The Amazing VFP2C32 Library

Eric Selje
Salty Dog Solutions, LLC
2505 Commonwealth Ave.
Madison, WI 53711
Voice: 608/213-9567
Email: Eric@SaltyDogLLC.com
WebSite/Blog: SaltyDogLLC.com
LinkedIn/Facebook/Twitter:EricSelje

Imagine having the entire Windows API at your disposal, allowing you to do things with your Visual FoxPro application you didn't think were possible. You can, with the VFP2C32 library written by Christian Ehlscheid . It slices (your development time)! It dices (your effort)! It even pureés (I'm not sure what this means)! Watch as it makes using the Windows API as easy as calling native functions.

In this whitepaper you will learn:

- What is the Windows API?
- How does it compare to .NET?
- How can I figure out what's in the Windows API?
- How do I manually call methods in the Windows API?
- What other tools are available to make Windows API calls?
- What does VFP2C32.fll bring to the table?
- Samples of existing projects that use VFP2C32, including projects on VFPX

What is the Windows API?

The Windows Application Programming Interface (API) enables applications to exploit the power of Windows. Using this API, you can develop applications that run successfully on all versions of Windows while taking advantage of the features and capabilities unique to each version. (Note that this was formerly called the Win32 API, and was introduced with Windows95 and NT. The name Windows API more accurately reflects its roots in 16-bit Windows and its support on 64-bit Windows.)[i]

Windows makes this API available to all developers for free and with no licensing requirements like some APIs. This means that you, as a FoxPro developer, can access the same internals as a developer that a C, C++, or even a .Net developer can. The trick is knowing how to do this, and that's what this whitepaper will discuss.

What is this "API" thing? *By Joel Sposky (Joel on Software)*

If you're writing a program, say, a word processor, and you want to display a menu, or write a file, you have to ask the operating system to do it for you, using a very specific set of function calls which are different on every operating system. These function calls are called the API: it's the interface that an operating system, like Windows, provides to application developers, like the programmers building word processors and spreadsheets and whatnot. It's a set of thousands and thousands of detailed and fussy functions and subroutines that programmers can use, which cause the operating system to do interesting things like display a menu, read and write files, and more esoteric things like find out how to spell out a given date in Serbian, or extremely complex things like display a web page in a window. If your program uses the API calls for Windows, it's not going to work on Linux, which has different API calls. Sometimes they do approximately the same thing. That's one important reason Windows software doesn't run on Linux. If you wanted to get a Windows program to run under Linux, you'd have to reimplement the entire Windows API, which consists of thousands of complicated functions: this is almost as much work as implementing Windows itself, something which took Microsoft thousands of person-years. And if you make one tiny mistake or leave out one function that an application needs, that application will crash.

How does the Windows API differ from the .Net framework?

While ".Net" means a lot more than can be discussed in one whitepaper, the classes that make up the framework of ".Net" do perform a similar function as the Windows API. One advantage of these .Net framework classes is they are organized into heirarchical classes that make a lot more sense to object-oriented developers. For example, here's a comparison of the Windows API and .NET functions that compares strings[ii]:

Windows API Call	.NET method

`CompareString(__in  LCID Locale,`
`  __in  DWORD dwCmpFlags,`
`  __in  LPCTSTR lpString1,`
`  __in  int cchCount1,`
`  __in  LPCTSTR lpString2,`
`  __in  int cchCount2)`
	`System.String.Compare(string s1, string s2)`

Now at first you may think these are pretty similar, and they are, but there are also some subtle differences. The .NET method named Compare is in the "System.String" namespace, which means there may be another method called Compare in the "System.DateTime" namespace (and indeed there is). Also, because of its object-oriented nature, the .NET System.String.Compare() method can be overloaded to allow for all sorts of variations, while the Windows API call has to explicitly name each variation differently, eg.

lstrcmp() to compare strings with case-sensitivty, and

lstrcmpi() to compare strings without case-sensitivity. *(Both of these, incidentally, call the above CompareString method and sends in the current Locale ID).*

Coincidentally, not everything in the Windows API is available via the .NET framework, so .NET developers must do the same things we as FoxPro developers do and make explicit Interop calls into the Windows API.

Calling .NET methods from within Visual FoxPro is really a companion piece to this white paper, and serendipitously Rick Strahl is doing a session named "Extending Visual FoxPro by Calling .NET Components."

In addition to the Windows API and .NET, you have the ability to call methods and use APIs that are exposed anywhere on the Internet via web services. Check out my other session, "Extending Web Apps Using VFP" for more information about taking advantage of a web app's API.

What's in the Windows API?

Essentially, almost everything that Windows can do is encapsulated in one of the eight categories that the API encompasses. They are:

- Administration and Management (install, configure, and service applications or systems)
- Diagnostic Information (troubleshoot application or system problems and monitor performance)
- Graphics and Multimedia (incorporate formatted text, graphics, audio, and video)

- Networking (enable communication between applications on different computers on a network. You can also create and manage access to shared resources, such as directories and network printers, on computers in the network)
- Security (enable password authentication at logon, discretionary protection for all sharable system objects, privileged access control, rights management, and security auditing.)
- System Services (give applications access to the resources of the computer and the features of the underlying operating system, such as memory, file systems, devices, processes, and threads.)
- Windows User Interface (create and manage a user interface.)

Within each of these eight categories are multiple subcategories. All told, thousands of functions are available to you and it can be a little overwhelming to know everything that's in there.

How can we possibly know what's in the Windows API?

The Windows API is very well documented. The internet is rife with information about its contents, FAQs, tips and tricks (although some is outdated, misleading or just plain wrong). Here are a few great sites that will help you get a handle on what's available:

MSDN

The Microsoft Developers Networks (MSDN) is *the* authoritative resource in regards to the Windows API. Use this hyperlink, http://msdn.microsoft.com/en-us/library/aa383723%28VS.85%29.aspx, to jump past all the .NET hype and right to the Windows API section.

KiloFox

An excellent book, **1001 THINGS YOU WANTED TO KNOW ABOUT VISUAL FOXPRO** (Akins, Kramek, & Schummer, 2000, Hentzenwerke Publishing), affectionately referred to as "KiloFox," has a section in Chapter 13 called "Windows API Calls." It has many examples of calling Windows API functions within Visual FoxPro.

