Interface Definition Language

Jim Fawcett

CSE 775 - Distributed Objects

Error Codes

- Error codes are returned as HRESULTS by all COM interface functions, with the exception of AddRef() and Release().
  - Visual studio smart pointer class _com_ptr simulates return by value and throws exceptions on errors by wrapping the proxy's interface functions in wrapper classes that take care of those details.

- Test HRESULTS using the macros:
  - #define SUCCEEDED(hr) (long(hr) >= 0)
  - #define FAILED(hr) (long(hr) < 0)

- Specific Error Codes:
  - S_OK : successful normal operation
  - S_FALSE : return logical false as a success code
  - E_FAIL : generic failure
  - E_OUTOFMEMORY : memory allocation failed
  - E_NOTIMPL : method not implemented
  - E_UNEXPECTED : method call at incorrect time

Data Types

- Marshaling depends on exact knowledge of the sizes of data types.
- C and C++ do not define the sizes of their types. Each compiler and platform may define the sizes as they wish.
- COM bases its types on the NDR (Network Data Representation) types which have specified sizes.
- COM types suitable for marshaling are defined in wtypes.idl. These declarations include a lot of Windows specific data types as well as types useful for general COM programming.
- We can use all these types if we import wtypes.idl in our IDL file.

Decorations

- In order to marshal efficiently COM needs to know the direction of data flow, e.g.:

  [in], [out], [in, out], [out, retval]

  and will marshal data only in the direction(s) specified.

- For languages that have run-time support like Java and Visual Basic, an out parameter may be decorated with retval, indicating that those environments make the call look like a function return value:

  IDL : HRESULT Method([in] short arg, [out, retval] short *ret)

  Visual Basic: Function Method(arg as Integer) As Integer

  Since C++, has no such support, its interface looks like this:

  C++ : virtual HRESULT __stdcall Method(short arg, short *ret)

Memory Allocation

- Memory for [in] parameters is always allocated and freed by caller. Can use any kind of allocation, e.g., stack, heap, static.
- Memory for [out] parameters is always allocated by the method and always freed by the caller.
  - Method : CoTaskMemAlloc(ULONG size);
  - Client : CoTaskMemFree(LPVOID pv);
- Memory for [in,out] is allocated by caller, may be reallocated by method using:
  - CoTaskMemRealloc(LPVOID pv, ULONG size)
- Must be freed by caller.

Pointer Decorations

- In order to marshal efficiently COM needs to know how pointers will be used:
  - [ref] : pointers are initialized with valid (non-null) addresses at method invocation. This value can not change during method execution. All [out] pointers must be [ref]
  - [unique] : pointers may be null, can not be aliased.
  - [ptr] : same as unique except it can be aliased – requires much more work of marshaler, as it requires duplicate detection.

- Example:

```
HRESULT method([in,out,ref] int *pInt);
```

Strings

- All characters in COM are represented using the OLECHAR data type:
  - typedef wchar_t OLECHAR
- IDL uses the string decoration to tell the marshaler that a null terminated wide char string is being sent:
  - HRESULT method([in,string] const OLECHAR *pOC);
- You can initialize an OLECHAR string this way:
  - Const OLECHAR *pOC = OLESTR(“this is a string”);
- The C Run-Time Library provides two conversion functions:
  - size_t mbstowcs(wchar_t *pOC, const char *pC, size_t count);
  - Size_t wcstombs(char *pC, const wchar_t *pOC, size_t count);

