



.NET Architecture

Albrecht Wöß

Table of Contents

- new terms, features, general considerations, ...
- assemblies, modules, libraries, applications
- multi-language interoperability & cross-language debugging
- Virtual Execution System (VES)
 - dynamic loading
- .NET security
 - Code Access Security (CAS)
 - configuration
- Appendix:
 - important directories and configuration files
 - source code and IL disassembler view of example program

CLI - Common Language Infrastructure



- CLI defines the basic platform for the .NET architecture
- described by ECMA Standard 335
<http://www.ecma-international.org/publications/standards/ecma-335.htm>
- Microsoft's .NET Framework
 - contains Common Language Runtime (CLR)
= Microsoft's implementation of the CLI standard
 - completely fulfill the standard
 - and offers still more, e.g.
 - ASP.NET
 - ADO.NET
 - Web Services
 - extended class library
 - ...



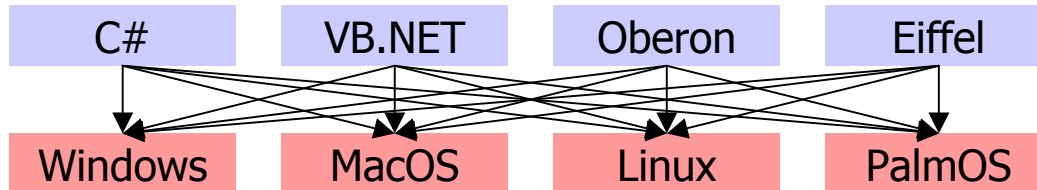
CLI Features

(buzzwords and acronyms ☺)

- common intermediate language ⇒ **CIL**
- common type system ⇒ **CTS**
- special deployment unit ⇒ **Assembly**
- extensible type information ⇒ **Metadaten**
- integration of many programming languages ⇒ **CLS**
- code-based security model ⇒ **CAS**
- automatic memory management ⇒ **GC**
- just-in-time compilation (*NEVER* interpretation!) ⇒ **JIT**
- dynamic type loading ⇒ **VES**

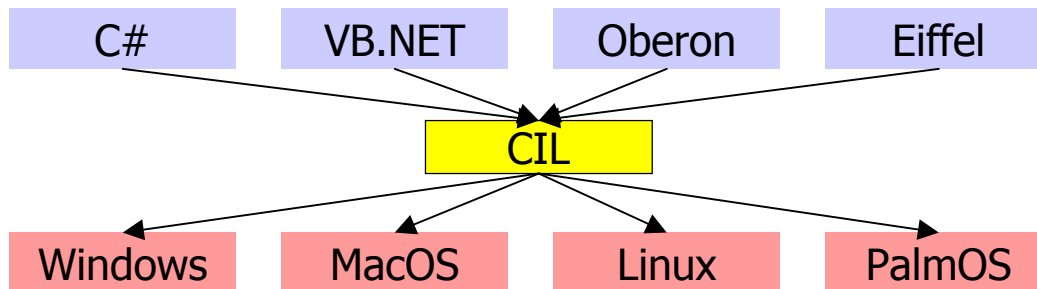
Advantages of a Virtual Machine

- portability (platform and language independence)
 - w/o VM: compilers for each language on each platform



e.g. $4 \times 4 = \underline{16}$

- w/ VM: translation into intermediate language (with .NET: CIL)
 - one compiler per language and
 - one CLR (JIT compiler) per platform



e.g. $4 + 4 = \underline{8}$

- compactness of intermediate code
- better optimizations for native code at JIT compile time

Comparing VMs: CLR vs. JVM

- What's in the CLR that is *not* in the JVM?
 - objects on the stack (records, unions), custom value types
 - reference parameters (CALL-BY-REFERENCE)
 - varargs
 - function pointers
 - block matrices
 - overflow checking
 - tail calls
 - "native size" data types
 - boxing of value types
 - ...

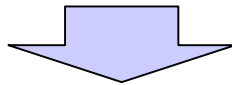
➔ CLR better suited for use with numerous different programming languages

The [☺] Example: Hello, .NET-World!



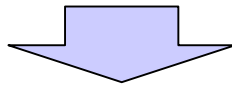
HelloWorld.cs:

```
class HelloWorldApp {  
    static void Main () {  
        System.Console.WriteLine("Hello, .NET-World!");  
    }  
}
```

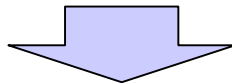


Compile and create assembly (w/ C# compiler):

```
> csc HelloWorld.cs
```

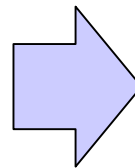


Assembly: HelloWorld.exe (3072 Byte!)



*View metadata and CIL code
(w/ IL disassembler):*

```
> ildasm HelloWorld.exe
```



