Contents

Overview

The VBDLLCTL Custom Control

Properties

Events

Writing Library Programs

Calling Library Functions

VBFunctionCall Function

VBFunction Function

Passing Parameters

Tools And Samples

Library Manager

Client VB Project

Server VB Project

General Information

Introduction

The VB/DLL Library Builder for Visual Basic is a system that allows a Visual Basic program to function in the same manner as a Windows DLL A Visual Basic custom control is used to "export" functions from the library program which can be called by other executing Windows programs with a DLL function call. This lets developers write functions in Visual Basic which can be directly executed by other Windows applications. Callers can pass parameters to the library functions which can then modify their values if desired. Since the same code is used by multiple applications, it doesn't need to be copied from program to program to be re-used. This results in improved applications design which will save time for developers by simplifying software maintenance.

Uses for VB/DLL

Client/Server Programming - VB/DLL is ideal for communicating between client and server programs. With only a single control instance each, both programs can call each other as if they were the same program. This eliminates the need for complicated cross task programming. The next release of VB/DLL will support calls across TCP/IP networks. Library programs will then be able to be placed on network servers for use by multiple computers without any modification of existing library code.

Alternative to DDE or OLE Automation - VB/DLL is a great alternative to DDE and OLE Automation. It's much easier to use than either. It is better suited for cross-application programming than DDE and, unlike OLE, takes only about 40K of RAM as opposed to megabytes. VB won't even be able to create OLE Automation servers until version 4.

Greater Design Flexibility - If there's more than one .EXE in a project, chances are there will be common tasks both will have to perform. For example, a project could include a forms collection library program. This would allow other programs in the project to have menu options that display forms from the library. Another example is a database access module. All database access would occur in the library program. Until VB/DLL, the only way to re-use VB code was with DDE or subclassing. This type of solution has many pitfalls and usually requires a great deal of special knowledge about the Windows messaging system whereas VB/DLL programs completely shield the programmer from such nastiness. Also, everything needed for library programs to run on other computers can be redistributed without any royalty fees.

Building Library Products - VB/DLL can be used to build library products in VB that will be used by other programmers. Before VB/DLL, these types of products had to be distributed with source code. This put the developer in the unenviable position of having to debug their users' modified source code. With VB/DLL, all their source can be in .EXE format so it can't be viewed or modified. Or, "stub" functions can be written in VB to call the library functions that source code will be distributed for. This will insulate the user from having to make DLL calls.

Step-By-Step Implementation

Overview - This gives a step-by-step description of how to implement a CheckMail function in a mail server program. Client programs will call the function to check for new mail. The function will take the user name and password as parameters and return an error code.

Write the Function - Open the mail server project under VB. Code and test the CheckMail function. When finished, copy the VB function declaration code to the clipboard for later use.

Give CheckMail a Home - Decide which control instance will define the CheckMail function. If a new control is desired, it should be placed on a form that will always be loaded when the function is called. To guarantee availability, put the control on the first form that loads for the mail server program. Then, click the right mouse button on the control the function will be placed in. The Edit Declarations Dialog will display with the proper control as the current selection. Then click the Paste New button to add the declaration from the clipboard. The function name will be added to the list of functions displayed for the control.

Write the Event Code - Select the function in the Edit Declarations Dialog. Then click the Show Code button to display the sample event processing code and copy any desired code to the clipboard. Click the Goto Code button to display the CallProc event code and paste any desired code in the proper place. Then, copy the Type...End Type parameter structure from the Show Code window into one of the .BAS modules for the mail server project. If mail server has a .BAS file for VB/DLL exported functions, copy the Show Code window's Declare statement into it.

Write the Calling Code - If the caller is a VB program, open the project and paste in the sample calling code from the ShowCode window, modifying it if necessary. Non-VB callers can call library functions by using the VBFunctionCall Function that's exported from the VBDLLCTL.VBX file as a Windows DLL function. Refer to the documentation for the product being used about how to call DLLs.

Test Test Test - Nothing more need be said. See the Helpful Hints section for remedies to common ailments.

Properties

EditDeclarations Property

This property does not have a value. It's used to trigger the Library Manager to define the functions a control instance will export as well as generate and display sample code to call and implement particular library functions. To operate, select the ShowCode property and click the '...' box to the right of the property display. The Library Manager can also be triggered by clicking the right mouse button on any VBDLLCTL control. Since the Library Manager program runs as a separate application instead of a modal dialog, its windows don't need to be closed before other properties are selected or other projects are loaded.

Events

CallProc Event

This event fires when the VBFunctionCall Function or VBFunction Function is called for the control. The ParmPnt As Long parameter passed to this event is used by the CopyParmsToVB Function and the CopyParmsFromVB Function to access the parameters passed from the caller.

