Where do we go from here?

- **Visual Basic**
  - language
  - development environment
  - building GUI's
  - scripting
    - embedding
    - viruses
- **component-based software**
  - libraries and software re-use
  - COM
    - creating your own components
  - other approaches to components
    - CORBA, RMI
  - C# and .NET
    - the next generation
- **XML and related acronyms**

Visual Basic

- **Windows graphics model similar to X Windows**
  - big library, with graphics primitives at the bottom
  - event loop
  - graphical components
- **but different in many respects**
  - not distributed, not portable
  - more complicated
  - large library interface
- **Visual Basic for building GUI's**
  - a language at about the same level as Java
    - also usually interpreted
  - controls analogous to Java Swing
    - similar properties, methods, events
  - interactive development environment
    - draw the interface on the screen
      - generally don't use layout managers
    - code templates for binding actions to events
    - create the code, run, debug within the environment

Why study / use Visual Basic?

- **one of the most widely used languages / systems**
- **very easy to start with**
- **very easy to do useful things**

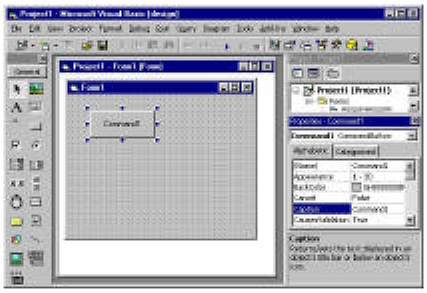
  http://www.cs.princeton.edu/courses/archive/
  fallxx/cs109/labs/VB1 and VB2

- **easy access to Windows environment**
  - can do almost anything that can be done in Windows
    - may not be fast
    - may not scale up to big programs or big data
- **embedded in other tools as extension mechanism**
  - Word, Excel, Powerpoint, ..., all contain VB
  - can easily augment their capabilities
  - scripting language for controlling other programs (VBScript)
- **at the heart of a class of computer viruses**

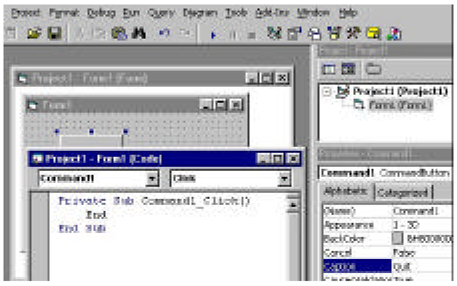
Visual Basic components

- **Visual Basic programming language**
  - modern dialect of Basic (Basic created in 1964 by John Kemeny ('47, *49) and Tom Kurtz (*56))
  - reasonable control flow, data types, arrays, structures
  - a bit bulky, verbose, clumsy
  - good error checking at "compile" and run time
- **toolkit / components**
  - standard library for math, file I/O, text manipulation
  - user interface components: buttons, text, menus, ...
  - extensible:
    - access to Windows API and existing objects
    - can add own C/C++ code and create new controls
  - "glue" language for assembling from pre-built pieces
- **integrated development environment**
  - interactive system for building and testing VB programs (~1991)
    - draw interface by dragging and dropping components
    - fill in behaviors in code templates
    - set properties like size, color, position, ...
    - manage/edit source code and other resources
    - run in controlled environment for testing and debugging
    - compile and export as .EXE file

Visual Basic environment



(Name)	Command1
Appearance	1 - 3D
BackColor	&H8000000F
Cancel	False
Caption	Command1
CausesValidation	True



```
Private Sub Command1_Click()
    End
End Sub
```

(Name)	Command1
Appearance	1 - 3D
BackColor	&H8000000
Cancel	False
Caption	Quit

Visual Basic language

- **variables & constants**
  - Boolean Integer Single Double String Const

    ```
    Dim s As String, i As Integer, d As Double
    ```
  - Byte Date Currency
  - Object Variant user-defined

- **arrays**
  - fixed size

    ```
    Dim ar(100) as Integer
    ```
  - dynamic

    ```
    Dim dyn() as Integer       ' declaration
    Redim dyn(10)              '  set size
    ```
  - reset size, preserve old contents