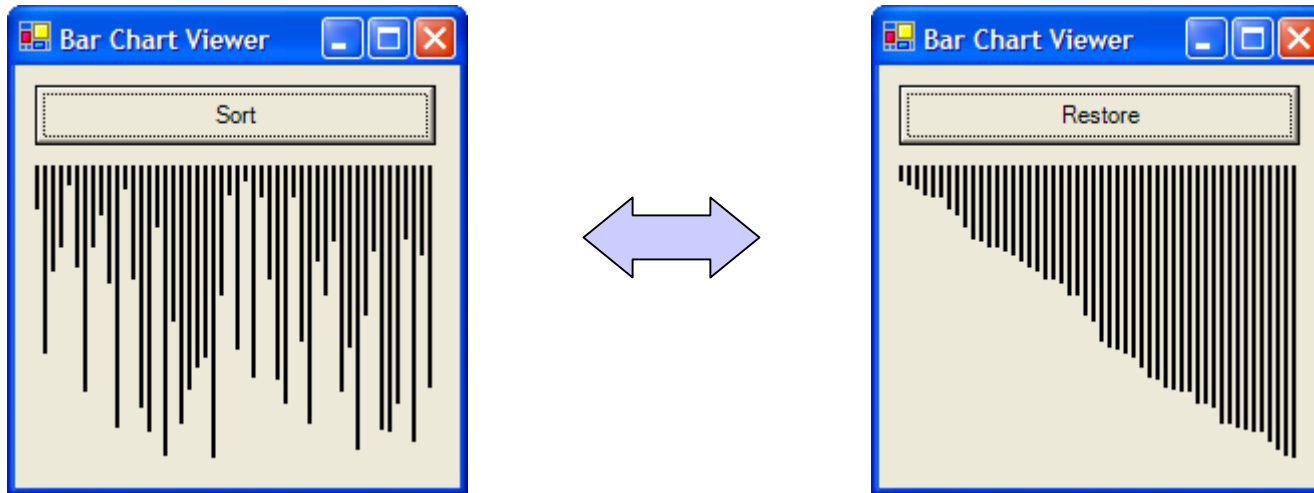
Assemblies and Modules

- Managed module =
 - **physical unit**: 1 .NET module = 1 .NET PE-file
 - contains type definitions with Metadata & CIL-code
 - is generated by a .NET compiler
- Assembly =
 - **logical unit** for:
deployment, encapsulation, versioning, security
 - **component** (in terms of component-oriented software development)
 - combines modules and resource files (see \Rightarrow **Manifest**)
 - is generated by a .NET compiler or the assembly linker (al.exe)
- Manifest
 - holds information about all the parts of an assembly
e.g. which files it consists of, where to find these files, which types
get exported outside the assembly, ...

Multi-Language Interoperability



- Demo: Visualization of a bar chart
 - button click toggles between unsorted and sorted view



- Implementation with different programming languages
 - C# ⇒ Windows Control (Chart)
 - Visual Basic .NET ⇒ Windows Form (connecting Chart ↔ Button)
 - J# ⇒ Application (set window size, Main method)

Modules, Libraries, and Applications



- .NET compiler can produce each of the following:
 - modules (*.netmodule): not an assembly, not executable
 - dynamic link libraries (*.dll): not executable assembly
 - application (*.exe): executable assembly

BarChartControl.cs is compiled into a *.NET module*:

```
csc /out:bar.mod /target:module /debug BarChartControl.cs
```

BarChartGUI.vb is compiled into a *.NET library* and uses bar.mod:

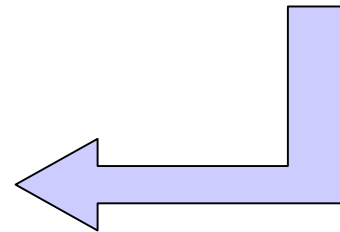
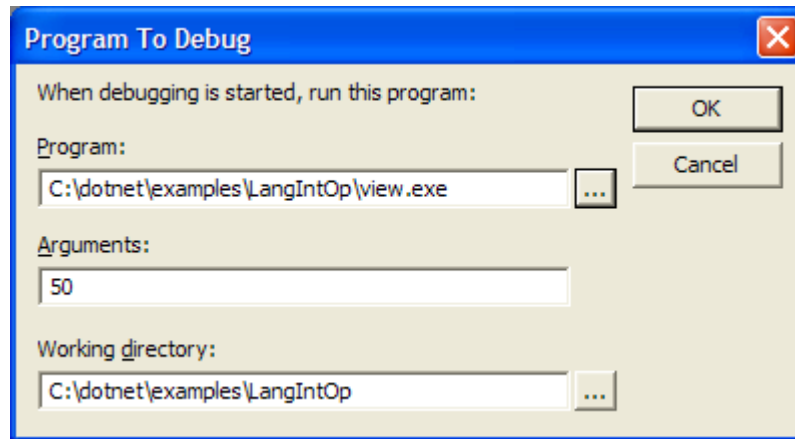
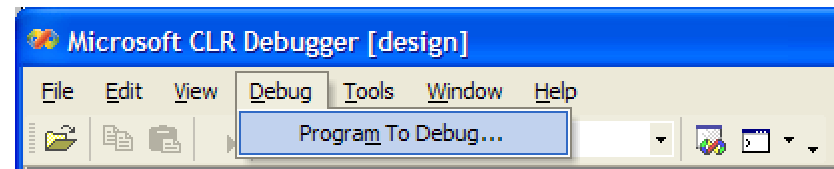
```
vbc /out:gui.dll /target:library /addmodule:bar.mod  
/r:System.dll,System.Drawing.dll,System.Windows.Forms.dll /debug  
BarChartGui.vb
```

BarChartApp.jsl is compiled into a *.NET Windows application* and uses gui.dll:

```
vjc /out:view.exe /target:winexe /reference:gui.dll  
/r:System.Windows.Forms.dll /debug BarChartApp.jsl
```

GUI-Debugger (1)


- start GUI-Debugger
 - `.NET-SDK*\GuiDebug\DbgCLR.exe`
- select program to debug

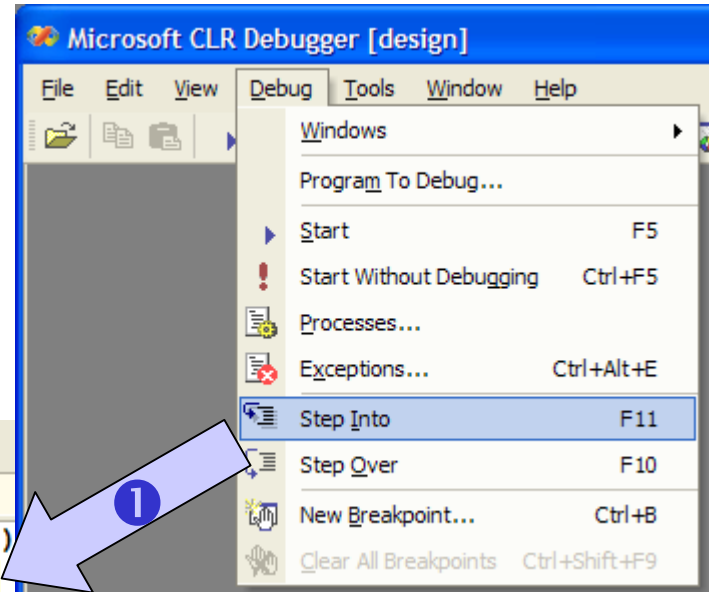
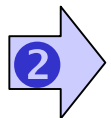


