C# Compiler

Edit page on GitHub (https://github.com/mono/website/blob/gh-pages/docs/about-mono/languages/csharp.md)

Introduction

The Mono C# compiler is considered feature complete for C# 1.0, C# 2.0, C# 3.0, C# 4.0, C# 5.0 and C# 6.0 (ECMA (/docs/about-mono/languages/ecma/)) and it has partial support for C# 7.

Historically, various version of same compiler existed.

- **gmcs**: compiler to target the 2.0 mscorlib.
- **smcs**: compiler to target the 2.1 mscorlib, to build Moonlight (/docs/web/moonlight/) applications.
- **dmcs**: compiler to target the 4.0 mscorlib.

Starting with Mono version 2.11 a new unified compiler **mcs** is available. It replaces all previous runtime specific compilers (gmcs, dmcs, smcs). They still exist (as scripts only) to ease the migration path to mcs but we strongly recommend to use mcs.

Bug reports or any compiler issue can be filed on our bug tracking system. (/community/bugs/)

The compiler is able to compile itself and many more C# programs (there is a test suite included that you can use). The compiler is routinely used to compile Mono, roughly four million lines of C# code and a few other projects.

The compiler is also fairly fast. On a IBM ThinkPad t40 it compiles 18,000 lines of C# code per second.

Compiler Service

Starting with Mono 2.2 it supports a Compiler Service (/docs/tools+libraries/tools/repl/) that applications can consume.

The compiler can be used as a service by using the Mono.CSharp.Evaluator class in the Mono.CSharp.dll assembly.

Both a console and GUI read-eval-print (/docs/tools+libraries/tools/repl/) shells are distributed as part of Mono 2.2 and are both built on top of the above service.

State of the Compiler

The (**mcs**) compiler defaults to the latest language specification available.

All C# 2.0 features are supported including:

- Anonymous methods (http://msdn2.microsoft.com/library/0yw3tz5k.aspx)
- Iterators (http://msdn2.microsoft.com/library/dscyy5s0.aspx)
- Partial classes (http://msdn2.microsoft.com/library/wa80x488.aspx)
- Generics (http://msdn2.microsoft.com/en-US/library/512aeb7t(vs.80).aspx)
- Nullable types (http://msdn2.microsoft.com/library/1t3y8s4s.aspx)
- Friend assemblies (http://msdn2.microsoft.com/library/0tke9fxk(en-us,vs.80).aspx)
- Static classes (http://msdn2.microsoft.com/library/79b3xss3.aspx)
- Covariance and contravariance (http://msdn2.microsoft.com/library/ms173174.aspx)
- Access modifiers on properties (http://msdn2.microsoft.com/library/75e8y5dd.aspx)
- Fixed buffers (http://msdn2.microsoft.com/library/zycewsya.aspx)
- External assembly alias (http://msdn2.microsoft.com/library/e59b22c5(en-us,vs.80).aspx)
- namespace alias qualifier (http://msdn2.microsoft.com/library/htccxtad(en-us,vs.80).aspx)
- Inline warning control (http://msdn2.microsoft.com/library/441722ys.aspx).

All C# 3.0 features are supported including:

- Extension methods
- Query expressions (LINQ)
- Expression trees
- Automatically implemented properties
- Lambda expressions
- Anonymous types
- Collection initializers
- Object initializers
- Implicitly typed arrays
- Partial methods

All C# 4.0 features are supported including:

- Dynamic binding support
- Generic type variance
- Optional parameters
- Named arguments

All C# 5.0 feature are supported including:

- Asynchronous programming support
- Caller info attributes

All C# 6.0 features are supported including:

- Auto-property initializers
- Await in catch and finally blocks
- Exception filters
- Expression-bodied function members
- Index initializers
- Null-conditional operator
- Nameof operator
- String interpolation
- Using static

Following C# 7 features are supported:

- Tuples
- Discards
- Out variables declaration
- Ref returns and locals
- Expression-bodied constructors, finalizers and accessors
- Throw expression
- Binary Literals
- Digit Separators
- Generalized async return types
- Default literal expressions
- Non-trailing named arguments
- ref struct
- Readonly struct
- Pattern matching (limited to simple usage)

If you want to limit the mcs compiler to be a strict 2.0 compiler, use the -langversion:ISO-2 flag, further options are available for each language version, inlcuding minor version starting with version 7.1.

Experimental features that are being brainstormed for future versions of C# when using `-langversion: experimental` are also available

Specification

The C# 2.0 specification is available on the third edition of the ECMA 334 standard (http://www.ecma-international.org/publications/standards/Ecma-334.htm).

An on-line and hyperlinked version of the C# 1.0 specification is available from Jon Jagger's (http://www.jaggersoft.com) site here (http://www.jaggersoft.com/csharp_standard/).

The specification shipped with Monodoc, and available on our web downloads is based on Jon's original XML documentation that he extracted from the ECMA 334 specification.

Working with MCS

Obtaining MCS

The Mono C# compiler is part of the `mono' module in the Mono Git you can get it from our source code (/community/contributing/source-code-repository/) server, or you can get nightly download page (/download/nightly/).

You can also browse or download a snapshot of the compiler alone:

- browse the sources (https://github.com/mono/mono/tree/master/mcs/mcs/).

If you are interested in developing the C# compiler, the C# Compiler as a Service or the interactive shell on Windows we provide Visual Studio solutions that work on VS 2008 and 2010 and allow developers to quickly get the compiler up and running.

