Module 11

"Reflection"

TEKNOLOGISK
INSTITUT

TEKNOLOGISK
INSTITUT

Agenda

- Using Reflection
- Attributes
- Using the CodeDOM

TEKNOLOGISK
INSTITUT

Introducing Reflection

- Reflection constitute the .NET classes facilitating
  - Programmatic inspection and enumeration of types
  - Inspection and processing of metadata such as attributes
  - Creating types and code dynamically, e.g.
    - Creating dynamic assemblies
      - Generate IL code dynamically
    - Creating dynamic types
    - Creating adaptive code
    - Plug-in architecture building
    - Dynamically subscribing to events
    - …

TEKNOLOGISK
INSTITUT

Reflection on Assemblies

- Assemblies are one of the starting points of Reflection

- The `Assembly` class
  - `Load()` static
  - `LoadFrom()` static
  - `ReflectionOnlyLoad()` static
  - `ReflectionOnlyLoadFrom()` static

  - `GetExecutingAssembly()` static
  - `GetEntryAssembly()` static
  - `GetTypes()` non-static

Reflection on Types

TEKNOLOGISK INSTITUT

- The `Type` class is another starting points of Reflection
  - `GetMembers()`
    - `GetFields()`
    - `GetProperties()`
    - `GetEvents()`
  - `GetMethods()`
    - `GetConstructors()`
- `MemberInfo`
  - `FieldInfo`
  - `PropertyInfo`
  - `EventInfo`
- `MethodBase`
  - `MethodInfo`
  - `ConstructorInfo`

```
Type type = typeof( Player );

FieldInfo[] fields = type.GetFields();
foreach( FieldInfo fi in fields )
{
   Console.WriteLine( "Field: {0}", fi );
}

MethodInfo[] methods = type.GetMethods();
foreach( MethodInfo mi in methods )
{
   Console.WriteLine( "Method: {0}", mi );
}
```

TEKNOLOGISK INSTITUT

Binding Flags

- The `BindingFlags` enumeration provides filtering
  - `Default` Equivalent to not specifying `BindingFlags`
  - `DeclaredOnly` Ignores inherited members
  - `FlattenHierarchy` Declared, inherited, and protected members
  - `IgnoreCase` Case-insensitive matching
  - `Instance` Instance type members are included
  - `Public` Public members are included
  - `NonPublic` Protected and internal members are included
  - `Static` Static members are included

```
BindingFlags flags = BindingFlags.NonPublic |
                     BindingFlags.Public |
                     BindingFlags.Instance;

FieldInfo[] fields = type.GetFields( flags );
```

MethodInfo Class

TEKNOLOGISK INSTITUT

- `MethodInfo.Invoke()`
  - Can invoke members on objects and classes
    - Use `null` for static methods

```
MethodInfo setPositionMethod = t.GetMethod( "SetPosition" );
setPositionMethod.Invoke( player,
                          new object[] { Position.Midfielder } );
```

- `MethodInfo` properties are many, e.g.
  - `IsAbstract`
  - `IsConstructor`
  - `IsFinal`
  - `IsGeneric`
  - `IsStatic`
  - `IsVirtual`

TEKNOLOGISK
INSTITUT

Agenda

- Using Reflection
- **Attributes**
- Using the CodeDOM

TEKNOLOGISK
INSTITUT

Introducing Attributes

- Attributes are metadata
  - Inserted into the assembly at compilation time
  - Can be retrieved and handled at runtime

- Examples include
  - `[Serializable]` Read by the .NET serialization engine
  - `[DebuggerHidden]` Read by Visual Studio 2012
  - `[AssemblyFileVersion]` Read by Windows Explorer

- You can define custom attributes yourself if needed

TEKNOLOGISK
INSTITUT

Assembly Attributes

- Assembly attributes include e.g.
  - `[AssemblyCompany]`
  - `[AssemblyCopyright]`
  - `[AssemblyConfiguration]`
  - `[AssemblyDescription]`
  - `[AssemblyVersion]`
  - `[AssemblyFileVersion]`
  - …

```
[assembly: AssemblyVersion( "1.0.0.0" )]
[assembly: AssemblyCompany( "Wincubate ApS" )]
```