*) see Appendix A: Important Directories

GUI-Debugger (2)



- 1 start debugging
 - Step Into (F11)
- 2 set breakpoint
 - click into left border
- 3 run to breakpoint 



```
BarChartApp.jsl  
  
public static void main (String[] args)  
    BarChartApp app = new BarChartApp ();  
  
    int nVals = 8;  
    if (args.length > 0) nVals = Convert.ToInt32(args[0]);  
  
    int[] values = new int[nVals];  
    Random gen = new Random();  
    for (int i = 0; i < nVals; i++)  
        values[i] = gen.Next(app.Chart.get_Height());  
    app.Chart.set_Values(values);  
  
    System.Windows.Forms.Application.Run(app);  
}
```

Cross-Language Debugging



The screenshot shows two windows in Visual Studio. The top window, titled 'BarChartApp.jsl', contains JavaScript code. The line `app.Chart.set_Values(values);` is highlighted in yellow. A red arrow points to this line. The bottom window, titled 'BarChartControl.cs', contains C# code. The line `values = original = value;` is highlighted in yellow. A red arrow points to this line. A blue arrow labeled 'Step Into' points from the JavaScript code to the C# code.

```
public static void main (String[] args) {
    BarChartApp app = new BarChartApp();

    int nVals = 8;
    if (args.length > 0) nVals = Convert.ToInt32(args[0]);

    int[] values = new int[nVals];
    Random gen = new Random();
    for (int i = 0; i < nVals; i++)
        values[i] = gen.Next(app.Chart.get_Height());
    app.Chart.set_Values(values);
}

System.Windows.Forms.Application.Run(app);
}
```

```
public class BarChartControl : UserControl {
    int[] values = {}; // points to the list to be displayed
    int[] original = {}; // original (unsorted) list of the values
    int[] sorted = {}; // sorted representation of the original values

    public int[] Values {
        set {
            values = original = value;
            sorted = (int[]) original.Clone(); Array.Sort(sorted);
            Refresh();
        }
    }
}
```



CIL and Metadata - Exported Types

IL-Assembler-Code of main method of view.exe:

```
.method public hidebysig static void main (string[] args) cil managed {  
  .entrypoint  
  .locals init ( class BarChartApp app, int32 nVals, int32[] values, . . . )  
  . . .  
  ldloc.0 | local variable 0 | local variable 2  
  ldfld class [gui]BarChartControl [gui]BarChartGui::Chart  
  ldloc.2  
  callvirt instance void [gui]BarChartControl::set_Values(int32[])  
  . . .  
}
```

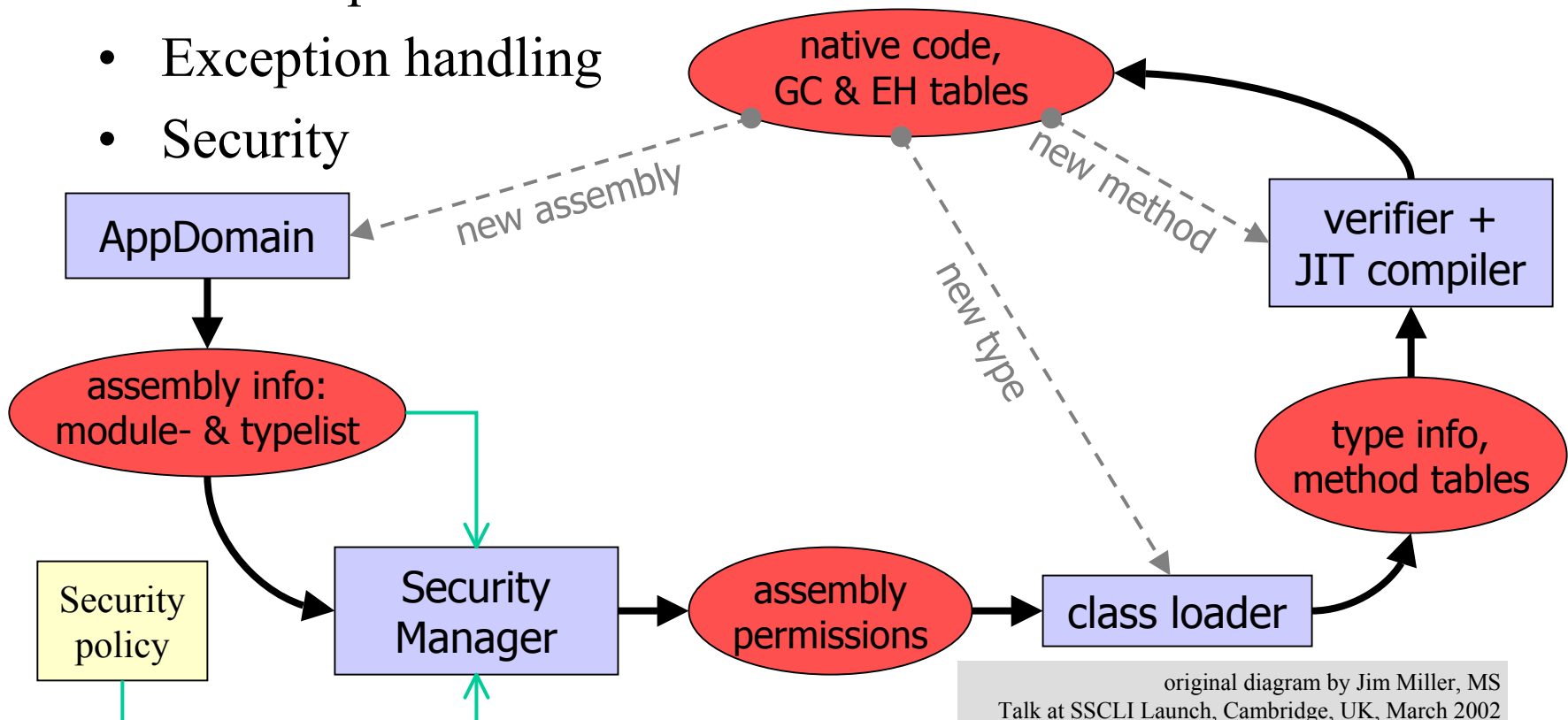
Manifest of gui.dll:

```
.assembly gui { . . . }  
.module gui.dll  
.module extern bar.mod  
.file bar.mod  
  .hash = ...  
.class extern public BarChartControl { .file bar.mod  
  .class 0x02000002 }
```

