Autodesk User Group International
AUGI Training Program (ATP)

This course (and every course you are registered for) will only continue if you re-register for it after completing every segment. To do this (and you can re-register for all the courses you're taking at one time for each segment) use the ATP Registration webpage. It will be updated with each segment (A, B, C, D and E) and you will need to re-register using it to keep the courses you signed up for going this semester. No one is forcing you to do this, but if you don't, your courses may get cancelled. It's that simple.

If you have a question or comment about the content of any segment, we encourage you to post a message to your course's corresponding ATP Mail List. Teachers moderate their class mail list and may take up to 2 days to respond to questions you put to them. Proper use of any ATP Mail List is outlined on the ATP Mail List web page. If you have a question or comment about the ATP program in general, you can write to atpstaff@augi.com. Please do not write to atpstaff@augi.com with questions about course content unless there is a typo or a technical problem with a file.

You are welcome to publish the content of this or any ATP course segment for the benefit of your Local User Group or co-workers so long as AUGI and the course instructor receive full credit for generating it. The content of this course may not be sold in any form and is copyrighted. This lesson and other AUGI Training Program materials have been generated by the faculty and staff of Autodesk User Group International, a non-profit user-focused organization which depends on your participation.

-The ATP Staff

ATP611A – "Using the Windows API in VBA"
Faculty: Joe Sutphin

In this session I will outline the basics of what is needed for you to take advantage of the Windows API. While this is not a definitive guide to the Windows API, it will give enough information to get you started including some common examples such as the Open and SaveAs common dialogs.

Declares

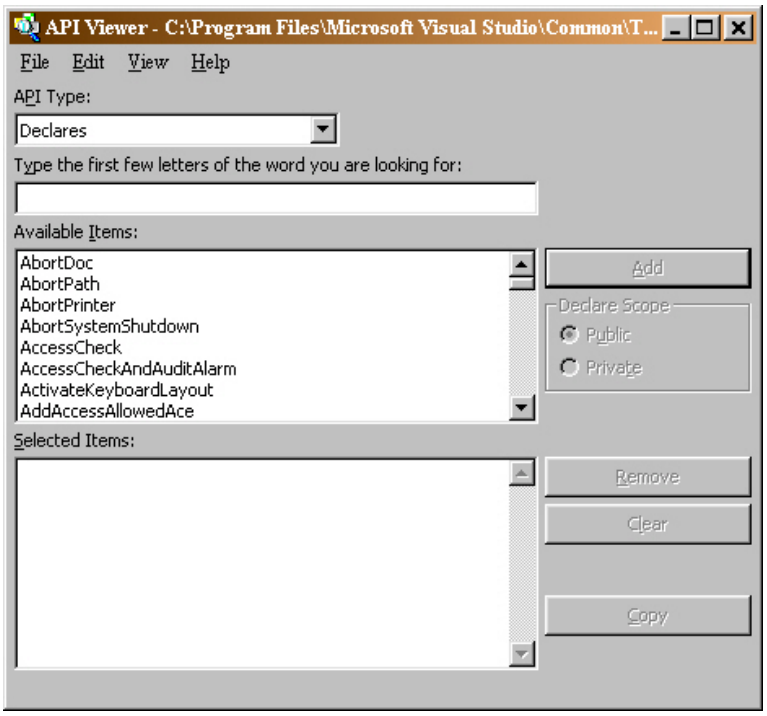
Every Windows API function that you use must be declared, period! Why? Because, there is no type library for your function calls to be resolved against like those used in VBA. So, in order for them to be resolved (and all function calls must be resolved) you have to declare each one before you use it as the following illustration demonstrates

```
Public Declare Function SetForegroundWindow Lib "user32.dll" (ByVal hwnd As Long) As Long
```

The breakdown of the syntax is as follows:

[Scope] Declare Function <FunctionName> Lib <DLL Filename String> (List of Parameters) As <DataType>

The easiest way to get the declare is to use the API Text Viewer as follows



Windows Data Structures

First of all, what is a data structure? Well, it is a collection of related information that is accessible through a single variable name. Data structures are widely used in the C (and some C++) language but for modern programming practices has been replaced by *classes*. The major difference being classes support the definition of functions that work on the data of the class thus allowing the programmer to hide variables and

functionality from the outside world. On the other hand, data structures are just that – data structures and nothing more. However, you're in luck because the Windows API has a ton of them such as the following

```
Public Type POINTS
  x  As Integer
  y  As Integer
End Type
```

DLL to Visual Basic Calling Conventions

To call DLL function procedures from Visual Basic you will need to convert the C language syntax used to document them into valid Declare statements that can be called from Visual Basic using the correct parameter data type declarations.

The C data types must be converted into Visual Basic data types. Also, you will need to specify whether the calling convention is ByVal [ByValue] or ByRef [ByReference]. The following table illustrates the conversions for 32-bit Windows C language data types to Visual Basic.