CopyParmsToVB Function

Overview

This function is intended for use only inside the CallProc Event. It uses the ParmPnt As Long variable passed to the event to copy the parameters passed to the library function into VB variables so they can be accessed by the library function. See CallProc Event Processing for examples. It returns 0 if successful.

Parameters

VB Buffer Pointer - This variable is usually a user defined type variable declared with the Dim statement. It can be a normal VB type variable (declared either with Dim or a type declaration character e.g. & for As Long) if only a single parameter is being passed and its data type is not As String.

VBX Access Pointer - This parameter must be the ParmPnt As Long value passed into the CallProc Event.

CopyParmsFromVB Function

Overview

This function is intended for use only inside the CallProc Event. It uses the ParmPnt As Long variable passed to the event to send values back to the caller from the library program's VB variables. See CallProc Event Processing for examples. It returns 0 if successful.

Parameters

VB Buffer Pointer - This variable is usually a user defined type variable declared with the Dim statement. It can be a normal VB type variable (declared either with Dim or a type declaration character e.g. & for As Long) if only a single parameter is being passed and its data type is not As String. This parameter can be 0 if there are no parameters or the parameters don't need to be passed back to the caller.

VBX Access Pointer - This parameter must be the ParmPnt As Long value passed into the CallProc Event.

Return Value Buffer - This parameter is a pointer to the buffer to be passed back to the caller as the function return value. If a return value is desired, this parameter should be the VB variable that's storing it. The variable should be the same data type as the return value defined in the function's declaration as set up in the Edit Declarations Dialog. This parameter can be 0 if there is no return value or the return value doesn't need to be passed back to the caller.

VCopyParmsToVB

This is an alternate way to call the CopyParmsToVB Function function. The difference is that the VCopyParmsToVB function can accept varying numbers of parameters like the VBFunction Function function. This eliminates the need to have a VB Type..End Type data structure for the VB/DLL library function in the library program. VB Declare statements are generated for individual VB/DLL library functions by the Show Code Window in the Library Manager program.

VCopyParmsFromVB

This is an alternate way to call the CopyParmsFromVB Function function. The difference is that the VCopyParmsFromVB function can accept varying numbers of parameters like the VBFunction Function function. This eliminates the need to have a VB Type..End Type data structure for the VB/DLL library function in the library program. VB Declare statements are generated for individual VB/DLL library functions by the Show Code Window in the Library Manager program.

CallProc Event Processing

Overview

The CallProc Event is triggered when the VBFunctionCall Function or the VBFunction Function is called identifying the proper library and function names. The ParmPnt As Long parameter it receives allows it to access the passed parameters as well as interact in various ways with the VBX. Here is a typical sequence:

If the function has parameters, dimension a variable to store passed parameters using the user-defined type in the Type statement generated by the Show Code Window. Then call the CopyParmsToVB Function to load the parameter structure.

Do the function processing. If a function already exists, this will probably be just one line to call the function. If the function doesn't already exist, one should be written to call although it can be implemented directly in the event processing.

If the function has a return value or parameters that need to be sent back to the caller, call the CopyParmsFromVB Function to send the parameters and return value back to the caller.

Sample Code

Sample code for function named TypeTest$ with a declaration property value of TypeTest$(fnd%, lng&,sng!, rl#, st$, fnd2 as integer, lng2 as long, sng2 as single, rl2 as double, st2 as string).

```
Declare Function CopyParmsToVB Lib "vbdllctl.vbx" (Dest As Any, ByVal pnt&) As Integer
Declare Function CopyParmsFromVB Lib "vbdllctl.vbx" (Dest As Any, ByVal pnt&, Retv As Any) As Integer
Type TypeTestType
    IntTemp As Integer
    LongTemp As Long
    sng As Single
    dbl As Double
    StrTemp As String
    IntTemp2 As Integer
    LongTemp2 As Long
    sng2 As Single
    dbl2 As Double
    StrTemp2 As String
End Type
Sub Function1_CallProc (ParmPnt As Long)
        Dim Parm As TypeTestType
        ErrCode% = CopyParmsToVB(Parm,   ParmPnt)
        retv$ = TypeTest$(Parm.fnd, Parm.lng, Parm.sng, Parm.rl, Parm.st, Parm.fnd2, Parm.lng2,
Parm.sng2, Parm.rl2, Parm.st2)
        ErrCode% = CopyParmsFromVB(Parm,   ParmPnt, retv$)
End Sub
```

Unloading Libraries

Overview

Libraries can be automatically shut down when all their caller programs terminate. This is done by exporting a VB/DLL function called "UnloadVbLib" which will be called when all the callers unload. It can also be called by user programs. The function must accept one integer parameter and cannot have a return value. It's the responsibility of the UnloadVbLib function to shutdown the library program. The VBX will not shut it down, just call the function. It's recommended that the library program not end during the function call but instead implement a delayed event to shut down using a timer control or some other available method. The parameter passed to this function is a flag indicating whether or not the library program was started by the VBX. If it's False, the library may have been started by interactive user action and possibly shouldn't be unloaded. If it's True (-1), then the library program was started by the VBX.