Virtual Execution System (VES)



- Type safety
- Memory management (memory layout of objects, GC, ...)
- JIT compilation
- Exception handling
- Security

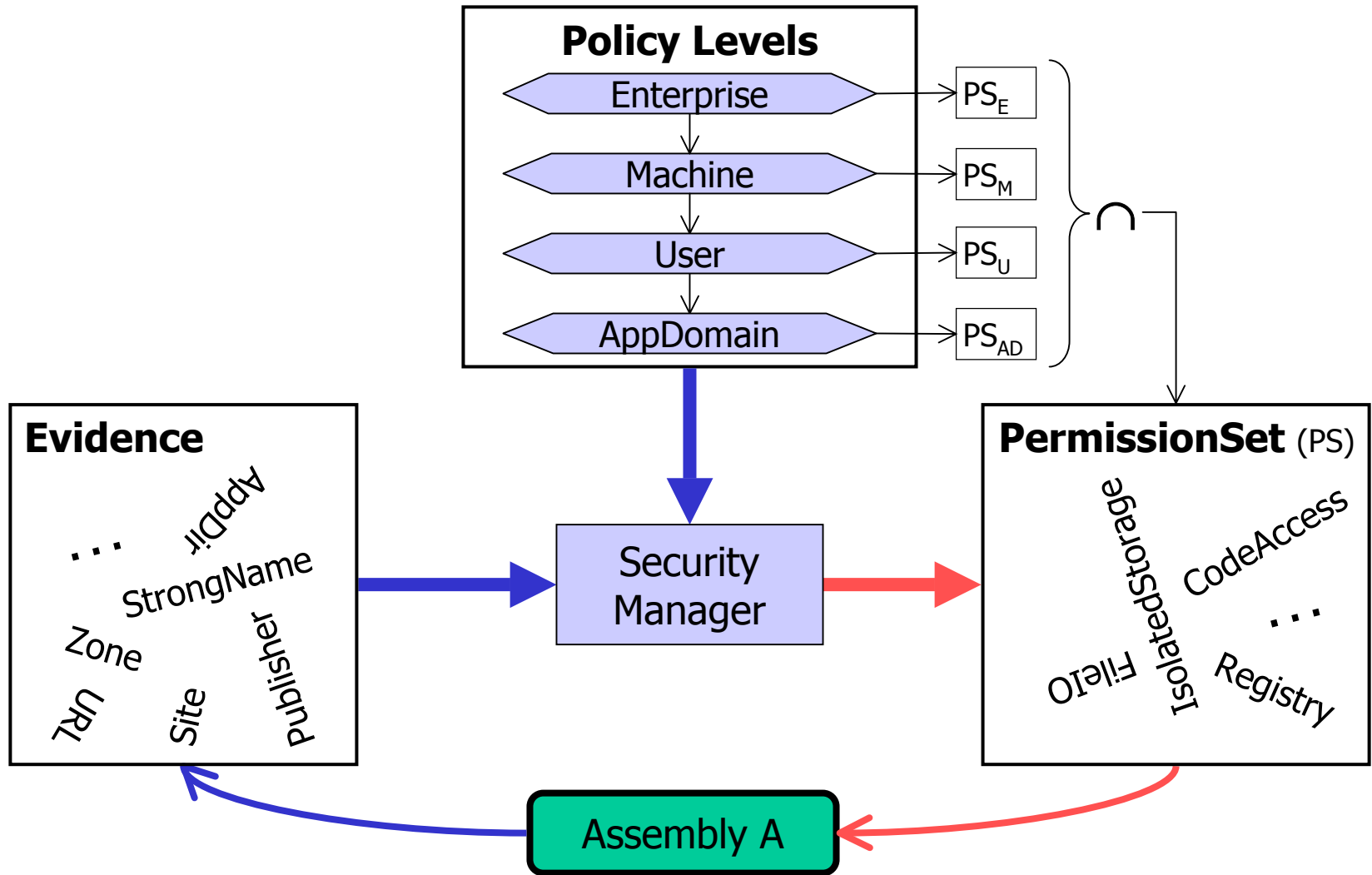


.NET Security



- role-based security
 - exists on most operation systems
 - permissions are assigned to a *user*
 - based on *group affiliation*
 - namespace System.Security.Principal has supporting interfaces
 - System.Security.Principal.IPrincipal → role, group
 - System.Security.Principal.IIdentity → user
- code-based security (*Code Access Security (CAS)*)
 - permissions are assigned to *assemblies*
 - based on *assembly information (evidence)* and *security policy*

