Using ActiveX DLLs in Liberty BASIC

Part 1
Originally published in NL 130.
-
DennisMcK

Using an ActiveX DLL

This tutorial will attempt to explain how to use an AciveX dll with Liberty BASIC and LB_DispHelper.

The subjects covered are:

1. Registering an ActiveX dll.
2. Initializing COM.
3. Uninitializing COM.
4. How to obtain the ProgID of the ActiveX dll.
5. Getting the IDispatch pointer to the ActiveX dll.
6. Releasing the IDispatch pointer to the ActiveX dll.
7. Calling the methods in the AciveX dll.
8. Getting return values from the ActiveX dll methods.
9. How to work with classes (coclasses) in the ActiveX dll.
10. Working with variant Dates.

LB_DispHelper is based on DispHelper, a free open source C library developed by xtmouse. DispHelper provides the necessary functions for interfacing with COM. The home of DispHelper is http://disphelper.sourceforge.net . The library cannot be used as-is with Liberty BASIC because it is a C library and uses unicode strings which LB does not support at this time. LB_DispHelper.dll was created to make DispHelper compatible with Liberty BASIC. Interfacing functions for some of the C library routines were added and modifications were made to convert strings from ansi to unicode and back. Variants are converted to the correct variable type within the library. It should be noted that ActiveX controls cannot be used with DispHelper at this time. The complete set of free LB_DispHelper ActiveX tools may be found at The Liberty Belle.
Download DLL and documentation here:

ActiveX Demo.zip

- Details
- Download
- 91 KB

The ActiveX dll used in this tutorial is Xzip.dll from http://www.xstandard.com . Xzip provides industry standard zip and unzip abilities for your application. This dll was chosen because it is fairly simple to implement and is well documented.

Registering the DLL

Before you can use an ActiveX dll it must registered with regsvr32.exe. The dll should be installed into a folder that will be its permanent location. If the dll is moved to a different folder after it's registered it will no longer work. If you decide to rename the folder it occupies or move it to a new location it must be unregistered first, then registered again after it is moved.
To register the dll:
1. Move the dll to a directory like: "C:\Program Files\XStandard\Bin\".
2. Open a command prompt and cd to the directory where the dll is located.
3. Type regsvr32 XZip.dll

To unregister the dll:
1. Open a command prompt and cd to the directory where the dll is located.
2. Type regsvr32 -u XZip.dll

With some systems you may need to type the full path to both regsvr32 and XZip.

Code to Use the DLL

Now that the dll is registered we can begin writing some code to use it. The LB_DispHelper.dll functions that are used will be explained as we progress. Most of the dll functions return pointers or take pointers to variables so structures are used within the program for this purpose.

```
Struct comObj, obj As Ulong 'for receiving pointers to COM objects
Struct LVAL, x As Long
'for receiving numeric returns (long integers)
                'from COM methods
Struct STRVAL, x As Ptr
'for receiving string returns from COM methods
```

To use COM it must be initialized, and when the program is finished it must be Uninitialized. Since LB_Disphelper and oleaut32.dll should also be opened and closed at these same times, a couple of subs can be used to simplify things.

```
'Call at the start of the program.
'Opens LB_dispHelper, initializes COM,
'and turns on DispHelper error reporting.
Sub BeginCOM
    Open "oleaut32.dll" For Dll As #ole
'Required for variant Date conversion.
    Open "LB_dispHelper.dll" For Dll As #com
    Calldll #com, "dhInitializeCom", FALSE As Long, r As Long
    Calldll #com, "dhToggleExceptions", 1 As Long, r As Long
End Sub

'Call at the end of the program.
'Closes LB_dispHelper, uninitializes COM.
Sub EndCOM
    Close #ole
    Calldll #com, "Uninitialize_COM", 1 As Long, r As Void
    Close #com
End Sub
```

Difference from Normal DLLs

In Liberty BASIC, dlls are normally opened with 'Open "this.dll" for dll as #x', but this will not work with AcitiveX dlls. Instead, an IDispatch pointer for the AcitiveX dll must be obtained. Don’t worry about what an IDispatch pointer is, it’s a number similar to a function pointer or a window handle and it's used to identify a COM object in your program. To get this pointer you must know the VersionIndependentProgID or ProgID for the ActiveX dll. This is a string that should be provided by the ActiveX developer but in many cases it isn't. In cases where the ActiveX dll has a both a ProgID and an VersionIndependentProgID, you should use the VersionIndependentProgID. The Xzip API reference lists it as "XStandard.Zip".
If you use a dll that isn't documented or doesn't give you the one of the ProgIDs you can use the LB_DispHelper Object Browser to create your own documentation.
Now that we know the ProgID we can get the IDispatch pointer. This next function will take care of this.