Sample Code

```
Sub Function1_CallProc (ParmPnt As Long, FunctionName As String)
If FunctionName = "UnloadVbLib" Then
    ErrCode% = CopyParmsToVB(AutoStarted%, ParmPnt)
    If AutoStarted% = True Then
        Unload Me
        End
    End If
    Exit Sub
End If
```

Debugging the Library Program

Since library programs can be executed in the VB development environment, it's to your benefit to debug library programs this way as much as possible. For this to work, a filename with a .MAK extension must be passed in the Library Name parameter of the VBFunctionCall Function. The Library Name parameter (without the drive, path and extension) must also match the library application's title as entered in the Make Exe dialog. This ensures that the library manager can locate a library that is loaded in the Visual Basic environment. Debugging can be simplified by bypassing the VBFunctionCall Function calling method and using the VB Call statement to directly execute the CallProc Event or actual library function (if one exists). This way, VB can enter break mode without the caller having to wait. Another technique is to use the VBFunctionCall Function in the library program to call its own functions. Remember to use Debug.Print statements in spots where entering break mode is not feasible.

VBFunctionCall Function

Overview

The VBFunctionCall function is located in the VBDLLCTL.VBX library in the Windows system directory. It's purpose is provide a standard DLL function interface to call the VB/DLL library functions. See the VBFunction Function for an alternate way to call VB/DLL library functions.

Parameters

The VBFunctionCall function accepts four parameters.

Library Name - A null-terminated string identifying the drive, path and filename of the library executable file. e.g. C:\LIB\LIBRARY.EXE. If a .MAK file extension is specified, and the proper library is currently running in the VB development environment, the library function will execute in the development environment, allowing the debugger to be used. Otherwise, the parameter is used as a command string to start the library program if it's not already started.

Function Name - A null-terminated string identifying the library function to execute. This name must match the name of a function defined in the Edit Declarations Dialog of a VBDLLCTL custom control residing on a loaded form.

Return Value - A pointer to storage that will store the library function return value. In Visual Basic, this will be a variable defined with the Dim or Global statement or with a type shorthand character (like $ for String). If a return value is not desired, 0 can be passed. Otherwise, this value should point to a valid memory block for the caller that is large enough to store the library function return value. Otherwise, a General Protection Fault may occur.

Parameters - A pointer to a data structure containing parameter values to pass to the library function. In Visual Basic, this will be a variable defined with the Dim or Global statement. If the library function has parameters , this value should point to a valid memory block for the caller that is large enough to store the library function parameters. Otherwise, a General Protection Fault may occur.

Calling Considerations

It's ok to call DoEvents but don't display modal dialogs or call the Msgbox function in a VB/DLL library function. While modal forms cannot be displayed in the library function, a timer-style delayed event can be used to show the form. If the caller needs to wait for the form to be dismissed, it can use a second function to check on the state of the modal form which it calls in a DoEvents loop.

VB/DLL libraries do not support ByVal parameter passing. All parameter value changes made in the library function will be reflected in the callers variables.

Declaration Code

VB Declaration Code

```
Global Const ERR_CTL_APP_NOT_LOADED = 15000
Global Const ERR_LOADING_CONTROL_APP = 15001
Global Const ERR_SERVER_APP_NOT_LOADED = 15004
Global Const ERR_SERVER_FUNCTION_NOT_FOUND = 15006
Global Const ERR_FUNCTION_BUSY_TIMEOUT = 15008
Global Const ERR_FUNCTION_EXECUTION_TIMEOUT = 15009
Global Const ERR_CTL_MEM_ALLOC    = 15010
Declare Function VBFunctionCall Lib "VBDLLCTL.VBX" (ByVal lpString As String, ByVal lpString2 As String, lpRet As Any, lpParms As Any) As Integer
```

C Declaration Code

```
#define ERR_CTL_APP_NOT_LOADED 15000
#define ERR_LOADING_CONTROL_APP 15001
#define ERR_SERVER_APP_NOT_LOADED 15004
#define ERR_SERVER_FUNCTION_NOT_FOUND 15006
#define ERR_FUNCTION_BUSY_TIMEOUT 15008
#define ERR_FUNCTION_EXECUTION_TIMEOUT 15009
#define ERR_CTL_MEM_ALLOC           15010
short FAR PASCAL _export VBFunctionCall (char far *, char far *, void far *, void far *);
```

Error Codes

21000 - The DLLCTRL.EXE file either could not be found or an error was encountered starting it up. A re-install will copy it to the Windows system directory.

