IT-University of Copenhagen

Student project

The Future of C5

Developing the C5 Generic Collection Library for .NET 4.0 and beyond

Author:
Rasmus Nielsen
280877-1317
rnie@itu.dk

Supervisor:
Peter Sestoft
sestoft@itu.dk

May 24, 2011

Abstract

This project aims to investigate the future of the C5 Generic Collection Library for C♯ and CLI. We move C5 to Github and .NET 4.0 and make a number of optimizations in the process. We reduce the code base by removing redundant implementations and make C5 available to Silverlight. We prepare C5 for future upgrades and uncover a number of issues which developers will have to consider when developing future versions of C5.

Contents

Chapter 1

Introduction

> Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.
>
> Antoine de Saint-Exupery (1900 – 1944)

The C5 Generic Collection Library for C♯ and CLI (C5) is a generic collection library for the .NET Framework created by Niels Kokholm and Peter Sestoft from 2003 to 2006.

In the authors' opinion "C5 provides the most powerful, well-structured and scalable generic collections library available for C♯/CLI" [Kokholm and Sestoft, 2006, page 2].

Version 1.0 of the C5 library was first published in 2006 [Kokholm and Sestoft, 2010] and has since received a small number of bug fixes.

The current version of C5 is version 1.1.1 released 17 December, 2010 [Kokholm and Sestoft, 2010].

C5 is built on .NET 2.0 and currently supports both the Common Language Runtime (CLR)[1] and Mono[2].

Microsoft has continued to expand the Common Language Runtime (CLR) with every new version of the .NET Framework and has also expanded the framework beyond the desktop – mainly to the web browser with Silverlight in 2007 and the mobile phone with Windows Phone 7 in 2010.

This also means that a lot of the innovative features of C5 are now incorporated as part of the base class library (BCL)[3]. This will allow us to take away chunks of C5 without loosing functionality.

The Mono project in turn brings .NET to the Mac OS X and Linux desktops. Mono supports C5 by bundling it as a third party API[4].

This project will establish a staging ground for the continuous development of C5 for the foreseeable future.

---

[1] The CLR is Microsoft's implementation of the Common Language Infrastructure (CLI)

[2] Mono is an open source implementation of the CLI until recently sponsored by Novell [de Icaza, 2011].

[3] The BCL is a standard library available to all languages using the .NET Framework. It is comparable to the Java Class Library and the C++ Standard Template Library.

[4] http://www.mono-project.com/Release_Notes_Mono_2.0

1.1 The users

A number of users have suggested new features and enhancements for C5. These are:

- Alex Rønne Petersen
- Henrik Feldt
- Marcus Griep
- Kasper Overgård Nielsen
- Jack Addington
- Keith (surname unknown)

We will treat a number of these suggestions throughout the course of the project.

When their names are mentioned in the following text, it refers to email correspondence between them and the original authors of C5.

1.2 Project goals

This project has a number of goals for the future of C5 which concern many spectrums of software development: Availability, maintainability, upgradability, portability, and testability. The goals are:

1. Make C5 available through online open source repositories and package managers.
2. Upgrade the version control system (VCS) of C5 to a modern platform for enabling effective branching and development.
3. Upgrade C5 to .NET 4.0 and make it easier to support upcoming versions of the CLR and Mono.
4. Make C5 available to multiple platforms including Silverlight, Windows Phone, and Mono.
5. Consider the use of a continuous integration server to make building and testing C5 more efficient.
6. Discuss the majority of the incoming feedback, suggestions, and bug reports received from the users since C5's inception.

1.3 Modus operandi

The original authors of C5 have gone to great lengths to test every aspect of C5 during the development. The library includes over 1400 unit tests to prove this.

In order to secure the stability of the library going forward it is therefore very important to observe that these tests do not break, and whether they do provide a way to fix this.

This project aims to plan for creating a *future* version of C5, hence backwards compatibility will not take precedence over goals like Common Language Specification (CLS) compliance and adherence to newer .NET naming conventions.

This project is a precursor for a planned master's thesis on the future of C5. The plan is to execute the pending goals and objectives found throughout this project during the course of future thesis.

However, this project has become more than a mere thought experiment, as it quickly became clear that it would be more feasible to complete many of the goals during the project as oppose to postponing them for the thesis.

1.4 A note on continuity

The project report is divided into chapters where similar objectives are treated as a whole. The path to version 2.0 of C5 has been more rugged, which is why the reader will find C5 2.0 published to NuGet in section 2.2.1 before the library is retargeted to .NET 4.0 in section 4.2 and ported to Silverlight in chapter 6.

Chapter 2

Availability

To ensure the success of C5 it is paramount that the community and the potential consumers of C5 have easy access to information about C5, news, and the current build of the library.

Currently C5 resides on a static web page on the IT-University of Copenhagen's (ITU) servers at: http://www.itu.dk/research/c5/.

The source code is controlled using the aging Concurrent Versions System (CVS)[1] on an internal server at the Faculty of Life Sciences at Copenhagen University (KU LIFE)[2].

C5 is only available by downloading either a compiled .dll or the source from the aforementioned ITU web site.

2.1 Upgrade version control system and make C5 publicly available

The authors have received a number of suggestions to move the trunk of C5 to a publicly available server. Some have even made an SVN server available, others have forked C5 and uploaded the source to Github repositories without the author's consent[3].

Since C5 is published under an open source license [Sestoft and Kokholm, 2007] it would seem like a natural step forward to make the library publicly available through one of the free open source hosting facilities which have sprouted on the Internet in recent years.

Currently there are a number of providers of open source hosting, most notably:

- Github at github.com using Git[4].
- Bitbucket at bitbucket.org using Mercurial[5].

[1] http://savannah.nongnu.org/projects/cvs

[2] Cf. Peter Sestoft 15 April 2011

[3] Cf. Email from Henrik Feldt to Peter Sestoft, 15 November, 2009 and email from Marcus Griep to Peter Sestoft, 22 March, 2010.

[4] http://git-scm.com

[5] http://mercurial.selenic.com

- SourceForge at sourceforge.net using either Git, Mercurial or Apache™ Subversion® (SVN)[6].
- Google Code at code.google.com using either Mercurial or SVN.
- CodePlex at codeplex.com using either Mercurial, SVN, or Microsoft Team Foundation Server (TFS)[7].

Discussing the advantages and disadvantages of these is beyond the scope of this report. However, we recommend the "Comparison of open source software hosting facilities" article on Wikipedia [Wikipedia, 2011c] for further information.

Conclusion

Due to the vibrant and very large community of Github [Github, 2011; Wikipedia, 2011c] and our previous experiences with Git, we have decided to move C5 to Github.

C5's new home is: https://github.com/sestoft/C5.

2.2 NuGet

In late 2010 Microsoft launched an open source package manager called NuGet [Guthrie, 2010]. The goal of NuGet is to make the process of incorporating open source libraries into a solution as easy as "Add Reference" [Hanselman and Haack, 2011][8] using either a graphical user interface or a PowerShell-based console from inside Visual Studio 2010.

NuGet makes it trivial for a consumer of open source libraries to keep up to date with the latest versions. Dependencies are also resolved automatically much like using `apt-get` on a Debian-based Linux distribution[9].

If the NuGet C5 package is called "C5" one can install it and use it in a project by typing `Install-Package C5` in the Package Manager Console[10].

2.2.1 Publishing C5 to NuGet

Having created a NuGet account on http://nuget.org and installed the NuGet Package Explorer and the NuGet Command Line executable from http://nuget.codeplex.com publishing to NuGet is a straight-forward process:

1. `> nuget pack C5.csproj -symbols`

[6] http://subversion.apache.org

[7] http://www.microsoft.com/visualstudio/en-us/products/2010-editions/team-foundation-server

[8] NuGet also supports in-house closed-source development. For an entertaining in-depth introduction to NuGet the author would recommend a presentation at MIX 11 by Scott Hanselman and Phil Haack [Hanselman and Haack, 2011]

[9] http://wiki.debian.org/Apt

[10] The C5 source depends on NUnit, however it is not supplied with the source. I was able to bring it down by just typing `Install-Package` which allowed the C5 source to compile without errors.

2. Open the generated `C5.2.0.nupkg` and `C5.2.0.symbols.nupkg` in the NuGet Package Explorer to edit metadata and fix assembly references for Silverlight and Windows Phone 7. See figure A.5 for details.

3. `> nuget push C5.2.0.nupkg`

Ad 1: The `-symbols` switch creates a symbols package which is uploaded to http://www.symbolsource.org/. This allows the users of C5 to debug straight into C5 as if the source was on their own machine. To do this two settings must be changed in Visual Studio cf. figure A.7 and A.8 [Hanselman and Haack, 2011].

Conclusion

C5 is now available through NuGet at http://nuget.org/List/Packages/C5.

It can be referenced using the "Add Library Package Reference" dialog as shown in figure A.6 or by typing:

```
PM> Install-Package C5
```

in the "Package Manager Console".

Chapter 3

Maintainability

3.1 File structure

The current source for C5 is organized in very few, but very very long files – some of them thousands of lines long.

Many files contain multiple classes, structs, interfaces, delegates, and enumerations.

This makes the code very hard to read and also hard to maintain.

For a future version of C5, we move to refactor the code by putting one and only one class, struct, interface, delegate, or enumeration in each file.

These files can then be organized into folders instead.

3.2 Naming conventions

Throughout C5 a multitude of naming conventions have been used.

There a number of public methods which internally call methods with the same names but in lower case. This practice ought to be abandoned in future releases.

Method names were devised before the advent of LINQ and many even before System.Collections.Generic. Therefore, there are naturally clashes as shown in table 3.1.

Table 3.1: Differences between C5 and CLR method names

C5	CLR
void AddAll(IEnumerable<T> items)	void AddRange(IEnumerable<T> collection)
void Reverse() IDirectedEnumerable<T> Backwards()	void Reverse()
bool Exists(Func<T, bool> p)	bool Exists(Predicate<T> match) bool Any(Func<T, bool> p)*
bool IsEmpty**	bool Any()*
IEnumerable<T> Filter(Func<T, bool> p)	IEnumerable<T> Where(Func<T, bool> p)*
void Apply(Action<T> action)	void ForEach(Action<T> action)

* LINQ extension method

** lst.IsEmpty ∼ !lst.Any()

Refactoring C5 by renaming these methods and properties will introduce breaking changes. Also, there are subtle differences between `Reverse` and `Backwards` which cannot be ignored.

