vbug

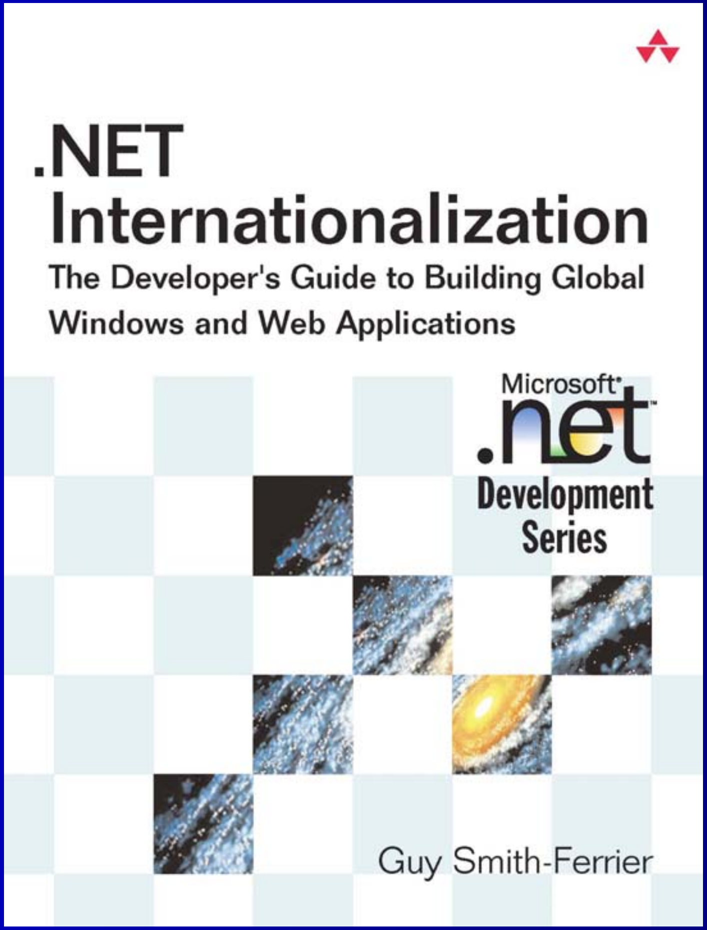
Writing Custom FxCop Rules

Microsoft CERTIFIED Professional

Guy Smith-Ferrier
guy@guysmithferrier.com
Blog: http://www.guysmithferrier.com

Author of…

- .NET Internationalization, Addison Wesley, ISBN 0321341384
- Visit http://www.dotneti18n.com to download the complete source code


.NET Internationalization
The Developer's Guide to Building Global Windows and Web Applications

Agenda

- Overview Of FxCop
- Writing FxCop Rules
  - Writing an FxCop rule which walks code instruction by instruction
  - Writing an FxCop rule which walks code by overriding "visit" methods

Overview

- "FxCop" is an abbreviation for framework police
- FxCop is a free static analysis tool for Visual Studio 2003 and Visual Studio 2005
- FxCop tests rules against assemblies and reports on failed rules
  - FxCop can be applied to any .NET language because it works on assemblies and not code
  - The rules included with FxCop are based upon the "Microsoft .NET Framework Design Guidelines"

How To Get FxCop

- FxCop is included with Visual Studio 2005 Team Edition For Software Developers
- FxCop can be downloaded from:-

  http://www.gotdotnet.com/team/fxcop/
- You can post messages directly to the FxCop team and other interested parties at Microsoft’s FxCop Forum:-

  http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=98&SiteID=1

FxCop Versions

	FxCop 1.35	**FxCop 1.32**
Analyzes Assemblies From	.NET 1.1 and 2.0	.NET 1.1
Loads Rules Assemblies From	.NET 2.0	.NET 1.1

FxCop Interfaces

- FxCop supports two interfaces:-
  - A GUI interface
    - Intended for interactive use
      - Is built into Visual Studio 2005
      - Is available as a separate executable (FxCop.exe) for Visual Studio 2003 and Visual Studio 2005
  - A Command Line interface
    - Intended for use in scripts and build processes
      - Is a separate executable (FxCopCmd.exe) for both Visual Studio 2003 and Visual Studio 2005