    ```
    Redim Preserve dyn(100)    ' like realloc
    ```

- **operators & expressions**

```
+   -    *    /    \   mod  ^
=   <>   >   >=   <   <=
And  Or  Not
```

Types, declarations, conversions

- **variables declared with Dim statement**

```
Dim i as Integer, s as Single,
  d as Double, str as String
```

  - Integer: 32 bits
  - Single, Double: approximately 6 or 15 digits with fractional part
    - 3.14159, 3.14159265358979323846
  - String: "any number of characters within quotes"
  - Object: object in same sense as Java or C++

- **VB usually infers types from context, does conversions automatically**
  - sometimes have to be explicit:
    - CInt(string) if can't tell from context that string is meant as a number
    - CStr(double) to produce a string value
  - **Variant** type holds any type

Control Flow

- **If Then Else**

```
If i >= 0 Then
    print i, " is positive"
ElseIf i = 0 Then
    print i, " is zero"
Else
    print i, " is negative
End If
```

- **For Next loop**

```
For i = 1 To 10
   print i, i * i, 2 ^ i
Next i
```

  can go forward or backward, any step size

- **Do While loop**

```
i = 1
Do While i <= 10
   print i, i * i, 2 ^ i
   i = i + 1
Loop
```

  test at top or bottom; use **While** or **Until**;
  early exit with **Exit Do**

Subroutines and functions

```
Sub ask (s As String)
  Dim stat As String
  stat = MsgBox("Another game?", vbYesNo)
   If stat = vbYes Then ...
   ...
End Sub

Function Randint(n As Integer) As Integer
    Randint = Int(n * Rnd) + 1
        // function name => return value
End Function
```

- **call by reference by default**
  - **ByVal** to specify call by value

- **Exit Sub and Exit Function for early exit**

Standard VB libraries

- **strings**
  - Len(s), Mid(s, p, n), InStr(target, pat), ...
  - s1 Like pat (shell-like pattern match)
- **math**
  - Sqr, Rnd, Sin, Cos, ...

- **I/O, etc.**

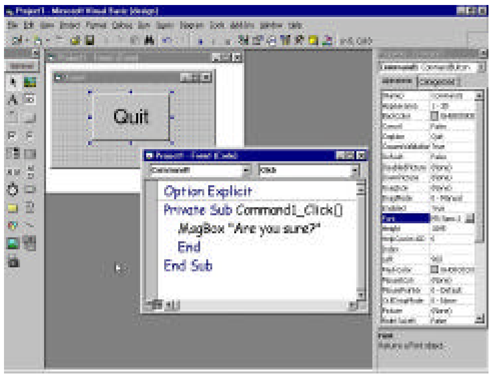
```
Open fin For Input As #1
Open fout For Output As #2
Do Until EOF(1)
   Line Input #1, textline
   Print #2, textline
Loop
Close #1
Close #2
```

- **run processes**

  Call Shell("command...", 1)

Controls: Interface components

- **buttons, sliders, labels, text boxes, …**
  - about 25 in basic set
  - instances normally created at design time
  - if in an array, new ones can be added and deleted at run time
  - menubar builder
  - dialog controls
- **each control has a fixed set of properties, events, and methods**
- **properties:**
  - size, position, color, caption, name, …) for what it is
  - set when drawn (usually) or when program is running
    by assignments or functions in your program
- **methods:**
  - the operations it will do, appropriate to what it is
- **events:**
  - external stimuli that it responds to
    mouse click, typing, scrolling, size change, window close
  - when an event occurs, VB calls the subroutine associated with it
    e.g., Button_Click(), TextBox_KeyPress(), etc.
  - what you write in the subroutine determines what the program does:
    you define what the behavior is



```
Option Explicit
Private Sub Command1_Click()
    MsgBox "Are you sure?"
    End
End Sub
```

Software re-use

- **how do we re-use code that others have written?**
  - "If I have seen further than others, it is because I have stood on the shoulders of giants."

