April

Fourth

2002

From: Bob Zale, President

PowerBASIC, Inc.

PowerBASIC Gazette #28

==========

Subject: An Introduction to the Windows API

==========

Today, I'm pleased we can offer you a very informative article from one of our best-known engineers, Lance Edmonds. He's helped many of of us over and over, both directly and via the PowerBASIC BBS. The article is primarily directed as an introductory text, but the most experienced of us can use a refresher, too! {smile} As always, we hope you find this Gazette to be interesting reading...

An introduction to the Windows API - By Lance C. Edmonds

==========

This article is intended as an introduction to calling Windows API functions, primarily for programmers contemplating their first venture into the world of Windows programming.

Before we begin to discuss API programming concepts, we'll start off with a few words of advice...

First, understanding how to call API functions is an essential

prerequisite to writing professional Windows GUI applications. DOS programmers who have already made the transition across to Windows programming will often mention that their first steps into Windows GUI programming were the most daunting, but then it got a lot easier once some of the key concepts were understood.

In addition, a basic understanding of how to make Windows API calls can even benefit the programmers who rely heavily on visual design tools, simply because the extent and power of the Windows API can stretch the horizons of the every Windows programmer.

However, along with this additional programming power comes some added complexity too. As a result, giving up on the idea of migrating to Windows can seem like the easy way out. With this in mind, our final words of advice are simply "Take it step by step", and "Don't give up!" - you will get there if you persist - and you'll be glad you did!

After all, if the author of this article can make the transition from "die-hard" DOS programmer to a dedicated Windows programmer, then you can do it too!

What is an "API"?

The often heard term "API" is an acronym for the collective phrase "Application Programming Interface". Following on, the term "Windows API" is used to describe a collection of Functions and Subs that Microsoft Windows(tm) provides for Windows application programmers to exploit.

Simply put, the Windows API can be visualized as nothing more than a large set of programmer's libraries. In reality, Windows offers an almost unbelievable number of API Sub/Functions that can be

used to do everything from setting the clock, sending email, printing, playing WAV and MP3 files, to drawing graphical windows on the screen, etc.

One oft-overlooked advantage of learning Windows programming is that Windows applications can be launched from DOS applications (via SHELL) to perform tasks that are just not possible or practical in DOS. For example, a Windows application that sends an email message or performs a file transfer over the Internet can be launched directly from a DOS application without the need to install a DOS TCP stack and write mountains of code. Learning Windows programming does not necessarily mean abandoning DOS programming - it can be a means of enhancing existing DOS apps significantly!

Subs, Functions and APIs

Calling an API function is absolutely no different than calling a PowerBASIC Sub or Function using regular BASIC code. For example, consider the PB function:

```
FUNCTION ABC(X AS STRING) AS LONG
  FUNCTION = VAL(x)
END FUNCTION
```

You would call this with normal PB code, something like this:

```
a$ = "12345"
result& = ABC(a$)
```

Simple, right? As far as the programmer is concerned, calling an API is no different! Lets discuss a little theory, then move along to some of the more interesting aspects.

Library files

>From a programming perspective, API Subs and Functions are no different than the regular Subs and Functions you have probably written in your own BASIC programs hundreds of time before. The key difference is that API Subs and Functions are not actually built into your program file (EXE file). Rather, Windows API Subs and Functions reside in a set of modules or library files with curious names like KERNEL32, USER32, etc.

To call an API, a program simply calls the required Sub/Function that is stored in a library file.

To make it even easier, Windows transparently handles much of the library file management for us!

So how does PowerBASIC know which library files to use?

When PowerBASIC compiles programs that make API calls, PowerBASIC automatically builds an "import table" directly into the compiled program file. The import table contains the names of the API functions that the program will use, along with the name of the library file that contains the API. Simply put, whenever a program is launched, Windows examines the program's import table and uses that information so it can load all of the required libraries files into memory, right along with the program itself. Windows then "links" the library files and the program code together, before allowing the program to finally start running.