News2News.com

My favorite reference for the Windows API is News2News.com. This searchable site has hundreds if not thousands of examples. You do need to pay a subscription to get 100% access to everything on the site, but most of it's free to look at and should give you plenty of information.

The best part of News2News is that the examples are *FoxPro-centric*. That means you don't have to translate C# examples to FoxPro like most sites. Spend some time on this site and you'll be impressed by its depth.

Taking Advantage of the Windows API

Ok, now you know what the Windows API is and where to find the components that you can use. But how, exactly, can you call those functions from within Visual FoxPro? Essentially you can either write the calls manually or find a utility where someone has already done the gruntwork for you.

Manually Calling Windows API functions

Once you know what you want to call, making the actual call itself isn't all that difficult. It involves using the DECLARE (dll) command in Visual FoxPro (not to be confused with the DECLARE <array> command – oh how you FoxPro love to mess with us!).

Here's a most simple example, which has no parameters and returns the handle to the active Visual FoxPro window:

```
DECLARE INTEGER GetActiveWindow IN user32
? GetActiveWindow()
```

This should return the same thing as _VFP.Hwnd or ThisForm.Hwnd if a form is running, and it is likely that those properties were actually set by a call to the GetActiveWindow function.

Tip:

Here's a tip I remember Drew Speedie mentioning a few years back. Because API functions with the same name as FoxPro functions take precedence in the calling heirarchy, you can dynamically load API functions on-demand by wrapping them in a VFP function of the same name. The first time you call it, the VFP function will be called, the API function will load and get called, but on subsequent calls the API function will be called directly.

```
? getActiveWindow()

FUNCTION getActiveWindow
   DECLARE INTEGER GetActiveWindow IN user32
   RETURN GetActiveWindow()
```

Be careful though, because if you declare a DLL from the API that has the same name as a built-in VFP command, the VFP command still takes precedence. E.g.

```
DECLARE INTEGER RealDriveType IN shell32 AS DriveType;
    INTEGER iDrive,;
    INTEGER fOKToHitNet
? DriveType(3, 0)
&& This will fail because VFP's DriveType() takes precedence and expects a call
like DriveType("C:")
```

Here's a slightly more involved example, which returns a GUID.

```
* Create GUID
DECLARE INTEGER CoCreateGuid IN ole32 STRING @pguid
DECLARE INTEGER StringFromGUID2 IN ole32;
    STRING rguid, STRING @lpsz, INTEGER cchMax
LOCAL cGUID, cGUIDString
cGUID = REPLICATE(CHR(0), 16)  && 128 bits
cGUIDString = ""
IF CoCreateGuid(@cGUID) = 0
    LOCAL cBuffer, nBufsize
    nBufsize=128
    cBuffer = REPLICATE(CHR(0), nBufsize*2)
    StringFromGUID2(cGUID, @cBuffer, nBufsize)
    cBuffer = SUBSTR(cBuffer, 1, AT(CHR(0)+CHR(0), cBuffer))
    * Convert UNICODE to double-byte
   cGUIDString = STRCONV(cBuffer, 6)
ENDIF
RETURN cGUIDString
```

You can see how even this fairly simple example still gets a bit complicated, with a call to two separate API functions, character-type conversion, creating buffers, and throwing pointers around. You'll see how much easier VFP2C32 makes this call in a little bit. And, of course, the methods get *way* more complicated than this, very quickly.

