Invariants Runtime Checking Example

Abstract

This example shows the runtime checking of invariants. It also points out a subtle issue of when an invariant check is *not* done. The example consists of three classes: a base class BasicBankAccount, a subtype WallStreetAccount, and a client, Client that creates bank accounts and calls methods on them.

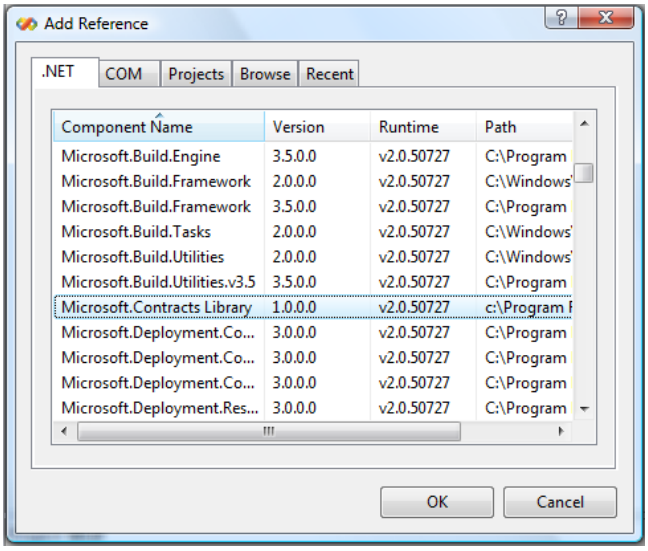
1 Adding the Contract Library Reference

If you are using Visual Studio 2008, or if you for some reason want to target a pre-v4 .NET runtime, then you need to:

- Change the target framework of the project.
- Manually add a reference to Microsoft.Contracts.dll

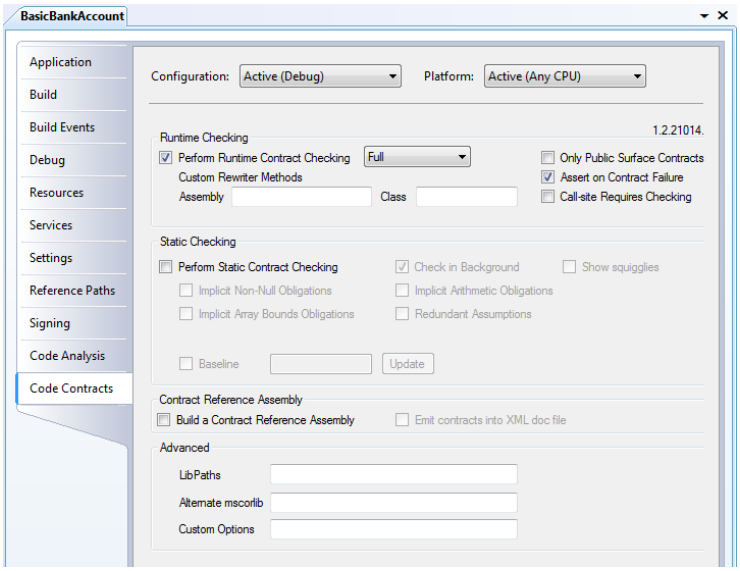
Otherwise, you may skip this section and go directly the next section!

To add the reference, open the Invariants solution and right-click on References in the Invariants project and select Add Reference. Find the Microsoft.Contracts library in the .NET tab as shown below and click OK.



2 Sample Walkthrough

Go to the Properties of project BasicBankAccount, select the Code Contracts pane (at the bottom), and enable runtime checking by clicking on the runtime checking box, as shown in this screenshot:

