Predefined macros

Article06/11/202413 contributors Feedback

In this article

The Microsoft C/C++ compiler (MSVC) predefines certain preprocessor macros depending on the language (C or C++), the compilation target, and the chosen compiler options.

MSVC supports the predefined preprocessor macros required by the ANSI/ISO C99, C11, and C17 standards, and the ISO C++14, C++17, and C++20 standards. The implementation also supports several more Microsoft-specific preprocessor macros.

Some macros are defined only for specific build environments or compiler options. Except where noted, the macros are defined throughout a translation unit as if they were specified as `/D` compiler option arguments. When defined, the preprocessor expands macros their specified values before compilation. The predefined macros take no arguments and can't be redefined.

Standard predefined identifier

The compiler supports this predefined identifier specified by ISO C99 and ISO C++11.

- `__func__` The unqualified and unadorned name of the enclosing function as a function-local **static const** array of `char`.

C++ Copy

```
void example()
{
    printf("%s\n", __func__);
} // prints "example"
```

Standard predefined macros

The compiler supports these predefined macros specified by the ISO C99, C11, C17, and ISO C++17 standards.

- `__cplusplus` Defined as an integer literal value when the translation unit is compiled as C++. Otherwise, undefined.

- `__DATE__` The compilation date of the current source file. The date is a constant length string literal of the form *Mmm dd yyyy*. The month name *Mmm* is the same as the abbreviated month name generated by the C Runtime Library (CRT) asctime function. The first character of date *dd* is a space if the value is less than 10. This macro is always defined.

- `__FILE__` The name of the current source file. `__FILE__` expands to a character string literal. To ensure that the full path to the file is displayed, use **`/FC`** (Full Path of Source Code File in Diagnostics). This macro is always defined.

- `__LINE__` Defined as the integer line number in the current source file. The value of this macro can be changed by using a `#line` directive. The integral type of the value of `__LINE__` can vary depending on context. This macro is always defined.

- `__STDC__` Defined as 1 when compiled as C and if the /Za compiler option is specified. Starting in Visual Studio 2022 version 17.2, it's defined as 1 when compiled as C and if the /std:c11 or /std:c17 compiler option is specified. Otherwise, undefined.

- `__STDC_HOSTED__` Defined as 1 if the implementation is a *hosted implementation*, one that supports the entire required standard library. Otherwise, defined as 0.

- `__STDC_NO_ATOMICS__` Defined as 1 if the implementation doesn't support optional standard atomics. The MSVC implementation defines it as 1 when compiled as C and one of the /std C11 or C17 options is specified.

- `__STDC_NO_COMPLEX__` Defined as 1 if the implementation doesn't support optional standard complex numbers. The MSVC implementation defines it as 1 when compiled as C and one of the /std C11 or C17 options is specified.

- `__STDC_NO_THREADS__` Defined as 1 if the implementation doesn't support optional standard threads. The MSVC implementation defines it as 1 when compiled as C and one of the /std C11 or C17 options is specified.

- `__STDC_NO_VLA__` Defined as 1 if the implementation doesn't support standard variable length arrays. The MSVC implementation defines it as 1 when compiled as C and one of the /std C11 or C17 options is specified.

- `__STDC_VERSION__` Defined when compiled as C and one of the **`/std`** C11 or C17 options is specified. It expands to `201112L` for /std:c11, and `201710L` for /std:c17.

- `__STDCPP_DEFAULT_NEW_ALIGNMENT__` When /std:c17 or later is specified, this macro expands to a `size_t` literal that has the value of the alignment guaranteed by a call to alignment-unaware `operator new`. Larger alignments are passed to an alignment-aware overload, such as `operator new(std::size_t, std::align_val_t)`. For more information, see /Zc:alignedNew (C++17 over-aligned allocation).

- `__STDCPP_THREADS__` Defined as 1 if and only if a program can have more than one thread of execution, and compiled as C++. Otherwise, undefined.

- `__TIME__` The time of translation of the preprocessed translation unit. The time is a character string literal of the form *hh:mm:ss*, the same as the time returned by the CRT asctime function. This macro is always defined.

Microsoft-specific predefined macros

MSVC supports other predefined macros:

- `__ARM_ARCH` Defined as an integer literal that represents the ARM architecture version. The value is defined as 8 for the Armv8-A architecture. For 8.1 and onwards, the value is scaled for minor versions, such as X.Y, by using the formula X * 100 + Y as defined by the ARM C language extension. For example, for Armv8.1, `__ARM_ARCH` is 8 * 100 + 1 or 801. To set the ARM architecture version, see /arch (ARM64). This macro was introduced in Visual Studio 2022 version 17.10.