For example, suppose you wanted to set the create date/time of a file with the current time. Thankfully if you go to News2News.com and type in "set file time" you immediately get a nice example[iii] of how to do this:

```
*| PROCEDURE  WinSetFileTime
LPARAMETERS m.uFl, m.cTimeType, m.nYear, m.nMonth,;
    m.nDay, m.nHour, m.nMinute, m.nSec, m.nThou

#DEFINE OF_READWRITE     2

    LOCAL m.lpFileInformation, m.cS, m.nPar, m.fh,;
        m.lpFileInformation, m.lpSYSTIME, m.cCreation,;
        m.cLastAccess, m.cLastWrite, m.cBuffTime, m.cBuffTime1,;
        m.cTT,m.nYear1, m.nMonth1, m.nDay1, m.nHour1,;
        m.nMinute1, m.nSec1, m.nThou1

    m.nPar=PARAMETERS()
    IF m.nPar < 1
        RETURN .F.
    ENDIF

    DO decl        && declare external functions

    m.cTT=IIF(m.nPar>=2 AND TYPE("m.cTimeType")="C";
        AND !EMPTY(m.cTimeType),LOWER(SUBSTR(m.cTimeType,1,1)),"c")

    m.nYear1=IIF(m.nPar>=3 AND TYPE("m.nYear")$"FIN";
        AND m.nYear>=1800,ROUND(m.nYear,0),-1)
```

```
m.nMonth1=IIF(m.nPar>=4 AND TYPE("m.nMonth")$"FIN";
    AND BETWEEN(m.nMonth,1,12),ROUND(m.nMonth,0),-1)

m.nDay1=IIF(m.nPar>=5 AND TYPE("m.nDay")$"FIN";
    AND BETWEEN(m.nDay,1,31),ROUND(m.nDay,0),-1)

m.nHour1=IIF(m.nPar>=6 AND TYPE("m.nHour")$"FIN";
    AND BETWEEN(m.nHour,0,23),ROUND(m.nHour,0),-1)

m.nMinute1=IIF(m.nPar>=7 AND TYPE("m.nMinute")$"FIN";
    AND BETWEEN(m.nMinute,0,59),ROUND(m.nMinute,0),-1)

m.nSec1=IIF(m.nPar>=8 AND TYPE("m.nSec")$"FIN";
    AND BETWEEN(m.nSec,0,59),ROUND(m.nSec,0),-1)

m.nThou1=IIF(m.nPar>=9 AND TYPE("m.nThou")$"FIN";
    AND BETWEEN(m.nThou,0,999),ROUND(m.nThou,0),-1)

m.lpFileInformation = REPLI (CHR(0), 53)    && just a buffer
m.lpSYSTIME = REPLI (CHR(0), 16)            && just a buffer

IF GetFileAttributesEx (m.uFl, 0, @lpFileInformation) = 0
    RETURN .F.
ENDIF

m.cCreation   = SUBSTR(m.lpFileInformation,5,8)
m.cLastAccess = SUBSTR(m.lpFileInformation,13,8)
m.cLastWrite  = SUBSTR(m.lpFileInformation,21,8)
m.cBuffTime   = IIF(m.cTT="w",m.cLastWrite,;
    IIF(m.cTT="a",m.cLastAccess,m.cCreation))

FileTimeToSystemTime(m.cBuffTime, @lpSYSTIME)

m.lpSYSTIME=;
    IIF(m.nYear1>=0,Int2Word(m.nYear1),SUBSTR(m.lpSYSTIME,1,2))+;
    IIF(m.nMonth1>=0,Int2Word(m.nMonth1),SUBSTR(m.lpSYSTIME,3,2))+;
    SUBSTR(m.lpSYSTIME,5,2)+;
    IIF(m.nDay1>=0,Int2Word(m.nDay1),SUBSTR(m.lpSYSTIME,7,2))+;
    IIF(m.nHour1>=0,Int2Word(m.nHour1),SUBSTR(m.lpSYSTIME,9,2))+;
    IIF(m.nMinute1>=0,Int2Word(m.nMinute1),SUBSTR(m.lpSYSTIME,11,2))+;
    IIF(m.nSec1>=0,Int2Word(m.nSec1),SUBSTR(m.lpSYSTIME,13,2))+;
    IIF(m.nThou1>=0,Int2Word(m.nThou1),SUBSTR(m.lpSYSTIME,15,2))

SystemTimeToFileTime(m.lpSYSTIME,@cBuffTime)

m.cBuffTime1=m.cBuffTime
LocalFileTimeToFileTime(m.cBuffTime1,@cBuffTime)

DO CASE
CASE m.cTT = "w"
    m.cLastWrite=m.cBuffTime
CASE m.cTT = "a"
    m.cLastAccess=m.cBuffTime
OTHERWISE
    m.cCreation=m.cBuffTime
    ENDCASE

    m.fh = _lopen (m.uFl, OF_READWRITE)

    IF m.fh < 0
        RETURN .F.
    ENDIF

    SetFileTime (m.fh,m.cCreation,;
        m.cLastAccess, m.cLastWrite)

    _lclose(m.fh)
RETURN .T.

PROCEDURE Int2Word
LPARAMETERS m.nVal
RETURN Chr(MOD(m.nVal,256)) + CHR(INT(m.nVal/256))

PROCEDURE  decl
    DECLARE INTEGER SetFileTime IN kernel32;
        INTEGER hFile,;
        STRING  lpCreationTime,;
        STRING  lpLastAccessTime,;
        STRING  lpLastWriteTime

    DECLARE INTEGER GetFileAttributesEx IN kernel32;
        STRING  lpFileName,;
        INTEGER fInfoLevelId,;
        STRING  @ lpFileInformation

    DECLARE INTEGER LocalFileTimeToFileTime IN kernel32;
        STRING LOCALFILETIME,;
        STRING @ FILETIME

    DECLARE INTEGER FileTimeToSystemTime IN kernel32;
        STRING FILETIME,;
        STRING @ SYSTEMTIME

    DECLARE INTEGER SystemTimeToFileTime IN kernel32;
        STRING  lpSYSTEMTIME,;
        STRING  @ FILETIME

    DECLARE INTEGER _lopen IN kernel32;
        STRING  lpFileName, INTEGER iReadWrite

    DECLARE INTEGER _lclose IN kernel32 INTEGER hFile
```

You can see how something as simple as setting the file time requires 7 API functions to accomplish. In additon, if you need to call a method that requires a STRUCT parameter, as many do, things get ugly quickly. There must be easier ways, right?

Take advantage of FoxTools.fll

The original way FoxPro developers took advantage of a few of the functions in the Windows API was via FoxTools.fll. In fact the most used functions in FoxTools have migrated into the base FoxPro language, eliminating the need for most of us to even think about FoxTools anymore. We shouldn't forget that it's still there however, and a few of the functions still have some life left in them:

Function Name	Use	Comment
RegFn	`nFuncHandle = RegFn( cFuncName, cParamTypes, cRetType, cLocation )`	Access and call functions from any DLL
GetProStrg	`GetProStrg(cSection, cEntry, cValue)`	Could also use the functions in Registry.vcx
PutProStrg	`PutProStrg(cSection, cEntry, cValue)`	

ValidPath	`? ValidPath("C:\Program Files\Visual FoxPro 9\") && Returns .T.`
	
	`? ValidPath("C:\Program Files\Visual FoxPro 9\*") && Returns .F., because of the asterisk.`
		Determines whether a drive, path, file name and extension could be valid, following path naming restrictions. Does not test for the *existence* of the path or file.

CleanPath	`? cleanpath("C:\/=temp*")`
	
	`&& Returns C:\Temp`
		Makes a best guess at the correct path and file name, removing invalid characters.

< Clipboard Functions>	`CountClipF(), EnumClipFm(), GetClipDat(),GetClipFmt(),IsClipFmt(), SetClipDat()`	These functions give you insight into the info on your clipboard.

FoxPro Foundation Classes

This treasure trove of class libraries that come with Visual includes many that classes that are wrappers around Windows API functions. For example, the aforementioned Registry.vcx has classes to read/write registry and INI values. Here's how you can see what the font is for your Locals window in the debugger:

```
#DEFINE HKEY_CURRENT_USER -2147483647
LOCAL cValue
SET CLASSLIB TO HOME()+"ffc\registry.vcx"
oReg=CREATEOBJECT("Registry")
oReg.getregkey("LocalsFontName",@cValue,"Software\Microsoft\VisualFoxpro\9.0\Options"
,HKEY_CURRENT_USER)
? cValue
```

(Note: all of those #DEFINEs can be found in the accompanying .h files in the FFC folder)

From the best I can tell, these class libraries contain functions that utilize the Windows API:

FFC Class Library Name	General Functionality
_crypt.vcx	String encryption/decryption

_environ.vcx	Run external apps via ShellExecute()
_frxcursor	Gets "DeviceCaps", ie printer information
_gdiplus.vcx	Calls to the graphics system
_multimedia.vcx	Plays media (.avi and .mov files)
_system.vcx	Call "Common dialogs", get user information
Registry.vcx	Read/Write registry settings, INI file settings, enumerate ODBC functionality

You can see a lot more of these in action by running the Solution app:

```
DO HOME(2)+"Solution\Solution.app"
```

VFPX projects

There are some fantastic projects up on VFPX[iv] that also takes advantage of the Windows API. Here are some that I found; there may be even more than this, but I didn't inspect the source code of every project out there:

Project Name	Use
ParallelFox	Create parallel threads in VFP
SubFox	Integrate FoxPro projects w/ Subversion
GDIPlusX[v]	Even better than _GDIPlus.vcx
Ctl32_StatusBar	A great statusbar for VFP forms
Ctl32_Scontainer	Scrollable containers
Desktop Alerts	Outlook-style alerting system
Themed Controls[vi]	Make your controls obey the Windows theme
Windows 7 Taskbar Library[vii]	Use new Win7 U/I features in VFP

The Amazing VFP2C32 Library

Up to now, we've seen what the Windows API is, how to call it manually from VFP, and a variety of tools that wrap the API's functionality into class libraries or take advantage of it. Wouldn't it be great if there was "one library to rule them all?" It would, but alas there is not. But VFP2C32 comes close!