objZip = CreateObject("XStandard.Zip")

```
'Create an instance of ObjName$ on the local machine
Function CreateObject(ObjName$)
    Calldll #com, "dhCreateObject", ObjName$ As Ptr, _NULL As Long, _
    comObj As Struct, r As Ulong
    CreateObject = comObj.obj.struct
    comObj.obj.struct = 0
End Function
```

This is very important, every object (IDispatch pointer) obtained must also be released before your

program ends. Again, a simple sub can be used.

```
Sub SetNothing Object
    Calldll #com, "dhReleaseObject", Object As Ulong, r As Void
End Sub
```

To release the objZip IDispatch pointer:

```
Call SetNothing objZip
```

Using the ActiveX DLL with LB_DispHelper

Now we can move on to using the ActiveX dll. It's common for the methods in these dlls to use optional arguments and LB_DispHelper uses methods that allow for a variable number of arguments. The documentation for Xzip lists all of the subs, functions, classes, properties, and constants that can be used. This tutorial only uses a few of them.

The first one we will examine is

```
Sub Pack(sFilePath As String, sArchive As String, [
bStorePath As Boolean = False], [sNewPath As String], [
lCompressionLevel As Long = -1])
Add file or folder to
 an archive. Compression level 1 is minimum, level 9
 is maximum, all other values default to level 6.
```

sFilePath is the path to the file to be added to the zip file.
sArchive is the full path to the zip file.

The arguments within brackets [ ] are optional. To call this sub from LB we use the dhCallMethod function in LB_DispHelper. dhCallMethod is used to call methods that do not return values, in other words, Subs. For now we will ignore the optional arguments and only use sFilePath As String, and sArchive As String.

```
 Sub Zip.Pack XZipObject, src$, zip$
    Calldll #com, "dhCallMethod", XZipObject As Ulong, _
    ".Pack(%s, %s)" As Ptr,_
    src$ As Ptr, zip$ As Ptr, r As Long
```

```
End Sub
```

Examine this breakdown of the above call to dhCallMethod.

```
Calldll #com, "dhCallMethod", _
    XZipObject As Ulong, _
'The IDispatch pointer of the COM object.
    _                   'In this case objZip.
    ".Pack(%s, %s)" As Ptr,_
'We are calling sub Pack with two string arguments.
    src$ As Ptr, _            'Arg 1: sFilePath As String.
    zip$ As Ptr, _            'Arg 2: sArchive As String.
    r As Long                 'LB_DispHelper return, 0 = success.
```

The second argument to dhCallMethod specifies the method name that is being called along with the types of the arguments for sub Pack. ".Pack(%s, %s)". DispHelper uses a C printf style format string to identify the arguments and their types. Each argument is specified by a type identifier. Below is a list of the type identifiers supported by LB_DispHelper.

Identifier Type

%d Long
%D Date (variant)
%u Ulong
%e Double
%b Boolean
%s String
%o IDispatch pointer
%p LPVOID - Use for HANDLEs, HWNDs.
%m Missing argument place-holder. Use when an omitted optional argument precedes a used optional argument.

Therefore if you were calling a method with 3 arguments consisting of a string, long, and boolean the syntax would be: ".MethodName(%s, %d, %b)".

When calling a method with no arguments the format string is omitted: ".MethodName".

The next method of Xzip we'll need is

```
Sub UnPack(sArchive As String, sFolderPath As String, [
sPattern As String])
```

Extract contents of an archive to a folder.

sArchive is the full path to the zip file.
sFolderPath is the path to the folder where the unzipped files will be placed.
Our wrapper:

```
Sub Zip.UnPack XZipObject, zip$, destination$
    Calldll #com, "dhCallMethod", XZipObject As Ulong, _
    ".UnPack(%s, %s)" As Ptr,_
    zip$ As Ptr, destination$ As Ptr, r As Long
End Sub
```

This next method of Xzip is different, and a little complicated. To use it requires an understanding of two additional classes and their properties.

```
Function Contents(sArchive As String) As Items
```

Get a list of files and folders in the archive.