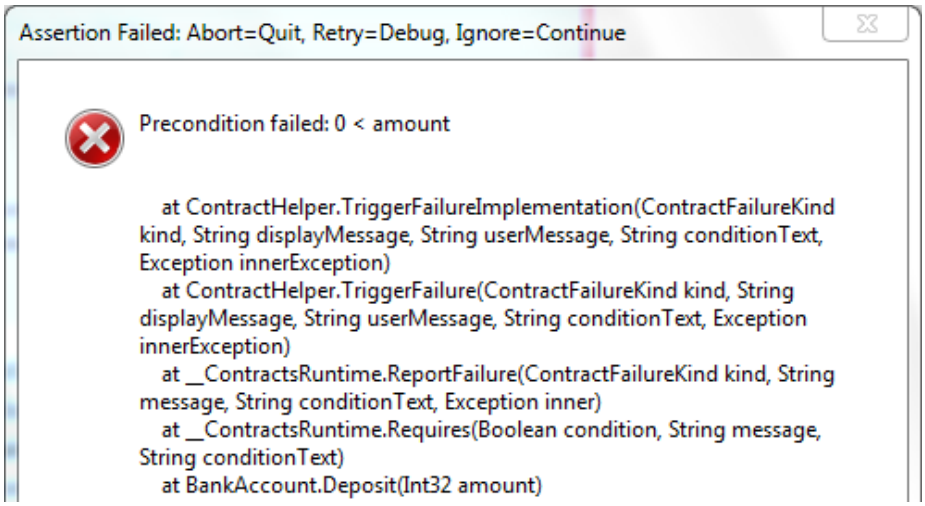


Then build the entire solution. The build should succeed. Now let's look at some of the code in the solution.

First, take a look at MainStreet.cs which contains the very simple class BankAccount. It has one method for putting money into the account, Deposit, another for removing money from the account, Withdraw, and a property, Balance, that has only a getter for returning the amount of money that is in the account.

What is different (aside from the contracts in each of those methods) is that an object invariant has been added to the class, a method named GoodAccount. (The full details about invariant methods and how to write them can be found in the general documentation.) This method gets checked at the end of every public method in the class.

Now run the program by, e.g., pressing F5 to start the debugger. You should see the following "assert" dialog box:



This failed contract is easily explained if you look at the code in Deposit:

```
public virtual void Deposit(int amount) {
  Contract.Requires(0 < amount);
  this.balance += amount;
}
```

and then notice that the client was calling it with a negative number. Fix that in Program.cs by changing −5 to some positive number. To see it in action, press F10 to start the debugger and have it start at the first line in the program. Press F10 twice to get to the call to Deposit. Now press F11 to step into the method call. Press F11 four more times and you'll see that you are now entering the object invariant:

```
[ContractInvariantMethod]
private void GoodAccount() {
  Contract.Invariant(0 <= balance);
}
```

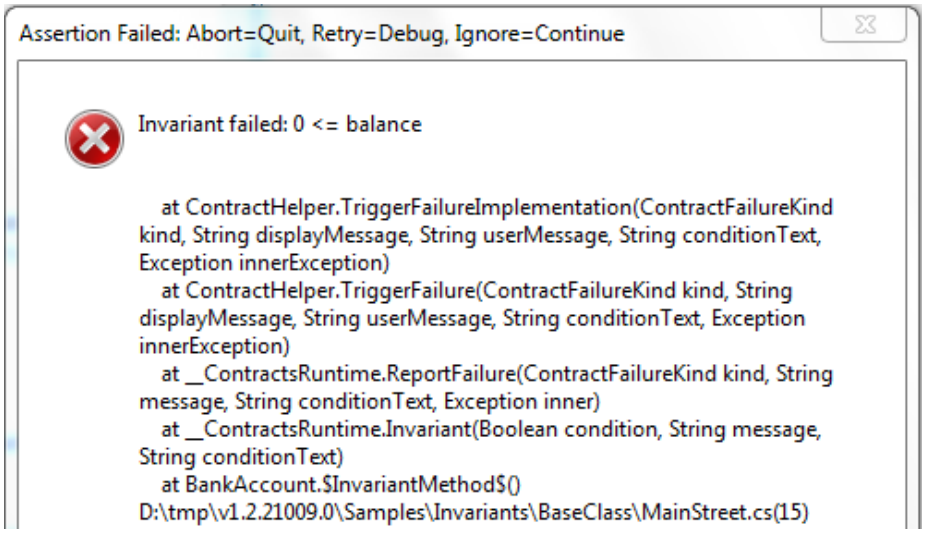
Continue execution by pressing F5 and the program will run to completion without an error.

However, if you look at the code in WallStreet.cs, you'll see that the override it has for Deposit does not maintain the invariant of its supertype.

```
public override void Deposit(int amount) {
  if (100 < amount) {
    this.slushFund += amount;
    this.balance -= amount;
  } else {
    base.Deposit(amount);
  }
```

```
}
```

The problem is that runtime checking was enabled for the base class, but not for the subtype. So now turn on the runtime checking for the project WallStreet and press F5 to run the program again. This time you should see another error dialog, this time for the invariant violation:



Note that the invariant check is injected into the subtype even though it has no contracts at all — not even a reference to the contract library!

The subtle point that is important to realize is that if you go back to the settings for the BasicBankAccount project and now turn *off* the runtime checking, then even though runtime checking is turned on for WallStreet, the invariant check will not be injected into the code and no invariant violation will be found.