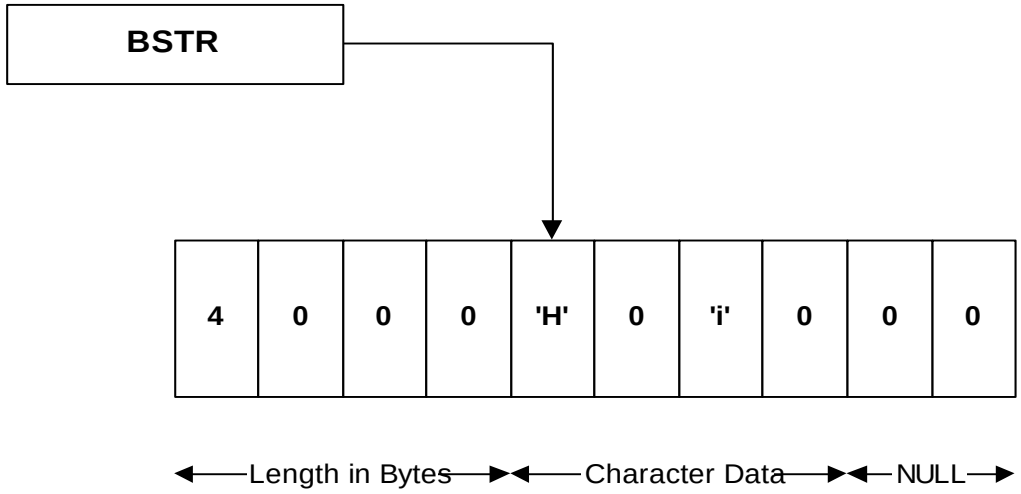
Support for OLECHAR Strings

- The C Run-Time Library provides wide char string support that parallels its ANSI char string support, e.g.:
  - wcslen : return number of characters in string (not equal to number of bytes)
  - wcscpy : copy a wide source string to a wide destination string. You have to allocate enough memory for destination.
  - wcscspn : find a substring in a wide char string
  - wcschr: : find first occurrence of a char in a wide char string.
  - wcsrchr : find the last occurrence of a char in wide char string.

- The C++ Standard library iostreams and strings module also provide support with:
  - wcout, wcin
  - wstring

BSTRs

- The BSTR type is a derived type used in Visual Basic and Microsoft Java (and presumably C#). BSTRs are recognized by the standard marshalers and used frequently by COM developers.
- BSTRs are length-prefixed, null terminated strings of OLECHARs.



BSTR Memory Allocation

- COM expects BSTRs to use a COM memory allocator, and provides several API functions for handling BSTRs, declared in oleauto.h:

  // allocate and initialize
  - BSTR SysAllocString(const OLECHAR *pOC);
  - BSTR SysAllocStringLen(BSTR *pBSTR, const OLECHAR *pOC, UINT count);

  // reallocate and initialize
  - INT SysReAllocString(BSTR *pBSTR, const OLECHAR *pOC);
  - INT SysReAllocStringLen(BSTR *pBSTR, const OLECHAR *pOC, UINT count);

  // free a BSTR
  - void SysFreeString(BSTR bstr);

  // peek at length count as OLECHAR count or byte count
  - UINT SysStringLen(BSTR bstr);
  - UINT SysStringByteLen(BSTR bstr)

BSTR Memory Management

- When passing BSTRs as [in] parameters, the caller invokes SysAllocString prior to calling the method and SysFreeString after the method has completed.
- When passing strings from a method as an [out] parameter, it is the responsibility of the method to call SysAllocString before passing back the string. The caller releases the memory by calling SysFreeString.
- When passing BSTRs as [in, out] parameters, you treat them like [in] parameters.
- Reference: If you are going to use BSTRs in your project code, make sure you look carefully at “Strings the OLE Way”, Bruce McKinney, in MSDN online or in help.
- CComBSTR class provides a lot of help handling BSTRs. Check it out in MSDN.

Arrays

- Fixed arrays have sized determined at compile-time:
  HRESULT method([in] double arr[8]);

- Conformal arrays have size determined at run-time:
  HRESULT method([in] long dim, [in,size_is(dim)] double *da);

- Varying array sends only part of array:
  HRESULT method([in,out] long *first, [in,out] long *last,
  [in,out,first_is(first),length_is(last-first+1),size_is(100)] long *la);

- Open array sends part of array – size is determined at run-time. Same as above, except argument of size_is( ) is a variable.

Other Data Types

- We will encounter the Variant and Safe Array data types when we discuss Automation and the IDispatch interface.

- A variant is a discriminated (tagged) union that will hold any of a large subset of the IDL data types. There are a set of system functions designed to help manipulate variants. These are declared in oleauto.h

- Reference: “The Ultimate Data Type”, Bruce McKinney, MSDN

- A Safe Array is a structure that holds, possibly multi-dimensioned, arrays with descriptors of their sizes. There are a set of system functions designed to help manipulate safe arrays. These are declared in oleauto.h

- Reference: “The Safe OLE Way of Handling Arrays”, Bruce McKinney, MSDN

References for IDL

- MSDN/Platform SDK/Component Services/Microsoft Interface Definition Language
- Essential IDL, Martin Gudgin, Addison Wesley, 2001
- Essential COM, Don Box, Addison Wesley, 1998
- COM IDL & Interface Design, Al Major, WROX, 1999