Retrieving Attributes

TEKNOLOGISK INSTITUT

- Attributes on assemblies can be retrieved via
  - `Assembly.GetCustomAttributes()`

```
Assembly assembly = Assembly.GetExecutingAssembly()
foreach( Attribute attribute in assembly.GetCustomAttributes(false) )
{
   if( attribute.GetType() == typeof( AssemblyCopyrightAttribute ) )
   {
      Console.WriteLine( "Copyright: {0}",
         ( attribute as AssemblyCopyrightAttribute ).Copyright );
   }
}
```

- Attributes can be retrieved on any type via
  - `MemberInfo.GetCustomAttributes()`

Creating Custom Attributes

TEKNOLOGISK INSTITUT

- Define your own attributes by deriving from **`System.Attribute`**

```
[AttributeUsage(AttributeTargets.Class)]
public class DeveloperInfoAttribute : System.Attribute
{
   public DeveloperInfoAttribute( string developer )
   {
      Developer = developer;
   }

   public string Developer { get; set; }
   public string Date { get; set; }
   public int Revision { get; set; }
}
```

TEKNOLOGISK
INSTITUT

Agenda

- Using Reflection
- Attributes
- **Using the CodeDOM**

TEKNOLOGISK INSTITUT

Introducing CodeDOM

- `System.CodeDOM` defines models of code
  - `CodeCompileUnit` class — Main entry point
  - `CodeNamespace` class
  - `CodeTypeDeclaration` class
  - `CodeMemberMethod` class

- Generate source code from CodeDOM model
  - `CodeDomProvider`
    - `CSharpCodeProvider`
    - `VBCodeProvider`
    - `JScriptCodeProvider`

- Main features
  - Generate source code from CodeDOM model
  - Compile source code and generate assembly

Defining the CodeDOM Model

TEKNOLOGISK INSTITUT

- Define a **`CodeCompileUnit`** and add types and members
  - The structure in the DOM is "parallel" to the program's structure
- More than 100 different CodeDOM classes for creating a model

```
CodeCompileUnit unit = new CodeCompileUnit();
...
CodeTypeDeclaration type = new CodeTypeDeclaration( "Program" );
CodeEntryPointMethod main = new CodeEntryPointMethod();
main.Statements.Add(
   new CodeMethodInvokeExpression(
      new CodeTypeReferenceExpression( "Console" ),
         "WriteLine",
         new CodePrimitiveExpression( "Hello World!" )
   )
);
type.Members.Add( main );
```

Generating Source Code from the CodeDOM Model

TEKNOLOGISK INSTITUT

- Invoke `CodeDomProvider.GenerateCodeFromCompileUnit()` using
  - `CodeGenerationOptions`
  - `IndentedTextWriter`

```
CSharpCodeProvider code = new CSharpCodeProvider();
using( StreamWriter writer = new StreamWriter(@"C:\Tmp\Program.cs") )
{
   using( IndentedTextWriter itw = new IndentedTextWriter( writer ) )
   {
      CodeGeneratorOptions options = new CodeGeneratorOptions
      {
         BlankLinesBetweenMembers = true
      };
      code.GenerateCodeFromCompileUnit( unit, itw, options );
   }
}
```

Compiling the Source Code

TEKNOLOGISK INSTITUT

- Compile using `CodeDomProvider.CompileAssemblyFrom`*`Xxx`*`()`
  - Set `CompilerParameters`

```
CompilerParameters parameters = new CompilerParameters()
{
   GenerateExecutable = true,
   OutputAssembly = @"C:\Tmp\HelloWorld.exe"
};
parameters.ReferencedAssemblies.Add( "System.dll" );

CompilerResults results = code.CompileAssemblyFromFile(
   parameters,
   @"C:\Tmp\Program.cs"
);
```

- Consult `CompilerResults.Errors` after compilation!

TEKNOLOGISK
INSTITUT

Summary

- Using Reflection
- Attributes
- Using the CodeDOM

TEKNOLOGISK
INSTITUT

Question

- You are creating an application and need to access the currently running assembly for reflection purposes.

  Which code segment should you use?

a)
```
var asm = Assembly.GetAssembly( this );
```

b)
```
var asm = Assembly.GetEntryAssembly();
```

c)
```
var asm = Assembly.GetExecutingAssembly();
```

d)
```
var asm = Assembly.Load();
```

TEKNOLOGISK
INSTITUT