Module 20

"Diagnostics and Managing Assemblies"

TEKNOLOGISK INSTITUT

Agenda

TEKNOLOGISK
INSTITUT

- Logging and Tracing
- Performance Counters
- Managing Assemblies

Instrumentation

- The addition of diagnostic code into applications is called "instrumenting" the code
  - Debug information
  - Trace information
  - Event logging
  - Audit logging
  - Performance counters
  - …

- A good instrumentation of your code is essential to monitoring and maintaining the system

Windows Event Logs

TEKNOLOGISK
INSTITUT

- Event logs are used in Windows to collect log information by means of the operating system
  - Event logs are always present
    - For all applications and services
    - All major OS service use the event logs
    - Log info can be collected by e.g. IT departments

- Built-in event logs include e.g.
  - `System`
  - `Security`
  - `Application`

- Event Viewer in Windows
  - Allows viewing and maintaining event logs and sources

EventLog Class

- The **`EventLog`** Class
  - An event source must first be created before events can be logged in existing event log

- Methods
  - `CreateEventSource()` static
  - `WriteEntry()`

- Properties
  - `Log`
  - `LogDisplayName`
  - `MachineName`
  - `Source`
  - `Entries`

Creating Custom Event Logs

- `CreateEventSource()` can also create custom event logs
  - If log with supplied name does not exist
- Permissions
  - Creating/deleting event logs requires administrative privileges
    - Usually done during install/uninstall
  - Any users with `EventLogPermission` can read/write to logs
- Important difference between `EventLog` and `EventSource`

Compiler Directives

- Build Configurations
  - Debug
  - Release

```
#if DEBUG
         Console.WriteLine( "DEBUG mode...");
#else
         Console.WriteLine( "RELEASE mode...");
#endif
```

- Preprocessor defines
  - `DEBUG`
  - `TRACE`
  - *`XXX`*

- `[Conditional]` attribute

Debug Class

- The **Debug** class emits information in Debug builds only!
  - As opposed to Release builds
- Static methods
  - `Write()`
  - `WriteIf()`
  - `WriteLine()`
  - `WriteLineIf()`
  - `Assert()`
    - Ensures that a condition is true
- Static properties
  - **`AutoFlush`**
  - **`Listeners`**

```
void Method1( int n )
{
   Debug.Assert( n == 87, "Illegal number" );

   Method2( "Look, Mom. I'm in Method1" );
}
```

```
void Method2( string s )
{
   Debug.WriteLine( "Method2() called!" );

   Console.WriteLine( s );
}
```

Debugger Class and Attributes

- The `Debugger` class provides access to the debugger (if present)

- Static methods
  - `Break()`
  - `Launch()`
  - `Log()`

- Static properties
  - `IsAttached`

- `Debugger` attributes
  - `[DebuggerHidden]`
  - `[DebuggerStepThrough]`

Trace

- The `Trace` class receives information to be emitted regardless of build type
  - Both Debug and Release builds
- Supports almost exactly the same set of methods and properties as the `Debug` class
- The `Trace.Listeners` property is important!

Trace Listeners

- Trace listeners receive the Debug and Trace information emitted
  - `Debug.Listeners` property
  - `Trace.Listeners` property

- Concrete **`TraceListener`** classes
  - `ConsoleTraceListener`
  - `TextWriterTraceListener`
  - `XmlWriterTraceListener`
  - `EventLogTraceListener`
  - `EventSchemaTraceListener`
  - `DelimitedListTraceListener`
  - **`DefaultTraceListener`**

- Can be manipulated in the application configuration files

Agenda

- Logging and Tracing
- **Performance Counters**
- Managing Assemblies

Introducing Performance Counters

- Very many operating system and server features are instrumented by performance counters
  - Inspected by
    - Resource Monitor (Windows XP)
    - Reliability and Performance Monitor (Windows Vista)
    - Performance Monitor (Windows 7 + 8)

- Performance counters are accessible programmatically via the `PerformanceCounter` class

Using Performance Counters

- Performance counter values can be accessed through the **`RawValue`** property

```
PerformanceCounter counter =
   new PerformanceCounter( "ProcessorPerformance",
                           "percentage",
                           "PPM_Processor_0" );
...
Console.WriteLine( counter.RawValue );
```

- Similarly, they can be updated via **`RawValue`** or thread-safe methods like
  - `Increment()`
  - `IncrementBy()`
  - `Decrement()`

Custom Performance Counters

- The **`PerformanceCounterCategory`** class facilitates the creation of custom performance counters

```
PerformanceCounterCategory category =
   PerformanceCounterCategory.Create(
      "Wincubate",
      "Counters for the 70-483 course",
      PerformanceCounterCategoryType.MultiInstance,
      "A instances",
      "Number of live A objects in the application" );
```

- Creation of sets of counters are eased by
  - `CounterCreationData`
  - `CounterCreationDataCollection`

Agenda

- Logging and Tracing
- Performance Counters
- Managing Assemblies

Assemblies

- An assembly is a collection of types and resources
- An assembly is a versioned deployable unit
- An assembly can contain:
  - IL code
  - Resources
  - Type metadata
  - Manifest

- Assemblies are versioned
  - *<major version>.<minor version>.<build number>.<revision>*

Global Assembly Cache

TEKNOLOGISK INSTITUT

- Global Assembly Cache a.k.a. "GAC"
  - Central location in Windows where shared assemblies are stored
    - `C:\Windows\Assembly` (but it is not a regular folder...)
  - Advantages are
    - Shared by all applications
    - Side-by-side installation
    - Improved loading time

- Assemblies must be **strong-named** to be added to the GAC
  - "`sn -k`" generates a key pair
  - Visual Studio 2012 can also generate key pair and sign

- Install to GAC with **`gacutil.exe`**

```
gacutil –i assembly.exe
```

Assembly Redirection

- `assemblyBinding` redirects the assembly references

```
<configuration>
   <runtime>
      <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
         <dependentAssembly>
            <assemblyIdentity
name="Wincubate.Module20.Slide17.FancyClass"
                              publicKeyToken="052694033bde0a15"
                              culture="neutral" />
            <bindingRedirect oldVersion="1.0.0.0"
                             newVersion="2.0.0.0"/>
            <codeBase version="2.0.0.0"
               href="http://www.wincubate.net/70-
483/Wincubate.Module20.Slide17.FancyClass.dll"/>
         </dependentAssembly>
      </assemblyBinding>
   </runtime>
</configuration>
```

Summary

- Logging and Tracing
- Performance Counters
- Debugging and Managing Assemblies

Question

- You are developing an assembly which will be used by a number of distinct applications. You need to install the assembly into Global Assembly Cache.

  Which actions can be used to accomplish this? (Each correct answer presents a complete solution. Choose two.)

a) [✓ Use Windows Installer 2.0 or later to install assembly]
b) [✓ Use gacutil.exe to add assembly to the cache]
c) Use regasm.exe to register assembly in the cache
d) Use regsvr32.exe to register assembly in the cache
e) Use sn.exe to copy assembly to the cache

TEKNOLOGISK
INSTITUT