- **source**
  - e.g., Open Source movement
- **libraries**
  - e.g., `-lsocket` on Unix,
    DLL's on Windows,
    Java packages
- **classes**
  - C++ Standard Template Library
  - Java Collection classes

- **objects**

- **components**

Libraries

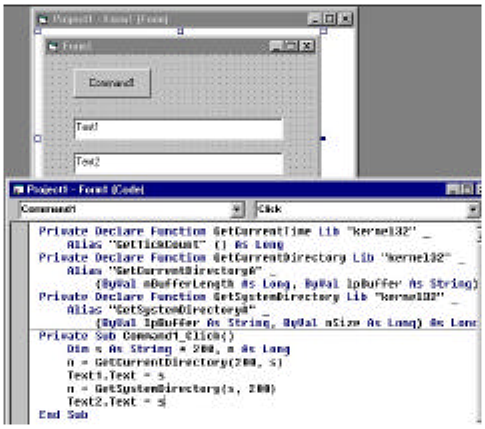
- **linking to previously compiled code**
- **static linking**
  - all called routines are included in executable
- **dynamic linking**
  - called routines located and linked in on demand
    - shared libraries on Unix
    - dynamic link libraries (DLL's) on Windows
- **lots of advantages**
  - no cost if a particular routine is not called
  - minor startup cost for initialization when called
  - minimal cost when running (extra indirection for call)
  - library code is shared among all simultaneous uses

- **DLL's very much used in Windows**

- **some disadvantages**
  - DLL hell: inconsistencies among versions,
    - especially after installation then uninstallation
  - a single-language solution, more or less
    - VB can call C/C++ DLL's
  - DLL runs in same address space
    - protection issues
    - not distributed

Extending VB by calling libraries

- **can call any DLL from the Windows API**



```
Private Declare Function GetCurrentTime Lib "kernel32" _
    Alias "GetTickCount" () As Long
Private Declare Function GetCurrentDirectory Lib "kernel32" _
    Alias "GetCurrentDirectoryA" _
        (ByVal nBufferLength As Long, ByVal lpBuffer As String)
Private Declare Function GetSystemDirectory Lib "kernel32" _
    Alias "GetSystemDirectoryA" _
        (ByVal lpBuffer As String, ByVal nSize As Long) As Long
Private Sub Command1_Click()
    Dim s As String * 200, n As Long
    n = GetCurrentDirectory(200, s)
    Text1.Text = s
    n = GetSystemDirectory(s, 200)
    Text2.Text = s
End Sub
```

- **can create and call your own DLL's**

COM: Microsoft's component object model

- **binary standard for creating & using components**
  - components can be written in any language
    - IDL (interface definition language) to describe arguments and return values, generate necessary code
  - components can be in same process,
    separate process on same machine,
    or on some other machine (DCOM)
    - DCOM transports include TCP/IP and HTTP
  - supporting libraries marshal arguments, call functions, retrieve results
    - all happens transparently to process that uses it
  - integral part of Microsoft systems
    - available on non-MS operating systems (sort of?)
- **COM components are objects with interfaces**
  - interface: functions that provides access to methods
    - based on C++ virtual function calls
    - implementable in any language
  - interface is also a contract between implementor and user about what the methods do
  - 128-bit ID's identify and guarantee uniqueness
    - stored in Windows registry so others can find it
- **COM has had several names, continues to evolve**
  - .NET is the next version / replacement

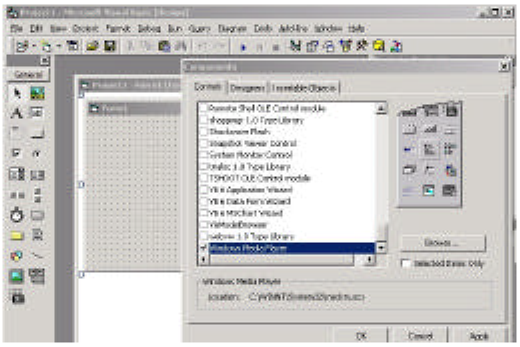
Using COM components in VB

- **a large industry creates 3rd-party controls**
  - much modern PC software is packaged as objects whose methods and properties can be accessed from VB and other programs
- **to add a component to a project**
  - Project / Components / Controls / Add MediaPlayer
- **examine its properties, methods, events**
  - View / Object browser / MediaPlayer
- **write code to use it**

