Hash table (associative array) COM object

- take existing C hash table code
- put a C++ / COM veneer on it
  - using Microsoft Visual C++
  - ATL Wizard to create framework and lots of files
  - insert semantics into framework
    ```c
    // insert your code here
    ```
- use it in VB applications
  - add reference to Hashcom object:
    ```vb
    Dim h as Object
    Set h = New Hashtable
    h.put name, val
    s = h.get(name)
    if h.member(s) then ...
    ```
- use this in Excel, scripts, etc.

Existing hash table code

```c
typedef struct Array Array;
Array *Anew(int n);
/* make a new empty array with size n */
int Aput(Array *A, char *s, char *v);
/* put an element into an array:
   A[s] = copy of d */
/* returns 0 if no room, 1 if installed,
   2 if already there */
char *Aget(Array *A, char *s);
/* get an element: return A[s],
   or 0 if not there */
int Amember(Array *A, char *s);
/* return 1 if A[s] is present, 0 if not */
int Asize(Array *A);
/* return number of current elements */
int Adelete(Array *A, char *s);
/* delete item, return new size */
```
Files created by VC++

IDL: Interface definition language

- COM defines binary format of interface
- IDL is a language for defining these interfaces
- specifies
  - type of each argument (int, float *, pointer, etc.)
  - role of each argument in call (in, out, inout, retval)
  - return type of function
  - miscellaneous other stuff

```c
interface IHashtable : Idispatch {
  [id(1), helpstring("method put")]
  HRESULT put([in] BSTR name, [in] BSTR val,
               [out,retval] int*stat);
  [id(2), helpstring("method get")]
  HRESULT get([in] BSTR name,
              [out,retval] BSTR *val);
  [id(3), helpstring("method member")]
  HRESULT member([in] BSTR name,
                  [out,retval] int *stat);
... }
```

- IDL compiler converts specification into function templates and code to marshal arguments for function calls
C++ generated by MIDL

MIDL_INTERFACE("24942DFC-6E32-48A0-AF77-C0C009EEC328")
IHastable : public Idispatch {
public:
    virtual /* [helpstring][id] */
    HRESULT STDMETHODCALLTYPE put(
        /* [in] */ BSTR name,
        /* [in] */ BSTR val,
        /* [retval][out] */ int __RPC_FAR *stat)=0;
    virtual /* [helpstring][id] */
    HRESULT STDMETHODCALLTYPE get(
        /* [in] */ BSTR name,
        /* [retval][out] */ BSTR __RPC_FAR *val)=0;
    virtual /* [helpstring][id] */
    HRESULT STDMETHODCALLTYPE member(
        /* [in] */ BSTR name,
        /* [retval][out] */ int __RPC_FAR *stat)=0;
    ...

• UUID: universally unique 128-bit identifier
  24942DFC-6E32-48A0-AF77-C0C009EEC328
  - every COM object has one
  - guaranteed unique across everything
  - used to identify objects regardless of where they are

Interface specification (IDL)
Add semantics to framework

BSTR string data type

• most scalar data types based on C++ types
  • strings are special: COM uses BSTR
    - 16-bit Unicode characters
    - 4-byte length field before the first character
    - small, irregular set of functions for manipulating them
  • Visual Basic, etc. all use BSTR
  • Windows API uses either Unicode (but not BSTR)
    or ASCII (8-bit, not 16)

```c
char *bstr2a(const BSTR cmd) // convert cmd to ASCII
{
  int n, i;
  char *buf;

  n = SysStringLen(cmd); // length of input
  buf = (char *) malloc(n+3); // wide to narrow
  for (i = 0; i < n; i++) buf[i] = (char) cmd[i];
  buf[i] = 0;
  return buf;
}

BSTR a2bstr(const char *buf1) // cvt buf1 to BSTR
{
  int i, n = strlen(buf1);
  BSTR buf2 = SysAllocStringLen(NULL, n);
  for (i = 0; i < n; i++) // narrow to wide
    buf2[i] = buf1[i];
  buf2[i] = 0;
  return buf2;
}```
Calling a COM object

- conceptually, what happens when a COM object is called from a program...

  • first time
    - find its code
      look up in Windows registry
    - do any initialization
      registered during install or when created or by explicit call
    - link it into current program (if a DLL)
      fill in calls with pointer to real code: vtbl

  • each subsequent method call
    - collect arguments into proper form ("marshalling")
    - call function
    - convert return value and output arguments into proper form

  • when done
    - do any finalization
    - release resources
      last user tells Windows that DLL is no longer in use

---

DLL startup code excerpt (machine generated)

// DLL Entry Point

extern "C"
BOOL WINAPI DllMain(HINSTANCE hInstance,
DWORD dwReason, LPVOID /*lpReserved*/)
{
if (dwReason == DLL_PROCESS_ATTACH)
{
    __Module.Init(ObjectMap, hInstance,
    &LIBID_HASHCOMLib);
    DisableThreadLibraryCalls(hInstance);
}
else if (dwReason == DLL_PROCESS_DETACH)
    __Module.Term();
return TRUE;  // ok
}

// Used to determine whether the DLL can be unloaded by OLE
STDAPI DllCanUnloadNow(void)
{
    return (_Module.GetLockCount()==0)
    ? S_OK : S_FALSE;
}
Use hashtable in VB

Automatically generated usage info
Use in Excel

```vbscript
Dim h As HashTable

Public Function hashinit() As Integer
    Set h = New HashTable
End Function

Public Function hashput(n As Range, v As Range) As Integer
    If h Is Nothing Then Set h = New HashTable
    For i = 1 To n.Count
        h.put n.Cells(i, 1), v.Cells(i, 1)
    Next i
    hashput = n.Count
End Function

Public Function hashget(n As Range) As String
    hashget = h.get(n.Cells(1, 1))
End Function

Public Function hashsize() As String
    hashsize = h.Size()
End Function

Public Function hashmember(n As Range) As Integer
    Dim c As Integer
    c = 0
    For i = 1 To n.Count
        If h.member(n.Cells(i, 1)) = 1 Then c = c + 1
    Next i
    hashmember = c
End Function
```