Potential Terminology Confusion

- In FxCop a "project" is an FxCop Project
  - It is not a Visual Studio project
  - It describes the targets, rules and exclusions for any given analysis
- In FxCop the assembly to be analyzed is called a "target"

FxCop Demo

- Create a Windows Forms application and build it
- Start FxCop.exe, select Project | Add Targets… and select the new Windows Forms assembly (e.g. WindowsApplication1.exe)
- Click the Analyze button
- Select all of the errors, right click and select Exclude

FxCop Demo (continued)

- Add a new enum to Form1:-

```
public enum CheeseEnum
     {SmokedAustrian, JapaneseSageDerby, VenezuelanBeaverCheese};
```

- Build the project
- In FxCop click Analyze again and observe the new error
- Fix the error, analyze it again and show the error is no longer reported

FxCop And Visual Studio 2005 Team Edition For Software Developers

- Visual Studio 2005 Team Edition For Software Developers supports including FxCop in the build process
  - When a build is performed Visual Studio also runs FxCop
    - Errors are shown in Visual Studio's Output window
    - Errors prevent the build from being successful
- To enable code analysis (i.e. FxCop) in Visual Studio:-
  - Right click the project in Solution Explorer, select Properties, select the Code Analysis page and check the Enable Code Analysis checkbox

FxCop, Visual Studio 2005 And Visual Studio 2003

- To integrate FxCop into Visual Studio 2005 or Visual Studio 2003:-
  - In Visual Studio select Tools | External Tools
  - Click Add and set:-
    - Title to "FxCopCmd"
    - Command to "C:\Program Files\Microsoft FxCop 1.35\FxCopCmd.exe"
    - Arguments to "/f:$(TargetPath) /r:rules /c"
    - Initial Directory to "C:\Program Files\Microsoft FxCop 1.35"
    - Check the "Use Output window" checkbox
  - Click OK
- To run FxCop select Tools | FxCopCmd
  - Errors show up in the Output window

The Problem

- Assume that we want all threads to be created by a thread factory
- So instead of writing something like this:-

```
Thread thread = new Thread(new ThreadStart(Work));
```

- We want our developers to write something like this:-

```
Thread thread = ThreadFactory.CreateThread(new ThreadStart(Work));
```

We need a rule to catch any instance where the developer uses the Thread class's constructor directly

  – Our rule will be called ThreadNotProvidedByFactory

Custom FxCop Rules Overview

- FxCop rules are contained within .NET assemblies
  - Create a .NET Class Library
- FxCop rules must be described in an XML document which is embedded in the assembly as a resource
- FxCop rules are classes which inherit from BaseIntrospectionRule

Custom Rules

- Create a new class library and call it CompanyRules
  - Add a Reference to FxCopSdk.dll and Microsoft.Cci.dll (both in the FxCop folder)
- In Solution Explorer right click the project, select Add | Add New Item…, select XML File and name it RuleData.xml
  - In the Properties Window change Build Action to Embedded Resource
- Add the following rule definition to RuleData.xml:-

Custom Rules (continued)

```
<Rules>
   <Rule TypeName="ThreadNotProvidedByFactory" Category="Threads" 
CheckId="C0001">
      <Name>Thread not provided by ThreadFactory</Name>
      <Description>A Thread has been constructed using a Thread 
constructor instead of ThreadFactory.CreateThread</Description>
      <Owner>Guy Smith-Ferrier</Owner>
      <Url></Url>
      <Resolution>Construct new Thread objects using 
ThreadFactory.CreateThread</Resolution>
      <Email></Email>
      <MessageLevel Certainty="99">Warning</MessageLevel>
      <FixCategories>Breaking</FixCategories>
   </Rule>
</Rules>
```

Custom Rules (continued)