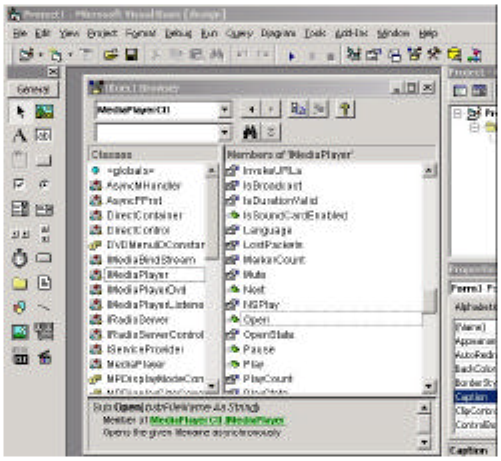
```
Private Sub Command1_Click()
    MediaPlayer1.Open (filename)
End Sub
```

- **you can make your own controls**
  - using VB, C++, etc.

Adding a component to a VB form:

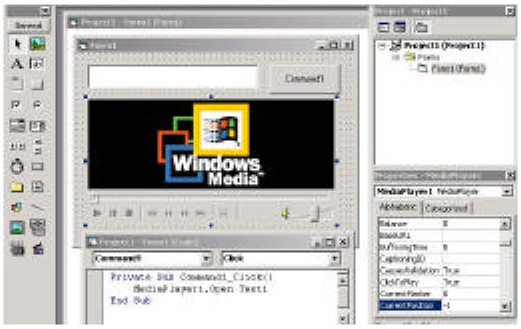


Check its methods & properties



Sub Open(bstrFileName As String)
Member of MediaPlayerCtl.IMediaPlayer
Opens the given filename asynchronously

After adding a media player control:



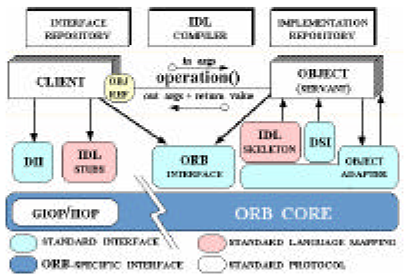
```
Private Sub Command1_Click()
    MediaPlayer1.Open Text1
End Sub
```

ActiveX

- **Microsoft's marketing name for technologies and services based on COM**
- **ActiveX components are COM objects**
  - executable code that packages an object as
    - .EXE (standalone executable)
    - .DLL (dynamic link library)
    - .OCX (VB-like control)
  - can run anywhere (client or server)
- **ActiveX controls**
  - COM components with user-interface aspects
  - written in C++, Java, VB, ...
  - can be used in web pages (analogous to applets)
  - can be controlled with VBScript, JScript and other scripting languages
- **ActiveX documents**
  - lets users view and edit non-HTML documents through the browser
  - integrates existing documents into browser or any other application

CORBA (Common Object Request Broker Architecture)

- **an alternate approach to the same problem**
  - industry consortium (OMG or Object Management Group)
- **client -server model, using objects**
- **object-request broker (ORB)**
  - communicates client requests to target objects
  - finds object implementation, activates it if necessary, delivers request, and returns response
- **IDL (interface definition language) and compiler for specifying and implementing interfaces**
  - names, arguments, return values



Java RMI and Java Beans

- **RMI (Remote Method Invocation)**
  - a remote procedure call mechanism
  - call objects located (usually) on other systems
  - very loosely equivalent to (D)COM
  - can pass objects, not just primitive types

- **Java Beans**
  - a marketing name for Java components
  - an API for writing component software in Java
  - components expose features (methods & events)
  - visual application builder tools determine properties by "introspection"
    - can query an object about its properties
  - loosely analogous to ActiveX components

- **attempting to solve many of the same problems as COM and CORBA, but entirely within Java**
  - access to non-Java code through JNI (Java Native Interface)

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

```
// insert your code here
```

- **use it in VB applications**
  - add reference to Hashcom object:

