Interface Contracts Static Checking Example

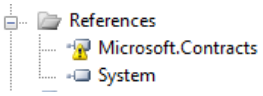
Abstract

This example shows how to use contracts on interfaces.

1 Adding the Contract Library Reference

1.1 Visual Studio 2008

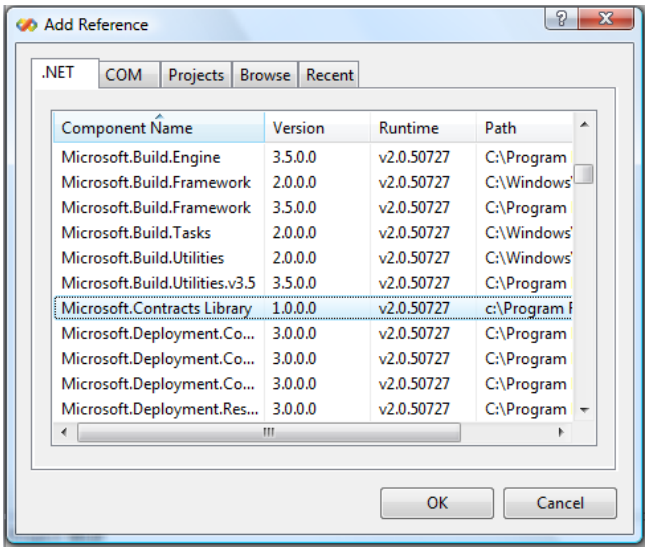
Before getting started with the sample, we need to make sure we have the proper reference for Microsoft.Contracts.dll. Look at the references of project InterfaceContracts:



If Visual Studio cannot find the Microsoft.Contracts.dll (indicated by the yellow warning), delete the reference and follow the steps below. Otherwise, you are ready to go to Section 2.

For new projects to use contracts, we need to add a reference to the Microsoft.Contracts library to the project.

Assuming you have installed the code contract tools, open the InterfaceContracts solution and right-click on References in the InterfaceContracts project and select Add Reference. Find the Microsoft.Contracts library in the .NET tab as shown below and click OK.



1.2 Visual Studio 2010

If you want to try the sample with Visual Studio 2010 you have two choices:

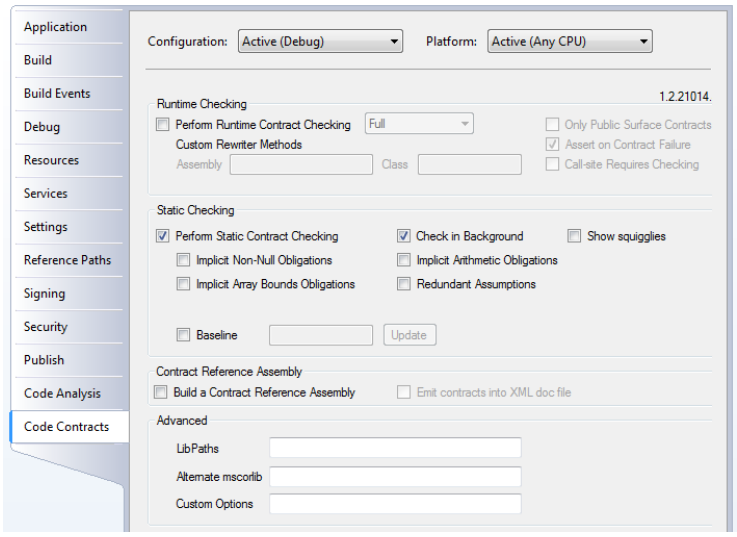
- Target the v3.5 framework (the sample default). In this case, follow the instructions under Visual Studio 2008 as this mode will require the Microsoft.Contracts library to be referenced.
- Target the v4.0 framework. In this case, the contract class is defined in mscorlib and *no* reference to Microsoft.Contracts.dll is necessary.

Thus, to target v4.0 you need to perform the following steps:

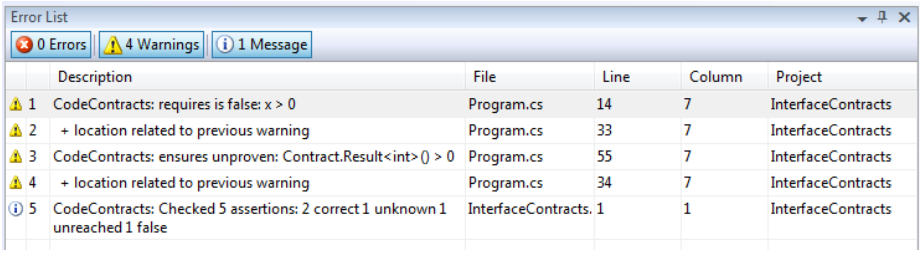
- Delete the Microsoft.Contracts.dll reference from all projects
- Change the target framework of all projects to v4.0.

2 Sample Walkthrough

After adding the proper reference, go to the Properties of project InterfaceContracts, select the Code Contracts pane (at the bottom), and enable static checking by clicking on the checkbox as shown in this screenshot:



Then build the example. The static checker should warn about two problems:



The first problem is in the call `f1.Foo(0)`. Given that `FooImplementation1` implements interface `IFoo` and `IFoo` has a contract written in the `IFooContract` class, the `Foo` method of `FooImplementation1` automatically inherits these contracts, namely:

```
Contract.Requires(x > 0);
Contract.Ensures(Contract.Result<int>() > 0);
```

At the call site flagged by the checker, we are passing a value of 0, which is clearly not positive.

Note how contracts are associated with an interface: take a look at the interface declaration for `IFoo`. You see the attribute

```
[ContractClass(typeof(IFooContract))]
```

This attribute informs the contract tools that the contract for interface `IFoo` is to be found in the `IFooContract` dummy class. Now look at the `IFooContract` class: it similarly has an attribute stating what interface it annotates. This is for consistency and documentation.

```
[ContractClassFor(typeof(IFoo))]
```

The return value can be anything, or the methods can throw NotImplementedException.

Now take a look at the second error: it warns that the implementation of Foo in class FooImplementation2 is not conforming to the IFoo contracts. Namely, the method may return 0, if the argument is 1. Since the contract states that the return value should be positive, the contract is not satisfied. In contrast, FooImplementation1 properly implements this contract, as it returns the argument x, which is already known to be positive given the pre-condition.