- `__ATOM__` Defined as 1 when the /favor:ATOM compiler option is set and the compiler target is x86 or x64. Otherwise, undefined.

- `__AVX__` Defined as 1 when the /arch:AVX, /arch:AVX2, /arch:AVX512 or /arch:AVX10.1 compiler options are set and the compiler target is x86 or x64. Otherwise, undefined.

- `__AVX2__` Defined as 1 when the /arch:AVX2, /arch:AVX512 or /arch:AVX10.1 compiler option is set and the compiler target is x86 or x64. Otherwise, undefined.

- `__AVX512BW__` Defined as 1 when the /arch:AVX512 or /arch:AVX10.1 compiler option is set and the compiler target is x86 or x64. Otherwise, undefined.

- `__AVX512CD__` Defined as 1 when the /arch:AVX512 or /arch:AVX10.1 compiler option is set and the compiler target is x86 or x64. Otherwise, undefined.

- `__AVX512DQ__` Defined as 1 when the /arch:AVX512 or /arch:AVX10.1 compiler option is set and the compiler target is x86 or x64. Otherwise, undefined.

- `__AVX512F__` Defined as 1 when the /arch:AVX512 or /arch:AVX10.1 compiler option is set and the compiler target is x86 or x64. Otherwise, undefined.

- `__AVX512VL__` Defined as 1 when the /arch:AVX512 or /arch:AVX10.1 compiler option is set and the compiler target is x86 or x64. Otherwise, undefined.

- `__AVX10_VER__` Defined as an integer that represents version of AVX10 when the /arch:AVX10.1 compiler option is set and the compiler target is x86 or x64. Otherwise, undefined.

- `_CHAR_UNSIGNED` Defined as 1 if the default **`char`** type is unsigned. This value is defined when the **`/J`** (Default char type is unsigned) compiler option is set. Otherwise, undefined.

- `__CLR_VER` Defined as an integer literal that represents the version of the Common Language Runtime (CLR) used to compile the app. The value is encoded in the form `Mmmbbbbb`, where `M` is the major version of the runtime, `mm` is the minor version of the runtime, and `bbbbb` is the build number. `__CLR_VER` is defined if the /clr compiler option is set. Otherwise, undefined.

C++ Copy

```
// clr_ver.cpp
// compile with: /clr
using namespace System;
int main() {
   Console::WriteLine(__CLR_VER);
}
```

- `_CONTROL_FLOW_GUARD` Defined as 1 when the **`/guard:cf`** (Enable Control Flow Guard) compiler option is set. Otherwise, undefined.

- `__COUNTER__` Expands to an integer literal that starts at 0. The value increments by 1 every time it's used in a source file, or in included headers of the source file. `__COUNTER__` remembers its state when you use precompiled headers. This macro is always defined.

  This example uses `__COUNTER__` to assign unique identifiers to three different objects of the same type. The `exampleClass` constructor takes an integer as a parameter. In `main`, the application declares three objects of type `exampleClass`, using `__COUNTER__` as the unique identifier parameter:

  C++ Copy

```
// macro__COUNTER__.cpp
// Demonstration of __COUNTER__, assigns unique identifiers to
// different objects of the same type.
// Compile by using: cl /EHsc /W4 macro__COUNTER__.cpp
#include <stdio.h>

class exampleClass {
    int m_nID;
public:
    // initialize object with a read-only unique ID
    exampleClass(int nID) : m_nID(nID) {}
    int GetID(void) { return m_nID; }
};

int main()
{
    // __COUNTER__ is initially defined as 0
    exampleClass e1(__COUNTER__);

    // On the second reference, __COUNTER__ is now defined as 1
    exampleClass e2(__COUNTER__);

    // __COUNTER__ is now defined as 2
    exampleClass e3(__COUNTER__);

    printf("e1 ID: %i\n", e1.GetID());
    printf("e2 ID: %i\n", e2.GetID());
    printf("e3 ID: %i\n", e3.GetID());

    // Output
```

```
    // ------------------------------
    // e1 ID: 0
    // e2 ID: 1
    // e3 ID: 2

    return 0;
}
```

- `__cplusplus_cli` Defined as the integer literal value 200406 when compiled as C++ and a /clr compiler option is set. Otherwise, undefined. When defined, `__cplusplus_cli` is in effect throughout the translation unit.

  C++ Copy

```
// cplusplus_cli.cpp
// compile by using /clr
#include "stdio.h"
int main() {
   #ifdef __cplusplus_cli
      printf("%d\n", __cplusplus_cli);
   #else
      printf("not defined\n");
   #endif
}
```

- `__cplusplus_winrt` Defined as the integer literal value 201009 when compiled as C++ and the **`/ZW`** (Windows Runtime Compilation) compiler option is set. Otherwise, undefined.

