Chunker Example

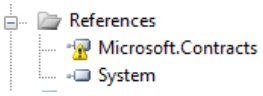
Abstract

This example shows how to use invariants to explicate implicit assumptions in data structures and how they allow one to satisfy contracts on other APIs, such as System.String.

1 Adding the Contract Library Reference

1.1 Visual Studio 2008

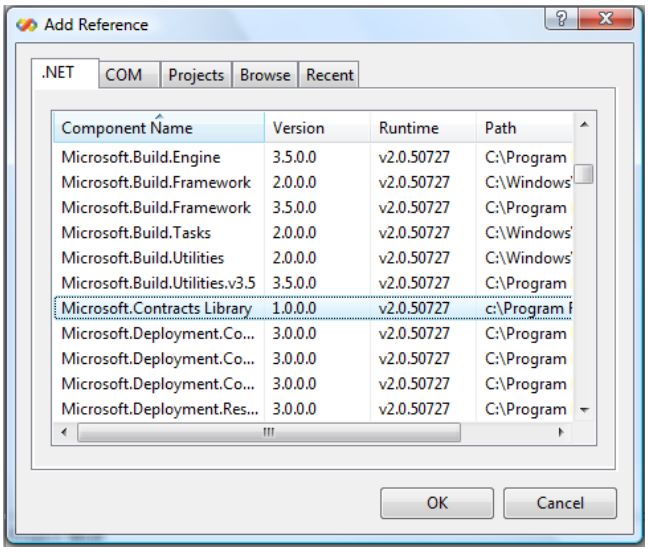
Before getting started with the sample, we need to make sure we have the proper reference for Microsoft.Contracts.dll. Look at the references of project Chunker:



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 Chunker solution and right-click on References in the Chunker 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:

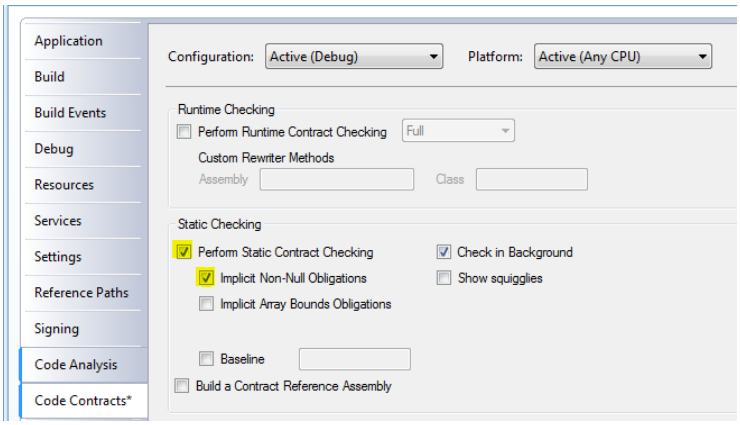
- Target the v3.5 framework (the sample default). In this case, follow the instructions under Visual Studio 2008 as this mode will require the Microsoft.Contracts library to be referenced.
- Target the v4.0 framework. In this case, the contract class is defined in mscorlib and *no* reference to Microsoft.Contracts.dll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

After adding the proper reference, go to the Properties of project Chunker, select the Code Contracts pane (at the bottom), and enable static checking by clicking on the static checking box. Also enable non-null checking if you wish.



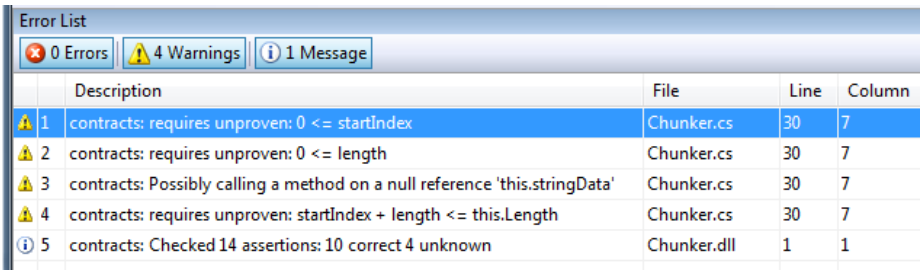
3 Overview

The Chunker class provides a way to split a string into equal size sub-strings, each holding a fixed number (chunkSize) of characters. The chunks are obtained by repeated calls to NextChunk

A chunker object holds on to the original string in stringData. This value is never modified. The size of each chunk is stored in chunkSize and also does not vary over the running time. Finally, returnedCount holds the number of characters returned from stringData so far. Alternatively, we can think of it as the index into stringData at which to return the next chunk.

4 First Attempt

Build the example. The build should succeed. After a moment[1], the static checker should warn about the call to Substring in NextChunk.



The documentation (and our corresponding contracts) on String.Substring(int, int) state that startIndex + length must be within the string extent. Furthermore, startIndex and length must be non-negative.

[1] The static checker runs in the background after the regular build.