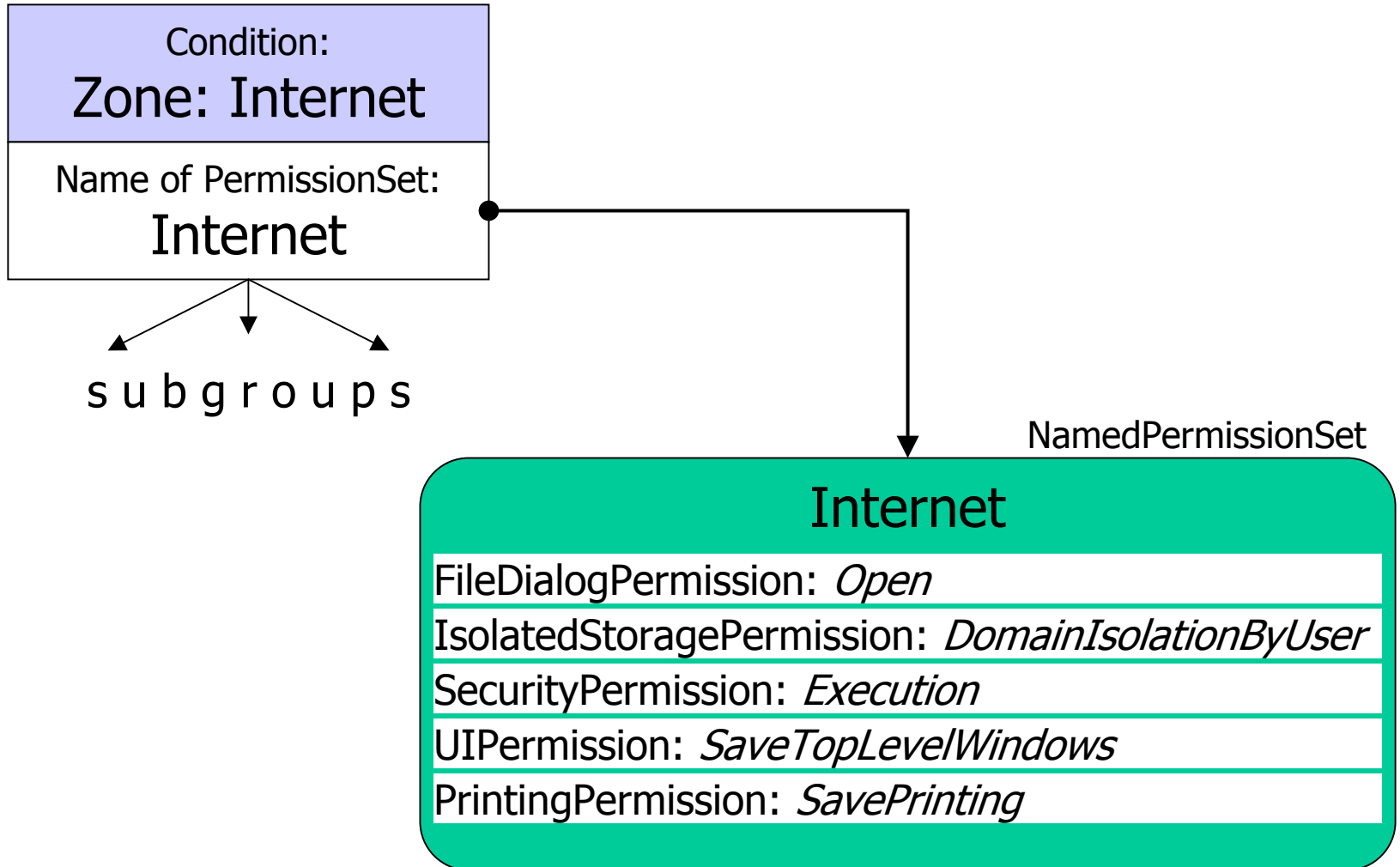
Code Access Security (CAS)



