NAME

mcs − Turbo C# Compiler

SYNOPSIS

mcs [option] [source-files]

DESCRIPTION

mcs is the Turbo C# compiler (also known as the Mono C# compiler), it is an implementation of the ECMA-334 language specification. You can pass one or more options to drive the compiler, and a set of source files. Extra options or arguments can be provided in a response file. Response files are referenced by prepending the @ symbol to the response file name.

The *mcs* compiler is used to compile against the latest Mono Base Class Library version and fully implements C# 1.0, 2.0, 3.0, 4.0, 5.0 and 6.0 specifications with partial support for C# 7.0.

See the section on packages for more information.

The Turbo C# compiler accepts the same command line options that the Microsoft C# compiler does. Those options can start with a slash or a dash (/checked is the same as -checked). Additionally some GNU-like options are supported, those begin with "--". All MCS-specific flags which are not available in the Microsoft C# compiler are available only with the GNU-style options.

C# source files must end with a ".cs" extension. Compilation of C# source code requires all the files that make up a library, module or executable to be provided on the command line. There is no support for partial compilation. To achieve the benefits of partial compilation, you should compile programs into their own assemblies, and later reference them with the "-r" flag.

The Turbo C# compiler generates images (.exe files) that contain CIL byte code that can be executed by any system that implements a Common Language Infrastructure virtual machine such as the Microsoft .NET runtime engine on Windows or the Mono runtime engine on Unix systems. Executables are not bound to a specific CPU or operating system.

The Turbo C# compiler by default only references three assemblies: mscorlib.dll, System.dll and System.Xml.dll. If you want to reference extra libraries you must manually specify them using the -pkg: command line option or the -r: command line option. Alternatively if you want to get all of the System libraries, you can use the -pkg:dotnet command line option.

OPTIONS

−−about

Displays information about the Turbo C# compiler

−−addmodule:MODULE1[,MODULE2]

Includes the specified modules in the resulting assembly. Modules are created by calling the compiler with the -target:module option

-checked, -checked+

Sets the default compilation mode to ‘checked’. This makes all the math operations checked (the default is unchecked).

-checked-

Sets the default compilation mode to ‘unchecked’. This makes all the math operations unchecked (this is the default).

-clscheck-, -clscheck+

Disables or enables the Common Language Specification (CLS) checks (it is enabled by default).

The Common Language Specification (CLS) defines an interoperable subset of types as well as conventions that compilers (CLS producers) and developers must follow to expose code to other programming languages (CLS consumers).

-codepage:ID

Specifies the code page used to process the input files from the point it is specified on. By default files will be processed in the environment-dependent native code page. The compiler will also

automatically detect Unicode files that have an embedded byte mark at the beginning.

Other popular encodings are 28591 (Latin1), 1252 (iso-8859-1) and 65001 (UTF-8).

MCS supports a couple of shorthands: "utf8" can be used to specify utf-8 instead of using the cryptic 65001 and "reset" restores the automatic handling of code pages. These shorthands are not available on the Microsoft compiler.

−define:SYMLIST, -d:SYMLIST
Defines the symbol listed by the semi-colon separated list SYMLIST SYMBOL. This can be tested in the source code by the pre-processor, or can be used by methods that have been tagged with the Conditional attribute.

−debug, −debug+
Generate debugging information. To obtain stack traces with debugging information, you need to invoke the mono runtime with the '--debug' flag. The debugging information is stored in a MDB file located in same output folder as produced assembly.

−debug-
Do not generate debugging information.

−delaysign+
Only embed the strongname public key into the assembly. The actual signing must be done in a later stage using the SN tool. This is useful to protect the private key during development. Note that delay signing can only be done using a strongname key file (not a key container). The option is equivalent to including [assembly: AssemblyDelaySign (true)] in your source code. Compiler option takes precedence over the attributes.

−delaysign-
Default. Strongname (sign) the assembly using the strong name key file (or container). The option is equivalent to including [assembly: AssemblyDelaySign (false)] in your source code. Compiler option takes precedence over the attributes.

−doc:FILE
Extracts the C#/XML documentation from the source code and stores in in the given FILE.