Regarding `Exists`, the .NET Framework counterpart only exists on `System.Collections.Generic.List<T>` and it is identical in functionality to the LINQ extension method `Any`. The C5 implementation of `Exists` is time asymptotically linear in the number of items so performance wise it should be on par with `Any`. `Exists` could therefore be safely removed.

Regarding `Filter`, it is identical to the LINQ extension method `Where`. It could also be safely removed.

One might argue that some of the names in C5 are better than what Microsoft came up with for `System.Collections.Generic` and `System.Linq` – `Filter` vs `Where` and `Apply` vs `ForEach` – but the Microsoft names are part of an industry standard (and there is no way we can change those), so it would be better for the consumer of C5 if names where the same.

For future versions of C5 we recommend synchronizing names between C5 and the .NET Framework and removing redundant implementations.

3.3 Testing

The unit tests for C5 are very extensive and very exhaustive.

There are, however, a number of issues which could be optimized for future versions:

1. Tests do not follow the common AAA pattern (Arrange, Act, Assert) (Marcus Griep).
2. Some tests contain NO assertions
3. Many tests contain MULTIPLE assertions
4. Test names are very generic and non-descriptive
5. Some tests are very complex and there is a host of support classes for the test project.

Ad 4: We suggest using a three part naming convention for unit tests: MethodUnderTest_Scenario_ExpectedResult [Osherove, 2009, 7.3.1]. An example can be seen in listing 3.1.

Listing 3.1: Naming unit tests

```
[TestFixture]
public class HashSetTests
{
    [Test]
    public void Constructor_EqualityComparerIsNull_ExceptionIsThrown()
    {
        Assert.Throws<ArgumentNullException>(() => new HashSet<int>(null));
    }
}
```

Using this naming convention will make it very easy to read the outputs from a test runner and one can instantly see where and what the problem is if a test breaks.

Ad 5: In future versions this could maybe be solved by introducing parameterized tests (`[RowTest]` in NUnit) [Osherove, 2009, 7.2.5].

It would be a huge undertaking to change all the existing unit tests, but it is important to deal with this problem in future versions of C5.

3.4 License

All the source files in C5 contain a copy of the license. Although the license is not subject to change it is still cumbersome to maintain with many source files – especially when applying the refactoring proposed in section 3.1.

We have opted to remove the license in the future and just add two lines referring to the license file instead – see listing 3.2.

Listing 3.2: License header

```
// This file is part of the C5 Generic Collection Library for C# and CLI
// See https://github.com/sestoft/C5/blob/master/LICENSE.txt for licensing details.
```

3.5 Conclusion

The preceding section describes objectives which should be observed when writing a future version of C5. We will not go about refactoring the existing code base just for the sake of refactoring it.

Chapter 4

Upgradability

The major reason for revisiting C5 is to upgrade the library to .NET 4.0 and enable the use of all the new features of the framework.

We do not aim to be backwards compatible with older versions, which means that users which have not yet migrated to .NET 4.0 will have to continue using the 1.0 branch of C5.

4.1 Overview of the new features in C♯ 3.0 and 4.0

4.1.1 C♯ 3.0 features

C♯ 3.0 introduces a number of new features as shown in table 4.1 [Wikipedia, 2011a].

Table 4.1: C♯ 3.0 features

LINQ (Language-Integrated Query)
Object initializers
Collection initializers
Anonymous types
Local variable type inference
Lambda expressions
Expression trees
Automatic properties
Extension methods
Partial methods

Most of these features provide "syntactic sugar" for C♯ that allow developers to write cleaner and more readable code – foreach statements can be replaced by LINQ, delegates by lambda expressions, and `public readonly` fields can be replaced by automatic properties.

C5 should be upgraded to take advantage of some or all of these features in order to provide a cleaner source code that is easier to read, test, and maintain.

The most interesting feature of C♯ 3.0 is arguably LINQ which provides a SQL-style way to query a database, XML file, or object collection. As C5 is created for object collections it is paramount that future versions of C5 support these features.

4.1.2 C♯ 4.0 features

C♯ 4.0 introduces a number of new features as shown in table 4.2 [Wikipedia, 2011b].

Table 4.2: C♯ 4.0 features

Dynamic member lookup
Covariant and contravariant generic type parameters
Optional parameters and named arguments

The interface `IEnumerable<T>` in .NET 4.0 has been redefined to `IEnumerable<out T>`. This means that every class that implement `IEnumerable<Derived>` for some sub class derived from `Base` is compatible with `IEnumerable<Base>`.

In the current version of C5 the authors have previously allowed this by adding extra type parameters in a number of interfaces like: `void AddAll<U>(IEnumerable<U> xs)where U:T`. With .NET 4.0 this can now be written as `void AddAll<T>(IEnumerable<T>)` without loosing any functionality.

4.2 Retargeting C5 to .NET 4.0

In order to retarget C5 to .NET 4.0 we will follow these steps:

1. Create a new branch: `git branch net4`
2. Switch branch: `git checkout net4`
3. Open C5 in Visual Studio
4. Build the project
5. Run all unit tests
6. Change target type of all projects in the C5 solution to ".NET Framework 4.0 Client Profile", secondarily ".NET Framework 4.0"
7. Rebuild the project
8. Run all unit tests
9. Commit the changes: `git commit --all --message "Upgraded to .NET 4.0"`
10. Push everything up to Github: `git push --all`

Ad 4: The project cannot be built as it lacks a reference to NUnit[1]. With NuGet in place this is resolved very quickly through the *Package Manager Console*: `PM> Install-Package NUnit`.

[1] http://www.nunit.org/

Ad 5: There are some initial problems with the unit tests of C5. 1435 of 1438 tests pass which is actually very good. We just need it to be 100% going forward.

Figure A.1 shows the initial output of the NUnit test runner.

First, there are the three failing tests. They test the behavior of enumerating a `HashDictionary<K, V>` and assume that the output of enumerating the contents of a dictionary will be yielded in a specific order. The failing tests are shown in listing B.1. We have been unable to find anything in the specification which would require this, furthermore it would seem like a very dangerous thing to assume while using the interface, as a dictionary normally is agnostic to the order of its content[2].

To fix the failing tests and actually test what is relevant, the tests have been rewritten as shown in listing B.2.

Second, there are a number of methods which have been erroneously marked with a `[Test]` attribute. These are methods belonging to the `C5UnitTests.Templates.Events` namespace, which will be called by other tests but which are not unit tests themselves. See 3.3 for an in-depth discussion of this.

This means that the `[Test]` attributes should not have been there in the first place, and we therefore remove them.

Ad 6: All projects could be safely retargeted to the ".NET Framework 4.0 Client Profile". C5 can therefore be used in client applications on computers which do not have the full .NET Framework installed. This includes the vast majority of desktops as Microsoft has pushed the ".NET Framework 4.0 Client Profile" as a recommended update for some time[3].

Ad 8: Figure A.2 shows the final output of the NUnit test runner.

Conclusion

The upgrade went pretty smoothly and we can now start using all the new features of C♯ 3.0 and 4.0 in the C5 implementation.

As an added bonus we get LINQ for free in all collections, which implement `IEnumerable<T>`.

[2] C5 provides an `ISortedDictionary<K, V>` interface for this purpose, which the `HashDictionary<K, V>` does **not** implement

[3] http://support.microsoft.com/kb/982670

Chapter 5

Portability

In this chapter we will discuss porting C5 to other platforms and ensuring CLS compliance to make the library callable from other languages on the .NET Framework like IronPython, IronRuby, and F♯.

5.1 Other platforms

Users have requested that C5 be ported to different platforms like Silverlight (Jack Addington) and the .NET Compact Framework (Kasper Overgård Nielsen).

5.1.1 Silverlight

"Microsoft Silverlight is an application framework for writing and running rich Internet applications, with features and purposes similar to those of Adobe Flash" [Wikipedia, 2011f].

Silverlight supports only a subset of the .NET Framework and one can therefore often not use a library in Silverlight which is not specifically targeting it.

Conclusion

The C5 library does not target Silverlight and it uses some features not available to Silverlight. In the following sections we will discuss whether these features can be omitted or rewritten in order for C5 to be used in a Silverlight project.

5.1.2 Windows Phone 7 and the .NET Compact Framework

"Windows Phone 7 (WP7) is a mobile operating system developed by Microsoft, and is the successor to its Windows Mobile platform" [Wikipedia, 2011g].

WP7 is built upon the .NET Compact Framework (.NET CF) [Kidambi, 2010] which "is a version of the .NET Framework that is designed to run on resource constrained mobile/embedded devices such as personal digital assistants (PDA's), mobile phones, factory controllers, set-top boxes, etc." [Wikipedia, 2011d]

WP7 supports Silverlight and Microsoft XNA, which is the Microsofts game development platform, however WP7 only implements a subset of Silverlight and is thus even more limited in functionality than Silverlight itself compared to the .NET Framework.

Conclusion

The C5 library does not target WP7. In the following sections we will discuss whether these features can be omitted or rewritten in order for C5 to be used in a WP7 Silverlight or XNA project.

5.1.3 Mono

"Mono is an open source implementation of Microsoft's .NET Framework based on the ECMA standards for C♯ and the Common Language Runtime." [Mono, 2011].

Mono runs on multiple platforms, including Microsoft Windows, Linux, Mac OS X, and mobile platforms like iOS and Android, and the browser with an open source implementation of Silverlight called Moonlight.

Conclusion

C5 in its current version can already be built on Mono and the C5 will continue to support Mono in the future.

5.1.4 .NET Micro Framework

"The .NET Micro Framework (.NET MF) is an Open Source .NET platform for resource-constrained devices with at least 256 KBytes of flash and 64 KBytes of RAM." [Wikipedia, 2011e]

The current version of C5 is *only* about 300 Kb, but given the constraints of .NET MF C5 would have to be reduced by a factor 10 to be usable.