- Replace all of the code in Class1.cs with:-

```
using System;
using Microsoft.Cci;
using Microsoft.FxCop.Sdk;
using Microsoft.FxCop.Sdk.Introspection;
namespace CompanyRules
{
    public class ThreadNotProvidedByFactory: BaseIntrospectionRule
    {
        public ThreadNotProvidedByFactory()
            : base("ThreadNotProvidedByFactory",
"CompanyRules.RuleData", typeof(ThreadNotProvidedByFactory).Assembly)
        {
        }
    }
}
```

Custom Rules (continued)

- Save and build the assembly
- In FxCop select Project | Add Rules…and select CompanyRules.dll
- In the Rules tab expand the CompanyRules.dll node to reveal the "Thread not provided by ThreadFactory" rule
  - Double click the rule to see all of the information which was supplied in RuleData.xml
- Close FxCop because it locks the rule assemblies preventing them from being rebuilt

Strategies For Writing Rules

- Strategy 1
  - Walk through IL instructions one by one looking for offending instructions
- Strategy 2
  - Override "visit" methods which are called for each offending instruction

BaseInspectionRule.Check Overrides

```
public virtual ProblemCollection Check(Member member)

public virtual ProblemCollection Check(Module module)

public virtual ProblemCollection Check(Parameter parameter)

public virtual ProblemCollection Check(Resource resource)

public virtual ProblemCollection Check(TypeNode type)

public virtual ProblemCollection Check(
    string namespaceName, TypeNodeList types)
```

Overriding Check Methods

```
public override ProblemCollection Check(Member member)
{
    Method method = member as Method;

    if (method != null &&
        ! TypeIsSubClassOf(method.DeclaringType,
        "Company.Threading.ThreadFactory"))
    {
        if (CheckMethodForNewObj(method, new string[] {
            "System.Threading.Thread"}).Count > 0)
        {
            Resolution resolution = GetResolution(
                new string[] {method.Name.Name});
            Problems.Add(new Problem(resolution));
            return Problems;
        }
    }
    return base.Check(member);
}
```

Walking Through IL Instructions

```
protected virtual StringCollection CheckMethodForNewObj(
    Method method, string[] classNames)
{
    StringCollection classesFound = new StringCollection();
    for(int instructionNumber = 0; instructionNumber <
        method.Instructions.Length; instructionNumber++)
    {
        Microsoft.Cci.Instruction instruction =
            method.Instructions[instructionNumber];
        if (instruction.OpCode == OpCode.Newobj &&
        instruction.Value is Microsoft.Cci.InstanceInitializer)
        {
            Microsoft.Cci.InstanceInitializer instanceInitializer
         = (Microsoft.Cci.InstanceInitializer) instruction.Value;
```