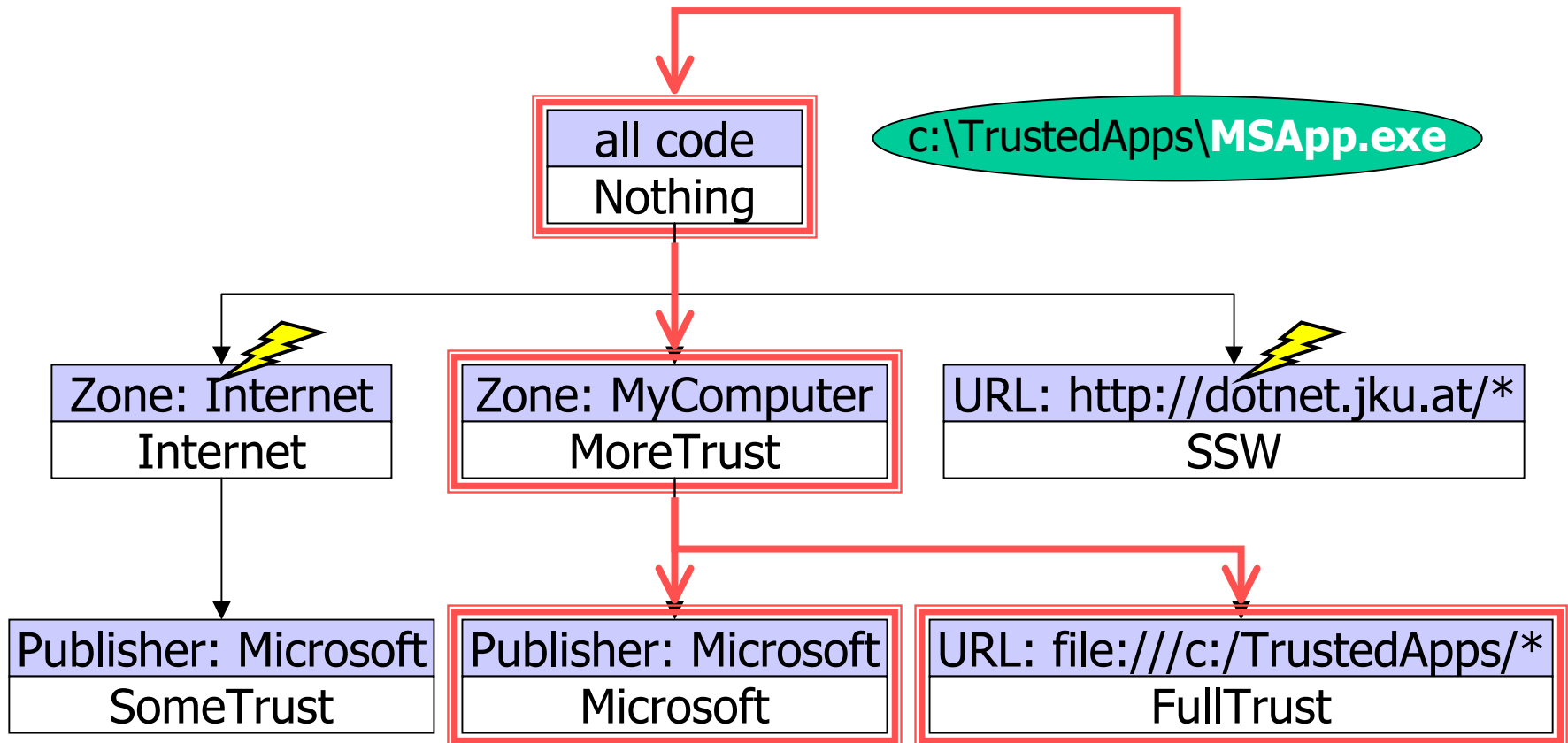
Code Groups & Permission Sets



CodeGroup

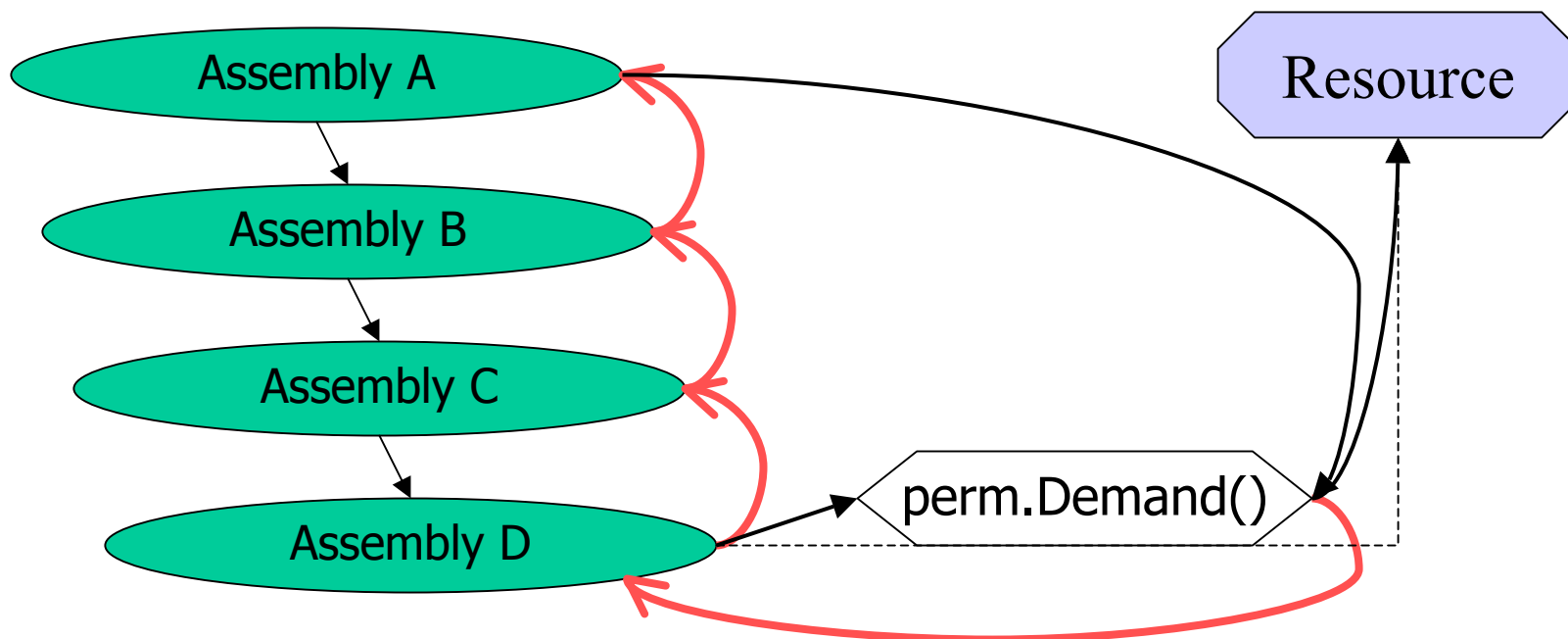


Levels of Security Policy



➔ Nothing ∪ MoreTrust ∪ Microsoft ∪ FullTrust

Security Stack Walk



- CLR checks permissions of all callers
- only if all callers have the required permissions perm, the resource may be accessed

Security Policy Configuration (1)



- Microsoft .NET Framework Configuration Tool
 - MMC (Microsoft Management Console) Snap-In
 - Start with
 - Start | Control Panel | Administrative Tools | Microsoft .NET Framework Configuration or
 - mscorcfg.msc at the command line
 - supports (with wizards)
 - creating/editing of code groups
 - creating/editing of of permission sets
 - determines permissions/code groups of an assemblies
- Code Access Security Policy Tool (caspol.exe)
 - command line tool
 - e.g. activate/deactivate security checks
caspol **-security** (on | off)

Security Policy Configuration (2)



.NET Framework Configuration

Datei Aktion Ansicht ?

My Computer

- Assembly Cache
- Configured Assemblies
- Remoting Services
- Runtime Security Policy**
 - Enterprise
 - Machine
 - Code Groups
 - All_Code
 - My_Computer_Zone
 - Microsoft_Strong_Name
 - ECMA_Strong_Name
 - WebKalendarApp
 - MyVSProjects
 - LocalIntranet_Zone
 - Internet_Zone
 - Restricted_Zone
 - Trusted_Zone
 - Permission Sets
 - FullTrust
 - SkipVerification
 - Execution
 - Nothing
 - LocalIntranet
 - Internet
 - Everything
 - SomeTrust
 - MoreTrust
 - SSW
 - Policy Assemblies
- User
- Applications

Code Access Security Policy

The common language runtime's code access security system determines an assembly's permissions to access protected resources. Each permission set granted to an assembly is based on the assembly's evidence (such as its URL or publisher certificate), which in turn is based on configurable security policy.