The obvious difference is that this is a function and returns a value. Not so obvious is the return type of Items. Further examination of the Xzip documentation shows that Items is a class with two properties.
Class: Items

Property Count As Long
(read-only)
Returns the number of members in a collection.

Property Item(Index As Long) As Item
(read-only)
Returns a specific member of a collection by position.

Notice that the property Item has a return type of Item. The Xzip documentation shows that Item is also a class, with five properties.

Class: Item

Property Date As Date
(read-only)
Last modified date.

Property Name As String
(read-only)
File name.

Property Path As String

(read-only)
Relative Path.

Property Size As Long
(read-only)
File size in bytes.

Property Type As ItemType
(read-only)
Type of object.

This last property says it returns ItemType.
From the Xzip documentation:
Enum ItemType

Const tFolder = 1
Item is a folder.

Const tFile = 2
Item is a file.

Ok, ItemType is just a value of either 1 or 2.
Did you notice that the Date property has a return type of Date? The Date data type is a number that will require further manipulations to become a human readable date and time. You will see how to do this a little later in this tutorial.

What all of this is used for is to retrieve a list of files in the zip archive, their relative paths (if any) and also some information about the files. We will need this information to show the user what's in the zip file. Starting with the function Contents, Contents returns a list of files and folders in the archive as Items. We can deduce that Items is a collection from the description of it's properties, so Contents returns an Items collection. A collection is a group of objects, similar to an array. Items is a class so it will have it's own IDispatch pointer, and each member of the Items collection is an Item, also a class and also with a unique IDispatch pointer. Each Item contains the information for one file or folder.

It's quite understandable if you are a bit confused at this point. If you are, then perhaps an analogy to a window will help clarify things. Imagine that you have a program that displays a window. The window has several buttons in it. In your program you write a function that gets all of the handles to the buttons and puts them into an array. You name this function Contents because it puts the contents (button handles) of the window into an array. You name this array Items because it contains the handles to each Item (button) in the window. Then you write a routine that uses each button handle (Item) and lists the properties of that button; the button text, width, height, position, etc. In short, you have a function (Contents) that gives you an Items collection (the Items array), and each Item (button handle) in the collection represents a button in the window. This is similar to what we are working with here.

Listing Files in the Zip Archive

Now that we possess all of this information a course of action can be charted to list the files in the zip archive:

1. Call Contents and get the IDispatch pointer to the Items collection.
2. Get the number of Item members contained in the Items collection from the Items Count property.
3. Get an IDispatch pointer for an Item in Items.
4. Get the file information from the Item properties.
5. Release the IDispatch pointer for this Item.
6. Repeat steps 3, 4, and 5 for every Item in Items.
7. Release the IDispatch pointer for Items.

Demo

Here is the code, step by step.