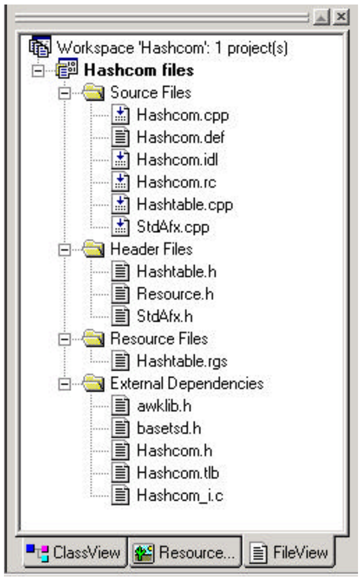
```
Dim h as Object
Set h = New Hashtable
h.put name, val
s = h.get(name)
if h.member(s) then …
```

- **use this in Excel, scripts, etc.**

Existing hash table code

```
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++



Workspace 'Hashcom': 1 project(s)
- **Hashcom files**
  - Source Files
    - Hashcom.cpp
    - Hashcom.def
    - Hashcom.idl
    - Hashcom.rc
    - Hashtable.cpp
    - StdAfx.cpp
  - Header Files
    - Hashtable.h
    - Resource.h
    - StdAfx.h
  - Resource Files
    - Hashtable.rgs
  - External Dependencies
    - awklib.h
    - basetsd.h
    - Hashcom.h
    - Hashcom.tlb
    - Hashcom_i.c

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

```
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);
  ...
 };
```

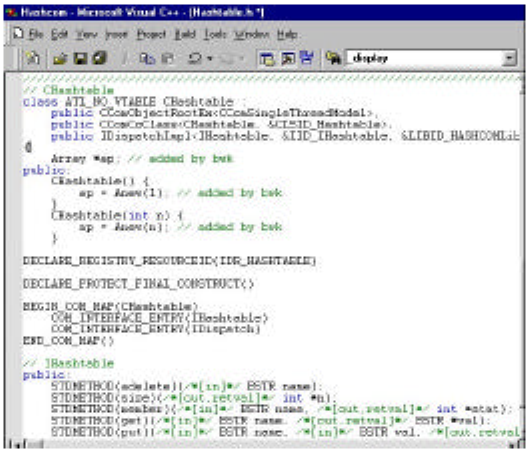
- **IDL compiler converts specification into function templates and code to marshal arguments for function calls**

C++ generated by MIDL

```
MIDL_INTERFACE("24942DFC-6E32-48A0-AF77-
                              C0C009EEC328")
IHashtable : 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

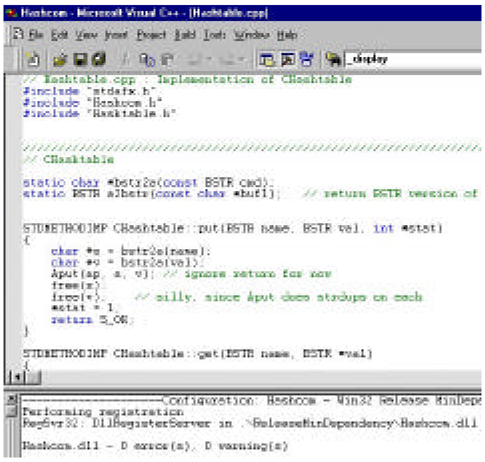
Interface specification (IDL)



Hashcom - Microsoft Visual C++ - [Hashtable.h *]
File Edit View Insert Project Build Tools Window Help
display

```
/////////////////////////////////////////////////////////////////////////////
// CHashtable
class ATL_NO_VTABLE CHashtable :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<CHashtable, &CLSID_Hashtable>,
    public IDispatchImpl<IHashtable, &IID_IHashtable, &LIBID_HASHCOMLib
{
    Array *ap; // added by bwk
public:
    CHashtable() {
        ap = Anew(1); // added by bwk
    }
    CHashtable(int n) {
        ap = Anew(n); // added by bwk
    }

DECLARE_REGISTRY_RESOURCEID(IDR_HASHTABLE)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CHashtable)
    COM_INTERFACE_ENTRY(IHashtable)
    COM_INTERFACE_ENTRY(IDispatch)
END_COM_MAP()

// IHashtable
public:
    STDMETHOD(adelete)(/*[in]*/ BSTR name);
    STDMETHOD(size)(/*[out,retval]*/ int *n);
    STDMETHOD(member)(/*[in]*/ BSTR name, /*[out,retval]*/ int *stat);
    STDMETHOD(get)(/*[in]*/ BSTR name, /*[out,retval]*/ BSTR *val);
    STDMETHOD(put)(/*[in]*/ BSTR name, /*[in]*/ BSTR val, /*[out,retval
```