To learn more about the code access security model, refer to the Microsoft .NET Framework SDK documentation.

The wizards and task links below will help you set and distribute security policy. For complete control of security policy, use the tree views under this node.

Tasks

[Increase Assembly Trust](#)

Use the Trust an Assembly wizard to increase the level of trust granted to a particular assembly. This wizard modifies security policy based on information about the evidence of the assembly selected.

[Adjust Zone Security](#)

Use the Security Adjustment Wizard to modify the level of trust granted to all assemblies coming from a particular zone, such as Internet, Local Intranet, or My Computer.

[Evaluate Assembly](#)

Use the Evaluate an Assembly wizard to evaluate what permissions or code groups apply to a particular assembly. This is useful in determining the effect of current security policy on actual assemblies.

[Create Deployment Package](#)

Use the Deployment Package Wizard to create a policy deployment package. The wizard wraps a security policy level into a Windows Installer Package (.msi file) that can then be distributed using Group Policy or Microsoft Systems Management Server.

[Reset All Policy Levels](#)

Use this task to reset your security policy to the default settings. Note that this will obliterate all modifications that might have been made to default security policy.

Appendix A: Important Directories



- Windows-specific directories
 - *Windows*: e.g. c:\WINDOWS\ , c:\WINNT\
 - *Programs*: e.g. c:\Program Files\
 - *User*: e.g. c:\Documents and Settings*UserName*\
- .NET-specific directories
 - *.NET-Runtime*:
e.g. *Windows*\Microsoft.NET\Framework\v1.0.3705
 - *.NET-SDK*:
e.g. *Programs*\Microsoft.NET\FrameworkSDK or
Programs\Microsoft Visual Studio .NET\FrameworkSDK
 - *GAC* (Global Assembly Cache): e.g. *Windows*\assembly\

Appendix B: Configuration Files

- Machine configuration
 - *.NET-Runtime*\CONFIG\machine.config
- Application configuration
 - in the application directory
 - filename = application filename + .config
e.g. MyApp.exe → MyApp.exe.config
- Security policy
 - level ENTERPRISE: *.NET-Runtime*\CONFIG\enterprisesec.config
 - level MACHINE: *.NET-Runtime*\CONFIG\security.config
 - level USER: e.g.
User\Application Data\Microsoft\CLR Security Config\v1.0.03705\security.config
 - level APPDOMAIN:
in program code with `AppDomain.SetAppDomainPolicy (...)`

C#: BarChartControl (1)

BarChartControl.cs:

```
using System; using System.Drawing; using System.Windows.Forms;
```

```
public class BarChartControl : UserControl {  
    int[] values = {};           // points to currently displayed list (original or sorted)  
    int[] original = {};        // original (unsorted) list of the values  
    int[] sorted = {};          // sorted representation of the original list  
  
    public int[] Values {  
        set {  
            values = original = value;  
            sorted = (int[]) original.Clone(); Array.Sort(sorted);  
            Refresh();  
        }  
    }  
  
    public bool Sorted { get { return values == sorted; } }  
    ...  
}
```

C#: BarChartControl (2)

BarChartControl.cs (Forts.):

...

```

public void ToggleSorted () {
    if (values == sorted) values = original;
    else values = sorted;
    Refresh();
}

```

```

protected override void OnPaint (PaintEventArgs e) {
    Brush b = new SolidBrush(ForeColor); // fill color for bars
    int w = DisplayRectangle.Width / (values.Length*2-1); // width of bars
    int x = 0; // starting position
    foreach (int i in values) {
        e.Graphics.FillRectangle(b, x, 0, w, i);
        x += w * 2;
    }
}
}

```

Visual Basic .NET: BarChartGUI (1)

BarChartGUI.vb:

Imports System
Imports System.Drawing
Imports System.Windows.Forms

Public Class BarChartGui

Inherits Form

inherits from a class library type
 (unknown source language)

Private Dim WithEvents ToggleButton **As New** Button()

Protected Dim Chart **As New** BarChartControl()

type of field is
 implemented in C#

Public Sub New ()

ToggleButton.Location = New Point(10, 10)

ToggleButton.Size = New Size(200, 30)

ToggleButton.Text = "Sort"

Controls.Add(ToggleButton)

writing to a Property
 (unknown source language)

Chart.Location = New Point(10, 50)

Chart.Size = New Size(200, 150)

Controls.Add(Chart)

writing to a Property
 (source language C#)

End Sub

Visual Basic .NET: BarChartGUI (2)

BarChartGUI.vb (Forts.):

...

```
Private Sub Toggle_Click (ByVal sender As Object, ByVal e As EventArgs) _
    Handles ToggleButton.Click
```

```
Chart.ToggleSorted()
```

invocation of method
(source language C#)

```
If Chart.Sorted Then
```

```
    ToggleButton.Text = "Restore"
```

```
Else
```

```
    ToggleButton.Text = "Sort"
```

```
End If
```

```
End Sub
```