First we need to get a return value (the IDispatch pointer to Items) from the Contents method so we will use the function dhGetValue from LB_DispHelper.

```
'Step 1. Call Contents and get the IDispatch pointer to the Items coll
ection object.
Calldll #com, "dhGetValue", _
    "%o" As Ptr, _
'The variable type to return. In this case
    _
'an IDispatch pointer to the Items collection. (Items object).
    comObj As Struct, _
'The struct to recieve this type of return.
    objZip As Ulong, _
'Our IDispatch pointer to Xzip, because Contents
    _                   'is a method of Xzip.
    ".Contents(%s)" As Ptr, _ 'The method being called,
    _                   'with one string argument specified.
    zipFile$ As Ptr, _         'The argument, sArchive As String.
    r As Long                  'LB_DispHelper return, 0 = success

objItems = comObj.obj.struct
'Get the returned IDispatch pointer to Items
                    'from the struct and assign it to objItems.
comObj.obj.struct = 0          'Reset the struct value.
```

```
'Step 2. Next we need to get the number of Item objects contained in
'the Items collection from the Items Count property.
count = GetValueLong(objItems, ".Count")

'Get each Item object from the Items collection.
'If the Item represents a file, get the file name, size, and path.
tFile = 2
For Idx = 1 To count
    'Step 3. Get the IDispatch pointer for Item (Idx)
    Calldll #com, "dhGetValue", _
        "%o" As Ptr, _        'return an IDispatch pointer.
        comObj As Struct, _
'The struct to recieve this type of return
        objItems As Ulong, _  'The IDispatch pointer to the object
    _              'containing this method
        ".Item(%d)" As Ptr, _ 'The method being called,
    _              'with one long argument specified.
        Idx As Long, _        'The argument, Index number
        r As Long

    objItem = comObj.obj.struct: comObj.obj.struct = 0

    'Step 4. Get the file information from the Item properties.
    If GetValueLong(objItem, ".Type") = tFile Then 'it's a file
        fileName$ = GetValueStr$(objItem, ".Name")
        fileTime$ = GetValueDateTime$(objItem, ".Date")
        fileSize = GetValueLong(objItem, ".Size")
        filePath$ = GetValueStr$(objItem, ".Path")
    End If


'Step 5. We're done with this Item object, release it's IDispatch poin
ter.
    Call SetNothing objItem
Next Idx

'Step 6. We're done with the Items object, release it's IDispatch poin
ter.
Call SetNothing objItems
The routine above uses the following three support functions. All thre
e support functions use the dhGetValue function from LB_DispHelper.

Function GetValueLong(Object, method$)
    Calldll #com, "dhGetValue", _
        "%d" As Ptr, _      'Return a long
```

```
        LVAL As Struct, _   'The struct to recieve this type of return
        Object As Ulong, _  'The IDispatch pointer to the object
    _             'containing this method
        method$ As Ptr, _
'The method being called, with no arguments
        r As Long           'LB_DispHelper return, 0 = success

    GetValueLong = LVAL.x.struct
    LVAL.x.struct = 0
End Function


Function GetValueStr$(Object, method$)
    Calldll #com, "dhGetValue", _
        "%s" As Ptr, _      'Return a string
        STRVAL As Struct, _ 'The struct to recieve this type of return
        Object As Ulong, _  'The IDispatch pointer to the object
    _             'containing this method
        method$ As Ptr, _
'The method being called, with no arguments
        r As Long           'LB_DispHelper return, 0 = success

    x=STRVAL.x.struct           'Get the pointer to the string
    GetValueStr$ = Winstring(x) 'Retrieve the string

    'Free the memory containing the string,
    'this must be done to prevent memory leaks.
    Calldll #com, "FreeString", x As Ulong, r As Void
    STRVAL.x.struct = ""
End Function


Function GetValueDateTime$(Object, method$)
    TIME.NOSECONDS = 2

    Struct DT, x As Double
'A local struct guarantees a 0 starting value

    Struct st, _ 'SYSTEMTIME
    wYear As Word, _
    wMonth As Word, _
    wDayOfWeek As Word, _
    wDay As Word, _
    wHour As Word, _
    wMinute As Word, _
    wSecond As Word, _
```

```
    wMilliseconds As Word

    'Get the file date and time
    Calldll #com, "dhGetValue", _
        "%D" As Ptr, _      'Return a variant Date
        DT As Struct, _     'The struct to recieve this type of return
        Object As Ulong, _
'The IDispatch pointer to the object containing this
        _                   'method
        method$ As Ptr, _
'The method being called, with no arguments
        r As Long           'LB_DispHelper return, 0 = success
    vtDate = DT.x.struct

    'Convert the vtDate value into a human readable date and time.

    'Fill the systemtime structure using the vtDate value.
    Calldll #ole, "VariantTimeToSystemTime", vtDate As Double, _
    st As Struct, r As Long

    'Get a date string in the same format as used on this machine.
    dtBuf$ = Space$(50)
    Calldll #kernel32, "GetDateFormatA",
 _LOCALE_SYSTEM_DEFAULT As Ulong, _
        _NULL As Ulong, st As Struct, _NULL As Ulong, dtBuf$ As Ptr,
50 As Long, _
        r As Long
    dt$ = Left$(dtBuf$, r-1)

    'Get a time string, without seconds,
    'in the same format as used on this machine.
    dtBuf$ = Space$(50)
    Calldll #kernel32, "GetTimeFormatA",
 _LOCALE_SYSTEM_DEFAULT As Ulong, _
        TIME.NOSECONDS As Ulong, st As Struct, _NULL As Ulong,
 dtBuf$ As Ptr, _
        50 As Long, r As Long
    tm$ = Left$(dtBuf$, r-1)

    GetValueDateTime$ = dt$ + " " + tm$
End Function
```

That's enough to digest for now. In Part 2 we will build a simple Zip application that puts this information to good use. Until then I hope you will help yourself to the LB_DispHelper ActiveX tools and read the LB_DispHelper help file.