Interface Definition Language

- Types are problematic:
  - Weakly typed C language: short / int / long
  - Architecture-specific data types
- IDL is a strongly typed language
  - Concretely defined sizes for base types
- IDL uses Network Data Representation (NDR)
  - Architecture-independent network transmissions

Enumerated Types

- Enum keyword
  - Enums are transmitted as 16-bit values per default
  - [v1_enum] attribute generates 32-bit entities

```
Interface IWeek: IUnknown {
      typedef [v1_enum] enum DaysOfTheWeek
      {
                  Monday, Tuesday, Wednesday, Thursday,
                  Friday, Saturday, Sunday
      } DaysOfTheWeek;

      HRESULT Test(DaysOfTheWeek day);
    }
```

Directional Attributes

- Which variables need to sent to server?
- C++ does not indicate whether function changes pointer variables

- Attributes:
  - [in], [out], [in, out]
  - [in, out] models standard C++ behavior (extra network traffic)
  - [in] is default

Arrays

- Fixed Arrays
- Conformant Arrays
  - Several attributes to define size of arrays and data transmitted:
  - First_is, last_is, length_is,
    max_is, min_is, size_is
  - Caller specifies actual number of elements at runtime
- Varying arrays
  - Server may return less than the full number of elements
  - Maximum number of elements is bounded
- Open arrays
  - Caller/callee control size of memory blocks separately
- Multidimensional Arrays

Character Arrays

- Common programming practice...
  - Special [string] attribute

    ```
    HRESULT SendString1([in, string] wchar_t * myString);
    HRESULT SendString2([in] int cLength, [in, size_is(cLength)]
                                          wchar_t * myString );
    ```

  - Both versions are semantically equivalent
  - Client calls:

    ```
    wchar_t wszHello[] = L“My favourite String“;
    pTest->SendString1( wszHello );
    pTest->SendString2( wcslen( wszHello ), wszHello );
    ```

  - Problem: send/receive a string (memory allocation)

Pointers

- Full Pointers (problem of aliasing)
  - Stub code maintains dictionary of marshaled pointers
  - Avoid full pointers whenever possible
- Unique Pointers
  - Can point to any location
  - Can have the value null
  - Can change form non-null to null and vice-versa during a call
  - IDL ignores changes between non-null values during call
  - No aliasing

Pointers (contd.)

- Reference Pointers
  - Can point to any location
  - Cannot have the value null
  - No aliasing
  - Cannot change during a call
- Interface Pointers
  - C++ class and function pointers are off-limits to remote method calls
  - Access to code in different address spaces only through interface pointers
  - Problem: generic IF pointers, MIDL cannot generate stub code for void**
  - HRESULT GetInterfacePointer([out] IUnknown** ppvObject);
  - Client needs to call QueryInterface() afterwards

Interface Pointers (contd.)

- Use IDL attribute to identify type of IF pointer
  - HRESULT GetInterfacePointer([in] REFIID riid,
    [out], iid_is(riid)] void** ppvObject);
  - Client calls:

```
IMyCustomInterface * pCustomInterface;
pObject->GetInterfacePointer( IID_IMyCustomInterface,
        (void**) pCustomInterface );
```

- Special attributes to map non-remotable methods
  - [local] – directs MIDL not to generate stub code (for DLLs)
  - [call_as] – directs MIDL to treat parameter types differently