow interpreted code. Compiled code executes from 4 to 10 times faster than interpreted code.

Oh, and we forgot to mention one very important feature. It's shareware! That's right, you can download First BASIC from our World Wide Web site, CompuServe forum or Electronic Bulletin Board System (BBS) and try it for 30-days. We're so sure that you'll like it and want to register it that we didn't disable one single feature.

You get built-in statements for sorting arrays, adding and removing elements to arrays, even fast array searching. File I/O provides both file and record locking for multi-user programming. BCD variables let you easily create financial applications with fast, accurate math. Options are available to generate code for 8086 or 80286 CPUs and numeric co-processors. You can even link assembler object code (OBJ files) directly in to your program and much, much more.

Use the built-in editor to write your programs and the source-level debugger to test your code. Context-sensitive online help is only a keystroke away. The online help system covers every aspect of the language, including all of the new keywords we've added. You can even import the source code examples from the help system right into your program.

When you register First BASIC, you get the FBC command-line compiler which lets you compile your code from the DOS command line and the FBINST configuration utility which lets you customize certain features of the Integrated Development Environment (IDE), such as colors, editor hot keys etc. Registration also gets you an extensive User's Guide help file which walks you through various features of the compiler, including advanced topics like sequential, random and binary file I/O, graphics programming, event trapping, serial I/O, debugging your programs, and more. And, you get a free subscription to the PowerBASIC Gazette plus lots more source code examples.

First BASIC 1.0 requires DOS version 3.3 or later, 640k of RAM and at least one floppy drive (a hard drive is recommended). You can purchase the registered version for $25 plus $4 shipping and handling. Registration entitles the

Power Tip

If you are returning floating point values from a DLL to your Visual Basic application, don't forget to include the RETPRM statement in your function declaration. When Visual Basic calls a function in a DLL which returns a floating point value, it places a 'hidden parameter' on the stack which is a buffer for the return value. RETPRM tells PB/DLL to store the returned floating point value in that buffer:

```
FUNCTION Add!(BYVAL a!, BYVAL b!) EXPORT RETPRM
  FUNCTION = a! + b!
END FUNCTION
```

If you forget to include RETPRM, a "Bad DLL Calling Convention" error can occur in your Visual Basic program. RETPRM should not be used if you are returning integer or long integer values from your Function.

DLLs which use the Visual Basic convention for returning floating point values are not directly accessable from most other Windows programming languages. If you need to write code which supports multiple languages, use conditional compiling:

```
%VisualBasic = %TRUE
$IF %VisualBasic
  FUNCTION Add!(BYVAL a!, BYVAL  b!) EXPORT RETPRM
$ELSE
  FUNCTION Add!(BYVAL a!, BYVAL b!) EXPORT
$ENDIF
```

To compile a DLL compatible with Delphi or C/C++, simply change %VisualBasic to %FALSE and re-compile.

Book Report

Looking for reference material to help you get your programming done? Here are a few favorites from the staff here at PowerBASIC.

📖 *Visual Basic Programmer's Guide to the Windows API* by Daniel Appleman (ISBN 1-56276-073-7). If you are planning to use the Windows API in your PB/DLL code, this book is a must. It's easy to read and the VB code works with PB/DLL (usually without any necessary modifications).

📖 *Inside ODBC* by Kyle Geiger (ISBN 1-55615-815-7). If you want to use ACCESS databases, dBase databases or the Microsoft SQL Server from your PB/DLL code, this book gives you the inside scoop. Included on the companion CD are examples for VB which work well with PB/DLL.

📖 *Programming Windows version 3.1* by Charles Petzold (ISBN 1-55615-395-3). Known by most Windows programmers as The Windows programming bible. It's very easy to read and walks you step-by step through writing Windows applications using the Windows API. Source code examples are all in C, but are easily converted to PB/DLL if you have even a limited knowledge of C programming.

📖 *The Personal Computer from the Inside Out - Third Edition* by Murray Sargent and Richard Shoemaker (ISBN 0-201-62646-2). If you need to access hardware data ports this is your book. The bound-in disk comes with lots of assembler source code examples for communicating with switches, relays, A/D and D/A converters, motor controllers and more. Even example code for sending files from one PC to another

(Continued from PB/DLL on page 1)

editor for Visual Basic Programmer's Journal in the May 1996 issue. "Two thumbs up!"

PB/DLL brings a myriad of features from the award-winning PowerBASIC compiler to the Visual Basic programmer: Code pointers allow PB/DLL programmers to write callbacks and custom controls. Data pointers allow unlimited options for accessing data anywhere in memory. The In-line assembler allows programmers to mix assembler and Basic code together in the same source file. Built-in Array Sort, Scan, Insert and Delete for fast array manipulation. Unsigned Byte, Word, and Double-Word variables. 64-bit integers, extended precision (80-bit) floats, and currency variables with 2 or 4 digit precision. Native ASCIIZ string support for easy access to the Windows API. Hardware operations like Peek/Poke, Peek$/ Poke$, and Inp/Out. Direct bit operations like Shift and Rotate, as well as Bit Test, Set, Reset, and Toggle. Other BASIC firsts include Unions, and optional parameters in BASIC Subs and Functions.

"I have to admit I'm impressed! In about a day, I was able to port our Win16 Btrieve interface to PB/DLL, and it runs great!", reported Jim Barber, Product Analysis Editor of Btrieve Developer's Journal. "The code is much easier to read and maintain, and talk about quick compiles... One of my DSP DLL's is about 1K lines of VC++ source, and takes about 1.5 minutes to compile (in VC++). The PB/DLL equivelant takes less than 2 seconds."

The PB/DLL Compiler requires Windows 3.1, Windows 95, Windows NT or Win-OS/2 with at least four megabytes of RAM and a hard drive. PB/DLL has a retail price of $149

PB/DLL Features...

★ Native Machine Code
Compile Basic source code into true native machine code DLLs and EXEs for Windows. No run-time DLLs required.

★ Speed, Speed, Speed
PB/DLL was built with a single primary purpose. **Speed** and generates machine code optimized for 32-bit processors.

★ Security
Machine code DLLs and EXEs created by PB/DLL cannot be de-compiled back into Basic source code.

★ Compatibility
Move your Subs and Functions from Visual Basic, PowerBASIC for DOS or Microsoft Basic into true Windows DLLs and call them from VB 3.0, VB 4.0/16-bit, C/C++, Delphi, or any other Windows programming language. Even make calls to other DLLs and the Windows API.

★ Code & Data Pointers
Add speed and flexibility with pointers to both Code and Data. Access the target as any type of data. Use code pointers to create callbacks.

★ Huge Arrays
Say goodbye to 640k DOS limitations and hello to 16 megabytes of useable memory for arrays. Take advantage of built-in support for Sorting, Element Delete, Insert and Scan, and Redim Preserve.

★ 18 Data Types
Dynamic, ASCIIZ (null-terminated) and Fixed-length strings. User-Defined Types and Unions. Integers, Longs, Quad (64-bit) integers, Bytes, Words, and DWords. Single-precision, Double-precision, Extended-precision (80-bit) floats. Currency with two levels of precision.

★ Built-in Assembler
Hand optimize the most critical sections of your code for absolute performance. Just mix your BASIC and assembler code together PowerBASIC

Advertiser no longer in business