Threading in C#

Authored by : Sushant Banerjee
Email : sushantba@cybage.com

Presented by : Sushant Banerjee
Extn. 7221

Agenda

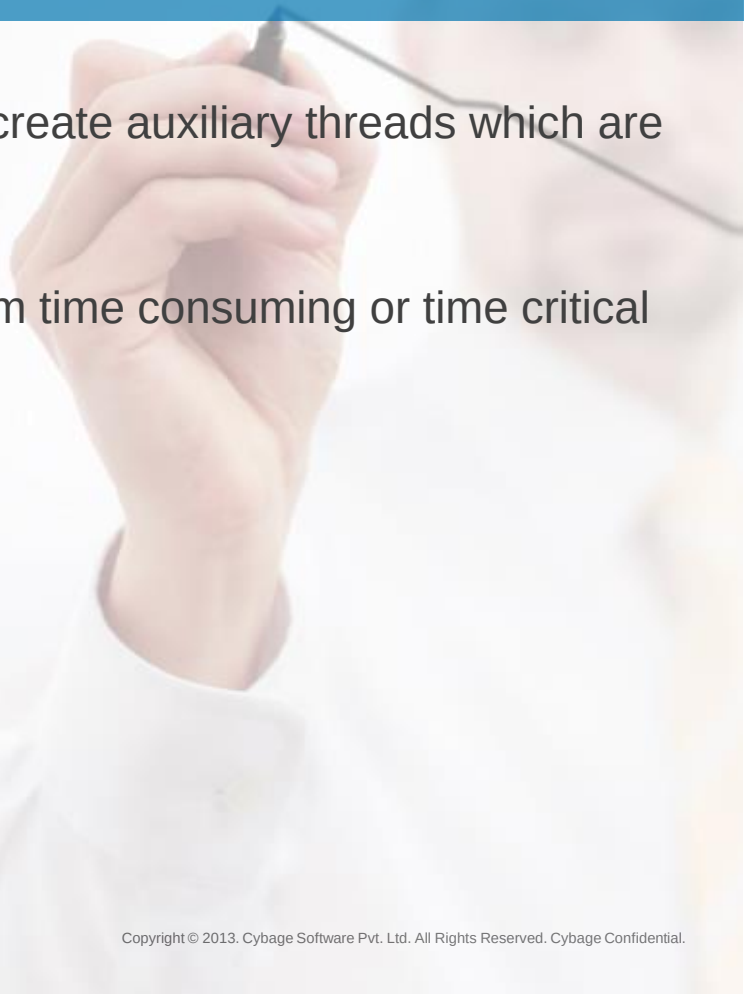
- Introduction
- Creating New Threads
- Multithreading
- Setting Priority
- Synchronization

What is Threading

- Threading enables you to perform **concurrent processing**.
- In other words, you can do **more than one operation** at a time.
- In .Net Framework **System.Threading** namespace used for threading.
- By default a program has one thread which is also known as **primary thread**.

Worker Threads

- In addition to primary thread you can create auxiliary threads which are also known as **worker threads.**
- Worker threads can be used to perform time consuming or time critical tasks.



Multithreading

- Multithreading is a process in which different tasks execute on **separate threads**.
- A multithreaded applications can **perform multiple tasks** at the same time.
- Multithreading is also known as **free threading**.
- In multithreaded applications the user interface **always remains active**.
- Multithreading can be used to write scalable applications which means that you can add new threads as the workload increases.

Creating Thread

```
//create a new worker thread and pass method reference
  to threadstart delegate
Thread workerThread = new Thread(PrintEvenNumbers);

//start worker thread
workerThread.Start();
```

Thread Methods

Method	Action
Start	Causes a thread to start to run.
Sleep	Pauses a thread for a specified time.
Suspend	Pauses a thread.
Abort	Stops a thread.
Resume	Restarts a suspended thread

Thread Properties

Method	Action
IsAlive	Contains the value True if a thread is active
Name	Gets or sets the name of a thread. Most frequently used to discover individual threads when you debug.
Priority	Gets or sets a value that is used by the operating system to prioritize thread scheduling.
ThreadState	Contains a value that describes a thread's state or states.

Thread Priorities

- The operating system allocates
  - Longer time slices to high priority threads and
  - Shorter time slices to low priority threads.
- You can use priority property of threads to set the priority level.
- Threads are scheduled for execution based on their priority.

```
workerThread.Priority = ThreadPriority.Highest;
```

Thread Priorities

Member name	Description
AboveNormal	The Thread can be scheduled after threads with Highest priority and before those with Normal priority.
BelowNormal	The Thread can be scheduled after threads with Normal priority and before those with Lowest priority.
Highest	The Thread can be scheduled before threads with any other priority.
Lowest	The Thread can be scheduled after threads with any other priority.
Normal	The Thread can be scheduled after threads with AboveNormal priority and before those with BelowNormal priority. Threads have Normal priority by default.

Thread Synchronization

- In multithreaded applications each thread executes asynchronously.
- That means each thread can access same resources such as files, networks and memory at the same time.
- This asynchronous nature of threads result into unpredictable data corruption.
- Thread synchronization helps you to handle these situations using lock and monitors.