VFP2C32 is an FLL that encapsulates Windows API functionality into functions that are easily called from VFP. The advantages to this FLL are

- You don't have to explicitly DECLARE each DLL you want to use
- The FLL takes care of converting the parameters into a STRUCT that the API understands, which relieves you of a huge burden
- It gracefully handles errors that the API returns
- All C++ source code is included

VFP2C32 was written by Christian Ehlscheid and placed in the public domain in 2005. Christian originally wrote it to provide better interop between C API development and VFP, since he was frustrated with the difficulty of using the only statement VFP had, the DECLARE statement.

Downloading and Installing VFP2C32

The official site for getting VFP2C32 is http://vfp2c.dyndns.org/ . You can tell by the DynDns domain that the site is on a home server using Dynamic DNS (a service that updates the world's DNS servers with your new IP address if it changes, which can happen with non-business Internet connections). I've had occassional difficulty connecting to it, so if you get on it, grab the files while you can! They should also be included in the session downloads for this conference.

The file itself comes as a .zip file (http://vfp2c.dyndns.org/uploads/files/vfp2c.zip). Once you've got it you simply unzip it to the folder of your choice. It creates a directory structure like this:

\VFP2C32   (The library itself and supporting programs)
\VFP2C32Source (The Visual Studio solution file and source code)
\VFP2Cexamples (Examples of usage)

The only files required for you to use VFP2C32 are VFP2C32.fll and MSVCR71.dll if you don't already have that on your system (you probably do).

Installing the IntelliSense

One nice feature of VFP2C32 is that it can install IntelliSense for itself. This gives you a big headstart in navigating all of the API functions and figuring out what parameters are required.

To install VFP2C32's Intellisense, just type `DO vfp2cintelli`

Instantiating VFP2C32

From that same VFPC32 subfolder, type

```
SET LIBRARY TO vfp2c32.fll ADDITIVE
```

And presto, much of the functionality of the Windows API is now available at your fingertips!

The first method you want to call is InitVFP2C32(features)

Eg. `InitVFP2C32(0Xffffffff)`

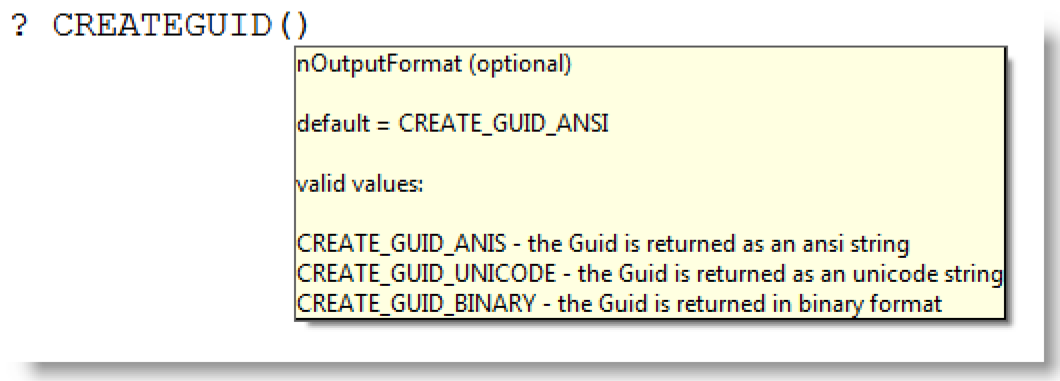
You may want to check the return value for that to make sure it started correctly (see the examples for a nice "Init" routine. You can selectively choose which categories of features to initialize in VFP2C32 if you know you just need certain features by sending the correct flag, and these work in an additive fashion, so you can mix and match.

VFP2C_INIT_MARSHAL	0x00000001
VFP2C_INIT_ENUM	0x00000002
VFP2C_INIT_ASYNC	0x00000004
VFP2C_INIT_FILE	0x00000008
VFP2C_INIT_WINSOCK	0x00000010
VFP2C_INIT_ODBC	0x00000020
VFP2C_INIT_PRINT	0x00000040
VFP2C_INIT_NETAPI	0x00000080
VFP2C_INIT_CALLBACK	0x00000100
VFP2C_INIT_SERVICES	0x00000200
VFP2C_INIT_WINDOWS	0x00000400
VFP2C_INIT_RAS	0x00000800
VFP2C_INIT_IPHELPER	0x00001000
VFP2C_INIT_URLMON	0x00002000
VFP2C_INIT_WININET	0x00004000
VFP2C_INIT_ALL	0xFFFFFFFF

InitVFP2C32() issues some `DECLARE DLL` commands, so you'll want to be very careful not to inadvertently issue a plain `CLEAR DLLS` with no parameters because you'll lose those declarations and things will stop working until they're reinitialized.

The simplest examples, revisited

Remember earlier when we wanted to create a GUID? We had to dive into News2News.com in order to find the function we wanted and what all the parameters meant. Let's see how difficult it is now that VFP2C32 is installed. Type this:



```
? CREATEGUID()
```

nOutputFormat (optional)

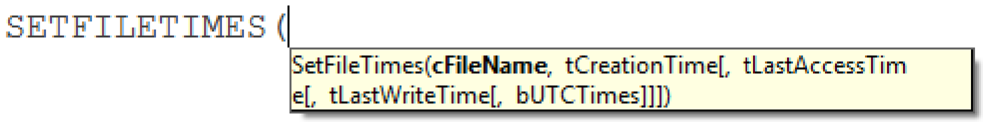
default = CREATE_GUID_ANSI

valid values:

CREATE_GUID_ANIS - the Guid is returned as an ansi string
CREATE_GUID_UNICODE - the Guid is returned as an unicode string
CREATE_GUID_BINARY - the Guid is returned in binary format

In one simple command, you know have what took two DECLARE DLLs and 15 more lines of code to accomplish before. AND you get the Intellisense, so you could discern that you had the option to get your value as ANSI, Unicode, or Binary GUID.