To do this, download the entire MCS tree from:

- https://github.com/mono/mono/tree/master/mcs/ (https://github.com/mono/mono/tree/master/mcs/).

And then open the Visual Studio solution on mcs/tools/csharp, this will build the Mono.CSharp.dll compiler as a service as well as the command line interactive C# shell "csharp".

Running MCS

MCS is written in C# and uses heavily the .NET APIs. MCS runs on Linux with the Mono runtime and on Windows with both the .NET runtime and the Mono runtime.

Reporting Bugs

You can submit bugs by filing the bugs against the C# compiler in our Bugs (/community/bugs/) page.

When you report a bug, try to provide a small test case that would show the error so we can include this as part of the Mono C# regression test suite. If the bug is an error or a warning that we do not flag, write a sample program called `csXXXX.cs' where XXXX is the code number that is used by the Microsoft C# compiler that illustrates the problem. That way we can also do regression tests on the invalid input.

Implementation details

The compiler is documented in the file mcs/docs/compiler (https://github.com/mono/mono/blob/main/mcs/docs/compiler.txt)

CIL Optimizations

The compiler performs a number of simple optimizations on its input: constant folding (this is required by the C# language spec) and can perform dead code elimination.

Other more interesting optimizations like hoisting are not possible at this point since the compiler output at this point does not generate an intermediate representation that is suitable to perform basic block computation.

Adding an intermediate layer to enable the basic block computation to the compiler should be a simple task, but we are considering having a generic CIL optimizer. Since all the information that is required to perform basic block-based optimizations is available at the CIL level, we might just skip this step altogether and have just a generic IL optimizer that would perform hoisting on arbitrary CIL programs, not only those produced by MCS.

If this tool is further expanded to perform constant folding (not needed for our C# compiler, as it is already in there) and dead code elimination, other compiler authors might be able to use this generic CIL optimizer in their projects reducing their time to develop a production compiler.

Open bugs

See the bugs page (/community/bugs/) for more information. A test suite is maintained to track the progress of the compiler and various programs are routinely compiled and ran.

Slides

Slides for the Mono C# Compiler presentation at .NET ONE are available here (http://primates.ximian.com/~miguel/slides-europe-nov-2002/Mono_C_Sharp_Overview_1007.sxi) in StarOffice format.

History

MCS was able to parse itself on April 2001, MCS compiled itself for the first time on December 28 2001. MCS became self hosting on January 3rd, 2002.

The Mono Runtime and the Mono execution engine were able to make our compiler self hosting on March 12, 2002.

On July 2003 work started on the generics support of mcs. Since the core of the compiler was used in production and in the development of Mono itself, a fork of the compiler was created. This fork is `gmcs’. The `gmcs’ fork merges all of the changes from mcs on a regular basis and will eventually become the default compiler.

Support for the third edition of the C# standard became available on the Mono 1.1.8 release in the summer of 2005.

The compiler was originally written as an attempt to learn the C# language, it began as a tokenizer for the language, and then evolved into a parser for the language and later got the semantic analysis and code generation features in place. This explains the difference in coding styles from the tokenizer (which is the earliest code) and the code generation (which came at later phases).

Miguel de Icaza started the compiler and did most of the early work. He was later was joined by Ravi Pratap who would take over attributes, delegates, and overload resolution.

Martin Baulig wrote the flow analysis code as well as the generics support. Harinath Raja fixed and refactored large parts of the compiler, rearchitected flow analysis and fixed many of the early design mistakes.

Harinath Raja has been the compiler maintainer for the last two years and was in charge of making the compiler conform to the standard and productized many areas in the compiler, from fixing the name resolution hacks, to performance, to correctness, to simplifying many of the complexity that had crawled into the compiler over the years

Marek Safar initially focused on improving the warnings, error messages and made the compiler more robust as part of his work to make the compiler CLS compliant (Common Language Specification errors and warnings). He later implemented extension methods, delegate type inference and has been a mentor to new compiler authors for a long time.

Atsushi Enomoto wrote the support for inline documentation.

Sebastien Pouliot wrote all the code required to sign assemblies, delay sign them and deal with keys.

Ben Maurer did a lot of profiling work on the mcs compiler and helped fixed some of the problems in it.

Scott Peterson contributed in 2007 many of the foundation blocks for C# 3 (var, support, anonymous types, automatic properties, variable type inference and object and collection initializers).

In 2007 Marek Safar took over the maintenance of the compiler and he completed the C# 3.0 implementation, lambdas, mutators, type inferencing and LINQ expressions.

In 2008 Miguel de Icaza turned the compiler into a reusable library (Mono.CSharp.dll) and introduced the Interactive C# Shell (/docs/tools+libraries/tools/repl/).

In 2009 Marek Safar evolved the compiler to support C# 4.0, adding support for the dynamic data type, named parameters, default parameter values as well as co/contravariance support.

Many other developers contributed fixes and other small components in the compiler.

Common Questions

What is the difference between dmcs, gmcs and smcs

They are the same compiler with three different set of defaults.

- **dmcs**: references the 4.0-profile libraries (the APIs as defined in .NET 4.0) and supports C# 4.0.
- **gmcs**: references the 2.0-profile libraries (the APIs as defined in .NET 2.0 and .NET 3.5) and exposes the full C# 3.0 language.
- **smcs**: references the 2.1-profile libraries (the APIs defined for Silverlight) and exposes the full C# 3.0 language. This is the compiler used for creating Silverlight/Moonlight (/docs/web/moonlight/) applications.