C language data type	Declare In Visual Basic as	Call with
ATOM	ByVal *variable* As Integer	An expression that evaluates to an Integer
BOOL	ByVal *variable* As Long	An expression that evaluates to a Long
BYTE	ByVal *variable* As Byte	An expression that evaluates to a Byte
CHAR	ByVal *variable* As Byte	An expression that evaluates to a Byte
COLORREF	ByVal *variable* As Long	An expression that evaluates to a Long
DWORD	ByVal *variable* As Long	An expression that evaluates to a Long
HWND, HDC, HMENU, etc. (Windows handles)	ByVal *variable* As Long	An expression that evaluates to a Long
INT, UINT	ByVal *variable* As Long	An expression that evaluates to a Long
LONG	ByVal *variable* As Long	An expression that evaluates to a Long
LPARAM	ByVal *variable* As Long	An expression that evaluates to a Long
LPDWORD	variable As Long	An expression that evaluates to a Long
LPINT, LPUINT	variable As Long	An expression that evaluates to a Long
LPRECT	variable As type	Any variable of that user-defined type

LPSTR, LPCSTR	ByVal *variable* As String	An expression that evaluates to a String
LPVOID	variable As Any	Any variable (use ByVal when passing a string)
LPWORD	*variable* As Integer	An expression that evaluates to an Integer
LRESULT	ByVal *variable* As Long	An expression that evaluates to a Long
NULL	As Any or ByVal *variable* As Long	ByVal Nothing or ByVal 0& or vbNullString
SHORT	ByVal *variable* As Integer	An expression that evaluates to an Integer
VOID	Sub procedure	Not applicable
WORD	ByVal *variable* As Integer	An expression that evaluates to an Integer
WPARAM	ByVal *variable* As Long	An expression that evaluates to a Long

Specifying the Library

Visual Basic knows where to find the .dll file that contains the procedures by using the *Lib* clause in the *Declare* statement. When you are declaring a function that uses one of the core Windows API libraries it is necessary to specify the filename extension .dll. However, for consistency I recommend you get in the habit as you will need to specify it for non-core API libraries that you use.

```
Public Declare Function SetForegroundWindow Lib "user32.dll" (ByVal hwnd As Long) As Long
```

For non-core API libraries you may specify a path in the *Lib* clause. If a path is not specified then Visual Basic will search for the file in the following order

- Directory containing the .exe file
- Current directory
- Windows system directory (usually C:\Windows\System)
- Windows directory (usually C:\Windows)
- Path environment variable

The Major Windows DLL's

The following is a table of the most commonly used libraries of Windows API functions.

Advapi32.dll	Advanced API services library supporting numerous APIs including many security and Registry calls
Comdlg32.dll	Common dialog API library
Gdi32.dll	Graphics Device Interface API library
Kernel32.dll	Core Windows 32-bit base API support

Lz32.dll	32-bit compression routines
Mpr.dll	Multiple Provider Router library
Netapi32.dll	32-bit Network API library
Shell32.dll	32-bit Shell API library
User32.dll	Library for user interface routines
Version.dll	Version library
Winmm.dll	Windows multimedia library
Winspool.drv	Print spooler interface that contains the print spooler API calls

Working with Windows API Procedures that Use Strings

The "*Alias*" clause in your Declare statements is required when calling Windows API procedures that use strings to specify the correct character set. There are actually two formats for procedures that contain strings: ANSI and Unicode.

For example, the SetWindowText function does not really exist but rather there are two separate functions that you use depending on whether your using *ANSI* or *Unicode*. The following illustrates the ANSI version

```
Private Declare Function SetWindowText Lib "user32" Alias "SetWindowTextA" (ByVal hwnd As Long, ByVal lpString As String) As Long
```

Note that the string that follows the Alias clause must be the true, case-sensitive name of the procedure.

You should specify the ANSI version of functions in Visual Basic because the Unicode versions are supported in Windows NT only. Use Unicode for those applications that you are certain will be running on Windows NT.

Passing Arguments by Value or by Reference

Visual Basic passes arguments *by reference* by default. Instead of passing the actual value of the argument a 32-bit address specifying the location of the value is passed. The ByRef keyword is not required however to make your code more readable it would be prudent to specify the exact method of passing the argument.

Many DLL procedures expect an argument to be passed *by value*. The function is expecting to receive the actual value instead of its memory location. If you pass the argument to the function using ByRef the function will be receiving information that it has no idea had handle.

To pass an argument by value, place the ByVal keyword in front of the argument declaration in the Declare statement. The InvertRect procedure accepts its first argument by value and its second by reference as in the following example

```
Declare Function InvertRect Lib "user32" Alias "InvertRectA" (ByVal hdc As Long, lpRect As RECT) As Long
```

Note - When you're looking at DLL procedure documentation that uses C language syntax, remember that C passes all arguments except arrays by value.