That one was probably too easy. What can we do that's more complicated? Let's revisit our more complicated example: touching the file creation time. Got VFP2C32 loaded? Then issue this command:



```
SETFILETIMES(
```

SetFileTimes(**cFileName**, tCreationTime[, tLastAccessTime[, tLastWriteTime[, bUTCTimes]]])

What took almost two pages of code (code we found on News2New2.com and didn't have to write ourselves, granted) now is accomplished in one command.

So, what all is in VFP2C32? Appendix A has the complete list, which I gleaned from parsing LIST STATUS . The website also enumerates the functions, but documentation beyond what's included in the IntelliSense doesn't exist for most of the functions in the library.

The best way to find out what's in there is to just empirically poke away at it. Here are some examples of the things that VFPC32.fll allows you to do (some of these are based on what's in the VPC2Examples subfolder):

Ex. 1: FileSystem
Copy MULTIPLE files selected in one dialog to a backup folder, with feedback

VFP's COPY command allows you to copy a table to another place, and its COPY FILE command allows you to copy other files and even include a wildcard. But you don't get any feedback while it's copying, and the overwriting dialog is feeble and only works if SET SAFETY is ON. Also if you have to copy multiple files that don't share a common extension you would have to call COPY FILE multiple times. With VFP2C32 loaded, you can now do it like this:

```
#include vfp2c.h
* Copy some files into a backup folder
* Default flags (1st parameter) are OFN_EXPLORER | OFN_NOCHANGEDIR |
OFN_NODEREFERENCELINKS | OFN_FILEMUSTEXIST | OFN_DONTADDTORECENT
* Get the filename(s) to backup
```

```
lnFiles = GETOPENFILENAME(0, "All Files" + CHR(0) +
"*.*","",FULLPATH(CURDIR()),"Example 1: Backup Multiple Files",0,"laFiles")
* Don't prompt if files exist, and make the backup folder if it's not there
lnFlags = FOF_NOCONFIRMATION+FOF_NOCONFIRMMKDIR
lcSource = ""
* 1st value of laFiles contains the path
* Reminaing values contain the filenames selected
IF lnFiles > 0
   IF lnFiles > 1
      FOR X=2 TO lnFiles
         lcSource = lcSource +  ADDBS(laFiles[1])+laFiles[X]+CHR(0)
      NEXT
      lcTarget = ADDBS(laFiles[1])+"backup"
   ELSE
      lcSource = laFiles[1]
      lcTarget = ADDBS(JUSTPATH(laFiles[1]))+"backup\"+JUSTFNAME(laFiles[1])
      lnFlags = lnFlags + FOF_MULTIDESTFILES
   ENDIF
   * Copy those files, and give user feedback while copying
   SHCOPYFILES(lcSource, lcTarget, lnFlags, "Copying selected files...")
ENDIF
```

One caveat: this doesn't copy tables that are opened exclusively, which VFP's plain old COPY can do (if you're the one that has the table open).

Tip:

One very nice side-effect of using these VFP2C32 functions rather than Visual Foxpro's built-in functions is that these functions always preserve case. If you've ever run into the problem where FoxPro decides to change the filename based on its whims, you'll know how frustrating this can be. One particular place where this is bad is in *nix-based source control systems, where file1.prg is different than file1.PRG which is different than FILE1.PRG.

Ex 2:ODBC
Get an aggregate result from an ODBC source without creating a cursor

How many times have you needed to get a SUM from a SQL Server database and said to yourself, "Why can't I just get the number; why do I need an entire cursor?" With VFP2C32, now you can, by sending that SQLEXECEX_DEST_VARIABLE flag on the end of a SQLEXECEX command (Note how Christian often adds an 'EX' to the FLL functions to avoid conflicts and indicate that these are 'extended').
This makes the functionality of SQL-PassThrough very similar to what we Visual FoxPro developers had with our non-SQL aggregate functions, like CALCULATE and SUM.

```
#INCLUDE vfp2c.h
&& store result into variables instead of a cursor (useful for aggregate functions)
LOCAL lnSum, lnCon, cConnString
```

```
cConnString="Driver={SQL Server Native Client 10.0};Server=<svr>; Database=<db>;
Uid=<hopefully not sa>;}"
lnCon = SQLSTRINGCONNECT(cConnString)
IF lnCon > 0
   lnRet = SQLEXECEX(lnCon,'SELECT SUM(numDues) FROM tblMemberDues', 'lnSum', '',
SQLEXECEX_DEST_VARIABLE)
   * The answer is now in lnSum, rather than a field in a cursor that I have to
maintain
   ? lnSum
ELSE
   ? "Invalid connection."
ENDIF
```

Ex 2a: Give feedback during SQLExec

It's not polite to have your application run off and do something without giving feedback as to what it's up to. Often after issuing a SQL Passthrough (SPT) statement with SQLEXEC, it just seems to hang there for a while. With SQLEXECEx, you can have your application give feedback while it's gathering your results:

```
#INCLUDE vfp2c.h
&& SQLExec with feedback.  This example just writes the progress to console, but you
could do something more interesting
LOCAL lnSum, cConnString
cConnString="Driver={SQL Server Native Client 10.0};Server=<svr>; Database=<db>;
Uid=<hopefully not sa>;}"
lnCon = SQLSTRINGCONNECT(cConnString)
IF lnCon > 0
   SQLEXECEX(lnCon,'SELECT * FROM largeTable', 'crsResults', '', 0, '', '',
'SQLCallBack', 100)
ELSE
   ? "Invalid connection."
ENDIF

FUNCTION SQLCallback(lnSet, lnRow, lnRowCount)
? "On set " + TRANSFORM(lnSet) + ", Row " + TRANSFORM(lnRow) + " out of " +
TRANSFORM(lnRowCount)
ENDFUNC
```

With the last two parameters of SqlExecEx, I specified a "callback" function to be invoked every 100 records (it also gets invoked at the beginning and end of the command). In this example it merely writes the progress out to the screen, but it'd be fairly easy to bind a progressbar or something even fancier to indicate your progress.