This process of loading libraries with applications is known as "load-time" or "dynamic" linking. The library files that contain

the API Subs/Functions are referred to as "Dynamic Link Libraries",
or DLLs for short.

Ok, so what about calling some API functions?

To make use of an API function, all you have to do is pass the
correct parameters, and retrieve the return value correctly. For
example, the following piece of BASIC code defines a variable, and
makes an API call to retrieve the number of milliseconds (1/1000th
seconds) since the last reboot occurred:

```
DIM MillisecondsSinceLastBoot AS DWORD
MillisecondsSinceLastBoot = GetTickCount()
```

But how does PowerBASIC know that the GetTickCount() function is
an API call, and not another variable or Function in the BASIC
code, or is the name of a function that is simply missing from
the program code?

The answer is simple: In order to call an API Sub/Function, you
must first provide a declaration (DECLARE statement) to describe
the target API Sub/Function for PowerBASIC. The DECLARE acts as
a template so that PowerBASIC can validate the return data type
and API parameter types, etc. Further, the DECLARE statement
indentifies the name of the library file, so PowerBASIC can build
the import table correctly.

Do I write them myself?

You can, but a HUGE number of the most common Windows API functions
have already been translated to PowerBASIC for you, ready for use.
These declarations, along with various equates and UDT definitions,

can be found in the Win32API.INC file located in your \PB\WINAPI folder.

Therefore, a Windows program simply needs to #INCLUDE the file WIN32API.INC in order to make good use of the multitude of API's declared in WIN32API.INC. For example, the following is a fully functional Windows application that uses an API Function to ask Windows to provide the folder/path name of the "root" Windows directory:

```
#COMPILE EXE
#INCLUDE "WIN32API.INC"
FUNCTION PBMAIN
  LOCAL szPath AS ASCIIZ * %MAX_PATH
  LOCAL Result AS DWORD
  Result  = GetWindowsDirectory(szPath, SIZEOF(szPath))
  IF ISTRUE Result THEN
     MSGBOX szPath
  ELSE
     MSGBOX "The API function returned an 'error' result"
  END IF
END FUNCTION
```

The central part of this code is the call to the API Function GetWindowsDirectory(). How did I know what that specific API function needs for its parameters, and how did I know what the return value signifies?

The procedure to determine this is relatively straightforward, yet can be a source of confusion at first. This is a good point in time to refresh your memory about it.

At this point, do you remember the advice we offered at the start of the aticle? You do? Great! Let's gets back to it!

First, I looked up the API function name in the WIN32.HLP file, which shows us that the API function in the above code (GetWindowsDirectory) is defined by Microsoft as follows, using C code syntax:

```
UINT GetWindowsDirectory(
    LPTSTR lpBuffer,      // address of buffer for directory
    UINT uSize            // size of directory buffer
    );
```

Since this is a fairly common API function, we can find a PowerBASIC DECLARE statement for this API in the WIN32API.INC folder, as follows:

```
DECLARE FUNCTION GetWindowsDirectory LIB "KERNEL32.DLL" _
    ALIAS "GetWindowsDirectoryA" (lpBuffer AS ASCIIZ, _
    BYVAL nSize AS DWORD) AS DWORD
```

Starting at the left of the C code definition, the UINT keyword tells us that the function returns a UINT data type. In C, a UINT is an unsigned 32-bit integer, which corresponds to a DWORD in PB. Therefore, we can readily tell that the GetWindowsDirectory() API will return a DWORD value to the calling code.

The 1st API parameter, an LPTSTR data type, tells us the parameter is expected to be "long pointer" to a zero-terminated string buffer -- which is directly equivalent to a pointer to an ASCIIZ string buffer in PowerBASIC. Now, passing a pointer to an ASCIIZ string is achieved by a BYREF ASCIIZ string pass... therefore the 1st API parameter is satisfied simply by passing a BYREF ASCIIZ string to it.

