Code Contracts and MSDN

Mike Barnett, Tom Ball, Manuel Fähndrich, Francesco Logozzo, Wolfram Schulte, Herman Venter, RiSE

Brian Grunkemeyer, Immo Landwerth, CLR

# Add Code Contracts to every MSDN Page

We propose to add a contract section to every MSDN page for which a contract exists. The contracts are automatically extracted from the code and persisted in XML files, using a tool called *CCDoc*. The IXP tools have already been modified to render the resulting XML content into HTML MSDN pages.

## What’s in a contract?

In terms of the MSDN documentation, Code Contracts are used to express method preconditions and postconditions. Each contract contains three (possibly) pieces of information:

1. The actual condition (boolean expression) representing the contract. For preconditions, these are the properties that must hold when a client calls the method (e.g., a parameter must be non-null). Postconditions are used to describe properties of the method’s return value and any out- or ref-parameters. They can also express the state that the object is in, which might be a precondition for other methods (e.g., a file is now in the open-state and can be read from). Contracts are written in the same source language as the programmer uses for the method’s implementation. However we can render the contract in any of the .NET languages (because they are persisted as IL).
2. (*OPTIONAL*) For preconditions, the exception that will be thrown if the precondition is violated. When this is not specified, the existence of the contract makes the client aware that it is a program error to violate the precondition, but the actual runtime behavior is not known. The code is allowed to throw an exception, raise an assertion violation, or just execute with unpredictable results returned to the client. This actually gives implementers much more freedom than forcing them to do parameter validation that might be redundant and expensive.
3. (*OPTIONAL*) A string description of the contract, again authored by the contract writer.

The presence of the second and third pieces depends on how the contracts are authored. There are three overloads of the method Contract.Requires, which is used to express method preconditions and two overloads of the method Contract.Ensures, which is used to express method postconditions. So if the contracts were written with the overloads that provide the additional information then that will also show up in the documentation: all three types of information are persisted in the XML file generated by CCDoc.

## Examples

Here are some examples of the contracts for some BCL methods. Note that the contracts are not full functional specifications; they specify only part of the behavior of the methods. Of course, there is nothing that prevents writing more contracts that cover more. They are shown written in C#.

### Array.Binary Search (mscorlib.dll)

[Pure]

public static int BinarySearch(Array array, object value)

{

Contract.Requires<ArgumentNullException>(array != null);

Contract.Requires(array.Rank == 1, "The array rank must be one!");

Contract.Ensures(array.GetLowerBound(0) - 1 <= Contract.Result<int>());

Contract.Ensures(Contract.Result<int>() <= array.GetUpperBound(0));

}

In this example, the first precondition specifies the type of exception, while the second precondition specifies the description. (Of course, one could specify both in the same precondition.) Both postconditions use the simplest form that contains only the condition. The two postconditions use the method Contract.Result to refer to the return value of the method. Note that since it is a generic method of no arguments, the type instantiation must be explicitly provided. The custom attribute, Pure, is a declaration that method does not change any user-visible state. Any method used within a contract must be pure. All property getters are assumed to be pure, as per the .NET coding guidelines.

### Stack<T>.Pop (System.dll)

[Pure]

public T Pop()

{

Contract.Requires(this.Count > 0);

Contract.Ensures(this.Count == Contract.OldValue(this.Count) - 1);

}

All of the contracts in this example use the simplest form. This example shows the use of the method Contract.Old that is used in postconditions to refer to the state of the method upon entry. It is also a generic method, but since the type can be inferred from its argument, it does not need to be explicitly declared.

## How many contracts are there?

Currently, there are over 20 assemblies in the .NET Framework that have contracts. Here is a list of the ones with the most coverage:

|  |  |  |
| --- | --- | --- |
| **Assembly** | **Requires** | **Ensures** |
| Mscorlib | 1750 | 1514 |
| System | 389 | 242 |
| System.Core | 973 | 449 |
| System.Xml | 225 | 187 |
| PresentationFramework | 104 | 679 |

# Code Contracts and Exception Tables

In many situations, the information in the contracts overlaps with that in the Exception table. However, the contracts have several advantages over the information contained in the Exception table.

