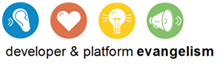
* 1. 

Demo Script

Code Contracts

* 1. Lab version: 1.0.0
  2. Last updated: 12/29/2010
  3. 

1. Contents

[Overview 3](#_Toc281405165)

[Key Messages 3](#_Toc281405166)

[Key Technologies 3](#_Toc281405167)

[Time Estimates 3](#_Toc281405168)

[Setup and Configuration 4](#_Toc281405169)

[Opening Statement 4](#_Toc281405170)

[Step-by-Step Walkthrough 5](#_Toc281405171)

[Segment #1 – Preconditions 5](#_Toc281405172)

[Segment #2 – Post Conditions 8](#_Toc281405173)

[Segment #3 – Object Invariants and configurable options 10](#_Toc281405174)

[Summary 12](#_Toc281405175)

Overview

* 1. Using Code Contracts, programmers provide method preconditions and post-conditions that enrich existing APIs with information that is not expressible in the type systems of .NET languages. Additionally, contracts specify object invariants, which define what allowable states an instance of a class may be in (i.e. its internal consistency.)
  2. The contracts are used for runtime checking, static verification, and documentation generation. Contracts also allow automatic documentation checking and improved testing. Code Contracts for .NET currently consists of three components: the static library methods used for expressing the contracts, a binary rewriter and a static checker.
  3. For tools and detailed instructions for using code contracts, see [Code Contracts](http://msdn.microsoft.com/en-us/devlabs/dd491992.aspx) on the MSDN DevLabs Web site.

### Key Messages

* 1. Code Contracts provide a language-agnostic way to express coding assumptions in .NET programs. The contracts take the form of pre-conditions, post-conditions, and object invariants.
  2. Preconditions are requirements that must be met when entering a method or property. They are specified using the Contract.Requires method.
  3. Post-conditions describe expectations at the time the method or property code exits. They are specified using the Contract.Ensures method.
  4. Object invariants describe the expected state for a class that is in a good state.

### Key Technologies

This demo uses the following technologies:

* 1. Microsoft Visual Studio 2010

### Time Estimates

* + Estimated time to complete the demo: 30 min

Setup and Configuration

## System Requirements

* + Microsoft Visual Studio 2010
  + [Data Contracts](http://msdn.microsoft.com/en-us/devlabs/dd491992.aspx) Premium Edition binaries

Opening Statement

* 1. Code contracts provide a way to specify preconditions, postconditions, and object invariants in your code. Preconditions are requirements that must be met when entering a method or property. Postconditions describe expectations at the time the method or property code exits. Object invariants describe the expected state for a class that is in a good state.
  2. Code contracts include classes for marking your code, a static analyzer for compile-time analysis, and a runtime analyzer. The classes for code contracts can be found in the **System.Diagnostics.Contracts** namespace. To use code contracts you must download an MSI from the code contracts web site in DevLabs (Premium Edition).
  3. All .NET Framework languages can immediately take advantage of contracts; you do not have to write a special parser or compiler. A Visual Studio add-in lets you specify the level of code contract analysis to be performed. The analyzers can confirm that the contracts are well-formed (type checking and name resolution) and can produce a compiled form of the contracts in Microsoft intermediate language (MSIL) format. Authoring contracts in Visual Studio lets you take advantage of the standard IntelliSense provided by the tool.
  4. The following is a brief description of the things that will be shown during this demo:
  5. Preconditions
  6. Post conditions
  7. Object Invariants