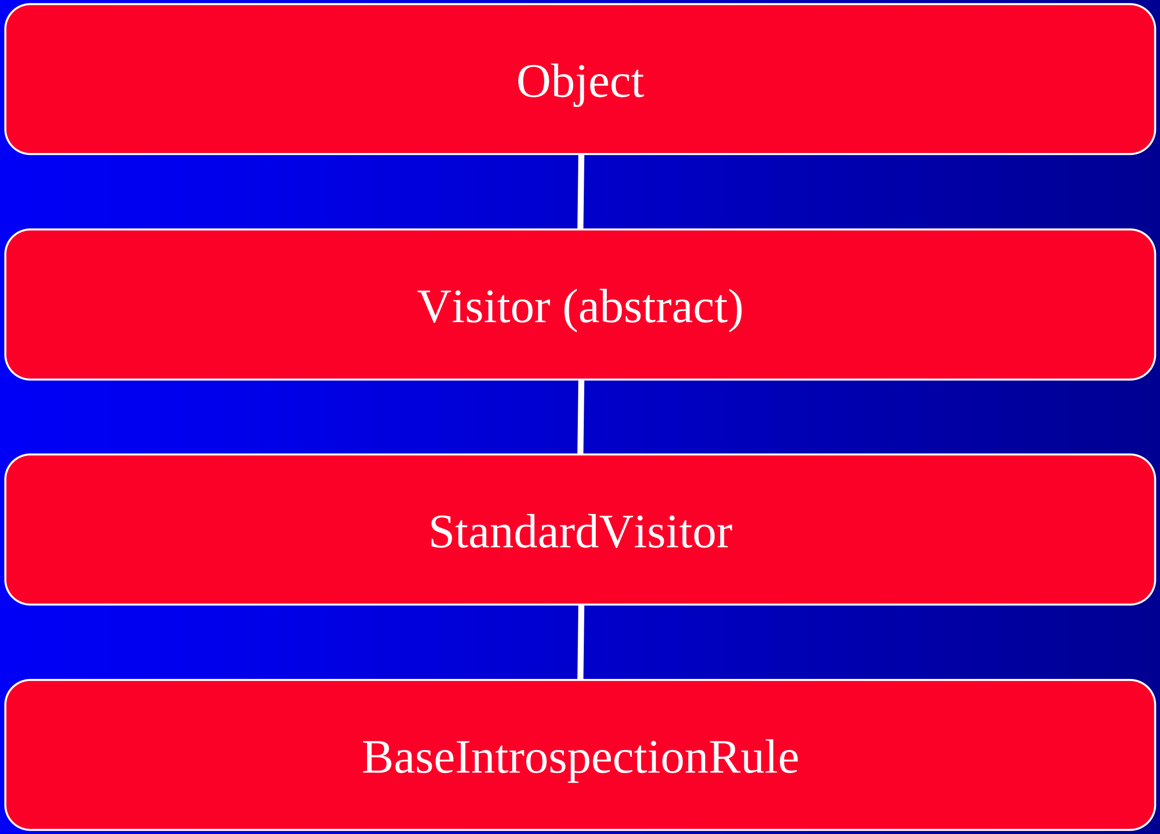
Walking Through IL Instructions (continued)

```
                foreach(string className in classNames)
                {
                    if (TypeIsSubClassOf(
                    instanceInitializer.DeclaringType,
                    className))
                        classesFound.Add(className);
                }
            }
        }
        return classesFound;
    }
```

Testing Type

```
protected virtual bool TypeIsSubClassOf(TypeNode type,string typeName)
{
    if (type.FullName == typeName)
        return true;
    else if (type.BaseType == null)
        return false;
    else
        return TypeIsSubClassOf(type.BaseType, typeName);
}
```

BaseIntrospectionRule Class Hierarchy



StandardVisitor's Visit Methods

- The StandardVisitor class includes 140 "Visit" methods
- Visit methods "visit" a node of a given type
  - VisitMethodCall visits method calls
- You begin the visiting process by calling a visit method with a broad scope
  - VisitMethod visits all nodes in a method (e.g. assignments, expressions, method calls, variable declarations)
- You override the Visit method that you are interested in

Overriding A Visit Method

- Replace the Check method with:-

```
public override ProblemCollection Check(Member member)
{
    Method method = member as Method;
    if (method != null)
    {
        classUsed = false;
        VisitMethod(method);
        if (classUsed)
        {
            Resolution resolution = GetResolution(
                new string[] {method.Name.Name});
            Problems.Add(new Problem(resolution));
            return Problems;
        }
    }
    return base.Check (member);
}
```

Overriding A Visit Method (continued)

- Add a private bool field called classUsed

```
public override Expression VisitConstruct(Construct cons)
{
    if (cons != null)
    {
        MemberBinding memberBinding =
            cons.Constructor as MemberBinding;
        if (memberBinding != null)
        {
            InstanceInitializer instanceInitializer =
                memberBinding.BoundMember as InstanceInitializer;
            if (instanceInitializer != null &&
                instanceInitializer.DeclaringType.FullName ==
                "System.Threading.Thread")
                classUsed = true;
        }
    }
    return base.VisitConstruct (cons);
}
```

Summary

- FxCop applies rules to assemblies
- FxCop includes a library of rules
- You can write your own rules to enforce your own standards
  - Writing rules efficiently requires a good understanding of FxCop
    - There is currently no documentation for the FxCop SDK so understanding this process is a case of trial and error
      - Try .NET Internationalization, Chapter 13 “Testing Internationalization Using FxCop”