1. **Contracts are more precise.** For instance, the documentation for [String.Remove(int startIndex, int count)](http://msdn.microsoft.com/en-us/library/d8d7z2kk.aspx) says that an ArgumentOutOfRangeException will be thrown if “*startIndex* plus *count* specify a position outside this instance”. Does that mean that every value in between *startIndex* and *startIndex*+*count* (inclusive) must be a valid index? Or does it mean that it is a half-open interval (*startIndex* <= i && i < *startIndex*+*count*)? It turns out that it is the latter.
2. **Contracts are more accurate.** Since they are derived from the code and checked at runtime, they often reveal bugs in the human-written information. (See the Documentation Bugs section for examples.)
3. **Contracts are the correct polarity.** If a programmer wishes to introduce a runtime test to ensure they are meeting the contract, then they can cut and paste the contract into their program. The information in the exception table must be negated, which is often a cause of errors. In particular, the contracts are (usually) expressed in a way that prevents arithmetic overflow, but that could easily be introduced by an unaware programmer.

We believe that it only helps users to have both sources of information available. Eventually, it could be that the exception tables no longer need to describe logic errors, but only cover runtime failures (i.e., no ArgumentExceptions, but only things like FormatException based on bad user input).

# Support from external users

We asked several of the most active members of the community for their input about having the contracts available in the documentation.

## Ken Muse, Vice President, Technology, [ecoInsight](http://www.ecoinsight.com/)

I would like to propose the addition of code contracts to the MSDN documentation for several reasons:

* Developers understand the contract “definition” more than the current written documentation. The syntax makes sense – even to those with limited English skills.
* Code contracts are more succinct and readable than long written descriptions. They are also written in a language which all developers understand fluently – code. We have observed that offshore teams tend to have difficulty understanding the written documentation. The more detailed the contract is, the less likely they are to understand all of the complexities being communicated. We have found that they respond well to the code contracts.
* Code contracts are not open to interpretation – they are clearly written with no question in how to interpret the content.
* In our internal operations, we have found that the two most important items for understanding how to call a function are the signature and the contracts. Having those fully detailed on the MSDN site would save us time trying to re-expose that to our developers.
* Code contracts provide concise and easy to read details about what to expect in the function’s return.

## Terje Sandstrom, Chief Software Architect at [Inmeta](http://www.inmeta.com/) Consulting, AS, Norway

Two obvious reasons are that 1) it helps promote understanding of what contracts is and can do, and 2) it will help developers understand the assumptions (contracts) of the methods in the library. I do a lot of code reviews, and the developers do not always understand the underlying assumptions. Having contracts there will give a standard way of expressing those assumptions.

## Dave Sexton: MCC 2011, <http://davesexton.com/>

Adding FCL [BCL] contracts as part of the official MSDN online documentation would be of great benefit to Microsoft’s customers. As it stands now, most precondition contracts have to be inferred based on the list of exceptions for a given API or by digging through remarks. It would be much easier to understand an API if its contracts were shown as positive assertions, as opposed to negative consequences or side-effects. For example, it would be useful to see the contracts for IList<T>.Insert

Contract.Requires(!IsReadOnly)  
Contract.Requires(index >= 0)  
Contract.Requires(index <= Count)

right next to the Syntax section, rather than:

ArgumentOutOfRangeException - index is not a valid index in the IList(Of T).  
NotSupportedException - The IList(Of T) is read-only.

where an API's precondition contracts are lost in a sea of exception types and negative messages.

In other cases, contracts cannot be easily inferred from the documentation alone – they are only revealed through experimentation. For example, it would be useful to see the post-condition contracts for IList<T>.RemoveAt:

Contract.Ensures(Count == Contract.OldValue(Count) - 1)

stated explicitly near the Syntax section, rather than having to assume that RemoveAt only affects (and must affect) a single element. This behavior is currently undocumented, at least not as explicitly and succinctly as above.