Ex 3: Services

Check to see if a service is running, and start it if it's not

This is a good one if you use SQL Server Express on the local machine for your datastore and you want to make sure it's fired up before your application starts requesting data from it:

```
* Ex 3: Start a service (default to SQLExpress)
#INCLUDE vfp2c.h
LPARAMETERS cServiceName
```

```
cServiceName = EVL(cServiceName, 'MSSQL$SQLEXPRESS')
CLEAR DLLS
SET LIBRARY TO vfp2c32.fll ADDITIVE
INITVFP2C32(VFP2C_INIT_SERVICES)
LOCAL lnServiceHandle
lnServiceHandle = OpenService(cServiceName)
IF AServiceStatus('laStatus',lnServiceHandle) = 1 AND laStatus[3] = 1
   ? cServiceName + " was not running, starting it..."
   StartService(lnServiceHandle)
   ? cServiceName + " is now running."
ELSE
   ? "Service was already going."
ENDIF
```

Note that Christian has written a nice Service class that encapsulates all of the service functions in VFP2C\VFP2cExamples\services.prg.

Ex 4: Printer

Get some serious printer information

Suppose you want your application to automatically send the results of your large print job to a printer that has a specific features (say, 11x17 stock) and has the shortest wait queue. Although VFP's APRINTERS() function enumerates what printers are available for you, it does not give you enough information to figure out which printer to use:

- Printer name.
- Name of the port to which the printer is connected.
- Name of the printer driver.
- Printer comment.
- Printer location.

With VFP2C32's APRINTERSEX(), you can get all that plus this information for each printer:

- Sharename
- Seperation File
- Printprocessor
- Datatype
- Parameters
- Attributes
- Priority
- DefaultPriority
- Starttime
- Untiltime
- Status
- Jobs
- AveragePPM
- DeviceNotSelectedTimeout
- TransmissionRetryTimeout

With APRINTJOBS(), you can query a print queue for which jobs are running.

With APRINTERFORMS(), you can see which forms a printer has available.
With APRINTERTRAYS(), you can see tray information for a printer.

Combining this functions, you can manage your print queues in order to maximize your efficiency.

Ex 4:Network
Grab a file from the Internet
This is great for many things, eg. "screen scraping," so you can get information from a web site that doesn't have a formal API. This method also has a Callback parameter, so you could show the progress of the download just like Example 6 showed the progress of a file copy. It also has an "asynchronous" parameter, so you can start the download going, and it can continue going about your code without waiting for it to finish. When it's done, your callback function will then do something with the file.

```
LPARAMETERS cName
URLDOWNLOADTOFILEEX("http://swfox.net/whoscoming.aspx","C:\TEMP\Test.html")
cWho = FILETOSTR("C:\TEMP\Test.html")
RETURN ATC(cName, cWho) > 0

? Ex4("Slay, Matt")  && Returns .f.
```

Alternatively, pull down a file from an FTP site:

```
URLDOWNLOADTOFILEEX("ftp://ftp.dina.kvl.dk/pub/general/cvejdina.doc")
```
[1]

Ex. 5: Interop/Conversion
This category of functions can often be used to interact with C functions, including FLLs and DLLs, and therefore are used internally quite a bit. However there are some interesting functions here that we may want to use in our applications:

What time is it, UTC?

```
? dt2utc(DATETIME())
```

This would be handy in situations where your server timestamps records in the local time, but you want to run reports on those records based on the UTC datetime.

Perform calculations on an array

No need to scan the array when you have VFP2C32:

```
DIMENSION aValues[5]
aValues[1]=RAND()
aValues[2]=RAND()
aValues[3]=RAND()
```

[1] Some random file I found on a public FTP site.

```
aValues[4]=RAND()
aValues[5]=RAND()
? aSum(@aValues)
? aMax(@aValues)
? aMin(@aValues)
? aAverage(@aValues)
```

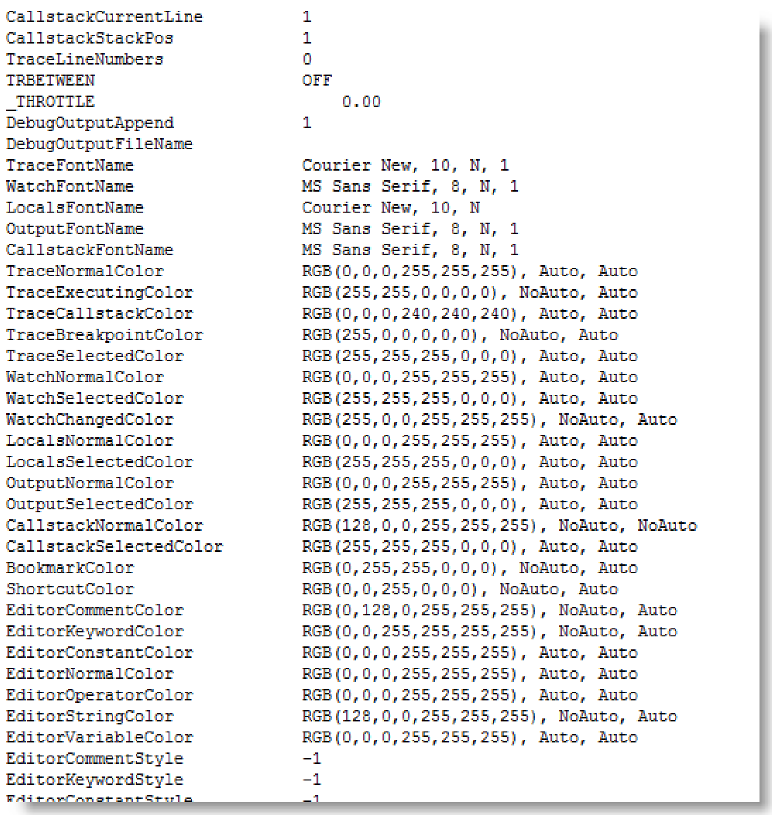
Ex. 6 Registry

Grab all the values from the FoxPro Options registry branch and put them into an array:

```
#INCLUDE vfp2c.h
SET LIBRARY TO vfp2c32.fll
INITVFP2C32(0Xffffff)
DIMENSION aRegValue[1,4]
iValues = aRegistryValues('aRegValues', HKEY_CURRENT_USER,
"SOFTWARE\Microsoft\VisualFoxPro\9.0\Options",3)

FOR x = 1 TO iValues
   ? PADR(aRegValues[X,1],30)+aRegValues[X,3]
NEXT
```

