**Ilasm.exe (MSIL Assembler)**

**.NET Framework 4.5**

The MSIL Assembler generates a portable executable (PE) file from Microsoft intermediate language (MSIL). (For more information on MSIL, see [Managed Execution Process](http://msdn.microsoft.com/en-us/library/k5532s8a.aspx).) You can run the resulting executable, which contains MSIL and the required metadata, to determine whether the MSIL performs as expected.

This tool is automatically installed with Visual Studio and with the Windows SDK. To run the tool, we recommend that you use the Visual Studio Command Prompt or the Windows SDK Command Prompt (CMD Shell). These utilities enable you to run the tool easily, without navigating to the installation folder. For more information, see [Visual Studio and Windows SDK Command Prompts](http://msdn.microsoft.com/en-us/library/ms229859.aspx).

* If you have Visual Studio installed on your computer: On the taskbar, click **Start**, click **All Programs**, click **Visual Studio**, click **Visual Studio Tools**, and then click **Visual Studio Command Prompt**.

-or-

If you have the Windows SDK installed on your computer: On the taskbar, click **Start**, click **All Programs**, click the folder for the Windows SDK, and then click **Command Prompt** (or **CMD Shell**).

* At the command prompt, type the following:

ilasm [options] filename [[options]filename...]

[**Parameters**](javascript:void(0))

|  |  |
| --- | --- |
| Argument | Description |
| *filename* | The name of the .il source file. This file consists of metadata declaration directives and symbolic MSIL instructions. Multiple source file arguments can be supplied to produce a single PE file with Ilasm.exe.   |  | | --- | | **Description: NoteNote** | | Ensure that the last line of code in the .il source file has either trailing white space or an end-of-line character. | |
| Option | Description |
| **/32bitpreferred** | Creates a 32-bit-preferred image (PE32). |
| **/alignment**=*integer* | Sets FileAlignment to the value specified by *integer* in the NT Optional header. If the .alignment IL directive is specified in the file, this option overrides it. |
| **/appcontainer** | Produces a .dll or .exe file that runs in the Windows app container, as output. |
| **/arm** | Specifies the Advanced RISC Machine (ARM) as the target processor.  If no image bitness is specified, the default is **/32bitpreferred**. |
| **/base**=*integer* | Sets ImageBase to the value specified by *integer* in the NT Optional header. If the .imagebase IL directive is specified in the file, this option overrides it. |
| **/clock** | Measures and reports the following compilation times in milliseconds for the specified .il source file:  **Total Run**: The total time spent performing all the specific operations that follow.  **Startup**: Loading and opening the file.  **Emitting MD**: Emitting metadata.  **Ref to Def Resolution**: Resolving references to definitions in the file.  **CEE File Generation**: Generating the file image in memory.  **PE File Writing**: Writing the image to a PE file. |
| **/debug**[=**IMPL**|**OPT**] | Includes debug information (local variable and argument names, and line numbers). Creates a PDB file.  **/debug** with no additional value disables JIT optimization and uses sequence points from the PDB file.  **IMPL** disables JIT optimization and uses implicit sequence points.  **OPT** enables JIT optimization and uses implicit sequence points. |
| **/dll** | Produces a .dll file as output. |
| **/enc**=*file* | Creates Edit-and-Continue deltas from the specified source file.  This argument is for academic use only and is not supported for commercial use. |
| **/exe** | Produces an executable file as output. This is the default. |
| **/flags**=*integer* | Sets ImageFlags to the value specified by *integer* in the common language runtime header. If the .corflags IL directive is specified in the file, this option overrides it. See CorHdr.h, COMIMAGE\_FLAGS for a list of valid values for *integer*. |
| **/fold** | Folds identical method bodies into one. |
| /**highentropyva** | Produces an output executable that supports high-entropy address space layout randomization (ASLR). (Default for **/appcontainer**.) |
| **/include**=*includePath* | Sets a path to search for files included with **#include**. |
| **/itanium** | Specifies Intel Itanium as the target processor.  If no image bitness is specified, the default is **/pe64**. |
| **/key:***keyFile* | Compiles *filename* with a strong signature using the private key contained in *keyFile*. |
| **/key:@***keySource* | Compiles *filename* with a strong signature using the private key produced at *keySource*. |
| **/listing** | Produces a listing file on the standard output. If you omit this option, no listing file is produced.  This parameter is not supported in the .NET Framework 2.0 or later. |
| **/mdv**=*versionString* | Sets the metadata version string. |
| **/msv**=*major***.***minor* | Sets the metadata stream version, where *major* and *minor* are integers. |
| **/noautoinherit** | Disables default inheritance from [Object](http://msdn.microsoft.com/en-us/library/system.object.aspx) when no base class is specified. |
| **/nocorstub** | Suppresses generation of the CORExeMain stub. |
| **/nologo** | Suppresses the Microsoft startup banner display. |
| **/output:***22* | Specifies the output file name and extension. By default, the output file name is the same as the name of the first source file. The default extension is .exe. If you specify the **/dll** option, the default extension is .dll.   |  | | --- | | **Description: NoteNote** | | Specifying **/output:**myfile.dll does not set the **/dll** option. If you do not specify **/dll**, the result will be an executable file named myfile.dll. | |
| **/optimize** | Optimizes long instructions to short. For example, **br** to **br.s**. |
| **/pe64** | Creates a 64-bit image (PE32+).  If no target processor is specified, the default is **/itanium**. |
| **/pdb** | Creates a PDB file without enabling debug information tracking. |
| **/quiet** | Specifies quiet mode; does not report assembly progress. |
| **/resource:***file.res* | Includes the specified resource file in \*.res format in the resulting .exe or .dll file. Only one .res file can be specified with the **/resource** option. |
| **/ssver**=*int*.*int* | Sets the subsystem version number in the NT optional header. For **/appcontainer** and **/arm** the minimum version number is 6.02. |
| **/stack**=*stackSize* | Sets the SizeOfStackReserve value in the NT Optional header to *stackSize*. |
| **/stripreloc** | Specifies that no base relocations are needed. |
| **/subsystem**=*integer* | Sets subsystem to the value specified by *integer* in the NT Optional header. If the .subsystem IL directive is specified in the file, this command overrides it. See winnt.h, IMAGE\_SUBSYSTEM for a list of valid values for *integer*. |
| **/x64** | Specifies a 64-bit AMD processor as the target processor.  If no image bitness is specified, the default is **/pe64**. |
| **/?** | Displays command syntax and options for the tool. |