Showing FCL contracts in official documentation will also guide developers to think in terms of designing by contract when writing their own APIs. DbC is a generally useful pattern for development that has been embraced at least partially with TDD, by guiding developers to think in terms of an API’s consumers. Documenting the FCL’s contracts is an important first step to introducing the concepts of DbC to all developers, regardless of whether Code Contracts is an official product or not. (But please make it official soon! ;)

Overall, adding FCL contracts adds tremendous value to your official documentation. In cases where it’s redundant, it’s clearer and more succinct to show contracts as explicit conditions. In cases where it’s not redundant, it’s useful information that would otherwise be undocumented. And ultimately, a set of documentation should guide developers to use good coding practices; thinking about an API in terms of its contracts is good practice for writing stable, manageable systems.

# Documentation Bugs

Having the contracts in addition to the existing information is a good way to find documentation bugs. This section presents a set of bugs that were found because of the discrepancies between the automatically generated contracts and the human-generated documentation. Note that these have been fixed as a result of Francesco reporting them to the documentation team.

## System.Collections.Generic.SortedSet<T>

The constructor (<http://msdn.microsoft.com/en-us/library/dd395024.aspx>) takes a parameter of type IComparer<T> and states that an exception is thrown if the argument is null, but that is not true. (Instead the default comparer is used.) On the other hand, the constructor (<http://msdn.microsoft.com/en-us/library/dd381810.aspx>) that takes a parameter of type IEnumerable<T> does not state that an exception will be thrown if the argument is null, but the code does throw such an exception. Probably the exception table for one is meant for the other?

## System.Xml.XmlConvert.ToDateTimeOffset

The documentation (<http://msdn.microsoft.com/en-us/library/bb558713.aspx>) does not state that an exception will be thrown when the first argument is null, but it does.

## System.IO.MemoryStream

There are two constructors whose exception information says that an exception is thrown when:

The sum of *index* and *count* is greater than the length of *buffer*.

But this can lead to overflow. They should be more precise, which is the case for the constructor (<http://msdn.microsoft.com/en-us/library/7y0t45kk.aspx>) which says it as:

The buffer length minus *index* is less than *count*.

(The code correctly avoids the overflow.)

## System.Math.Acos, System.Math.Asin

They had missing cases in describing their postconditions.

## System.Math.Atan2

It was missing several cases.

## System.Xml.Schema.Add

The exception table is completely wrong. <http://msdn.microsoft.com/en-us/library/system.xml.schema.xmlschemaobjectcollection.add.aspx>. It describes conditions on a parameter that isn’t even in the method.

## System.String.LastIndexOf(char)

This method used to have an exception table that said an exception would be thrown if its argument is null.

## System.String.LastIndexOf(char, int32, int32)

It was missing a conjunct for one of the exception cases.

# External projects using Code Contracts

We have started collecting a list of external projects (mostly open-source) that use Code Contracts. The up-to-date list is maintained at <http://research.microsoft.com/contracts>.

* [Autodiff](http://autodiff.codeplex.com/)
* [Boogie](http://boogie.codeplex.com/)
* [CCI](http://ccisamples.codeplex.com/)
* [Centricity Enterprise Archive](http://www3.gehealthcare.com/en/Products/Categories/Healthcare_IT/Medical_Imaging_Informatics_-_RIS-PACS-CVIS/Centricity_Enterprise_Archive) (Commercial medical DICOM\XDS archive)
* [Connected Properties](http://connectedproperties.codeplex.com/)
* [Facebook C# SDK](http://csharpsdk.org/)
* [DbExecutor](http://dbexecutor.codeplex.com/)
* [Elysium](http://elysium.codeplex.com/)
* [FT Toolkit](http://fttk.codeplex.com/)
* [Labs Framework](http://labs.codeplex.com/)
* [Mishra Reader](http://mishrareader.codeplex.com/)
* [NColony](http://ncolony.codeplex.com/)
* [Nito AsyncEx](http://nitoasyncex.codeplex.com/)
* [QuickGraph](http://quickgraph.codeplex.com/)
* [Extensions for Reactive Extensions](http://rxx.codeplex.com/)
* [Shweet](http://shweet.codeplex.com/)
* [Scrabble for WPF](http://wpfscrabble.codeplex.com/)