Furthermore, .NET MF implements only a small version of the .NET CLR and has not yet seen wide usage.

Conclusion

The scope of C5 will not currently take NETMF into consideration.

5.1.5 Portable Library Tools

Microsoft has recently released a new extension to Visual Studio 2010 called the "Portable Library Tools" (PLT) [Microsoft, 2011]. PLT aims to make it easier to built cross-platform libraries that are consumable by different implementations of the CLR, including .NET, Silverlight, Windows Phone, and Xbox 360.

This is done by automatically restricting the available namespaces in the project based on which platforms the developer has chosen to target.

For an in-depth demonstration of PLT see [Burke, 2010]

Conclusion

Future versions of C5 will be built using PLT for maximum portability.

5.2 CLS compliance

A goal for the authors of C5 is for the library to be compliant with the Common Language Specification (CLS) [Kokholm and Sestoft, 2006, p. 1] and therefore usable for any platform implementing the Common Language Infrastructure (CLI).

A future version of C5 should therefore also be CLS compliant.

To declare a project to be CLS compliant one adds the line `[assembly: CLSCompliant(true)]` to the `AssemblyInfo.cs` file of a project.

Compiling C5 produces a number of *not* CLS compliant warnings (Marcus Griep) which we will discuss in the following.

5.2.1 Non-CLS compliant types

The default comparers and equality comparers of `Builtin.cs` implement comparers for the standard .NET value types and the C♯ types `sbyte`, `ushort`, `uint`, and `ulong`, which are not CLS compliant.

To solve this there are two options:

1. Decorate the offending classes with the `[CLSCompliant(false)]` attribute
2. Remove the offending classes completely

According to [Microsoft, 2010a] when applying the `[CLSCompliant(false)]` attribute one must provide a compliant alternative but since CLS compliant languages do not know the types `sbyte`, `ushort`, `uint`, and `ulong`, they do not need alternatives.

Conclusion

The `[CLSCompliant(false)]` attribute has been applied to the classes[1]:

- `SByteEqualityComparer`
- `UShortEqualityComparer`
- `UIntEqualityComparer`
- `ULongEqualityComparer`

5.2.2 Non-CLS compliant overloads

The overloaded methods:

- `IDictionary<K,V>.Find(K key, out V value)`
- `IDictionary<K,V>.Find(ref K key, out V value)`

are not CLS compliant as they differ only on whether or not `key` is a `ref` parameter.

To solve this there are at first glance three options:

[1] The classes `SByteComparer`, `UShortComparer`, `UIntComparer`, `ULongComparer` are also not CLS compliant, but their visibility is *internal* so CLS compliance is irrelevant [Ecma, 2010, section 7.3, CLS Rule 1] .

1. Decorate one of the offending methods with the `[CLSCompliant(false)]` attribute
2. Remove one of the offending methods completely
3. Change the name of one of the offending methods so it is no longer an overload

Ad 1: According to [Ecma, 2010, section 8.9.4, CLS Rule 18]: *"CLS-compliant interfaces shall not require the definition of non-CLS compliant methods in order to implement them."*. Rule 18 effectively invalidates this option.

Ad 2: At first, this option seems a bit drastic, however the methods are very much the same. Also, there is only one implementation of this in `DictionaryBase<K, V>` where one of the methods will internally just call the other as seen in listing 5.1.

Listing 5.1: Non CLS compliant overloads of Find methods

```
public virtual bool Find(K key, out V value)
{
    return Find(ref key, out value);
}

public virtual bool Find(ref K key, out V value)
{
    KeyValuePair<K, V> p = new KeyValuePair<K, V>(key);

    if (pairs.Find(ref p))
    {
        key = p.Key;
        value = p.Value;
        return true;
    }
    else
    {
        value = default(V);
        return false;
    }
}
```

Ad 3: Changing the name of one of the Find methods to something more verbose will in any case be a breaking change. From our perspective it is already confusing to have two identically named methods that do almost the same. One will have to read the documentation carefully to choose the right one. This confusion is unlikely to diminish if one of the methods where to be called something like `FindWithOutRef` or even `Find2` instead.

Conclusion

We have removed the version without `ref` – `Find(K key, out V value)`.

Chapter 6

Building C5 as a portable library

Having installed the PLT as described in section 5.1.5 we can now continue to investigate building C5 for multiple platforms.

We initially created a new portable library and set it to target all available platforms – .NET Framework, Silverlight 4, Silverlight for Windows Phone 7, and XNA Framework 4.0 for Xbox 360.

Having done this reveals a host of problems that we will solve in this chapter.

6.1 System.Serializable

Problem: The `[Serializable]` attribute is not supported.

Solution: Remove the `[Serializable]` attribute completely.

If we consume C5 in a platform that supports serialization we can always convert the collection to one of the built-in generic collection types like `T[]`, `IEnumerable<T>`, and `IDictionary<T>`, provided `T` is serializable.

First, this means that we can convert any given collection to a built-in counterpart and serialize that – doing so does imply a small performance penalty, but this will at most time asymptotically linear in the number of items.

Second, one of the most common uses of serialization is for interoperability between different platforms and services, like returning a collection of objects from a web service using either XML or JSON. If we were to return a C5 collection this would enforce an extra – and most probably unnecessary – dependency on the caller.

Conclusion

We have removed the serialization attribute and thus removed the ability to directly serialize a C5 collection.

6.2 System.Comparison

Problem: The `Comparison<T>` delegate is not supported.

The `Comparison<T>` delegate is only used in a support class – `DelegateComparer` – that can create an `IComparer<T>` from a `Comparison<T>`.

Conclusion

Cf. section 7.10, we will remove `DelegateComparer` and replace it with a more useful `ComparerFactory<T>`.

6.3 System.ICloneable

Problem: The `ICloneable` interface does not exist.

Cloning an `IExtensible<T>` can be done in two ways as seen in listings 6.1 and 6.2:

Listing 6.1: Cloning a collection using ICloneable

```
var newCollection = (IList<string>) oldCollection.Clone();
```

Listing 6.2: Cloning a collection manually

```
IList<string> newCollection = new LinkedList<string>(oldCollection);
```

According to [Kokholm and Sestoft, 2006, secion 8.9] the automatic version is "usually more efficient" but it does require the developer to use an unsafe cast.

Furthermore a number of people, including Brad Abrams – one of the original designers of the .NET Framework and the CLR – consider the use of `ICloneable` to be bad coding practice mainly because the interface does not specify whether to create a *deep copy* or a *shallow copy* [Abrams, 2003, 2004]. This is probably the main reason why `ICloneable` is not available in Silverlight and XNA.

Conclusion

We have removed all implementations of `ICloneable`.

6.4 System.Console

The C5 library contains a large number of `Console.WriteLine` statements presumably left by the developers for logging purposes.

If one consumes the C5 library with an ASP.NET or Windows Presentation Foundation (WPF) application this will go about unnoticed, however, if one is writing a Console application these logging statements will propagate to the UI.

Instead of using `Console.WriteLine` one could use `System.Diagnostics.Debug.WriteLine`. This method will write the output to an instance of `System.Diagnostics.DefaultTraceListener`↩ if one is supplied by the consumer of the library.

Alternatively, the C5 library could incorporate an observer pattern [Gamma et al., 1994, p. 293], allowing the users of the library to choose a logging framework of their own.

This could be done simply by attaching a static singleton [Gamma et al., 1994, p. 127] of type Action<string> as shown in listing 6.3.

Listing 6.3: Simple logging observer

```
using System;

namespace C5
{
    /// <summary>
    /// Logging module
    /// </summary>
    public static class Logger
    {
        private static Action<string> _log;

        /// <summary>
        /// Gets or sets the log.
        /// </summary>
        /// <example>The following is an example of assigning a observer to the logging module:
        ///   <code>
        ///     Logger.Log = x => Console.WriteLine(x);
        ///   </code>
        /// </example>
        /// <remarks>
        /// If Log is not set it will return a dummy action
        /// <c>x => { return; })</c>
        /// eliminating the need for null-reference checks.
        /// </remarks>
        /// <value>
        /// The log.
        /// </value>
        public static Action<string> Log
        {
            get { return _log ?? (x => { return; }); }
            set { _log = value; }
        }
    }
}
```

And afterwards simply replacing all occurences of Console.WriteLine with Logger.Log.

This would allow the consumer of the library to choose if and how to log diagnostic messages, using the likes of Elmah[1] in a web application and Apache log4net[2] otherwise.

To crudely redirect all logging to the console one would simply write Logger.Log = x => Console.WriteLine(x);.

Ultimately it might be desirable to use a real pluggable logging architecture, which might be implemented using the Managed Extensibility Framework (MEF)[3], albeit MEF is currently not supported on Windows Phone 7 and Xbox 360.

Conclusion

We introduce the logger from listing 6.3 and replace all calls to Console.WriteLine with Logger.Log.

We will enter a suggestion on Github about using MEF for logging.

[1] Error Logging Modules and Handlers for ASP.NET: http://code.google.com/p/elmah
[2] http://logging.apache.org/log4net
[3] http://mef.codeplex.com

6.5 System.Exception

Problem: `System.Exception` is now an `abstract` class.

This is actually a very bold and interesting move by Microsoft which forces a developer to always throw a specific exception.

Upon closer inspection all instances of `System.Exception` in C5 are calls like this: `throw new Exception("The method or operation is not implemented.");`.

These calls should have been `throw new NotImplementedException();` in the first place.

Conclusion

All occurrences of `System.Exception` have been replaced by more specific exceptions.

6.6 System.Type.EmptyTypes

Problem: The `Type.EmptyTypes` property does not exist.

`Type.EmptyTypes` is just a clean way of writing `new Type[0]`.

Conclusion

We have replaced `Type.EmptyTypes` with `new Type[0]`.

6.7 System.Runtime.CompilerServices.Runtime-Helpers.GetHashCode

Problem: The `ReferenceEqualityComparer<T>` uses `RuntimeHelpers.GetHashCode(object o)`, which is not available in Silverlight.

The `RuntimeHelpers.GetHashCode` method always calls `Object.GetHashCode` non-virtually, even if the type has overridden the `Object.GetHashCode` method [Microsoft, 2010c].