Step-by-Step Walkthrough

1. This demo is composed of the following segments:
   1. Preconditions
   2. Post Conditions
   3. Object Invariants

### Segment #1 – Preconditions

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Open Microsoft Visual Studio 2010 from **Start | All Programs**.   2. Open the **CodeContractsDemo.sln** solution located under the **Source** folder of this demo (and choosing the folder that matches the language of your preference.)   3. Open the Program.cs file by double-clicking it in **Solution Explorer**. | * + First, notice that to use code contracts a reference to **System.Diagnostics.Contracts** is needed.   + The scenario for this demo is a **GetPassword** function that returns the password for a particular user ID. |  |
| * 1. Scroll to the **GetPassword** method. | * + Preconditions specify state when a method is invoked. The **Requires** method is simply a way to encode that a particular condition must be true upon entry to the method.   + In this example, we are specifying that the userId parameter must be zero or greater.   + It can only use data that is at least as visible as the method itself, so that callers might actually be able to satisfy the condition. They are generally used for specifying valid parameter values. The run-time behavior of failed preconditions is determined by the runtime analyzer. |  |
| * 1. Scroll to the **Main** method and show the invocation to the **GetPassword** method.   2. Press **CTRL+SHIFT+B** to build the solution.   3. Show the **Error List pane**, by clicking on the **Error List** tab located at the bottom left corner of Visual Studio.   4. Press **F5** to run the example.   5. Stop the execution of the application. | * + Let’s try the example to see the Preconditions in action. Notice that the call to the **GetPassword** method is done by passing a negative number as a parameter. Therefore, violating the precondition.   + When you build the solution, in the Error List pane, the violations to the condition detected at build time are shown as warnings.   + We will ignore the warnings and run the solution to see what happens.   + When a violation to a contract occurs, a ContractException exception is thrown, which specifies the condition that was not fulfilled. | * 3. image9.png |

### Segment #2 – Post Conditions

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Scroll to the **GetPassword** method and show the post-condition. | * + Now we will see post-conditions.   + Postconditions are contracts for the state of a method when it terminates. The postcondition is checked just before exiting a method. The run-time behavior of failed postconditions is determined by the runtime analyzer.   + In the example, we are requiring that the return value of the GetPassword method is not null , because later we are calling the GetLength method on this value.   + Unlike preconditions, postconditions may reference members with less visibility. A client may not be able to understand or make use of some of the information expressed by a postcondition using private state, but this does not affect the client's ability to use the method correctly. |  |
| * 1. Scroll to the Main method and show the invocation to the GetPassword method.   2. Change the parameter from -1 to 0.   3. Press **CTRL+ALT+B** to build the solution.   4. Show the Error List pane, by clicking on the Error List tab located at the bottom left corner of Visual Studio.   5. Press **F5** to run the example.   6. Stop the execution of the application. | * + In the call to the **GetPassword** method we will change the UserId to zero. This value does not violate the precondition, but in the method, we have a branch that returns a null value when the user id is zero.   + Let’s build the solution and look at the Error List pane. Notice that the number of violations has decreased, since it detected that the precondition is fulfilled.   + What will happen when we run this?   + A ContractException exception is thrown, because the post-condition is not fullfilled. |  |
| * 1. Scroll to the **Main** method and show the invocation to the **GetPassword** method.   2. Change the parameter from 0 to 2.   3. Press **F5** to run the example.   4. Press any key to finish the application execution. | * + Now we will use a parameter that meets both conditions, for example the User Id 2.   + If we run the solution, we can see that the number eight is printed, which is the length of the returned password. |  |

### Segment #3 – Object Invariants and configurable options

|  |  |  |
| --- | --- | --- |
| Action | Script | Screenshot |
| * 1. Write the following code to show an example of Object Invariants.   2. C#   3. [ContractInvariantMethod]   4. protected void ObjectInvariant ()   5. {   6. Contract.Invariant ( this.y >= 0 );   7. Contract.Invariant ( this.x > this.y );   8. ...   9. } | * + Let’s talk a little about object invariants.   + Object invariants are conditions that should be true for each instance of a class whenever that object is visible to a client. They express the conditions under which the object is considered to be correct.   + The invariant methods are identified by being marked with the **ContractInvariantMethodAttribute** attribute. The invariant methods must contain no code except for a sequence of calls to the Invariant method, each of which specifies an individual invariant. This is shown in the following code. |  |
| * 1. In **Solution Explorer**, right-click the project name, and then click **Properties**.   2. Go to the **Code Contracts** tab and show the options. | * + Finally, let’s see some configuration for Code Contracts. Installing the Code Contracts MSI enables the **Code Contracts** tab in project properties where you can set your preferences for Code Contracts use. For configurations where the runtime checking is not performed, the library methods are compiled away (via conditional compilation attributes on their definitions) so that your code pays no performance penalty at all for contracts that you do not want executed. |  |

Summary

In this demo, you examined Code Contracts. You saw a simple example where Preconditions and Post conditions were used. Additionally, you learnt about Object Invariants and the configurable options of Code Contracts.