21001 - See 21000.

21004 - The specified library program either could not be found or generated an error when started.

21006 - The specified library function could not be found. This means that no VBDLLCTL control existed on any of the library program's loaded forms whose function names as set up in the Edit Declarations Dialog matched the Function Name parameter and whose declaration was valid.

21008 - A time-out occurred waiting for the library function to execute on behalf of another caller.

21009 - A time-out occurred waiting for the library function to complete.

21010 - An error occurred allocating memory.

21011 - Error processing parameters.

VBFunction Function

Overview

The VBFunction function is located in the VBDLLCTL.VBX library in the Windows system directory. It's purpose is to provide a more convenient method than the VBFunctionCall Function for calling VB/DLL library functions. It can only be used with VB and other closely compatible languages that can use VBX's like Access Basic. This is because it generates VB runtime error message boxes to display error text for VB/DLL errors that occur. Another difference between VBFunction and VBFunctionCall is that VBFunction can have the VB/DLL library function parameters passed directly to it eliminating the need for the Type...End Type declarations. Callers from VB can have a Declare Function statement with the Alias "VBFunction" phrase for each library function, even if the functions take different numbers of parameters. This way, it seems to the VB caller that different DLL functions are being called with different parameter lists when in reality, only the VBFunction call is executing. VB Declare statements for individual library functions are generated by the Show Code Window in the Library Manager program. The VB/DLL library function parameters (if there are any) are passed to the function first, followed by the three described below which must always be present. In all other ways, VBFunction operates the same way as VBFunctionCall. See the Client VB Project for examples of how to use this function.

Parameters

The VBFunctionCall function accepts a varying number of parameters but these three must always be present and must always be last.

Library Name - A null-terminated string identifying the drive, path and filename of the library executable file. e.g. C:\LIB\LIBRARY.EXE. If a .MAK file extension is specified, and the proper library is currently running in the VB development environment, the library function will execute in the development environment, allowing the debugger to be used. Otherwise, the parameter is used as a command string to start the library program if it's not already started.

Function Name - A null-terminated string identifying the library function to execute. This name must match the name of a function as set up in the Edit Declarations Dialog for a VBDLLCTL custom control residing on a loaded form.

Return Value - A pointer to storage that will store the library function return value. In Visual Basic, this will be a variable defined with the Dim or Global statement or with a type shorthand character (like $ for String). If a return value is not desired, 0 can be passed. Otherwise, this value should point to a valid memory block for the caller that is large enough to store the library function return value. Otherwise, a General Protection Fault may occur.

Passing Parameters

Types Supported

VB Variants - 16 byte Visual Basic variant type. This type can store any VB data type including date/time and currency.

Integer - 16 bit signed integer. Type character is %.

Long - 32 bit signed integer. Type character is &.

Single - 32 bit floating point. Type character is !.

Double - 64 bit floating point. Type character is #.

String - Normal VB HLSTR-type string variable (not defined with * length declaration). Type character is $.

String * XX - Indicates that a pointer to a data buffer of length XX will be passed. This is mainly for non-VB callers. In VB, String * variables cannot be passed to directly to functions but they can be included in Type...End Type structure declarations. To pass a String * type to VBFunction, pass a normal VB string but define the parameter "ByVal As String" in the callers Declare Function statement (not in the library program's Declaration property). Make sure the buffer size is large enough to hold the passed string with its null-terminator or a General Protection Fault may result. Also, any changes made to the buffer by the library function will not affect the normal VB string passed in this manner.

Specifying a Return Value

If a library function has a return value specified in it's declaration, memory to store the return value must be allocated by the caller and passed to the VBFunctionCall Function. The return value generated by the library function must be passed to the CopyParmsFromVB function as parameter 3.

User Defined Types

User defined types are not directly supported by the library builder. They can be used if a function's declaration breaks down the user defined type into elemental data type components that are supported. This can be done easily by copying the VB Type..End Type declaration for the user defined type to the clipboard and pasting it into the Declaration text box in the Edit Declarations Dialog.