- `_CPPRTTI` Defined as 1 if the **`/GR`** (Enable Run-Time Type Information) compiler option is set. Otherwise, undefined.

- `_CPPUNWIND` Defined as 1 if one or more of the **`/GX`** (Enable Exception Handling), **`/clr`** (Common Language Runtime Compilation), or **`/EH`** (Exception Handling Model) compiler options are set. Otherwise, undefined.

- `_DEBUG` Defined as 1 when the /LDd, /MDd, or /MTd compiler option is set. Otherwise, undefined.

- `_DLL` Defined as 1 when the /MD or /MDd (Multithreaded DLL) compiler option is set. Otherwise, undefined.

- `__FUNCDNAME__` Defined as a string literal that contains the decorated name of the enclosing function. The macro is defined only within a function. The `__FUNCDNAME__` macro isn't expanded if you use the /EP or /P compiler option.

  This example uses the `__FUNCDNAME__`, `__FUNCSIG__`, and `__FUNCTION__` macros to display function information.

  C++ Copy

```
// Demonstrates functionality of __FUNCTION__, __FUNCDNAME__, and __FUNCSIG__
void exampleFunction()
{
    printf("Function name: %s\n", __FUNCTION__);
    printf("Decorated function name: %s\n", __FUNCDNAME__);
    printf("Function signature: %s\n", __FUNCSIG__);

    // Sample Output
    // -----------------------------------------------
    // Function name: exampleFunction
    // Decorated function name: ?exampleFunction@@YAXXZ
    // Function signature: void __cdecl exampleFunction(void)
}
```

- `__FUNCSIG__` Defined as a string literal that contains the signature of the enclosing function. The macro is defined only within a function. The `__FUNCSIG__` macro isn't expanded if you use the /EP or /P compiler option. When compiled for a 64-bit target, the calling convention is **`__cdecl`** by default. For an example of usage, see the `__FUNCDNAME__` macro.

- `__FUNCTION__` Defined as a string literal that contains the undecorated name of the enclosing function. The macro is defined only within a function. The `__FUNCTION__` macro isn't expanded if you use the /EP or /P compiler option. For an example of usage, see the `__FUNCDNAME__` macro.

- `_INTEGRAL_MAX_BITS` Defined as the integer literal value 64, the maximum size (in bits) for a non-vector integral type. This macro is always defined.

C++ Copy

```
// integral_max_bits.cpp
#include <stdio.h>
int main() {
    printf("%d\n", _INTEGRAL_MAX_BITS);
}
```

- `__INTELLISENSE__` Defined as 1 during an IntelliSense compiler pass in the Visual Studio IDE. Otherwise, undefined. You can use this macro to guard code the IntelliSense compiler doesn't understand, or use it to toggle between the build and IntelliSense compiler. For more information, see Troubleshooting Tips for IntelliSense Slowness.

- `_ISO_VOLATILE` Defined as 1 if the /volatile:iso compiler option is set. Otherwise, undefined.

- `_KERNEL_MODE` Defined as 1 if the **`/kernel`** (Create Kernel Mode Binary) compiler option is set. Otherwise, undefined.

- `_M_AMD64` Defined as the integer literal value 100 for compilations that target x64 processors or ARM64EC. Otherwise, undefined.

- `_M_ARM` Defined as the integer literal value 7 for compilations that target ARM processors. Undefined for ARM64, ARM64EC, and other targets.