|  |
| --- |
| **Description: NoteNote** |
| All options for Ilasm.exe are case-insensitive and recognized by the first three letters. For example, **/lis** is equivalent to **/listing** and **/res:**myresfile.res is equivalent to **/resource:**myresfile.res. Options that specify arguments accept either a colon (:) or an equal sign (=) as the separator between the option and the argument. For example, **/output:***file.ext* is equivalent to **/output=***file.ext*. |

[**Remarks**](javascript:void(0))

The MSIL Assembler helps tool vendors design and implement MSIL generators. Using Ilasm.exe, tool and compiler developers can concentrate on MSIL and metadata generation without being concerned with emitting MSIL in the PE file format.

Similar to other compilers that target the runtime, such as C# and Visual Basic, Ilasm.exe does not produce intermediate object files and does not require a linking stage to form a PE file.

The MSIL Assembler can express all the existing metadata and MSIL features of the programming languages that target the runtime. This allows managed code written in any of these programming languages to be adequately expressed in MSIL Assembler and compiled with Ilasm.exe.

|  |
| --- |
| **Description: NoteNote** |
| Compilation might fail if the last line of code in the .il source file does not have either trailing white space or an end-of-line character. |

You can use Ilasm.exe in conjunction with its companion tool, [Ildasm.exe](http://msdn.microsoft.com/en-us/library/f7dy01k1.aspx). Ildasm.exe takes a PE file that contains MSIL code and creates a text file suitable as input to Ilasm.exe. This is useful, for example, when compiling code in a programming language that does not support all the runtime metadata attributes. After compiling the code and running the output through Ildasm.exe, the resulting MSIL text file can be hand-edited to add the missing attributes. You can then run this text file through the Ilasm.exe to produce a final executable file.