The Chunker code written so far does not guarantee these conditions. E.g., the caller to the constructor could provide a non-positive chunkSize. Similarly, nothing is known about the relation between stringData.Length and returnedData.

5 Writing the Object Invariant

Let's write an object invariant that makes these relations explicit. In the Chunker class, at the member level, type cim TAB TAB to get an emtpy object invariant declaration:

```
[ContractInvariantMethod]
void ObjectInvariant() {
  Contract.Invariant(false);
}
```

Now fill in the first invariant, stating that chunkSize is positive (we don't want 0, as there are an infinite number of 0 length chunks we could extract).

Contract.**Invariant**(chunkSize > 0);

Under this invariant, write ci TAB TAB to get another empty invariant and fill it in to specify that returnedCount is similarly non-negative.

Contract.**Invariant**(returnedCount >= 0);

Add one more invariant, specifying that returnedCount is never more than the string length.

Contract.**Invariant**(returnedCount <= stringData.Length);

Finally, for good measure, let's also add the invariant that stringData should never be null.

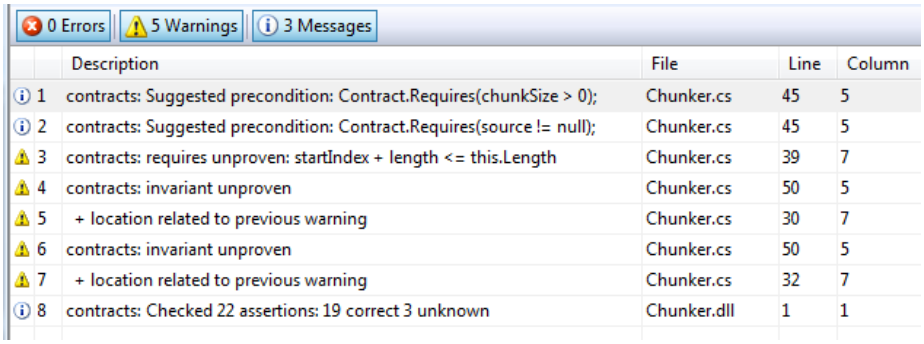
Contract.**Invariant**(stringData != **null**);

In fact, you should add this invariant *before* the invariant accessing stringData.Length, otherwise the checker will complain, and you might get a runtime null reference exception. Your object invariant should now look as follows:

```
[ContractInvariantMethod]
void ObjectInvariant() {
  Contract.Invariant(chunkSize > 0);
  Contract.Invariant(returnedCount >= 0);
  Contract.Invariant(stringData != null);
  Contract.Invariant(returnedCount <= stringData.Length);
}
```

6 Establishing the Object Invariant

If you build again, you see that the checker emits a new set of warnings:



0 Errors | 5 Warnings | 3 Messages

	Description	File	Line	Column
1	contracts: Suggested precondition: Contract.Requires(chunkSize > 0);	Chunker.cs	45	5
2	contracts: Suggested precondition: Contract.Requires(source != null);	Chunker.cs	45	5
3	contracts: requires unproven: startIndex + length <= this.Length	Chunker.cs	39	7
4	contracts: invariant unproven	Chunker.cs	50	5
5	+ location related to previous warning	Chunker.cs	30	7
6	contracts: invariant unproven	Chunker.cs	50	5
7	+ location related to previous warning	Chunker.cs	32	7
8	contracts: Checked 22 assertions: 19 correct 3 unknown	Chunker.dll	1	1

The two pre-conditions that length and startIndex must be non-negative are now satisfied in NextChunk. Before focusing on the remaining issue calling Substring, let's look at the constructor of Chunker. The checker warns that we may not establish the object invariant by the end of the constructor. In fact the first two messages suggest how to make sure we do, by adding the following pre-conditions to the Chunker constructor:

```
Contract.Requires(chunkSize > 0);
Contract.Requires(source != null);
```

Remember to use the shortcuts (cr TAB TAB for a general requires and crn TAB TAB for non-null requires).

7 Handling Border Cases

If you rebuild the project after adding the requires to the constructor, we should see the following remaining problem in NextChunk:

	Description	File	Line	Column
1	contracts: requires unproven: startIndex + length <= this.Length	Chunker.cs	39	7
2	contracts: Checked 22 assertions: 21 correct 1 unknown	Chunker.dll	1	1

The checker is complaining that returnedCount + chunkSize could be greater than stringData.Length. Of course, this situation may arise when we get near the end of the string. In that case, there may not be enough characters left. To fix this problem, we can change the code as follows:

```
public string NextChunk()
{
  string s;
  if (returnedCount + chunkSize <= stringData.Length)
  {
    s = stringData.Substring(returnedCount, chunkSize);
```

```
    }
    else
    {
      s = stringData.Substring(returnedCount);
    }
    returnedCount += s.Length;
    return s;
  }
```

Now the checker should not issue any further warnings.

The solution contains the file ChunkerFinal.cs (not compiled) that contains the final code and contracts.