Lock Statement

- The lock statement can be used to ensure that a block of code runs and completes its execution without interruption by other threads.

```
private System.Object lockThis = new System.Object();
        public void Process()
        {
            lock (lockThis)
            {
                // Access thread-sensitive resources.
            }
        }
```

Demo

Attributes in C#

Agenda

- Introduction
- Using Attributes
- Custom Attributes

What are Attributes

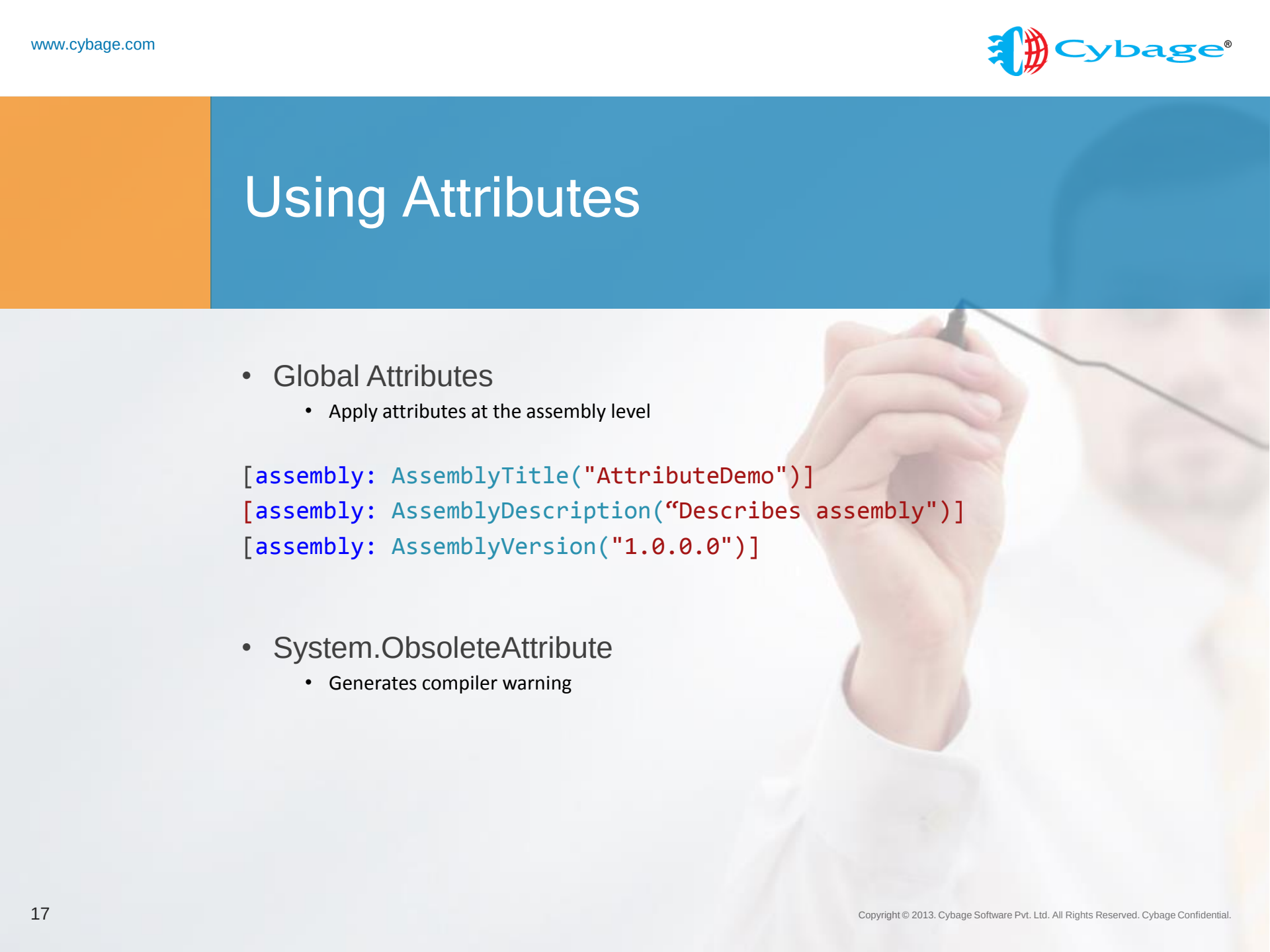
- An assembly contains your code converted into MSIL
- An assembly also contains metadata about your code
- Attributes can add extra information to the metadata
- The tool ildasm.exe allows you to look inside the assembly

Using Attributes

- Global Attributes
  - Apply attributes at the assembly level

```
[assembly: AssemblyTitle("AttributeDemo")]
[assembly: AssemblyDescription(“Describes assembly")]
[assembly: AssemblyVersion("1.0.0.0")]
```

- System.ObsoleteAttribute
  - Generates compiler warning



Custom Attributes

- Creating your own attribute
  - Derive from System.Attribute base class.
  - Add suffix “Attribute” (optional)
- Finally decorate your class adding
  - Attribute Parameters
  - Attribute Targets
  - AttributeUsage

Attribute Parameters

- **Positional Parameter**
  - A position parameter is mandatory
  - it should come before any named parameters.
- **Named Parameter**
  - A named parameter is optional
  - Can be specified in any order
  - Its should come after all positional parameters.

Attribute Targets

- The target of an attribute is the entity to which the attribute applies.
- By default an attribute applies to the element that it precedes.
- An attribute may target
  - Assembly
  - Field
  - Event
  - Method
  - Class
  - Struct
  - Enum
  - Delegate
  - property etc.

AttributeUsage

- AttributeUsage is an attribute that can be used to determine how a custom attribute class can be used.

- The default settings of AttributeUsage is as follows:

```
[System.AttributeUsage(System.AttributeTargets.All,
                       AllowMultiple = false,
                       Inherited = true)]
class NewAttribute : System.Attribute { }
```

Demo

Bibliography, Important Links

- http://msdn.microsoft.com/en-us/library/hyz69czz(v=vs.110).aspx
- http://msdn.microsoft.com/en-us/library/1c9txz50(v=vs.110).aspx
- http://www.codeproject.com/Articles/274062/Improved-Multi-Threading-Support-in-Net
- http://msdn.microsoft.com/en-IN/library/z0w1kczw.aspx

Any Questions?

Thank you!