This will give show you results like this, and a lot of these options are not settable via any FoxPro command or UI element: you *have* to change the registry setting explicitly:



```
CallstackCurrentLine          1
CallstackStackPos             1
TraceLineNumbers              0
TRBETWEEN                     OFF
_THROTTLE                         0.00
DebugOutputAppend             1
DebugOutputFileName
TraceFontName                 Courier New, 10, N, 1
WatchFontName                 MS Sans Serif, 8, N, 1
LocalsFontName                Courier New, 10, N
OutputFontName                MS Sans Serif, 8, N, 1
CallstackFontName             MS Sans Serif, 8, N, 1
TraceNormalColor              RGB(0,0,0,255,255,255), Auto, Auto
TraceExecutingColor           RGB(255,255,0,0,0,0), NoAuto, Auto
TraceCallstackColor           RGB(0,0,0,240,240,240), Auto, Auto
TraceBreakpointColor          RGB(255,0,0,0,0,0), NoAuto, Auto
TraceSelectedColor            RGB(255,255,255,0,0,0), Auto, Auto
WatchNormalColor              RGB(0,0,0,255,255,255), Auto, Auto
WatchSelectedColor            RGB(255,255,255,0,0,0), Auto, Auto
WatchChangedColor             RGB(255,0,0,255,255,255), NoAuto, Auto
LocalsNormalColor             RGB(0,0,0,255,255,255), Auto, Auto
LocalsSelectedColor           RGB(255,255,255,0,0,0), Auto, Auto
OutputNormalColor             RGB(0,0,0,255,255,255), Auto, Auto
OutputSelectedColor           RGB(255,255,255,0,0,0), Auto, Auto
CallstackNormalColor          RGB(128,0,0,255,255,255), NoAuto, NoAuto
CallstackSelectedColor        RGB(255,255,255,0,0,0), Auto, Auto
BookmarkColor                 RGB(0,255,255,0,0,0), NoAuto, Auto
ShortcutColor                 RGB(0,0,255,0,0,0), NoAuto, Auto
EditorCommentColor            RGB(0,128,0,255,255,255), NoAuto, Auto
EditorKeywordColor            RGB(0,0,255,255,255,255), NoAuto, Auto
EditorConstantColor           RGB(0,0,0,255,255,255), Auto, Auto
EditorNormalColor             RGB(0,0,0,255,255,255), Auto, Auto
EditorOperatorColor           RGB(0,0,0,255,255,255), Auto, Auto
EditorStringColor             RGB(128,0,0,255,255,255), NoAuto, Auto
EditorVariableColor           RGB(0,0,0,255,255,255), Auto, Auto
EditorCommentStyle            -1
EditorKeywordStyle            -1
EditorConstantStyle           -1
```

Note: LibCurl, which I discuss in my other session, takes advantage of a lot of the networking and Interop/Conversion functions of VFP2C32, including creating Callback functions to call during the processes. If you're interested in doing that sort of thing, look at the source code for LibCurl.vcx.

Ex. 7: System Info
Find another application that's running and bring it to the foreground

I once wrote an application that involved the user retyping some information from another application. We realized we could save a lot of time if our application could bring the other application to the foreground if a user clicked on a button, in order to save the user time from having to Alt+Tab through all the open windows to find it themselves. I wish I would have had VFP2C32 back then, because it would have been as easy as this (in this example I'll have Visual FoxPro bring Word to the forefront):

```
#INCLUDE vfp2c.h
SET LIBRARY TO vfp2c32.fll
INITVFP2C32(0Xffffff)
LOCAL iApps, iHwnd
DIMENSION aApps[1,5]
iApps = aWindowsEx('aApps', 'TW', 1)
iHwnd = 0

FOR x = 1 TO iApps
   IF ATC("Microsoft Word", aApps[X,1]) > 0
      iHwnd = aApps[x,2]
   ENDIF
NEXT
IF iHwnd > 0
   * Switch to that app. Hey, I have to use a Windows API function that
   * VFP2C32 doesn't already have!
   DECLARE INTEGER SetForegroundWindow IN user32;
          INTEGER hWindow
   setForeGroundWindow(iHwnd)
ENDIF
```

These examples only scrape the surface of the power of VFP2C32. I encourage you to check out the vfp2c32examples folder in your VFP2C installation folder for more great examples. And within those examples are even pointers to give you even more examples!

Redistributing VFP2C32

You are free to redistribute VFP2C32 with your applications, royalty-free. You will want to include the VFP2Cdummy.prg, which will make your application compile without complaint by explicitly declaring the functions that VFP2C32 uses.

Carlos Alloatti has made extensive use of VFP2C2 in his Ctl32 Projects.[viii] Two of these classes are on VFPX as well, Ctl32_SContainer and Ctl32_StatusBar.

Mr. Alloatti has also wrapped up the functionality of CURL into a class library called libcurl.vcx. See my other session on "Extending Web Apps with Visual FoxPro" for more information about that particular library. It uses a lot of the WININET features in the Windows API.

Summary

This whitepaper introduced the Windows API and gave examples of how to use it either via FoxTools, Foundation Classes, or directly DECLARE functions in the API. It then dove deep into the Christian Ehlscheid's VFP2C32 project to show a new way Visual FoxPro developers can more easily take advantage of the power that the Windows API.

Christian has placed VFP2C32 in the public domain. I would like to take it one step further and put it up on VFPX.   It might be nice to wrap a class library around it that breaks it into its component parts, the way the GDIPlusX library encompasses all of the Graphics functionality, but we would need the community to get involved in that. It would also be great to see VFP2C32's functions documented in more detail.