The ReferenceEqualityComparer has been included in C5 to allow enforcing a strict reference equality comparison when a type has overridden `GetHashCode()`[4].

The original authors are not aware that ReferenceEqualityComparer is used in production anywhere[5] and since it is a helper class it can be safely removed (and implemented by the consumer of C5 if needed).

Note: As we will describe in section 7.10, we have supplied a new `ComparerFactory<T>` which makes the use of custom comparers and equality compares much more simple using inline lambda expressions instead of custom implementations of `EqualityComparer<T>`.

Conclusion

We have removed the `ReferenceEqualityComparer<T>` class.

[4] Cf. email from Peter Sestoft to Niels Kokholm 7 December, 2004.
[5] Cf. Peter Sestoft 6 May, 2011

6.8 System.Reflection.Emit

Problem: There is a great amount of *experimental* code in C5 using Reflection.Emit to generate byte code at runtime. Although reflection is supported on all platforms it is not yet available in a portable library.

However, since the code is only used when compiled with an *experimental* flag, it is not needed.

Conclusion

We have removed all `#if EXPERIMENTAL` sections from the source.

6.9 System.Reflection.MethodBase

Problem: There are a couple of calls to `MethodBase.GetCurrentMethod()` in some internal diagnostics code of the `LinkedList<T>` class.

Inspecting the code it seems the author intended to propagate the name of the method causing the problem back to the caller.

First, according to [Microsoft, 2010b] `MethodBase.GetCurrentMethod()` always returns the name of the innermost method – it has no way of knowing who started the process (ultimately `void Main(string [] args)`!), so there is no point in using it in this context. The line:

```
throw new InternalException(MethodBase.GetCurrentMethod()+ "called on a view");
```

will always evaluate to:

```
throw new InternalException("Boolean checkViews()called on a view");.
```

Second, the way the .NET Framework handles exceptions one will (almost) always be able to debug the code and find the caller causing the exception quite easily – especially if employing *Intellitrace*[6].

Conclusion

We have updated the code to read `throw new InternalException("checkViews()called on a view");`.

6.10 Covariance and contravariance

In sections 7.4 and 7.5 we will discuss the .NET 4.0 optimizations for covariant and contravariant type parameters. Covariance and contravariance are also available in the portable library tools, however, for Microsoft forgot to declare `IEnumerable<T>` as covariant in Silverlight.

This has been fixed in Silverlight 5, but at the time of writing Silverlight 5 is only in beta and it will probably take a while before the update propagates to Windows Phone 7 and Xbox 360.

Conclusion

We have reverted the changes made in section 7.5.

[6] http://msdn.microsoft.com/en-us/magazine/ee336126.aspx

6.11 Conclusion

After all the above changes have been made we are able to convert C5 to a portable library and C5 now supports Silverlight, Windows Phone 7, and Xbox 360 development.

Chapter 7

Features

In this chapter we will discuss implementing a number of the features suggested by the users including some of our own ideas.

7.1 Replace C5 specific delegates with generic .NET types

C5 defines a number of delegates in `Delegates.cs`, namely `void Act(A1 x1)` to `void Act<A1, A2, A3, A4>(A1 x1, A2 x2, A3 x3, A4 x4)` and `R Fun<R>()` to `R Fun<A1, A2, A3, A4, R>(A1 x1, A2 x2, A3 x3, A4 x4)`.

These delegates precede the generic delegates `System.Action<in T>`[1] and `System.Func<TResult Func<out TResult>` introduced in .NET 3.5[2], however, they provide the exact same functionality. It has been noted by both the authors in the comments of the `Delegates.cs` file and by the users (Alex Rønne Petersen, Keith, Marcus Griep) that these constructs are redundant. They have remained in the library for backwards compatibility reasons.

The future version of C5 does not aim to be backwards compatible, which allows us to remove the `Act` and `Fun` delegates completely after properly updating the affected methods to use `System.Action` and `System.Func` instead.

7.1.1 Implementing System.Action/System.Func

To implement `System.Action` and `System.Func` we will:

1. Rename all `Act` delegates in `Delegates.cs` to `Action`
2. Rename all `Fun` delegates in `Delegates.cs` to `Func`
3. Delete the `Delegates.cs` file.
4. Rebuild solution
5. Run all unit tests

[1] `System.Action<T>` was introduced in .NET 2.0 but available with one and only one type parameter.

[2] The `in` and `out` co- and contravariant type parameters where added in C♯ 4.0

Conclusion

After doing some manual edits – primarily inserting appropriate `using` statements – We were able to build the solution and run the unit tests.

All tests passed and the new version has been committed to Github.

7.2 Crude timer

The C5 User Guide Examples library is practically littered with redundant classes called `Timer` (Henrik Feldt). These timer classes provide a crude stopwatch implementation using `System.DateTime`.

Microsoft already solved this problem by adding a `System.Diagnostics.Stopwatch` class in .NET 2.0.

As such there is no need for a redundant and crude timer class anywhere in C5.

Conclusion

We have replaced all the crude timer classes with stopwatches.

7.3 Updating the online documentation of C5

The C5 source includes a documentation project called *docNet*. It is a console application which can build an HTML documentation site for C5.

Microsoft has been working on a documentation tool called *Sandcastle*[3] since around 2006 [Wikipedia, 2010]. The project was officially released to the web (RTW) in January 2008. Sandcastle can build different types of documentation directly from the `.dll` and `.xml` files of a .NET project. Sandcastle is, however, quite difficult to use and lacks documentation, so the community has afterwards built and released the *Sandcastle Help File Builder*[4] which greatly simplifies the process.

See figures A.3 and A.4 for the output of running C5 through Sandcastle.

Conclusion

The website and help file outputs are nearly identical and very well structured. As a consequence we have chosen to remove the *docNet* project completely and have included a Sandcastle Help File Builder project instead – `C5.shfbproj`.

The HTML documentation will be made available online on the C5 home page later.

7.4 Redundant type constraints

In C5 there are a number of *Generic bulk methods* [Kokholm and Sestoft, 2006, 8.4]. These methods have generic type constraints which were inserted to overcome C♯ 2.0's lack of covariant generic type parameters.

[3] http://sandcastle.codeplex.com
[4] http://shfb.codeplex.com

C♯ 4.0 introduces covariant and contravariant generic type parameters – the `IEnumerable<T>` interface is now declared as:

```
public interface IEnumerable<out T> : IEnumerable.
```

Conclusion

We have removed all redundant type constraints:

`void AddAll<U>(SCG.IEnumerable<U> xs)where U : T` is changed to `void AddAll(SCG.IEnumerable<T> xs)`

The same optimization is applied to `AddSorted`, `ContainsAll`, `ContainsAny`, `InsertAll`, `RemoveAll`, and `RetainAll`.

Note: `void IDictionary<K, V>.AddAll<U, W>(SCG.IEnumerable<KeyValuePair<U, W>> entries)where U : K where W : V;` has not been changed as `KeyValuePair` is not, and cannot be covariant.

7.5 Covariance and contravariance

The interface `IDirectedEnumerable` can be declared as covariant: `IDirectedEnumerable<out T>` as it extends from `IEnumerable<out T>`.

7.6 Eliminating the preprocessor

C5 contains a number of collections which are so similar that the original authors have chosen to implement them in the same files using preprocessor directives – otherwise knows as *ifdefs*.

The collections are:

- `ArrayList` and `HashedArrayList`
- `LinkedList` and `HashedLinkedList`
- `RedBlackTreeSet` and `RedBlackTreeBag`

Using compiler directives this way has generated the need for a *preprocessor* implemented as a console application which must run before compiling C5.

Bjarne Stoustrup deliberately misquotes Cato in "The Design and Evolution of C++":

> Furthermore, I am of the opinion that Cpp must be destroyed
>
> Cato the Elder (Marcus Porcius Cato)
> [Stroustrup, 1994, ch. 18]

Before dedicating a whole chapter to why we must avoid using the preprocessor if at all possible.

Apart from being a security issue as noted by Stroustrup, some of the major problems using compiler directives in C5 are that they make the code harder to *read*, *write*, *test*, and *maintain*.

Generally we believe that the use of compiler directives is not necessary and should be avoided at all costs. In a high level language like C♯ one should be able to solve these problems much more elegantly using class inheritance and dependency injection [Fowler, 2004].

Conclusion

The preprocessor has been eliminated.

We need to consider refactoring the list/bag classes to eliminate the code duplication introduced in this operation. This could be partially done by extracting methods to a super class but it would probably not suffice[5].

7.7 Removing preprocessor directives

Following section 7.6 and cf. Peter Sestoft 15 April, 2011 and 26 April, 2011 there are a number of preprocessor directives which can be safely removed from C5:

1. `NCP` is always `true`
2. `SEPARATE_EXTRA` is always `false`
3. `EXPERIMENTAL` is always `false`
4. `LINEARPROBING` is always `false`
5. `REFBUCKET` is always `true`
6. `SHRINK` is always `false`
7. `INTERHASHING` is always `false`
8. `RANDOMINTERHASHING` is always `true`
9. `IMPROVED_COLLECTION_HASHFUNCTION` is always `true`
10. `MAINTAIN_SIZE` is always `true`
11. `EXPERIMENTAL` is always `false`
12. `STRONGNAME` is always `false`

All directives except `STRONGNAME` regard different implementations tested during development.

`STRONGNAME` allows for compilation with a strong name key. Since the code is open source, if a consumer needs a strong named version, they can download the source and compile it themselves.

Regarding `EXPERIMENTAL`, see section 6.8.

Conclusion

To simplify the code and build configuration we have opted to remove all preprocessor directives.

[5] Cf. Peter Sestoft 15 February, 2011

7.8 DEBUG symbol

The `DEBUG` symbol was used in the `HashSet<T>` class to allow for deterministic hashing in debug builds. This was used to test the hash collections – `HashSet<T>` and `HashBag<T>`.

We have removed the `DEBUG` symbol by:

1. Adding a `Debug` class as seen in listing 7.1[6].
2. Adding the line `[assembly: InternalsVisibleTo("C5.Tests")]` to `AssemblyInfo.cs`.
3. Modified the constructor of `HashBag<T>` to generate a deterministic hash factor `if (Debugging.UseDeterministicHashing)`.
4. Modified affected unit tests to call `Debugging.UseDeterministicHashing = true;` in their `[SetUp]` methods.