You can also use this technique to produce a single PE file from several PE files originally generated by different compilers.

|  |
| --- |
| **Description: NoteNote** |
| Currently, you cannot use this technique with PE files that contain embedded native code (for example, PE files produced by Visual C++). |

To make this combined use of Ildasm.exe and Ilasm.exe as accurate as possible, by default the assembler does not substitute short encodings for long ones you might have written in your MSIL sources (or that might be emitted by another compiler). Use the **/optimize** option to substitute short encodings wherever possible.

|  |
| --- |
| **Description: NoteNote** |
| Ildasm.exe only operates on files on disk. It does not operate on files installed in the global assembly cache. |

For more information about the grammar of MSIL, see the asmparse.grammar file in the Windows Software Development Kit (SDK).

**Version Information**

Starting with the .NET Framework 4.5, you can attach a custom attribute to an interface implementation by using code similar to the following:

[Copy](javascript:CodeSnippet_CopyCode('CodeSnippetContainerCode_bbf2a1b3-5dce-4bad-bdcb-42e7a21848cf');)

.class interface public abstract auto ansi IMyInterface

{

.method public hidebysig newslot abstract virtual

instance int32 method1() cil managed

{

} // end of method IMyInterface::method1

} // end of class IMyInterface

.class public auto ansi beforefieldinit MyClass

extends [mscorlib]System.Object

implements IMyInterface

{

.interfaceimpl type IMyInterface

.custom instance void

[mscorlib]System.Diagnostics.DebuggerNonUserCodeAttribute::.ctor() = ( 01 00 00 00 )

…

Starting with the .NET Framework 4.5, you can specify an arbitrary marshal BLOB (binary large object) by using its raw binary representation, as shown in the following code:

.method public hidebysig abstract virtual

instance void

marshal({ 38 01 02 FF })

Test(object A\_1) cil managed

For more information about the grammar of MSIL, see the asmparse.grammar file in the Windows Software Development Kit (SDK).

[**Examples**](javascript:void(0))

The following command assembles the MSIL file myTestFile.il and produces the executable myTestFile.exe.

ilasm myTestFile

The following command assembles the MSIL file myTestFile.il and produces the .dll file myTestFile.dll.

ilasm myTestFile /dll

The following command assembles the MSIL file myTestFile.il and produces the .dll file myNewTestFile.dll.

ilasm myTestFile /dll /output:myNewTestFile.dll

The following code example shows an extremely simple application that displays "Hello World!" to the console. You can compile this code and then use the [Ildasm.exe](http://msdn.microsoft.com/en-us/library/f7dy01k1.aspx) tool to generate an MSIL file.

**C#**

using System;

public class Hello

{

public static void Main(String[] args)

{

Console.WriteLine("Hello World!");

}

}

The following MSIL code example corresponds to the previous C# code example. You can compile this code into an assembly using the [Ilasm.exe (MSIL Assembler)](http://msdn.microsoft.com/en-us/library/496e4ekx.aspx) tool. Both MSIL and C# code examples display "Hello World!" to the console.

// Metadata version: v2.0.50215

.assembly extern mscorlib

{

.publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..

.ver 2:0:0:0

}

.assembly sample

{

.custom instance void [mscorlib]System.Runtime.CompilerServices.CompilationRelaxationsAttribute::.ctor(int32) = ( 01 00 08 00 00 00 00 00 )

.hash algorithm 0x00008004

.ver 0:0:0:0

}

.module sample.exe

// MVID: {A224F460-A049-4A03-9E71-80A36DBBBCD3}

.imagebase 0x00400000

.file alignment 0x00000200

.stackreserve 0x00100000

.subsystem 0x0003 // WINDOWS\_CUI

.corflags 0x00000001 // ILONLY

// Image base: 0x02F20000

// =============== CLASS MEMBERS DECLARATION ===================

.class public auto ansi beforefieldinit Hello

extends [mscorlib]System.Object

{

.method public hidebysig static void Main(string[] args) cil managed

{

.entrypoint

// Code size 13 (0xd)

.maxstack 8

IL\_0000: nop

IL\_0001: ldstr "Hello World!"

IL\_0006: call void [mscorlib]System.Console::WriteLine(string)

IL\_000b: nop

IL\_000c: ret

} // end of method Hello::Main

.method public hidebysig specialname rtspecialname

instance void .ctor() cil managed

{

// Code size 7 (0x7)

.maxstack 8

IL\_0000: ldarg.0

IL\_0001: call instance void [mscorlib]System.Object::.ctor()

IL\_0006: ret

} // end of method Hello::.ctor

} // end of class Hello