−errorreport
This flag is ignored by Mono's C# compiler and is present only to allow MCS to be used as a CSC replacement for msbuild/xbuild.

−−fatal
This is used for debugging the compiler. This makes the error emission generate an exception that can be caught by a debugger.

−filealign
This flag is ignored by Mono's C# compiler and is present only to allow MCS to be used as a CSC replacement for msbuild/xbuild.

−fullpaths
Any source code error or warning issued by the compiler includes file name only by default. This option causes compiler to issue absolute file path instead.

−keyfile:KEYFILE
Strongname (sign) the output assembly using the key pair present in the specified strong name key file (snk). A full key pair is required by default (or when using delaysign-). A file containing only the public key can be used with delaysign+. The option is equivalent to including [assembly: AssemblyKeyFile ("KEYFILE")] in your source code. Compiler option takes precedence over the attributes.

−keycontainer:CONTAINER
Strongname (sign) the output assembly using the key pair present in the specified container. Note that delaysign+ is ignored when using key containers. The option is equivalent to including [assembly: AssemblyKeyName ("CONTAINER")] in your source code. Compiler option takes

precedence over the attributes.

−langversion:TEXT

The option specifies the version of the language to use. The feature set is different in each C# version. This switch can be used to force the compiler to allow only a subset of the features. The possible values are:

Default

Instruct compiler to use the latest version. Equivalent is to omit the switch (this currently defaults to the C# 6.0 language specification).

ISO-1 Restrict compiler to use only first ISO standardized features. The usage of features such as generics, static classes, anonymous methods will lead to error.

ISO-2 Restrict compiler to use only the second ISO standardized features. This allows the use of generics, static classes, iterators and anonymous methods for example.

3 Restrict the compiler to use only the features available in C# 3.0 (a superset of ISO-1 and ISO-2).

4 Restrict the compiler to use only the features available in C# 4.0 specification.

5 Restrict the compiler to use only the features available in C# 5.0 specification.

6 Restrict the compiler to use only the features available in C# 6.0 specification.

experimental

Enables unstable features from upcoming versions of the language.

Notice that this flag only restricts the language features available to the programmer. A version of produced assemblies can be controlled using *SDK* option.

-lib:PATHLIST

Each path specified in the comma-separated list will direct the compiler to look for libraries in that specified path.

−L PATH

Directs the compiler to look for libraries in the specified path. Multiple paths can be provided by using the option multiple times.

−main:CLASS

Tells the compiler which CLASS contains the entry point. Useful when you are compiling several classes with a Main method.

−nostdlib, -nostdlib+

Use this flag if you want to compile the core library. This makes the compiler load its internal types from the assembly being compiled.

−noconfig, −noconfig+

Disables the default compiler configuration to be loaded. The compiler by default has references to the system assemblies.

−nowarn:WARNLIST

Makes the compiler ignore warnings specified in the comma-separated list WARNLIST>

-optimize, -optimize+, -optimize-

Controls compiler code generation optimizations on the code. Using -optimize or -optimize+ will turn on optimizations, -optimize- will turn it off. The default in mcs is to optimize-. The option can be mixed with -debug but for the best debugging experience it is recommended leave the options off.

-out:FNAME, -o FNAME

Names the output file to be generated.

−−parse
Used for benchmarking. The compiler will only parse its input files.

−pathmap:K=V[,Kn=Vn]
Sets a mapping for source path names used in generated output.

−pkg:package1[,packageN]
Reference assemblies for the given packages.

The compiler will invoke pkg-config --libs on the set of packages specified on the command line to obtain libraries and directories to compile the code.

This is typically used with third party components, like this:

```
$ mcs -pkg:gtk-sharp demo.cs
```

−pkg:dotnet
This will instruct the compiler to reference the System.* libraries available on a typical dotnet framework installation, notice that this does not include all of the Mono libraries, only the System.* ones. This is a convenient shortcut for those porting code.

−platform:ARCH
Used to specify the target platform. The possible values are: anycpu, anycpu32bitpreferred, arm, x86, x64 or itanium. The default option is anycpu.