This does solve one problem, but creates a new one: now we have deferred a debug check to runtime instead of compile-time.

Listing 7.1: Debug class

```
// This file is part of the C5 Generic Collection Library for C# and CLI
// See https://github.com/sestoft/C5/blob/master/LICENSE.txt for licensing details.

namespace C5
{
    /// <summary>
    /// Class containing debugging symbols - to eliminate preprocessor directives
    /// </summary>
    internal class Debug
    {
        /// <summary>
        /// Flag used to test hashing. Set to true when unit testing hash functions.
        /// </summary>
        internal static bool UseDeterministicHashing { get; set; }
    }
}
```

Conclusion

The `DEBUG` symbol has been eliminated.

For a future version of C5, we will optimize unit tests to test the contract, not the internal behavior. This will allow us to skip the debug check completely.

7.9 Comparers

The static class `Comparer<T>` provides a way to get a default (cached) comparer for a given type, however it duplicates the exact same functionality from `System.Collections.Generic.Comparer<T>.Default`, which also caches its values.

[6]The debug property cannot be in the `HashSet<T>` class as static properties are not shared between generic classes with different type parameters.

C5 also contains a number of comparers – `IntComparer`, `NaturalComparerO` etc. – which again duplicates the built in comparers for a given type.

All these comparers do exactly the same as the comparers shipped with the .NET Framework.

Conclusion

We have removed all the built-in comparers from C5.

7.10 Delegate comparer

C5 contains a helper class – `DelegateComparer<T>` – which can construct an `IComparer<T>` from a `Comparion<T>` delegate. This is helpful as implementing an `IComparer` or an `IEqualityComparer` otherwise requires a lot of *boilerplate* code.

However, instantiating a `Comparison<T>` is in itself cumbersome. This could be elegantly done using lambda expressions as described in [mark@lexparse.com, 2009].

Conclusion

We have removed the `DelegateComparer<T>` class and added a `ComparerFactory<T>` class along with two `internal` helper classes – `Comparer` and `EqualityComparer` which can create comparers and equality comparers from a lambda expressions – see listings B.3, B.4 and B.5.

7.11 Equality comparers

Like the newly removed comparers (see section 7.9), C5 also contains a somewhat redundant `EqualityComparer<T>` class.

There are also a `NaturalEqualityComparer<T>` and a `EquatableEqualityComparer<T>` which again will just return the same as `System.Collection.Generic.EqualityComparer<T>.Default`.

Conclusion

We have removed the redundant implementations but retained a modified `EqualityComparer<T>` for now (listing B.6), which keeps the sequenced equality comparers intact.

7.12 Special equality comparers

Apart from the removed `ReferenceEqualityComparer<T>` (see section 6.7), there are a number special comparers in C5:

- `ComparerZeroHashCodeEqualityComparer<T>`
- `SequencedCollectionEqualityComparer<T, W>`
- `UnsequencedCollectionEqualityComparer<T, W>`

7.12.1 Comparer zero hash code equality comparer

The `ComparerZeroHashCodeEqualityComparer<T>` is used to generate an `IEqualityComparer<T>` from a `IComparer<T>`.

This is used internally when a type implements `IComparer<T>` but not `IEqualityComparer<T>`.

It cannot be removed without introducing a number of breaking changes.

It could, however, be declared internal as it has no apparent use outside the library.

7.12.2 Sequenced and unsequenced collection equality comparer

These comparers are used for equality of sequenced and unsequenced collections. They are essentially just wrappers for the interface methods:

- `GetSequencedHashCode()`
- `SequencedEquals(ISequenced<T> otherCollection)`
- `GetUnsequencedHashCode()`
- `UnsequencedEquals(ICollection<T> otherCollection)`

This seems like what one would expect as the default behavior of `ISequenced<T>` and `ICollection<T>`.

Could this be solved by having `ISequenced<T>` implement `IEquatable<ISequenced<T>>` and `ICollection<T>` implement `IEquatable<ICollection<T>>`?

This will be a topic for future versions of C5.

Conclusion

We have kept the special equality comparers in C5 and modified the `EqualityComparer<T>` accordingly (listing B.6).

For future versions it should be investigated whether this can be removed completely in lieu of collections implementing `IEquatable<ICollection<T>>` directly.

7.13 Test attribute

The `TestedAttribute` has been removed – it was a device used during initial development, and it has lost its value[7].

[7] Cf. email from Niels Kokholm 21 February, 2011

Chapter 8

Future work

A number of topics and suggestions have come up during the project development which either cannot be implemented in the current version of C5 without introducing major breaking changes to the API or have been postponed as they are out of scope for this project.

These topics will be on the future work list for the next version of C5. Notable topics are:

- Optimizing unit testing for the AAA pattern and refactoring multiple asserts into more tests (section 3.3)
- ICollection should implement IEquatable<ICollection<T>> (section 7.12.2)
- ISequenced should implement IEquatable<ISequenced<T>> (section 7.12.2)
- Fixing problems with Silverlight not implementing proper covariant IEnumerable<out T> (section 6.10).
- Update the C5 home page with release notes and online documentation (section 7.3).
- Update the C5 book to reflect the changes and push the LaTeX source to Github.
- Update the C5 page on Wikipedia[1].
- Marcus Griep has suggested using Code Contracts[2] and Pex[3] for future development using a "Design by Contract" approach. This should be further investigated.
- Keith has suggested supporting Reactive Extensions (Rx)[4], which enables an interesting way of doing push-based collections and async programming[5].

[1] http://en.wikipedia.org/wiki/C5_Generic_Collection_Library_for_C_Sharp_and_CLI

[2] http://research.microsoft.com/en-us/projects/contracts

[3] http://research.microsoft.com/en-us/projects/pex

[4] http://msdn.microsoft.com/en-us/data/gg577609

[5] For an interesting take on Rx, we recommend a talk by Bart de Smet: "Rx: Curing your asynchronous programming blues" – http://channel9.msdn.com/posts/DC2010T0100-Keynote-Rx-curing-your-asynchronous-programming-blues

- C5 naming should follow the CLR and LINQ naming conventions (section 3.2).
- Using an automated build server for C5 (continuous integration) like TeamCity[6] which can immediately detect if one commits a change to C5 that breaks the build or the unit tests. This was part of the original goals for this project (section 1.2 but it has been postponed due to resource constraints.
- Alex Rønne Petersen notes that C5 library contains a number of `public readonly` fields. This style is deprecated in public APIs in favor of automatic properties with a `private` setter [Gunnerson, 2006], and should be fixed.

[6] http://www.jetbrains.com/teamcity/

Chapter 9

Conclusion

Through the course of this project we have upgraded C5 in many ways.

We have successfully upgraded C5 to .NET 4.0, making all the new language features for C♯ 3.0 and 4.0 available.

We have converted C5 to a portable library and thus made it useable for Silverlight, Windows Phone 7, and Xbox 360 development.

We have made major changes in the code and also reduced the code base considerably by removing a lot of redundant implementations – most notably the comparers and equality compares.

We have removed the complex build system and preprocessor directives making the C5 build process much simpler.

We have pushed C5 into the wild by publishing it to a public Github repository and the NuGet gallery.

We have maintained C5's reliability and trustability as all unit tests still pass and no new bugs have been introduced (that we know of).

All in all we believe that C5 is now ready for the future of .NET and C5 is now prepared a major rewrite which we intend to undertake in the near future.

Appendix A

Screen shots

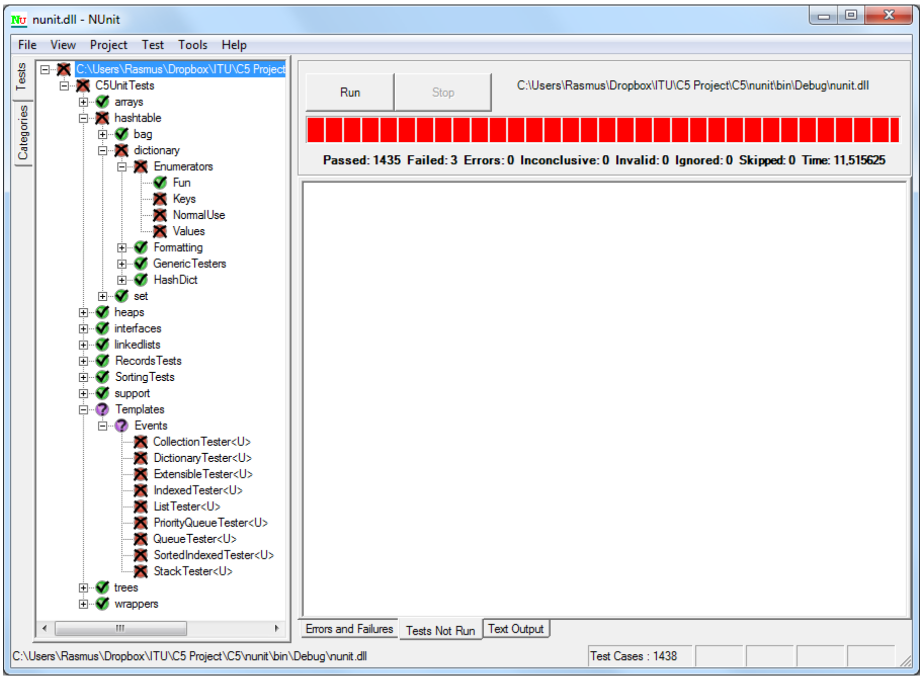

Figure A.1: Screen shot of the inital NUnit output.