- `_M_ARM_ARMV7VE` Defined as 1 when the /arch:ARMv7VE compiler option is set for compilations that target ARM processors. Otherwise, undefined.
- `_M_ARM_FP` Defined as an integer literal value that indicates which /arch compiler option was set for ARM processor targets. Otherwise, undefined.
  - A value in the range 30-39 if no **`/arch`** ARM option was specified, indicating the default architecture for ARM was set (`VFPv3`).
  - A value in the range 40-49 if **`/arch:VFPv4`** was set.
  - For more information, see **`/arch`** (ARM).
- `_M_ARM64` Defined as 1 for compilations that target ARM64. Otherwise, undefined.
- `_M_ARM64EC` Defined as 1 for compilations that target ARM64EC. Otherwise, undefined.
- `_M_CEE` Defined as 001 if any **`/clr`** (Common Language Runtime Compilation) compiler option is set. Otherwise, undefined.
- `_M_CEE_PURE` Deprecated beginning in Visual Studio 2015. Defined as 001 if the /clr:pure compiler option is set. Otherwise, undefined.
- `_M_CEE_SAFE` Deprecated beginning in Visual Studio 2015. Defined as 001 if the /clr:safe compiler option is set. Otherwise, undefined.
- `_M_FP_CONTRACT` Available beginning in Visual Studio 2022. Defined as 1 if the /fp:contract or /fp:fast compiler option is set. Otherwise, undefined.
- `_M_FP_EXCEPT` Defined as 1 if the /fp:except or /fp:strict compiler option is set. Otherwise, undefined.
- `_M_FP_FAST` Defined as 1 if the /fp:fast compiler option is set. Otherwise, undefined.
- `_M_FP_PRECISE` Defined as 1 if the /fp:precise compiler option is set. Otherwise, undefined.
- `_M_FP_STRICT` Defined as 1 if the /fp:strict compiler option is set. Otherwise, undefined.
- `_M_IX86` Defined as the integer literal value 600 for compilations that target x86 processors. This macro isn't defined for x64 or ARM compilation targets.
- `_M_IX86_FP` Defined as an integer literal value that indicates the /arch compiler option that was set, or the default. This macro is always defined when the compilation target is an x86 processor. Otherwise, undefined. When defined, the value is:
  - 0 if the `/arch:IA32` compiler option was set.
  - 1 if the `/arch:SSE` compiler option was set.
  - 2 if the `/arch:SSE2`, `/arch:AVX`, `/arch:AVX2`, `/arch:AVX512` or `/arch:AVX10.1` compiler option was set. This value is the default if an `/arch` compiler option wasn't specified. When `/arch:AVX` is specified, the macro `__AVX__` is also defined. When `/arch:AVX2` is specified, both `__AVX__` and `__AVX2__` are also defined. When `/arch:AVX512` is specified, `__AVX__`, `__AVX2__`, `__AVX512BW__`, `__AVX512CD__`, `__AVX512DQ__`, `__AVX512F_`

_, and __AVX512VL__ are also defined. When /arch:AVX10.1 is specified, __AVX__, __AVX2__, __AVX512BW__, __AVX512CD__, __AVX512DQ__, __AVX512F_ _, __AVX512VL__ and __AVX10_VER__ are also defined.

  - For more information, see **/arch** (x86).

- _M_X64 Defined as the integer literal value 100 for compilations that target x64 processors or ARM64EC. Otherwise, undefined.

- _MANAGED Defined as 1 when the /clr compiler option is set. Otherwise, undefined.

- _MSC_BUILD Defined as an integer literal that contains the revision number element of the compiler's version number. The revision number is the last element of the period-delimited version number. For example, if the version number of the Microsoft C/C++ compiler is 15.00.20706.01, the _MSC_BUILD macro is 1. This macro is always defined.

- _MSC_EXTENSIONS Defined as 1 if the on-by-default **/Ze** (Enable Language Extensions) compiler option is set. Otherwise, undefined.

- _MSC_FULL_VER Defined as an integer literal that encodes the major, minor, and build number elements of the compiler's version number. The major number is the first element of the period-delimited version number, the minor number is the second element, and the build number is the third element.

  For example, if the Microsoft C/C++ compiler version is 19.39.33519, _MSC_FULL_VER is 193933519. Enter cl /? at the command line to view the compiler's version number. This macro is always defined. For more information about compiler versioning, see C++ compiler versioning and specifically Service releases starting with Visual Studio 2017 for more information about Visual Studio 2019 16.8, 16.9, 16.10 and 16.11, which require _MSC_FULL_VER to tell them apart.

- _MSC_VER Defined as an integer literal that encodes the major and minor number elements of the compiler's version number. The major number is the first element of the period-delimited version number and the minor number is the second element. For example, if the version number of the Microsoft C/C++ compiler is 17.00.51106.1, the value of _MSC_VER is 1700. Enter cl /? at the command line to view the compiler's version number. This macro is always defined.

  To test for compiler releases or updates in a given version of Visual Studio or later, use the >= operator. You can use it in a conditional directive to compare _MSC_VER against that known version. If you have several mutually exclusive versions to compare, order your comparisons in descending order of version number. For example, this code checks for compilers released in Visual Studio 2017 and later. Next, it checks for compilers released in or after Visual Studio 2015. Then it checks for all compilers released before Visual Studio 2015:

  C++ Copy

```
#if _MSC_VER >= 1910
// . . .
#elif _MSC_VER >= 1900
// . . .
#else
```

```
// . . .
#endif
```

For more information about Visual Studio 2019 16.8 and 16.9, and 16.10 and 16.11, which share the same major and minor versions (and so have the same value for `_MSC_VER`), see Service releases starting with Visual Studio 2017.