Add semantics to framework


Hashcom - Microsoft Visual C++ - [Hashtable.cpp]
File Edit View Insert Project Build Tools Window Help
_display

```
// Hashtable.cpp : Implementation of CHashtable
#include "stdafx.h"
#include "Hashcom.h"
#include "Hashtable.h"


/////////////////////////////////////////////////////////////////////////////
// CHashtable

static char *bstr2a(const BSTR cmd);
static BSTR a2bstr(const char *buf1);    // return BSTR version of

STDMETHODIMP CHashtable::put(BSTR name, BSTR val, int *stat)
{
    char *n = bstr2a(name);
    char *v = bstr2a(val);
    Aput(ap, n, v); // ignore return for now
    free(n);
    free(v);      // silly, since Aput does strdups on each
    *stat = 1;
    return S_OK;
}

STDMETHODIMP CHashtable::get(BSTR name, BSTR *val)
{
```

```
--------------------Configuration: Hashcom - Win32 Release MinDepe
Performing registration
RegSvr32: DllRegisterServer in .\ReleaseMinDependency\Hashcom.dll

Hashcom.dll - 0 error(s), 0 warning(s)
```

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)**

```
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);
  for (i = 0; i < n; i++)     // wide to narrow
        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
    - registered during install or when created or by explicit call
  - do any initialization
    - Windows needs to keep track of what DLLs are in use
  - link it into current program (if a DLL)
    - fill in calls with pointer to real code: vtbl

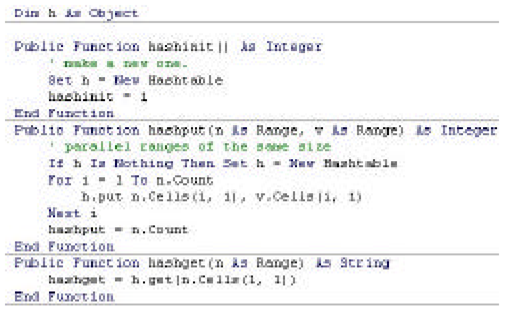
- **each subsequent method call**
  - collect arguments into proper form ("marshalling")
  - call function
  - convert return value and output arguments into proper form

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

DLL startup code excerpt (machine generated)

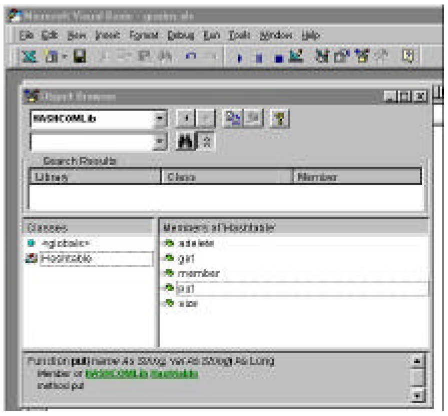
```
// DLL Entry Point

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



```
Dim h As Object

Public Function hashinit() As Integer
    ' make a new one.
    Set h = New Hashtable
    hashinit = 1
End Function
Public Function hashput(n As Range, v As Range) As Integer
    ' parallel ranges of the same size
    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
```

Automatically generated usage info



Function **put**(name As String, val As String) As Long
Member of HASHCOMLib.Hashtable
method put

Scripting

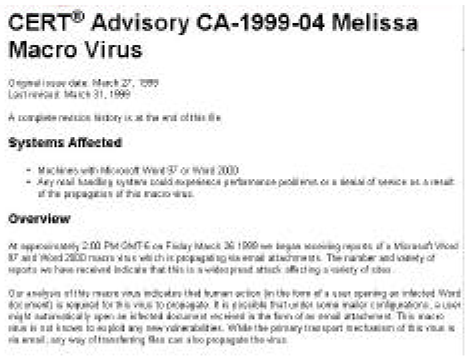
- **every component exposes what it can do as an object interface: methods, properties**
- **can control every object from a programming language that can access objects**
- **VBScript is a scripting version of VB for controlling scriptable objects**
  - can use it to control scriptable programs
- **Visual Basic for Applications (VBA) is a version of VB that lives inside some programs**
  - notably Word, Excel, Powerpoint, etc.
  - can use it to control them and other scriptable programs