Arrays

Passing VB arrays is not directly supported by the Library Builder. Individual VB array elements can be passed by coding the specific array element (e.g arr(24)) in the Parameters parm of the VBFunctionCall Function. An entire array can be passed by writing a library function to add passed elements to the array and placing the function call in a For..Next loop. If the array is a user defined type, see the previous section titled User Defined Types.

Library Manager

Overview

The Library Manager program serves several purposes. Mainly, it's used to define which functions will be exported by the library program and what parameters will be passed. It also displays information about library programs and functions and generates and displays sample code for calling and implementing library functions. It can be started stand alone or triggered by clicking on the EditDeclarations Property or by clicking the right mouse button on any design time VBDLLCTL control.

Edit Declarations Dialog

This is used to define the functions exported by the library program and their parameters. It's displayed by starting the Library Manager stand alone, by clicking on the EditDeclarations Property or by clicking the right mouse button on any design time VBDLLCTL control. Each loaded VBDLLCTL control is placed in the Control List. When a control is selected, the names of the functions exported by the selected control instance are placed in the Function List. When a function is selected, its declaration is displayed in the declaration text box where it can be edited. New functions are added by typing a new name in the Function Name text field.

Paste New Button - This button will use the clipboard contents to add a new function. The clipboard should contain an entire VB function declaration with or without the Declare Sub or Declare Function keywords.

Delete Button - This will delete the selected function from the selected control instance.

Goto Code Button - This will activate the Code Window for the selected control instance displaying the CallProc event code for the control.

Goto Ctl Button - This will activate the selected control instance in the VB design environment.

Show Code Button - This will display the Show Code Window for the selected function.

Close Button - Exits the Library Manager. Changes made in the dialog are automatically made to the corresponding controls. For changes to take permanent effect, the changed forms must be saved to disk by the VB design environment.

Show Code Window

Displays sample code generated for specific VB/DLL library functions. This window is opened by clicking the Show Code button in the Edit Declarations Dialog.

Copy Button - Copies the current window text to the clipboard.

Paste Button - Pastes the clipboard contents into the window at the caret position.

Goto Code Button - Displays the VB code window for the control. If the button is disabled, the control has been unloaded.

Close Button - Closes the Library Manager application.

Show Type/Show Menu Option - Toggles between sample code types.

Show Libraries Dialog

Available from both the Show Code Window and the Edit Declarations Dialog the will displays information about VB/DLL libraries and functions.

Library List - Displays currently loaded VB/DLL library programs.

Function List - Displays successfully loaded functions for the currently selected library program in the Library List.

Client VB Project

Overview

The Client project is copied to the installation directory during the install process. It is set up to demonstrate both the VBFunction Function method and the VBFunctionCall Function method for calling library functions.

Server VB Project

Overview

The Server project is copied to the installation directory during the install process. It's purpose is to show how library programs respond to the CallProc Event as well as how the custom control properties are used.

Helpful Hints

General Protection Faults - This usually means that parameters were mismatched between the caller and the library function.

Check Run-time Loading - Use the Show Libraries menu option in the Library Manager when the library program is running.

Check Return Codes - Check the VBFunctionCall Function return code and look it up in the error code section.

Check Show Code Window - This will tell if the function declaration was coded improperly, preventing the function from properly loading.

Licensing

A single user license for VB/DLL can be obtained directly from DataObjects, Inc. for $249 US dollars. MasterCard, Visa and Amex are accepted. The software comes with a 30 day money back guarantee.

Library Distribution

The only files that need to be distributed with library programs are the VBX control and its companion EXE. The file names are VBDLLCTL.VBX and DLLCTRL.EXE. These should be located in the WINDOWS\SYSTEM directory and can be distributed without any run-time fees. A technique that has been used is to write a .BAS file with short "stub code" functions to be distributed with a library. This will insulate users of the library from having to directly execute the VBFunctionCall Function

Uninstalling VB/DLL

To un-install VB/DLL take the following steps.

1) Determine the VB/DLL install directory. If not known, check the InstallDir setting in the VBDLLCTL.INI file in the Windows directory.
2) Delete all files from the install directory and remove it.
3) Delete the following files from the Windows system directory: VBDLLCTL.VBX, DLLCTRL.EXE, LIBMGDLL.DLL
4) Delete the VBDLLCTL.INI file from the Windows directory.

Credits

The VB/DLL Library Builder for Visual Basic was written by Scott Walters. Scott can be contacted at:

DataObjects, Inc. - Voice/FAX (615) 987-0421

Internet - waltersm@well.sf.ca.us

CIS - 76260,162

Special thanks to Chad Hower of Phoenix Business Enterprises and Eric Isaacson of Eric Isaacson software for their valuable contributions.