Learning By Example

This section will give you explicit examples of using the Windows API. The best way to learn how to use the Windows API is to follow the examples of others and try different situations on your own. These examples are some of the most commonly Windows API functions for AutoCAD developers and should provide you with enough information to pursue using the Windows API functions in your own application development.

OpenFile Common Control Dialog

Using the OpenFile common control dialog will add a look of consistency to your application design. The OpenFile dialog is part of the comdlg32.dll library of Windows API routines and is easily accessed. The following example illustrates using these routines to request a drawing file to open.

```
Private Declare Function GetOpenFileName Lib "comdlg32.dll" Alias "GetOpenFileNameA" (pOpenfilename As
  OPENFILENAME) As Long

Private Type OPENFILENAME
  lStructSize As Long
  hwndOwner As Long
  hInstance As Long
  lpstrFilter As String
  lpstrCustomFilter As String
  nMaxCustFilter As Long
  nFilterIndex As Long
  lpstrFile As String
  nMaxFile As Long
  lpstrFileTitle As String
  nMaxFileTitle As Long
  lpstrInitialDir As String
  lpstrTitle As String
  flags As Long
  nFileOffset As Integer
  nFileExtension As Integer
  lpstrDefExt As String
  lCustData As Long
  lpfnHook As Long
  lpTemplateName As String
End Type

Public Function ShowOpen(Filter As String, _
                         InitialDir As String, _
                         DialogTitle As String) As String

Dim OFName As OPENFILENAME

  'Set the structure size
  OFName.lStructSize = Len(OFName)
  'Set the owner window
  OFName.hwndOwner = 0
  'Set the filter
  OFName.lpstrFilter = Filter
  'Set the maximum number of chars
  OFName.nMaxFile = 255
  'Create a buffer
  OFName.lpstrFile = Space(254)
  'Create a buffer
  OFName.lpstrFileTitle = Space$(254)
  'Set the maximum number of chars
  OFName.nMaxFileTitle = 255
  'Set the initial directory
  OFName.lpstrInitialDir = InitialDir
  'Set the dialog title
```

```
  OFName.lpstrTitle = DialogTitle
  'no extra flags
  OFName.flags = 0
  'Show the 'Open File' dialog
  If GetOpenFileName(OFName) Then
    ShowOpen = Trim(OFName.lpstrFile)

    Else
      ShowOpen = ""
  End If
End Function
```

The following sample code illustrates using the ShowOpen routine that returns the filename selected as a string.

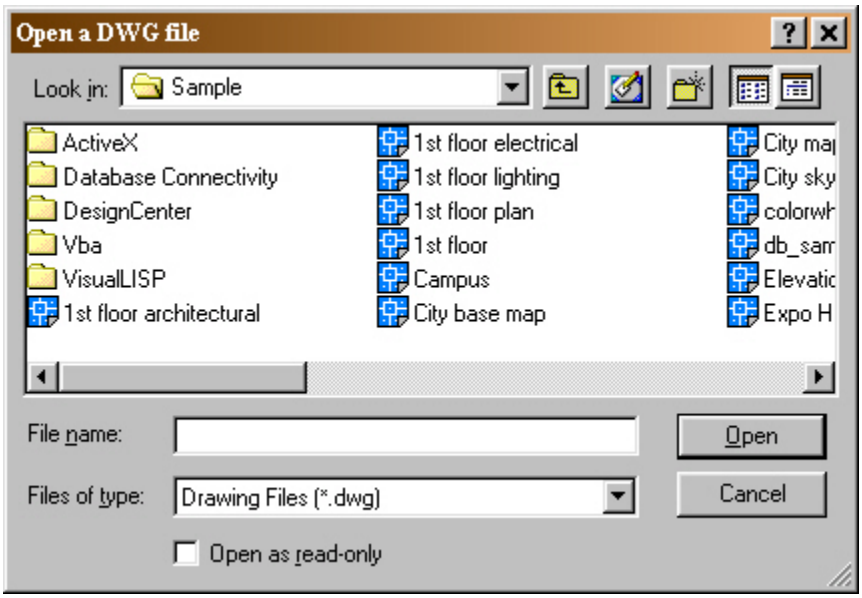
```
Dim OFName As New CommonFileDialog
Dim Filter As String
Dim InitialDir As String
Dim DialogTitle As String
Dim ReturnFile As String

  Filter = "Drawing Files (*.dwg)" + Chr$(0) + "*.dwg" + Chr$(0) + "All Files (*.*)" + Chr$(0) + "*.*" + Chr$(0)
  InitialDir = "C:\Program Files\AutoCAD 2002\Sample"
  DialogTitle = "Open a DWG file"

  ReturnFile = OFName.ShowOpen(Me, Filter, InitialDir, DialogTitle)
```

The Filter parameter is a string that details what filetypes by extension you want to display when the OpenFile dialog box is displayed. The InitialDir parameter specifies which directory will be displayed by default. You may choose to give a name to your OpenFile dialog box by using the DialogTitle parameter.