For more information about the history of compiler versioning, and compiler version numbers and the Visual Studio versions they correspond to, see C++ compiler versioning. Also, Visual C++ Compiler Version on the Microsoft C++ team blog.

- `_MSVC_LANG` Defined as an integer literal that specifies the C++ language standard targeted by the compiler. Only code compiled as C++ sets it. The macro is the integer literal value `201402L` by default, or when the /std:c++14 compiler option is specified. The macro is set to `201703L` if the /std:c++17 compiler option is specified. The macro is set to `202002L` if the /std:c++20 compiler option is specified. It's set to a higher, unspecified value when the /std:c++latest option is specified. Otherwise, the macro is undefined.
  The `_MSVC_LANG` macro and /std (Specify language standard version) compiler options are available beginning in Visual Studio 2015 Update 3.

- `__MSVC_RUNTIME_CHECKS` Defined as 1 when one of the /RTC compiler options is set. Otherwise, undefined.

- `_MSVC_TRADITIONAL`:
  - Available beginning with Visual Studio 2017 version 15.8: Defined as 0 when the preprocessor conformance mode /experimental:preprocessor compiler option is set. Defined as 1 by default, or when the /experimental:preprocessor- compiler option is set, to indicate the traditional preprocessor is in use.
  - Available beginning with Visual Studio 2019 version 16.5: Defined as 0 when the preprocessor conformance mode /Zc:preprocessor compiler option is set. Defined as 1 by default, or when the `/Zc:preprocessor-` compiler option is set, to indicate the traditional preprocessor is in use (essentially, `/Zc:preprocessor` replaces the deprecated `/experimental:preprocessor`).

  C++ Copy

  ```
  #if !defined(_MSVC_TRADITIONAL) || _MSVC_TRADITIONAL
  // Logic using the traditional preprocessor
  #else
  // Logic using cross-platform compatible preprocessor
  #endif
  ```

- `_MT` Defined as 1 when **`/MD`** or **`/MDd`** (Multithreaded DLL) or **`/MT`** or **`/MTd`** (Multithreaded) is specified. Otherwise, undefined.

- `_NATIVE_WCHAR_T_DEFINED` Defined as 1 when the /Zc:wchar_t compiler option is set. Otherwise, undefined.

- `_OPENMP` Defined as integer literal 200203, if the **`/openmp`** (Enable OpenMP 2.0 Support) compiler option is set. This value represents the date of the OpenMP specification

implemented by MSVC. Otherwise, undefined.

C++ Copy

```
// _OPENMP_dir.cpp
// compile with: /openmp
#include <stdio.h>
int main() {
    printf("%d\n", _OPENMP);
}
```

- `_PREFAST_` Defined as 1 when the /analyze compiler option is set. Otherwise, undefined.
- `__SANITIZE_ADDRESS__` Available beginning with Visual Studio 2019 version 16.9. Defined as 1 when the /fsanitize=address compiler option is set. Otherwise, undefined.
- `__TIMESTAMP__` Defined as a string literal that contains the date and time of the last modification of the current source file, in the abbreviated, constant length form returned by the CRT asctime function, for example, `Fri 19 Aug 13:32:58 2016`. This macro is always defined.
- `_VC_NODEFAULTLIB` Defined as 1 when the **`/Zl`** (Omit Default Library Name) compiler option is set. Otherwise, undefined.
- `_WCHAR_T_DEFINED` Defined as 1 when the default /Zc:wchar_t compiler option is set. The `_WCHAR_T_DEFINED` macro is defined but has no value if the **`/Zc:wchar_t-`** compiler option is set, and **`wchar_t`** is defined in a system header file included in your project. Otherwise, undefined.
- `_WIN32` Defined as 1 when the compilation target is 32-bit ARM, 64-bit ARM, x86, or x64. Otherwise, undefined.
- `_WIN64` Defined as 1 when the compilation target is 64-bit ARM or x64. Otherwise, undefined.
- `_WINRT_DLL` Defined as 1 when compiled as C++ and both **`/ZW`** (Windows Runtime Compilation) and **`/LD`** or **`/LDd`** compiler options are set. Otherwise, undefined.

No preprocessor macros that identify the ATL or MFC library version are predefined by the compiler. ATL and MFC library headers define these version macros internally. They're undefined in preprocessor directives made before the required header is included.

- `_ATL_VER` Defined in `<atldef.h>` as an integer literal that encodes the ATL version number.
- `_MFC_VER` Defined in `<afxver_.h>` as an integer literal that encodes the MFC version number.