Appendix A – All of the functions in VFP2C32.fll

COM

AsyncInvoke
CLSIDFromProgID
CLSIDFromString
CreateGuid
Invoke
IsEqualGuid
ProgIDFromCLSID
RegisterActiveObject
RegisterObjectAsFileMoniker
RevokeActiveObject
StringFromCLSID

FILESYSTEM

ADirEx
ADirectoryInfo
ADriveInfo
AFHandlesEx
AFileAttributes
AFileAttributesEx
CompareFileTimes
CopyFileEx
DeleteDirectory
DeleteFileEx
ExpandEnvironmentStr
FChSizeEx
FCloseEx
FCreateEx
FEoFEx
FFlushEx
FGetsEx
FHandleEx
FLockFile
FLockFileEx
FOpenEx
FPutsEx
FReadEx
FSeekEx
FUnlockFile
FUnlockFileEx
FWriteEx
GetFileAttributes

GetFileOwner
GetFileSize
GetFileTimes
GetLongPathName
GetOpenFileName
GetSaveFileName
GetShortPathName
GetSystemDirectory
GetWindowsDirectory
MoveFileEx
SHBrowseFolder
SHCopyFiles
SHDeleteFiles
SHMoveFiles
SHRenameFiles
SHSpecialFolder
SetFileAttributes
SetFileTimes

Interop / Conversion
AAverage
AMax
AMin
ASplitStr
ASum
BindEventsEx
Colors2RGB
CreateCallbackFunc
DT2FT
DT2ST
DT2Timet
DT2UTC
Decimals
DestroyCallbackFunc
Double2DT
Double2Str
FT2DT
Float2Str
Int64_Add
Int64_Div
Int64_Mod
Int64_Mul
Int64_Sub
Long2Str
Num2Binary
PG_ByteA2Str

PG_Str2ByteA
RGB2Colors
ST2DT
Short2Str
Str2Double
Str2Float
Str2Long
Str2Short
Str2ULong
Str2UShort
Timet2DT
ULong2Str
UShort2Str
UTC2DT
UnBindEventsEx
Value2Variant
Variant2Value

Library Info / Errorhandling
AErrorEx
FormatMessageEx
InitVFP2C32
VFP2CSys

Memory

AMemBlocks
AllocHGlobal
AllocMem
AllocMemTo
CompactMem
FreeHGlobal
FreeMem
FreePMem
FreeRefArray
LockHGlobal
ReAllocHGlobal
ReAllocMem
SizeOfMem
UnlockHGlobal
ValidateMem

Network
AIPAddresses
ANetFiles
ANetServers

AbortUrlDownloadToFileEx
GetServerTime
IcmpPing
Ip2MacAddress
ResolveHostToIp
SyncToSNTPServer
SyncToServerTime
UrlDownloadToFileEx

ODBC
ASQLDataSources
ASQLDrivers
ChangeSQLDataSource
CreateSQLDataSource
DeleteSQLDataSource
SQLExecEx
SQLGetPropEx
SQLSetPropEx

Printer
APrintJobs
APrinterForms
APrinterTrays
APrintersEx

Registry
ADialUpConnections
ARegistryKeys
ARegistryValues
CloseRegistryKey
CreateRegistryKey
DeleteRegistryKey
OpenRegistryKey
ReadRegistryKey
RegistryHiveToObject
RegistryValuesToObject
WriteRegistryKey

Services
AServiceStatus
AServices
CloseServiceHandle
ContinueService
ControlService
OpenService
PauseService

StartService
StopService

System Info

ADesktopArea
ADesktops
ADisplayDevices
AHeapBlocks
AProcessHeaps
AProcessModules
AProcessThreads
AProcesses
AResolutions
AResourceLanguages
AResourceNames
AResourceTypes
ATimeZones
AWindowProps
AWindowStations
AWindows
AWindowsEx
CancelFileChange
CancelRegistryChange
CenterWindowEx
FindFileChange
FindRegistryChange
GetCursorPosEx
GetSystemTime
GetWindowRectEx
GetWindowTextEx
ReadProcessMemoryEx
SetSystemTime
SetSystemTimeEx

DECLAREd API Functions used by VFP2C32

GetIUnknown
MarshalArrayCString
MarshalArrayCharArray
MarshalArrayDouble
MarshalArrayFloat
MarshalArrayInt
MarshalArrayLogical
MarshalArrayShort
MarshalArrayUInt
MarshalArrayUShort
MarshalArrayWCharArra

MarshalArrayWString
MarshalCursorCString
MarshalCursorCharArra
MarshalCursorDouble
MarshalCursorFloat
MarshalCursorInt
MarshalCursorLogical
MarshalCursorShort
MarshalCursorUInt
MarshalCursorUShort
MarshalCursorWCharAr
MarshalCursorWString
ReadBytes
ReadCString
ReadChar
ReadCharArray
ReadInt64AsDouble
ReadLogical
ReadPCString
ReadPChar
ReadPInt64AsDouble
ReadPLogical
ReadPPointer
ReadPUInt
ReadPUInt64AsDouble
ReadPWString
ReadPointer
ReadUInt
ReadUInt64AsDouble
ReadWCharArray
ReadWString
UnMarshalArrayCString
UnMarshalArrayCharArr
UnMarshalArrayDouble
UnMarshalArrayFloat
UnMarshalArrayInt
UnMarshalArrayLogical
UnMarshalArrayShort
UnMarshalArrayUInt
UnMarshalArrayUShort
UnMarshalArrayWCharA
UnMarshalArrayWString
UnMarshalCursorCStri
UnMarshalCursorCharA
UnMarshalCursorDoub
UnMarshalCursorFloat

UnMarshalCursorInt
UnMarshalCursorLogic
UnMarshalCursorShort
UnMarshalCursorUInt
UnMarshalCursorUShor
UnMarshalCursorWCh
UnMarshalCursorWStri
WriteBytes
WriteCString
WriteCharArray
WriteInt64
WriteLogical
WritePCString
WritePInt64
WritePLogical
WritePUInt64
WritePWChar
WritePWString
WriteUInt64
WriteWChar
WriteWCharArray
WriteWString

---

[i] http://msdn.microsoft.com/en-us/library/aa383723.aspx

[ii] I am aware that we don't need to call the Windows API or .Net Framework to compare strings in Visual FoxPro. I'm just looking for a good example.

[iii] http://news2news.com/vfp/index.php?example=128

[iv] www.codeplex.com/vfpx

[v] See Bo Durban's session, "Using Direct2D with Visual FoxPro"

[vi] See Doug Henning's session, "A Deep Dive into VFPX Themed Controls"

[vii] See Steve Ellenof's session, "Integrating Windows 7 taskbar functionality into VFP Applications"

[viii] http://www.ctl32.com.ar/default.asp