reading of a Property
(source language C#)

```
End Class
```

J#: BarChartApp

BarChartApp.jsl:

```
import System.*;
```

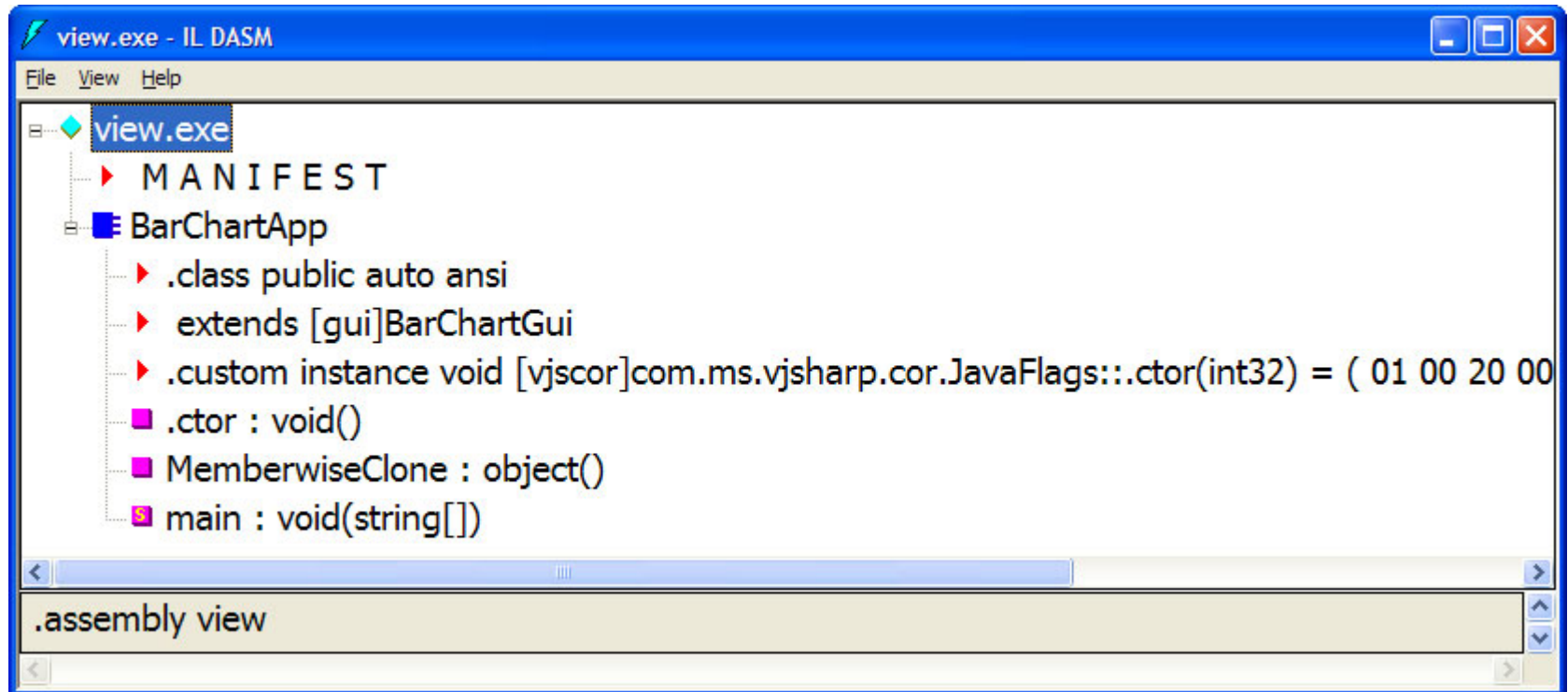
```
class BarChartApp extends BarChartGui {
  BarChartApp () {
    set_Text("Bar Chart Viewer");
    set_Width(230);
    set_Height(245);
  }
}
```

inherits from type implemented in C#

J# does not support C#-like Properties → use compiler-generated methods instead

```
public static void main (String[] args) {
  BarChartApp app = new BarChartApp();
  int nVals = 8;
  if (args.length > 0) nVals = Convert.ToInt32(args[0]);
  int[] values = new int[nVals];
  Random gen = new Random();
  for (int i = 0; i < nVals; i++) values[i] = gen.Next(app.Chart.get_Height());
  app.Chart.set_Values(values);
  System.Windows.Forms.Application.Run(app);
}
}
```

IL-Disassembler: view.exe



IL-Disassembler: gui.dll



```
gui.dll - IL DASM
File View Help
gui.dll
├── MANIFEST
├── BarChartGui
│   ├── .class public auto ansi
│   ├── extends [System.Windows.Forms]System.Windows.Forms.Form
│   ├── Chart : family class [.module bar.mod]BarChartControl
│   ├── _ToggleButton : private class [System.Windows.Forms]System.Windows.Forms.Button
│   ├── .ctor : void()
│   ├── Toggle_Click : void(object,class [mscorlib]System.EventArgs)
│   ├── get_ToggleButton : class [System.Windows.Forms]System.Windows.Forms.Button()
│   ├── set_ToggleButton : void(class [System.Windows.Forms]System.Windows.Forms.Button)
│   └── ToggleButton : class [System.Windows.Forms]System.Windows.Forms.Button()
└── .assembly gui
```

IL-Disassembler: bar.mod



The screenshot shows the IL DASM (Intermediate Language Disassembler) window for a file named 'bar.mod'. The window has a blue title bar and a menu bar with 'File', 'View', and 'Help'. The main area displays a tree view of the assembly's metadata. The tree is expanded to show the 'BarChartControl' class. The class is marked as 'public auto ansi beforefieldinit' and inherits from 'System.Windows.Forms.UserControl'. It contains three private fields: 'original', 'sorted', and 'values', all of type 'int32[]'. The class also has several methods: a constructor '.ctor : void()', 'OnPaint : void(class [System.Windows.Forms]System.Windows.Forms.PaintEventArgs)', 'ToggleSorted : void()', 'get_Sorted : bool()', and 'set_Values : void(int32[])'. Finally, there are two instance properties: 'Sorted : instance bool()' and 'Values : instance int32[]()'. The bottom of the window has a scroll bar.

```
bar.mod - IL DASM
File View Help
├─ bar.mod
│  └─ MANIFEST
│     └─ BarChartControl
│        ├── .class public auto ansi beforefieldinit
│        ├── extends [System.Windows.Forms]System.Windows.Forms.UserControl
│        ├── original : private int32[]
│        ├── sorted : private int32[]
│        ├── values : private int32[]
│        ├── .ctor : void()
│        ├── OnPaint : void(class [System.Windows.Forms]System.Windows.Forms.PaintEventArgs)
│        ├── ToggleSorted : void()
│        ├── get_Sorted : bool()
│        ├── set_Values : void(int32[])
│        ├── Sorted : instance bool()
│        └── Values : instance int32[]()
```