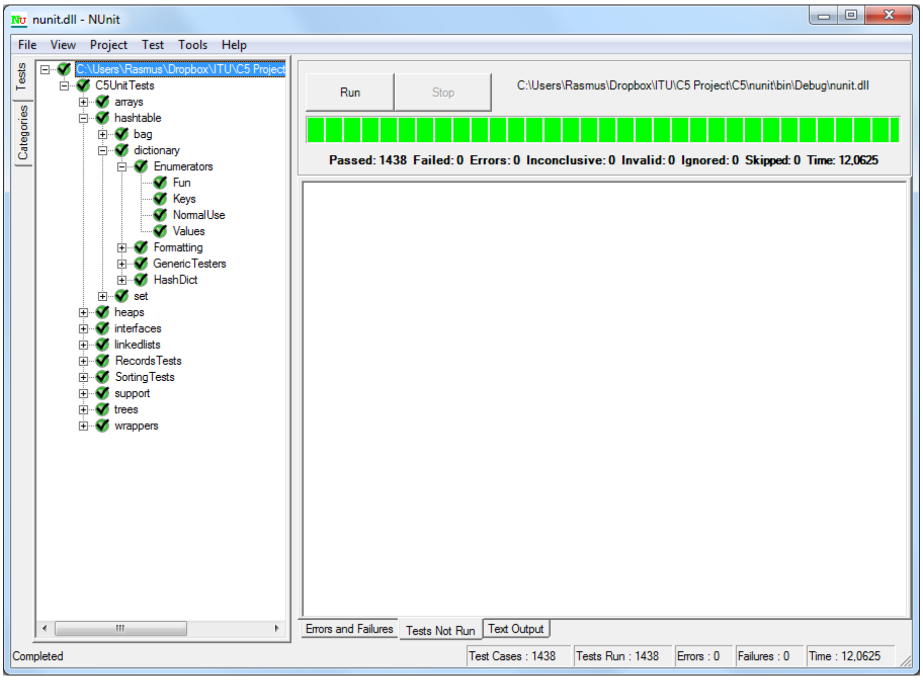

Figure A.2: Screen shot of the NUnit output after updating the enumerator tests and removing erroneously marked test methods.

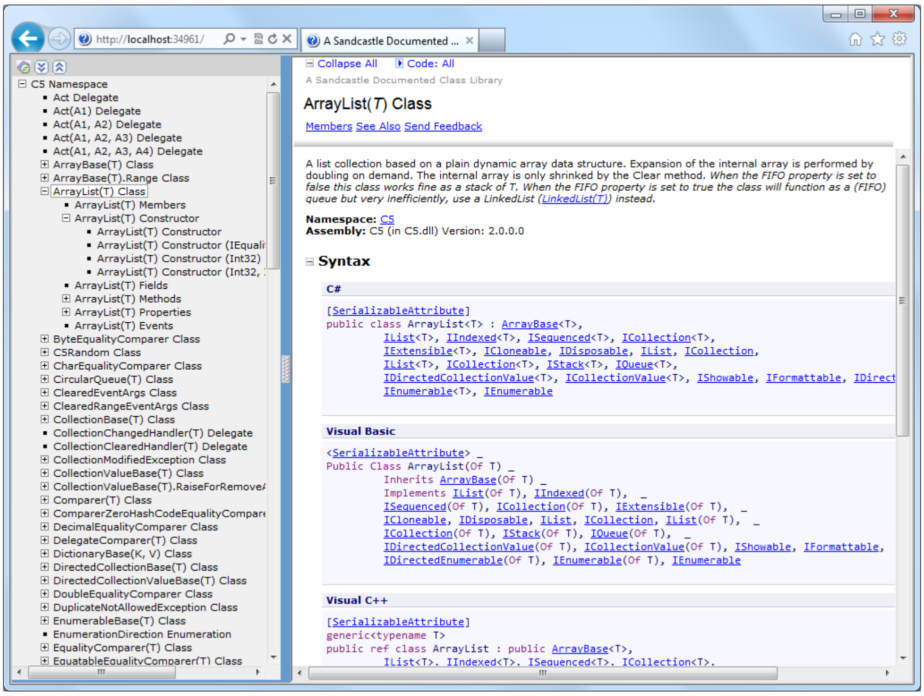

Figure A.3: Screen shot of the Sandcastle website output.

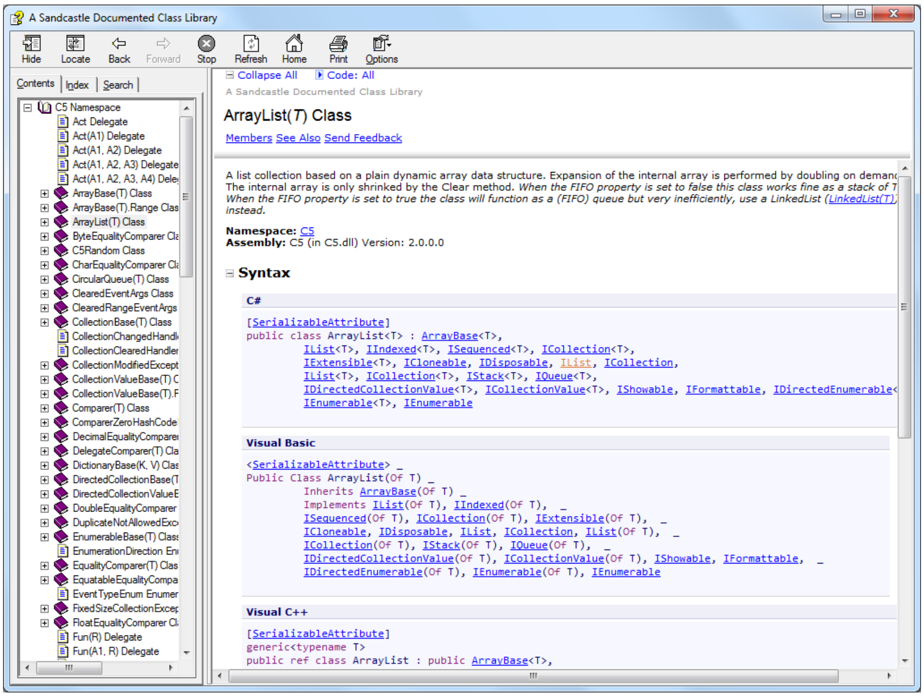

Figure A.4: Screen shot of the Sandcastle help file output.

