Windows API **Sample Help**

Sample Description: Windows API

Points of Interest

- Understanding Dynamic Link Libraries
- Declaring a Windows API Function
- Adding a Windows API Function to your Application

Control

For Help on Help, Press F1

Windows API

The Microsoft Windows environment defines more than 500 separate functions that programs can use to interact directly with the operating system environment. These Windows procedures are called *Application Programming Interface* (API) functions. The API procedures may include both Sub and Function procedures. The API procedures are stored in files known as *Dynamic Link Libraries* (DLL). The most important DLL files supplied with Windows are USER32.DLL, KERNEL32.DLL and GDI32.DLL. These DLL's are found in your Windows/System32 directory. Third party vendors also produce DLL files, which contain customized Windows procedures.

Envelop provides built-in features that make it unnecessary to use the API functions in most applications. There may be times, however, when you want to accomplish something that is not directly provided by Envelop.

In order to make an API function available to an Envelop application, the function must be declared. The exact syntax of the code that declares an API function varies for each function. You should also realize that Envelop is a 32-bit application, therefore it is designed to interface to 32-bit Windows API, not the older 16-bit API's.

To run the sample:

- Click the **Get Info** button. The corresponding API's are called and the results are displayed in labels on the form.

Declaring a Windows API Function

Declare statements can appear as a method to an object. In many cases, new objects are created to contain the declare statements. This way, their functions can be referenced by any form or object.

Once a Declare instruction associates a DLL procedure with an application, it can be invoked continually in the program code. No further reference to the Dynamic Link Library is necessary. A Declare instruction can define a procedure as a Sub or Function. As usual, Sub procedures do not return values, whereas Function procedures return values of a specified data type.

In the sample's module, the following objects were created with the following API function declarations. You can view these objects in the Object Viewer under the "module" view for this sample.

Object: APISysDir

Declare Function GetSystemDirectoryA Lib "kernel32" (ByVal file As String, ByVal size As Long) As Long

Object: APIWinDir

Declare Function GetWindowsDirectoryA Lib "kernel32" (ByVal file As String, ByVal size As Long) As Long

The general format to the Declare statement is as follows:

Declare Function **name** Lib "**libname**" (**ByVal var1** As **type**, ByVal var2 As type....) As type

This	Specifies this
name	The name of the DLL procedure. For Function procedures, this name can optionally include a type-declaration suffix.
libname	The name of the DLL containing the declared procedure by using a string literal.
ByVal	That the argument is passed by value. ByVal clauses are optional.
var1, var2,...	Envelop's variable names used as arguments to the DLL procedure.
type	The data type of the argument variable. The final As type clause specifies the data type of the DLL function. For variables and function names without type-declaration suffixes, an As type clause specifies a data type. As type clauses are also optional.

Adding an Windows API Function to your Application

Adding a Windows API function to your application's program code involves making a call to the API Object's declared function. The sample gets the names of the Windows and Windows System directories as shown in the program code below:

```
Sub btnGetInfo_Click()
    Dim buffer As String
    Dim totalchars As Integer

    buffer = Space(255)
    totalchars = APIWinDir.GetWindowsDirectoryA(buffer, 255)
    If totalchars == 0 Then
        lblWinDir.Caption = "Unable to determine Windows directory."
    Else
        lblWinDir.Caption = UCase(Left(buffer, totalchars))
    End If

    buffer = Space(255)
    totalchars = APISysDir.GetSystemDirectoryA(buffer, 255)
    If totalchars == 0 Then
        lblWinSysDir.Caption = "Unable to determine System directory."
    Else
        lblWinSysDir.Caption = UCase(Left(buffer, totalchars))
    End If
End Sub
```

The GetWindowsDirectoryA function returns the path of the Windows directory. The Windows directory contains windows applications, initialization, and help files. The GetSystemDirectoryA function provides the path of the Windows System subdirectory. The System subdirectory contains the Windows libraries, drives, and screen font files. The "buffer" variable points to the buffer that will receive the returned pathname string, while 255 represents the maximum size of the buffer.

Understanding Dynamic Link Libraries

Explaining the terms in Dynamic Link Library might give you a better understanding how these libraries work:

Dynamic

The DLL (which is an executable program) is not loaded into Windows memory until at least one application needs a procedure contained in the library. Windows examines its references to DLL procedures to determine when a particular DLL is needed by an application: Because DLLs are not loaded until run time, they are referred to as "dynamic."

Link

Once Windows loads the application program and the DLL, a special process known as "linking" occurs. This process connects the application program to the DLL. As a result, the application runs as though the DLL procedure was actually part of it.

Library

Libraries are similar to code modules in that both entities can contain functions and Sub procedures. Library procedures declared in code modules are globally available.