The 2nd parameter, a UINT data type, is a DWORD parameter. WIN32.HLP says that this parameter is used to inform the API function how many

bytes are available in the ASCIIZ string buffer. This parameter is not passed as a pointer (or it would show the parameter as LPUINT or UINT*) -- therefore we should pass the actual value directly (ie, this is a BYVAL parameter).

The additional LIB and ALIAS clauses in the PowerBASIC DECLARE statement simply tell the compiler which DLL file the API function is located within, along with the API's "internal function name". For the most part, these LIB and ALIAS clauses can be largely ignored since most every API functions you will want to use while learning how to write Windows applications will already have a suitable DECLARE statement (located in WIN32API.INC or one of the various .INC files included with PowerBASIC).

Finally, the return value of the GetWindowsDirectory() API function is shown in WIN32.HLP to return zero if an error occurs, so we can use that value to determine the success/failure of the API call.

As you may be starting to visualize, calling API functions means the simple act of calling a Sub or Function, like you do in normal BASIC programming. The hardest part is learning "when" to call API functions and "what" parameters to pass. To this end, the WIN32.HLP help file provides a wealth of useful background info on API functions, their parameter lists, and what they do.

Where else can I find API information

One useful reference is the "Comparative Data Types" table in the PBDLL.HLP and PBCC.HLP files. This can be used to assist with syntax and parameter translations of the information in the WIN32.HLP file.

PowerBASIC, Inc. does its best to provide a comprehensive set of

API Sub/Function, equate, and structure declarations, via the Win32API.INC and related .INC files. These files are updated and expanded on a regular basis, and the most recent version can always be downloaded from the PowerBASIC Web Site, which is located at http://www.powerbasic.com/files/pub/pbwin/win32api.zip

Since Windows is a product of Microsoft Corporation, Microsoft's own web site can be used to obtain the most up-to-date information on the Windows API. Microsoft makes the "Platform SDK" (PSDK) available for download (around 340Mb compressed!).

Microsoft also provides an on-line PSDK search facility, and can also provide a complete copy of the the PSDK on a CDROM for a very low cost. In either case, refer to http://www.msdn.microsoft.com for the latest dialog, search and purchase pricing information. Bear in mind that most API documentation is written for C programmers, since that is what Microsoft seems to predominantly use to create Windows applications. As I noted before, C code (and to some extent C++ code) that calls API functions uses a syntax that resembles BASIC code, so translation is often not too difficult.

Additionally, you can quickly get to grips with what parameters and data types to pass to an API Sub/Function by looking at the declaration for the API in the WIN32API.INC file.

You'll find that thousands of API functions have been discussed on the PowerBASIC BBS, so searching for the name of an API function will often yield good information and very often some example code.

Happy API'ing!

==========

Order online at https://www.powerbasic.com/shop/ or just send an
email with all pertinent information to sales@powerbasic.com
We'll take it from there!

Most PowerBASIC products (those without printed books) can now be
delivered by electronic mail. No wait for a package to arrive...
No high shipping costs... For just $6 per order, no matter how
many products, we'll deliver directly to your computer. If you're
outside the U.S., savings might be greater. You won't pay taxes or
duties to a freight company or postal service, because they aren't
involved in the delivery. Check your tax code to be sure, but some
countries charge no tax at all on transactions of this type. It
could just be your lucky day!

==========

Is your PowerBASIC Gazette Electronic Edition subscription coming to
you at home or work? If you don't want to miss a single issue, why
not subscribe from both email addresses?

Send your subscription request to email@powerbasic.com and please
include your name and all email addresses you'd like to add as well
as your Zip or Postal Code.

If you know someone else who would enjoy this newsletter please
forward a copy to them so they can subscribe.

==========

==========

PowerBASIC Gazette - Electronic Edition

Volume 1 - Issue 28

PowerBASIC, Inc. (888) 659-8000 Sales
2061 Englewood Road (941) 473-7300 Voice
Englewood, FL 34223 (941) 681-3100 Fax

Visit us on the World Wide Web at www.powerbasic.com
Email Sales: sales@powerbasic.com

This newsletter is best viewed with a fixed-width font.

==========