COM Types

Jim Fawcett

CSE775 - Distributed Objects

Spring 2007

IDL Base Types

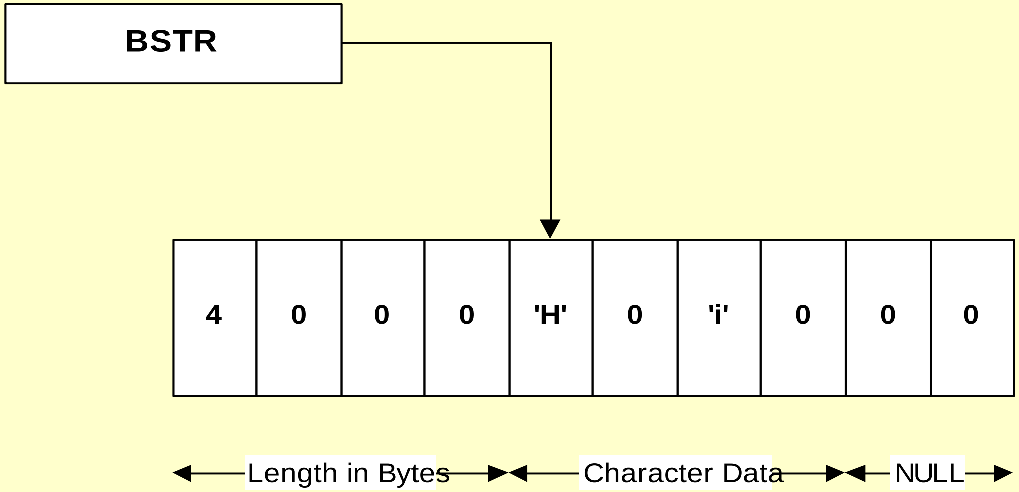
Boolean	A data item that can have the value TRUE or FALSE.
Byte	An 8-bit data item guaranteed to be transmitted without any change.
Char	An 8-bit unsigned character data item.
Double	A 64-bit floating-point number.
Float	A 32-bit floating-point number.
handle_t	A primitive handle that can be used for RPC binding or data serializing.
Hyper	A 64-bit integer that can be declared as either **signed** or **unsigned** Can also be referred to as **_int64**.
int	A 32-bit integer that can be declared as either **signed** or **unsigned**.
__int3264	A keyword that specifies an integral type that has either 32-bit or 64-bit properties.
Long	A modifier for **int** that indicates a 32-bit integer. Can be declared as either **signed** or **unsigned**.
Short	A 16-bit integer that can be declared as either **signed** or **unsigned**.
Small	A modifier for **int** that indicates an 8-bit integer. Can be declared as either **signed** or **unsigned**.
wchar_t	Wide-character type that is supported as a Microsoft® extension to IDL. Therefore, this type is not available if you compile using the **/osf** switch.

Automation Types

•BSTRs	- Basic Strings
•Variants	- Basic Data
•SafeArrays	- Basic Arrays

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

4	0	0	0	'H'	0	'i'	0	0	0

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.

BSTRS

- WCHAR = OLECHAR = wchar_t
- BSTR = wchar_t * = LPWSTR
- C language string = char *s = LPSTR

- BSTR is a pointer to the beginning of a sequence of wchar_t's
- HOWEVER, a BSTR always has four-byte length in front of the memory pointed to.
- You must always manage a BSTR's memory with the functions:
  - SysAllocString, SysFreeString, SysReallocString, ...

BSTR Rules

- Ref: “Strings the OLE Way”, Bruce McKinney
  - Allocate, destroy, and measure BSTRs only through the SysXXX functions
  - do what ever you like with the chars of strings you own, as long as you don’t write past the string buffer, measured by len
  - you may change the pointers to strings you own only through SysReAllocString or SysReAllocStringLen
  - you do not own any BSTR passed to you by value
  - you own any BSTR passed to you by reference as an in/out parameter
  - you must create any BSTR passed to you by reference as an out string, e.g., you are supplying a BSTR out parameter
  - you must create a BSTR in order to return it
  - a null pointer is an empty string, not just a pointer

Variant

- The variant type was developed for pre .Net Visual Basic, where it represented a data type that can hold, and convert between:
  - Strings, integers, floating point numbers, and objects of unspecified type.
- Programmatically, the variant is a discriminated union
- Variants are passed as arguments to Dispatch Interfaces. That is one of the few places you will see them used in this course.
- Another place is representing .Net objects on the COM side of a Runtime Callable Wrapper (RCW). The RCW is esentially a .Net object that is a COM client on the inside, and wraps some server the client has instantiated.

Variant Structure

- Variant is a discriminated union:

```
struct tagVARIANT {
  VARTYPE vt;
  WORD wReserved1; WORD wReserved2; WORD wReserved3;
  union {
    long lVal;              // VT_I4
    unsigned char bVal;     // VT_UI1
    short iVal;             // VT_I2
    float fltVal;           // VT_R4
    double dblVal;          // VT_R8
    VARIANT_BOOL boolVal; // VT_BOOL
    SCODE scode;            // VT_ERROR
    CY cyVal;               // VT_CY (currency)
    DATE date;              // VT_DATE
    BSTR bstrVal;           // VT_BSTR
    IUnknown *punkVal;      // VT_UNKNOWN
    IDispatch *pdispVal;    // VT_DISPATCH
    SAFEARRAY *parray;      // VT_ARRAY|*
     // other types that are windows specific
    VARIANT *pvarVal;       // VT_BYREF|VT_VARIANT
    void *byref;            // Generic ByRef
  };
};
```

Safe Arrays

- Safe Arrays also originated with Visual Basic. All pre .Net Visual Basic code represented arrays of data with Safe Arrays.
- A Safe Array is a structure:

```
struct SAFEARRAY {
  WORD cDims;               // number of dimensions
  WORD fFeatures;           // bit field describing attributes
  DWORD cbElements;         // size of array elements
  DWORD cLocks;             // lock reference count
  void * pvData;            // pointer to data on heap
  SAFEARRAYBOUND rgsabound[1];
};
```

- `Rgsabound[1]` is an array of boundary structures, that starts out life with one element, but may be expanded by safe array function calls.

References for VB Types

- Bruce McKinney’s articles:
  - Strings.htm
  - Variants.htm
  - SafeArrays.htm

ATL Support

- CComQIPtr
- CComBSTR
- CComSafeArray
- CComVariant