-resource:RESOURCE[,ID]
Embeds to the given resource file. The optional ID can be used to give a different name to the resource. If not specified, the resource name will be the file name.

-linkresource:RESOURCE[,ID]
Links to the specified RESOURCE. The optional ID can be used to give a name to the linked resource.

-r:ASSEMBLY1[,ASSEMBLY2], −reference ASSEMBLY1[,ASSEMBLY2]
Reference the named assemblies. Use this to use classes from the named assembly in your program. The assembly will be loaded from either the system directory where all the assemblies live, or from the path explicitly given with the -L option.

You can also use a semicolon to separate the assemblies instead of a comma.

-reference:ALIAS=ASSEMBLY
Extern alias reference support for C#.

If you have different assemblies that provide the same types, the extern alias support allows you to provide names that your software can use to tell those appart. The types from ASSEMBLY will be exposed as ALIAS, then on the C# source code, you need to do:

```
extern alias ALIAS;
```

To bring it into your namespace. For example, to cope with two graphics libraries that define "Graphics.Point", one in "OpenGL.dll" and one in "Postscript.dll", you would invoke the compiler like this:

```
mcs -r:Postscript=Postscript.dll -r:OpenGL=OpenGL.dll
```

And in your source code, you would write:

```
extern alias Postscript;
extern alias OpenGL;

class X {
        // This is a Graphics.Point from Postscrip.dll
        Postscript.Point p = new Postscript.Point ();
```

```
                // This is a Graphics.Point from OpenGL.dll
                OpenGL.Point p = new OpenGL.Point ();
        }
```

−recurse:PATTERN, --recurse PATTERN
Does recursive compilation using the specified pattern. In Unix the shell will perform globbing, so you might want to use it like this:

```
$ mcs -recurse:'*.cs'
```

−sdk:VERSION
Used to specify the version of Base Class Library assemblies used for compilation. Following predefined values are valid: 2, 4 (default) as well as any custom value. The predefined version number means which custom value is specified mcs will try to find Base Class Libraries in the mono installed location PREFIX/lib/mono/<value>.

−−shell
Starts up the compiler in interactive mode, providing a C# shell for statements and expressions. A shortcut is to use the *csharp* command directly.

−−stacktrace
Generates a stack trace at the time the error is reported, useful for debugging the compiler.

−target:KIND, −t:KIND
Used to specify the desired target. The possible values are: exe (plain executable), winexe (Windows.Forms executable), library (component libraries) and module (partial library).

−−timestamp
Another debugging flag. Used to display the times at various points in the compilation process.

−unsafe, -unsafe+
Enables compilation of unsafe code.

−v Debugging. Turns on verbose yacc parsing.

−−version
Shows the compiler version.

−warnaserror, −warnaserror+
All compilers warnings will be reported as errors.

−warnaserror:W1,[Wn], -warnaserror+:W1,[Wn]
Treats one or more compiler warnings as errors.

−warnaserror-:W1,[Wn]
Sets one or more compiler warnings to be always threated as warnings. Becomes useful when used together with -warnaserror.

−warn:LEVEL
Sets the warning level. 0 is the lowest warning level, and 4 is the highest. The default is 4.

−win32res:FILE
Specifies a Win32 resource file (.res) to be bundled into the resulting assembly.

−win32icon:FILE
Attaches the icon specified in FILE on the output into the resulting assembly.

−− Use this to stop option parsing, and allow option-looking parameters to be passed on the command line.

PACKAGES AND LIBRARIES

When referencing an assembly, if the name of the assembly is a path, the compiler will try to load the assembly specified in the path. If it does not, then the compiler will try loading the assembly from the current directory, the compiler base directory and if the assembly is not found in any of those places in the directories specified as arguments to the -lib: command argument.

The compiler uses the library path to locate libraries, and is able to reference libraries from a particular package if that directory is used. To simplify the use of packages, the C# compiler includes the -pkg: command line option that is used to load specific collections of libraries.