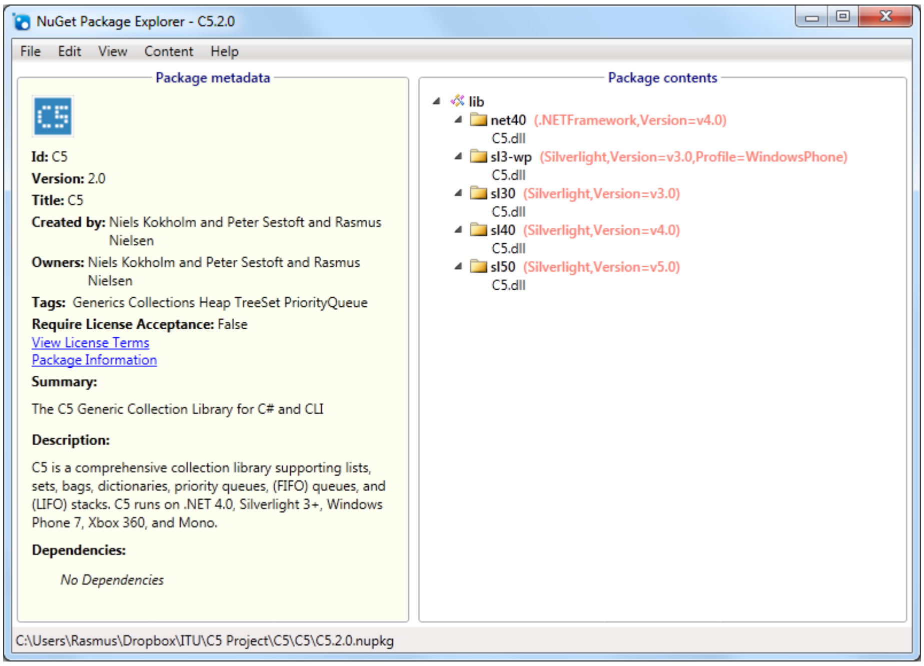

Figure A.5: Updating NuGet package metadata using the NuGet Package Explorer

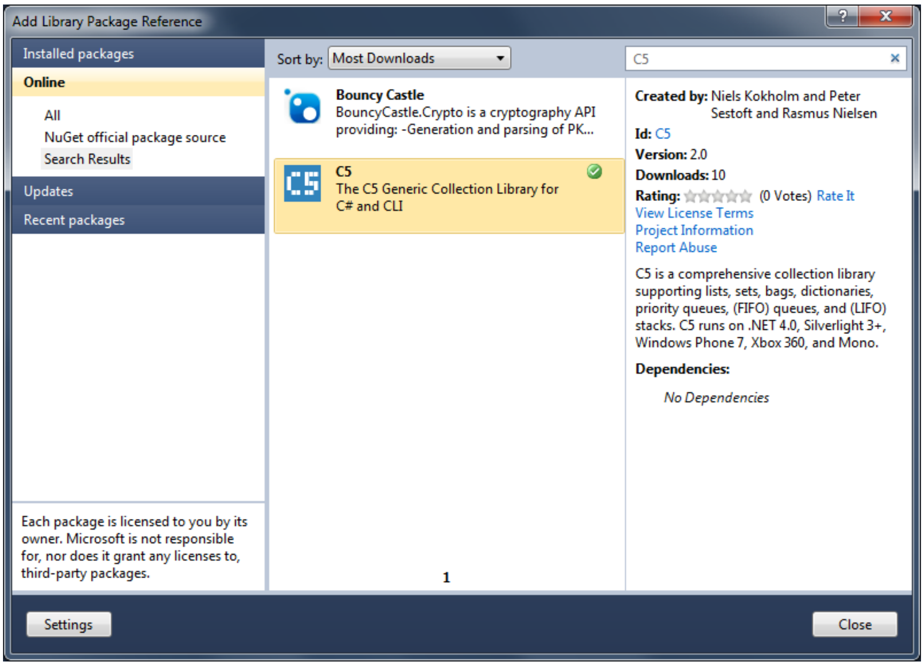

Figure A.6: Installing C5 using NuGet

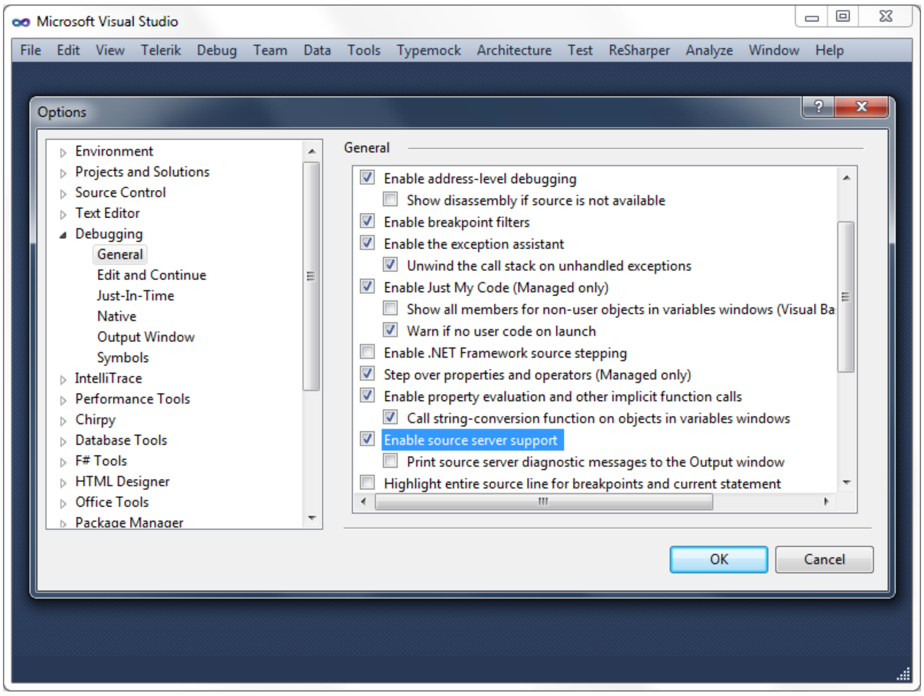

Figure A.7: Setting up source server support (1/2)

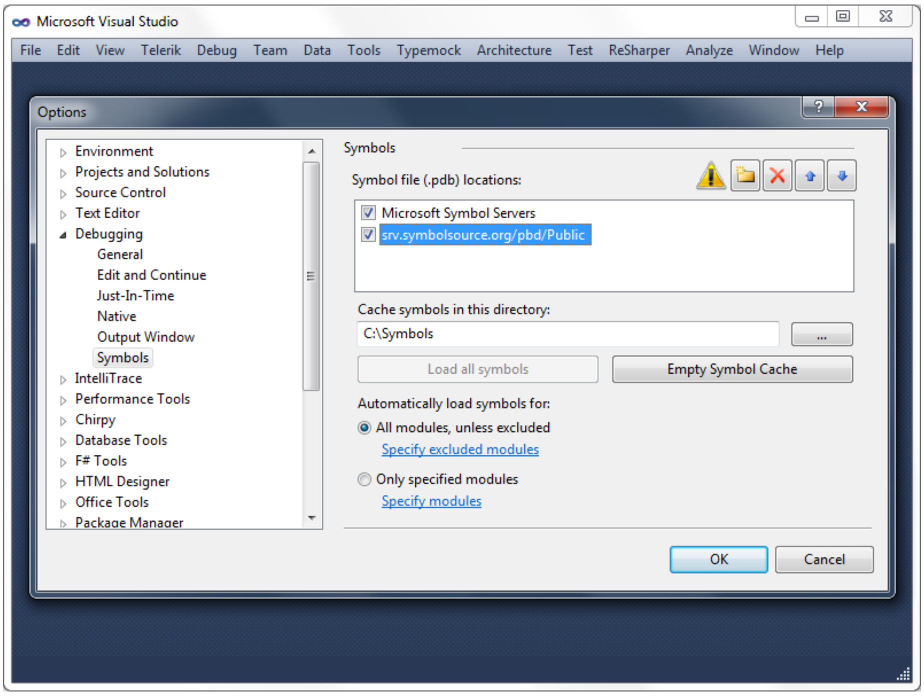

Figure A.8: Setting up source server support (2/2)

Appendix B

Code samples

Listing B.1: Original HashDictionary enumerators tests

```
namespace C5UnitTests.hashtable.dictionary
{
    [TestFixture]
    public class Enumerators
    {
        private HashDictionary<string, string> dict;

        private SCG.IEnumerator<KeyValuePair<string, string>> dictenum↩
            ;

        [SetUp]
        public void Init()
        {
            dict = new HashDictionary<string, string>();
            dict["S"] = "A";
            dict["T"] = "B";
            dict["R"] = "C";
            dictenum = dict.GetEnumerator();
        }

    [TearDown]
    public void Dispose()
    {
        dictenum = null;
        dict = null;
    }


    [Test]
    public void Keys()
    {
        SCG.IEnumerator<string> keys = dict.Keys.GetEnumerator();

        Assert.IsTrue(keys.MoveNext());
        Assert.AreEqual("R", keys.Current);
        Assert.IsTrue(keys.MoveNext());
        Assert.AreEqual("T", keys.Current);
        Assert.IsTrue(keys.MoveNext());
        Assert.AreEqual("S", keys.Current);
        Assert.IsFalse(keys.MoveNext());
    }


    [Test]
    public void Values()
    {
        SCG.IEnumerator<string> values = dict.Values.GetEnumerator();

```

```
            Assert.IsTrue(values.MoveNext());
            Assert.AreEqual("C", values.Current);
            Assert.IsTrue(values.MoveNext());
            Assert.AreEqual("B", values.Current);
            Assert.IsTrue(values.MoveNext());
            Assert.AreEqual("A", values.Current);
            Assert.IsFalse(values.MoveNext());
    }

        [Test]
        public void Fun()
        {
            Assert.AreEqual("B", dict.Fun("T"));
        }


        [Test]
        public void NormalUse()
        {
            Assert.IsTrue(dictenum.MoveNext());
            Assert.AreEqual(dictenum.Current, new KeyValuePair<string, string>("R", "C"));
            Assert.IsTrue(dictenum.MoveNext());
            Assert.AreEqual(dictenum.Current, new KeyValuePair<string, string>("T", "B"));
            Assert.IsTrue(dictenum.MoveNext());
            Assert.AreEqual(dictenum.Current, new KeyValuePair<string, string>("S", "A"));
            Assert.IsFalse(dictenum.MoveNext());
        }
    }
}
```

Listing B.2: Modified HashDictionary enumerators tests

```
namespace C5UnitTests.hashtable.dictionary
{
    [TestFixture]
    public class Enumerators
    {
        private HashDictionary<string, string> _dict;

        [SetUp]
        public void Init()
        {
            _dict = new HashDictionary<string, string>();
            _dict["S"] = "A";
            _dict["T"] = "B";
            _dict["R"] = "C";
        }

        [TearDown]
        public void Dispose()
        {
            _dict = null;
        }

        [Test]
        public void Keys()
        {
            var keys = _dict.Keys.ToArray();

            Assert.AreEqual(3, keys.Length);
            Assert.IsTrue(keys.Contains("R"));
            Assert.IsTrue(keys.Contains("S"));
            Assert.IsTrue(keys.Contains("T"));
        }


        [Test]
        public void Values()
        {
```

```
            var values = _dict.Values.ToArray();

            Assert.AreEqual(3, values.Length);
            Assert.IsTrue(values.Contains("A"));
            Assert.IsTrue(values.Contains("B"));
            Assert.IsTrue(values.Contains("C"));
        }

        [Test]
        public void Fun()
        {
            Assert.AreEqual("B", _dict.Fun("T"));
        }


        [Test]
        public void NormalUse()
        {
            var pairs = _dict.ToDictionary(pair => pair.Key, pair => ↩
                pair.Value);

            Assert.AreEqual(3, pairs.Count);
            Assert.IsTrue(pairs.Contains(new SCG.KeyValuePair<string, ↩
                string>("R", "C")));
            Assert.IsTrue(pairs.Contains(new SCG.KeyValuePair<string, ↩
                string>("S", "A")));
            Assert.IsTrue(pairs.Contains(new SCG.KeyValuePair<string, ↩
                string>("T", "B")));
        }
    }
}
```

Listing B.3: Comparer Factory

```
// This file is part of the C5 Generic Collection Library for C# and ↩
    CLI
// See https://github.com/sestoft/C5/blob/master/LICENSE.txt for ↩
    licensing details.

using System;
using System.Collections.Generic;

namespace C5.Comparers
{
    /// <summary>
    /// Factory class to create comparers and equality comparers using↩
        Func delegates
    /// </summary>
    /// <typeparam name="T">The type to compare</typeparam>
    public class ComparerFactory<T>
    {
        /// <summary>
        /// Create a new comparer.
        /// </summary>
        /// <param name="comparer">The compare function.</param>
        /// <returns>The comparer</returns>
        public static IComparer<T> CreateComparer(Func<T, T, int> ↩
            comparer)
        {
            return new Comparer<T>(comparer);
        }

        /// <summary>
        /// Creates a new equality comparer.
        /// </summary>
        /// <param name="equals">The equals function.</param>
        /// <param name="getHashCode">The getHashCode function.</param↩
            >
        /// <returns>The equality comparer.</returns>
        public static IEqualityComparer<T> CreateEqualityComparer(Func↩
            <T, T, bool> equals, Func<T, int> getHashCode)
        {
```

```
                return new EqualityComparer<T>(equals, getHashCode);
            }
        }
}
```

Listing B.4: Internal Comparer

```
// This file is part of the C5 Generic Collection Library for C# and CLI
// See https://github.com/sestoft/C5/blob/master/LICENSE.txt for licensing details.

using System;
using System.Collections.Generic;

namespace C5.Comparers
{
    /// <summary>
    /// Defines a method that a type implements to compare two objects.
    /// This class is intentionally declared internal - use the ComparerFactory to create an instance.
    /// </summary>
    /// <typeparam name="T">The type of objects to compare.</typeparam>
    internal class Comparer<T> : IComparer<T>
    {
        private readonly Func<T, T, int> _compare;

        /// <summary>
        /// Constructs a comparer using one Func delegate.
        /// </summary>
        /// <param name="compare">The compare function.</param>
        public Comparer(Func<T, T, int> compare)
        {
            _compare = compare;
        }

        /// <summary>
        /// Compares two objects and returns a value indicating whether one is less than, equal to, or greater than the other.
        /// </summary>
        /// <param name="x">The first object to compare.</param>
        /// <param name="y">The second object to compare.</param>
        /// <returns>A signed integer that indicates the relative values of x and y, as shown in the following table. Value Condition Less than zero x is less than y. Zero x equals y. Greater than zero x is greater than y.</returns>
        public int Compare(T x, T y)
        {
            return _compare(x, y);
        }
    }
}
```

