Getting the Most Out of ArcView GIS

You Make the Call in Avenue

By Todd Stellhorn, ArcView GIS Development Lead

A powerful way to enhance the capabilities of ArcView GIS is to include calls to external procedures defined in dynamic-link libraries (DLLs). Using a DLL in an Avenue application allows you to extend ArcView GIS tremendously. Virtually any function that is defined in a DLL can be called by an Avenue script.

ArcView GIS supports several other methods for taking advantage of the functionality in other applications—the use of system level commands and Remote Procedure Calls (RPCs). Both of these methods are available in Windows and UNIX environments. DLLs can be used only in a Windows environment, although the same functionality is available on UNIX by using shared libraries. Choosing the most appropriate method depends on the tasks to be performed and the level of application integration desired.

However, there are several advantages to using DLLs. They are linked at run time not at compile time. Many DLLs are commercially available or part of a product's application programming interface (API). Each major component of the Windows operating system is a DLL. Some of these Windows DLLs can be directly accessed by ArcView GIS. Custom DLLs can be created using languages such as C or Microsoft Visual C++. The online help for ArcView GIS does a very good job of documenting DLL procedures. Read the information provided by the online help for DLL and DLLProc classes and the topic "Writing Your Own DLL." A brief review of DLL functionality is included here.

Definitions

Before jumping directly into using DLLs, it's a good idea to review a few definitions. A DLL is an executable module that contains functions or procedures that can be called by an external program. Functions and procedures are essentially the same thing. The main difference between the two is that a function returns a value. Generally, at load time an executable needs to find all of the functions and procedures required for its execution. This process is called resolving the program's external dependencies and references. The dependencies are the DLLs, and the references are the functions or procedures defined in the DLLs. Resolving is performed by the linker module of the operating system dynamically at load time, hence the name dynamic-link library. However, Avenue employs another method for calling procedures in an external DLL. Avenue loads the DLL at run time and makes direct calls to it.

The Avenue Classes

The DLL and DLLProc classes are used to call external procedures in a DLL. The DLL class supports loading the external DLL and individual instances of the DLLProc class make calls to each of the desired DLL functions. Use the Make request to create an instance of the DLL class.

```
myDLL = DLL.Make
        (Œc:\temp\mydll.dllÓAsFileName)
```

DLL.Make returns a valid DLL object or NIL on failure. Failure to load the DLL can occur if the external DLL being called is not found or if that DLL references other DLLs that cannot be found. In the preceding example ("c:\temp\mydll.dll"), if mydll.dll contained external references to procedures defined in other DLLs that could not be found, mydll.dll would not load. All external dependencies and references must be resolved before the DLL will load. Consequently, verify the return value of the DLL. Make call because an error may be more involved than just an invalid file name.

Once a valid DLL object is created, creating instances of the DLLProc class for each external procedure to be called is the next task. Create the DLLProc instances using the DLLProc.Make request. Use the following syntax.

```
myDLLProc = DLLProc.Make
  (aDLL, aProcName, returnType,
  argumentList)
```

The table in Figure 1 explains each item in the Make request and lists the data type for each of the arguments. Both the arguments "returnType" and "argumentList" are specified as enumerations of type.

For each DLLProc created, the return type and the types of the arguments must be specified using DLLProcTypeEnum. In Avenue, the object representation will be either Number, String, or NIL. Common DLLProcTypeEnum values are shown in Figure 2.

The online help for ArcView GIS includes a complete list of DLLProcTypeEnum values. Note that you cannot pass or return double-precision floating point numbers using a DLLProc call.

Figure 1: The Make Request

myDLLProc	The returned instance of a DLLProc; NIL on failure
aDLL	A DLL object created using DLL.Make
aProcName	A string containing the name of the function or procedure that will be called in a DLL
returnType	Specifies the data type of the return value of the function or DLLPROC_TYPE_VOID in the case of a procedure
argumentList	Lists the data type for each of the arguments. Both the argument's returnType and argumentList are specified as enumerations of type.

Figure 2: Common Enumeration Values

#DLLPROC_TYPE_VOID
Used for procedures that return void (NIL).

#DLLPROC_TYPE_INT32
A 32-bit integer (Number)

#DLLPROC_TYPE_FLOAT
A single precision floating point number (Number)

#DLLPROC_TYPE_STR
A pointer to null-terminated character string (a C-type String)

You Make the Call

Once the Avenue DLL and DLLProc objects are created, an external function call can be made. Figure 3 contains an example from the online help. This example assumes a DLL named "math.dll" contains a function named "Add". The Add function takes two 32-bit integer values, adds them together, and returns the sum. In this example, "myDLL" is the Avenue DLL object, "add" is the Avenue DLLProc object, and "x" is the return value of calling the Add function from the math.dll.

Figure 3: An Example

```
myDLL = DLL.Make(Òc:\lib\math.dllÓAsFileName)
add = DLLProc.Make(myDLL, ÒAddLongsÓ, #DLLPROC_TYPE_INT32,
                   {#DLLPROC_TYPE_INT32, #DLLPROC_TYPE_INT32})
x = add.Call({500, 200})
MsgBox.Info(ÒThe result is Ò + x.asString, ÒÓ)
```

Common Errors

Far and away the most common error when using DLL and DLLProc classes is incorrectly specifying the data types for input arguments or return values. Arguments and returned values will be a specific data type that must be identified before the function can be called. Although ArcView GIS supports a variety of data types, some DLLs may use unsupported data types. Refer to the ArcView Class Hierarchy and the description of supported data types in the online help for more information. Once you've verified that the data types for arguments and return values are supported, make sure they are correctly listed in the code. There is no easy way to get this right other than checking, rechecking, and checking one more time. This is the most common error when using these classes and there is nothing ArcView GIS can do to help you. Using incorrect or unsupported data types can cause unpredictable results.

Another common error occurs when dealing with pointer types. For example, if the external function takes as an input argument a pointer to a character string and the evaluation of the function causes a result to be written to the memory pointed to by the character pointer, then that memory must be preallocated in the Avenue script before the call request is made on the DLLProc object. Use the MakeBuffer request on the String class to preallocate memory as shown in the following example.

```
aBuffer = String.MakeBuffer(1024)
```

A Real Example

The following code uses the standard Windows DLL, kernel32.dll, to create a new directory in the file system. The example assumes kernel32.dll is installed under the pathname c:\WinNT\System32. If the script is successful it will create a new directory named c:\temp\test.

```
Ô Example calling kernel32.dll function for creating a directory
kernel32 = DLL.Make(Òc:\WinNT\System32\kernel32.dllÓAsFileName)
if (kernel32 = nil) then
        MsgBox.Error(ÒError Þnding kernel32.dll.Ó, ÒCreateDirectory ExampleÓ)
    return nil
e n d
funName = ÒCreateDirectoryAÓ
Ô use the ANSI version of the function
mkdirProc = DLLProc.Make(kernel32, funName, #DLLPROC_TYPE_INT32,
    {#DLLPROC_TYPE_STR, #DLLPROC_TYPE_POINTER})
if (mkdirProc = nil)then
        MsgBox.Error(ÒCannot Þnd CreateDirectoryA function in kernel32.dll.Ó,
            ÒCreateDirectory ExampleÓ)
    return nil
e n d

newDir = Òc:\temp\testÓ
Ô name of the directory to create securityAtt = nil
Ô use default security attributes
stat = mkdirProc.Call({newDir, securityAtt})
if (stat = 0) then
        MsgBox.Error(ÒError calling CreateDirectoryA. Directory may already
exist.Ó ÒCreateDirectory ExampleÓ)
    return nil
e n d
```