VBScript example

```
Dim xl
Set xl = WScript.CreateObject("Excel.Application")

xl.Visible = TRUE
xl.WorkBooks.Add

xl.Columns(1).ColumnWidth = 20
xl.Columns(2).ColumnWidth = 30
xl.Columns(3).ColumnWidth = 40

xl.Cells(1, 1).Value = "Property Name"
xl.Cells(1, 2).Value = "Value"
xl.Cells(1, 3).Value = "Description"

xl.Range("A1:C1").Select
xl.Selection.Font.Bold = True
xl.Selection.Interior.ColorIndex = 1
xl.Selection.Interior.Pattern = 1 'xlSolid
xl.Selection.Font.ColorIndex = 2

xl.Columns("B:B").Select
xl.Selection.HorizontalAlignment = &hFFFFEFDD

Dim idx
idx = 2

Sub Show(strName, strValue, strDesc)
    xl.Cells(idx, 1).Value = strName
    xl.Cells(idx, 2).Value = strValue
    xl.Cells(idx, 3).Value = strDesc
    idx = idx + 1
    xl.Cells(idx, 1).Select
End Sub
```

VBScript example, page 2

```
Call Show("Name", WScript.Name,
     "Application Friendly Name")
Call Show("Version", WScript.Version,
     "Application Version")
Call Show("FullName", WScript.FullName,
     "Application Context: Fully Qualified Name")
Call Show("Path", WScript.Path,
     "Application Context: Path Only")
Call Show("Interactive", WScript.Interactive,
     "State of Interactive Mode")


Dim args
Set args = WScript.Arguments
Call Show("Arguments.Count", args.Count,
     "Number of command line arguments")

For i = 0 to args.Count - 1
    xl.Cells(idx, 1).Value = "Arguments(" & i & ")"
    xl.Cells(idx, 2).Value = args(i)
    idx = idx + 1
    xl.Cells(idx, 1).Select
Next

Call xl.Workbooks.Open("c:\temp\grades.xls")
```



CERT® Advisory CA-1999-04 Melissa Macro Virus

Original issue date: March 27, 1999
Last revised: March 31, 1999

A complete revision history is at the end of this file.

Systems Affected

- Machines with Microsoft Word 97 or Word 2000
- Any mail handling system could experience performance problems or a denial of service as a result of the propagation of this macro virus.

Overview

At approximately 2:00 PM GMT-6 on Friday March 26 1999 we began receiving reports of a Microsoft Word 97 and Word 2000 macro virus which is propagating via email attachments. The number and variety of reports we have received indicate that this is a widespread attack affecting a variety of sites.

Our analysis of this macro virus indicates that human action (in the form of a user opening an infected Word document) is required for this virus to propagate. It is possible that under some mailer configurations, a user might automatically open an infected document received in the form of an email attachment. This macro virus is not known to exploit any new vulnerabilities. While the primary transport mechanism of this virus is via email, any way of transferring files can also propagate the virus.

Security issues

- **VB embedding and scripting is a mixed blessing**
  - lots of nice useful properties
    - can easily extend capabilities
    - customize behaviors
  - lots of not so nice properties
    - viruses are very easy

- **scripts and plug-ins and applets let someone else run their code on your machine**
- **how can this be made safe (enough)?**

- **code-signing (Microsoft's "Authenticode")**
  - uses cryptographic techniques to assure that code comes from who it says it does
  - and that it hasn't been tampered with
  - but NOT that it works properly
    - doesn't protect against bugs, invasion of privacy, ...

- **sandboxing (Java model)**
  - isolate code inside virtual machine or similar
  - limits capabilities (e.g., Java applets)
  - doesn't protect against bugs in programs
  - or bugs in the security model and implementation

- **perfect security is not possible**