Libraries visible to the compiler are stored relative to the installation prefix under PREFIX/lib/mono/ called the PACKAGEBASE and the defaults for mcs, gmcs and smcs are as follows:

mcs References the PACKAGEBASE/1.0 directory

gmcs References the PACKAGEBASE/2.0 directory

smcs References the PACKAGEBASE/2.1 directory

Those are the only runtime profiles that exist. Although other directories exist (like 3.0 and 3.5) those are not really runtime profiles, they are merely placeholders for extra libraries that build on the 2.0 foundation.

Software providers will distribute software that is installed relative to the PACKAGEBASE directory. This is integrated into the *gacutil* tool that not only installs public assemblies into the Global Assembly Cache (GAC) but also installs them into the PACKAGEBASE/PKG directory (where PKG is the name passed to the -package flag to gacutil).

As a developer, if you want to consume the Gtk# libraries, you would invoke the compiler like this:

```
$ mcs -pkg:gtk-sharp-2.0 main.cs
```

The -pkg: option instructs the compiler to fetch the definitions for gtk-sharp-2.0 from pkg-config, this is equivalent to passing to the C# compiler the output of:

```
$ pkg-config --libs gtk-sharp-2.0
```

Usually this merely references the libraries from PACKAGEBASE/PKG.

Although there are directory names for 3.0 and 3.5, that does not mean that there are 3.0 and 3.5 compiler editions or profiles. Those are merely new libraries that must be manually referenced either with the proper -pkg: invocation, or by referencing the libraries directly.

SPECIAL DEFINES

The **TRACE** and **DEBUG** defines have a special meaning to the compiler.

By default calls to methods and properties in the System.Diagnostics.Trace class are not generated unless the TRACE symbol is defined (either through a "#define TRACE") in your source code, or by using the *--define TRACE* in the command line.

By default calls to methods and properties in the System.Diagnostics.Debug class are not generated unless the DEBUG symbol is defined (either through a "#define DEBUG") in your source code, or by using the *--define DEBUG* in the command line.

Note that the effect of defining TRACE and DEBUG is a global setting, even if they are only defined in a single file.

DEBUGGING SUPPORT

When using the "-debug" flag, MCS will generate a file with the extension .mdb that contains the debugging information for the generated assembly. This file is consumed by the Mono debugger (mdb).

ENVIRONMENT VARIABLES

MCS_COLORS

If this variable is set, it contains a string in the form "foreground,background" that specifies which color to use to display errors on some terminals.

The background is optional and defaults to your terminal current background. The possible colors for foreground are: **black, red, brightred, green, brightgreen, yellow, brightyellow,** blue, brightblue, magenta, brightmagenta, cyan, brightcyan, grey, white and brightwhite.

The possible colors for background are: black, red, green, yellow, blue, magenta, cyan, grey and

white.

For example, you could set these variable from your shell:

```
export MCS_COLORS
MCS_COLORS=errors=brightwhite,red
```

You can disable the built-in color scheme by setting this variable to "disable".

NOTES

During compilation the MCS compiler defines the __MonoCS__ symbol, this can be used by pre-processor instructions to compile Mono C# compiler specific code. Please note that this symbol is only to test for the compiler, and is not useful to distinguish compilation or deployment platforms.

AUTHORS

The Mono C# Compiler was written by Miguel de Icaza, Ravi Pratap, Martin Baulig, Marek Safar and Raja Harinath. The development was funded by Ximian, Novell and Marek Safar.

LICENSE

The Mono Compiler Suite is released under the terms of the GNU GPL or the MIT X11. Please read the accompanying 'COPYING' file for details. Alternative licensing for the compiler is available from Xamarin.

SEE ALSO

csharp(1), mono(1), pkg-config(1), sn(1)

BUGS

To report bugs in the compiler, you must file them on our bug tracking system, at: http://www.mono-project.com/community/bugs/

MAILING LIST

The Mono Mailing lists are listed at http://www.mono-project.com/community/help/mailing-lists/

MORE INFORMATION

The Mono C# compiler was developed by Novell, Inc (http://www.novell.com) and Xamarin Inc (http://www.xamarin.com) is based on the ECMA C# language standard available here: http://www.ecma.ch/ecma1/STAND/ecma-334.htm

The home page for the Mono C# compiler is at http://www.mono-project.com/docs/about-mono/languages/csharp/