With each of these parameters defined, executing this code will result in the following OpenFile dialog box being displayed



SaveAs File Dialog

Using the SaveAsFile common control dialog will add a look of consistency to your application design. The SaveAsFile dialog is part of the comdlg32.dll library of Windows API routines and is easily accessed. The following example illustrates using these routines to save a drawing file.

```
Private Declare Function GetSaveFileName Lib "comdlg32.dll" Alias "GetSaveFileNameA" (pOpenfilename As
  OPENFILENAME) As Long

Public Function ShowSave(FormName As Form, _
                         Filter As String, _
                         InitialDir As String, _
                         DialogTitle As String) As String

Dim OFName As OPENFILENAME

  'Set the structure size
  OFName.lStructSize = Len(OFName)
  'Set the owner window
  OFName.hwndOwner = 0
  'Set the filter
  OFName.lpstrFilter = Filter
  'Set the maximum number of chars
  OFName.nMaxFile = 255
  'Create a buffer
  OFName.lpstrFile = Space(254)
  'Create a buffer
  OFName.lpstrFileTitle = Space$(254)
  'Set the maximum number of chars
  OFName.nMaxFileTitle = 255
  'Set the initial directory
  OFName.lpstrInitialDir = InitialDir
  'Set the dialog title
  OFName.lpstrTitle = DialogTitle
  'no extra flags
  OFName.flags = 0
  'Show the 'SaveAs File' dialog
  If GetSaveFileName(OFName) Then
    ShowSave = Trim(OFName.lpstrFile)

    Else
      ShowSave = ""
  End If
End Function
```

The following sample code illustrates using the ShowSave routine.

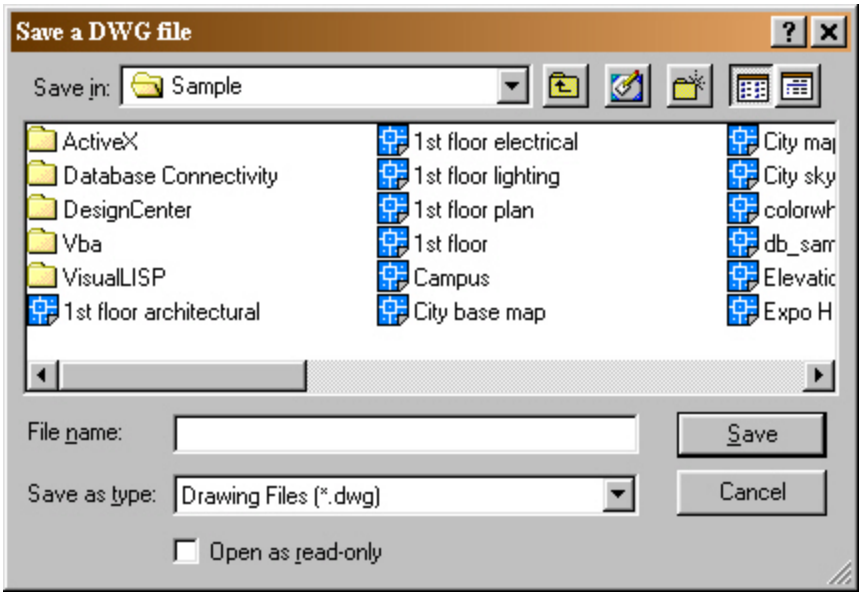
```
Dim OFName As New CommonFileDialog
Dim Filter As String
Dim InitialDir As String
Dim DialogTitle As String
Dim ReturnFile As String

  Filter = "Drawing Files (*.dwg)" + Chr$(0) + "*.dwg" + Chr$(0) + "All Files (*.*)" + Chr$(0) + "*.*" + Chr$(0)
  InitialDir = "C:\Program Files\AutoCAD 2002\Sample"
  DialogTitle = "Save DWG as file"

  ReturnFile = OFName.ShowSave(Me, Filter, InitialDir, DialogTitle)
```

The Filter parameter is a string that details what filetypes by extension you want to display when the SaveAsFile dialog box is displayed. The InitialDir parameter specifies which directory will be displayed by default. You may choose to give a name to your SaveAsFile dialog box by using the DialogTitle parameter. Also, an initial or default filename may be supplied using the InitialFile parameter.

With each of these parameters defined, executing this code will result in the following SaveAsFile dialog box being displayed.



OK, well that was a lot of material. Digest it, study it but most of all, try it! Next time we'll something equally useful.