Listing B.5: Internal equality comparer

```
// This file is part of the C5 Generic Collection Library for C# and CLI
// See https://github.com/sestoft/C5/blob/master/LICENSE.txt for licensing details.

using System;
using System.Collections.Generic;

namespace C5.Comparers
{
    /// <summary>
```

```
    /// Defines methods to support the comparison of objects for equality.
    /// This class is intentionally declared internal – use the ComparerFactory to create an instance.
    /// </summary>
    /// <typeparam name="T">The type of objects to compare.</typeparam>
    internal class EqualityComparer<T> : IEqualityComparer<T>
    {
        private readonly Func<T, T, bool> _equals;
        private readonly Func<T, int> _getHashCode;

        /// <summary>
        /// Constructs and equality comparer using two Func delegates.
        /// </summary>
        /// <param name="equals">The equals function.</param>
        /// <param name="getHashCode">The get hash code function.</param>
        public EqualityComparer(Func<T, T, bool> equals, Func<T, int> getHashCode)
        {
            _equals = equals;
            _getHashCode = getHashCode;
        }

        /// <summary>
        /// Determines whether the specified objects are equal.
        /// </summary>
        /// <param name="x">The first object of type T to compare.</param>
        /// <param name="y">The second object of type T to compare.</param>
        /// <returns>true if the specified objects are equal; otherwise, false.</returns>
        public bool Equals(T x, T y)
        {
            return _equals(x, y);
        }

        /// <summary>
        /// Returns a hash code for the specified object.
        /// </summary>
        /// <param name="obj">The System.Object for which a hash code is to be returned.</param>
        /// <returns>A hash code for the specified object.</returns>
        public int GetHashCode(T obj)
        {
            return _getHashCode(obj);
        }
    }
}
```

Listing B.6: Equality comparer supporting collection equality

```
using System;
using System.Linq;
using System.Reflection;
using SCG = System.Collections.Generic;

namespace C5
{
    /// <summary>
    /// Utility class for building default generic equality comparers.
    /// </summary>
    /// <typeparam name="T"></typeparam>
    public static class EqualityComparer<T>
    {
        private static SCG.IEqualityComparer<T> _default;

        readonly static Type SequencedCollectionEqualityComparer = typeof(SequencedCollectionEqualityComparer<,>);

```

```
        readonly static Type UnsequencedCollectionEqualityComparer = typeof(UnsequencedCollectionEqualityComparer<,>);

        /// <summary>
        /// A default generic equality comparer for type T. The procedure is as follows:
        /// <list>
        /// <item>If the actual generic argument T implements the generic interface
        /// <see cref="T:C5.ISequenced`1"/> for some value W of its generic parameter T,
        /// the equalityComparer will be <see cref="T:C5.SequencedCollectionEqualityComparer`2"/></item>
        /// <item>If the actual generic argument T implements
        /// <see cref="T:C5.ICollection`1"/> for some value W of its generic parameter T,
        /// the equalityComparer will be <see cref="T:C5.UnsequencedCollectionEqualityComparer`2"/></item>
        /// <item>Otherwise the SCG.EqualityComparer&lt;T&gt;.Default is returned</item>
        /// </list>
        /// </summary>
        /// <value>The comparer</value>
        public static SCG.IEqualityComparer<T> Default
        {
            get
            {
                if (_default != null)
                {
                    return _default;
                }

                var type = typeof(T);
                var interfaces = type.GetInterfaces();

                if (type.IsGenericType && type.GetGenericTypeDefinition().Equals(typeof(ISequenced<>)))
                {
                    return CreateAndCache(SequencedCollectionEqualityComparer.MakeGenericType(new[] { type, type.GetGenericArguments()[0] }));
                }

                var isequenced = interfaces.FirstOrDefault(i => i.IsGenericType && i.GetGenericTypeDefinition().Equals(typeof(ISequenced<>)));
                if (isequenced != null)
                {
                    return CreateAndCache(SequencedCollectionEqualityComparer.MakeGenericType(new[] { type, isequenced.GetGenericArguments()[0] }));
                }

                if (type.IsGenericType && type.GetGenericTypeDefinition().Equals(typeof(ICollection<>)))
                {
                    return CreateAndCache(UnsequencedCollectionEqualityComparer.MakeGenericType(new[] { type, type.GetGenericArguments()[0] }));
                }

                var icollection = interfaces.FirstOrDefault(i => i.IsGenericType && i.GetGenericTypeDefinition().Equals(typeof(ICollection<>)));
                if (icollection != null)
                {
                    return CreateAndCache(UnsequencedCollectionEqualityComparer.MakeGenericType(new[] { type, icollection.
```

```
                        GetGenericArguments()[0] }));
                }

                return _default = SCG.EqualityComparer<T>.Default;
            }
        }

        private static SCG.IEqualityComparer<T> CreateAndCache(Type ↩
            equalityComparertype)
        {
            return _default = (SCG.IEqualityComparer<T>)(↩
                equalityComparertype.GetProperty("Default", ↩
                BindingFlags.Static | BindingFlags.Public).GetValue(↩
                null, null));
        }
    }
}
```

Bibliography

Brad Abrams. Implementing ICloneable, 2003. URL http://blogs.msdn.com/b/brada/archive/2003/04/09/49935.aspx.

Brad Abrams. Should we Obsolete ICloneable (The SLAR on System.ICloneable), 2004. URL http://blogs.msdn.com/b/brada/archive/2004/05/03/125427.aspx.

Shawn Burke. 3-Screen Coding: Sharing code between Windows Phone, Silverlight, and .NET, 2010. URL http://channel9.msdn.com/Events/PDC/PDC10/CD11.

Miguel de Icaza. Announcing Xamarin, 2011. URL http://tirania.org/blog/archive/2011/May-16.html.

Ecma. *Standard ECMA-335 Common Language Infrastructure (CLI) 5th edition (December 2010)*. Ecma International, 2010. URL http://www.ecma-international.org/publications/standards/Ecma-335.htm.

Martin Fowler. Inversion of Control Containers and the Dependency Injection pattern. *martinfowler.com*, 2004. URL http://martinfowler.com/articles/injection.html.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

Github. Github Community, 2011. URL https://github.com/features/community.

Eric Gunnerson. Public readonly string vs. public readonly property?, 2006. URL http://blogs.msdn.com/b/ericgu/archive/2006/03/17/553934.aspx.

Scott Guthrie. Announcing NuPack, ASP.NET MVC 3 Beta, and WebMatrix Beta 2, 2010. URL http://weblogs.asp.net/scottgu/archive/2010/10/06/announcing-nupack-asp-net-mvc-3-beta-and-webmatrix-beta-2.aspx.

Scott Hanselman and Phil Haack. NuGet In Depth: Empowering Open Source on the .NET Platform, 2011. URL http://channel9.msdn.com/Events/MIX/MIX11/FRM09.

Srivatsan Kidambi. .NET Compact Framework for Windows Phone 7 series, 2010. URL http://blogs.msdn.com/b/srivatsan/archive/2010/03/16/net-compact-framework-for-windows-phone-7-series.aspx.

Niels Kokholm and Peter Sestoft. *The C5 Generic Collection Library for C♯ and CLI*. IT University of Copenhagen, Copenhagen, Denmark, 2006.

Niels Kokholm and Peter Sestoft. The C5 Generic Collection Library for C♯ and CLI, 2010. URL http://www.itu.dk/research/c5.

mark@lexparse.com. C♯ Lambdas: Never implement IComparer and IEqualityComparer again, 2009. URL http://www.lexparse.com/2009/11/02/c-lambdas-never-implement-icomparer-and-iequalitycomparer-again/.

Microsoft. CLSCompliantAttribute Class, 2010a. URL http://msdn.microsoft.com/en-us/library/system.clscompliantattribute(v=VS.100).aspx.

Microsoft. MethodBase.GetCurrentMethod Method, 2010b. URL http://msdn.microsoft.com/en-us/library/system.reflection.methodbase.getcurrentmethod.aspx.

Microsoft. RuntimeHelpers.GetHashCode Method (Object), 2010c. URL http://msdn.microsoft.com/en-us/library/11tbk3h9.aspx.

Microsoft. Portable Library Tools, 2011. URL http://visualstudiogallery.msdn.microsoft.com/b0e0b5e9-e138-410b-ad10-00cb3caf4981.

Mono. Mono, 2011. URL http://www.mono-project.com.

Roy Osherove. *The Art of Unit Testing with Examples in .NET*. Manning Publications Co., 2009.

Peter Sestoft and Niels Kokholm. C5 License, 2007. URL http://www.itu.dk/research/c5/LICENSE.txt.

Bjarne Stroustrup. *The Design and Evolution of C++*. Addison-Wesley, 1994.

Wikipedia. Sandcastle, 2010. URL http://en.wikipedia.org/wiki/Sandcastle_(software).

Wikipedia. C♯ 3.0, 2011a. URL http://en.wikipedia.org/wiki/C_Sharp_3.0.

Wikipedia. C♯ 4.0, 2011b. URL http://en.wikipedia.org/wiki/C_Sharp_4.0.

Wikipedia. Comparison of open source software hosting facilities, 2011c. URL http://en.wikipedia.org/wiki/Comparison_of_open_source_software_hosting_facilities.

Wikipedia. .NET Compact Framework, 2011d. URL http://en.wikipedia.org/wiki/.NET_Compact_Framework.

Wikipedia. .NET Micro Framework, 2011e. URL http://en.wikipedia.org/wiki/.NET_Micro_Framework.

Wikipedia. Microsoft Silverlight, 2011f. URL http://en.wikipedia.org/wiki/Microsoft_Silverlight.

Wikipedia. Windows Phone 7, 2011g. URL http://en.wikipedia.org/wiki/Windows_Phone_7.