Nadeau software consulting

C/C++ tip: How to detect the operating system type using compiler predefined macros

January 3, 2012
Topics: C/C++

Table of Contents

How to list predefined macros

See How to list compiler predefined macros for instructions on getting a list of macros for the compilers referenced here.

How to detect the operating system type

Throughout the following sections note:

- Red text indicates deprecated macros that don't start with an underscore. C++ compilers, and C compilers in standards compliance mode, do not define them.
- **Green text** indicates recommended macros that are well-supported and useful for detecting a specific OS.

AIX

Developer: IBM
Distributions: AIX
Processors: POWER

```
#if defined(_AIX)
        /* IBM AIX. --------------------------------------------- */

#endif
```

AIX

Macro	GNU GCC/G++	IBM XL C/C++
_AIX	yes	yes
__unix		yes
__unix__	yes	yes

Notes:

- See IBM's notes on Using the GNU C/C++ compiler on AIX.

BSD

Developer: Open source
Distributions: DragonFly BSD, FreeBSD, OpenBSD, NetBSD
Processors: x86, x86-64, Itanium, POWER, SPARC, etc.

```
#if defined(__unix__) || (defined(__APPLE__) && defined(__MACH__))
#include <sys/param.h>
#if defined(BSD)
        /* BSD (DragonFly BSD, FreeBSD, OpenBSD, NetBSD). ---------- */

#endif
#endif
```

BSD

	Clang/LLVM				GNU GCC/G++		
Macro	***DragonFly BSD***	***FreeBSD***	***NetBSD***	***OpenBSD***	***FreeBSD***	***NetBSD***	***OpenBSD***
unix	yes	yes		yes	yes		
__unix	yes	yes		yes	yes		
__unix__	yes	yes	yes	yes	yes	yes	yes
__DragonFly__	yes						
__FreeBSD__		yes			yes		
__NetBSD__			yes			yes	
__OpenBSD__				yes			yes

Notes:

- Compilers for the old BSD base for these distributions defined the **__bsdi__** macro, but none of these distributions define it now. This leaves no generic "BSD" macro defined by the compiler itself, but all UNIX-style OSes provide a **<sys/param.h>** file. On BSD distributions, and only on BSD distributions, this file defines a **BSD** macro that's set to the OS version. Checking for this generic macro is more robust than looking for known BSD distributions with **__DragonFly__**, **__FreeBSD__**, **__NetBSD__**, and **__OpenBSD__** macros.

- Apple's OSX for the Mac and iOS for iPhones and iPads are based in part on a fork of FreeBSD distributed as Darwin. As such, OSX and iOS also define the **BSD** macro within **<sys/param.h>**. However, compilers for OSX, iOS, and Darwin do not define **__unix__**. To detect all BSD OSes, including OSX, iOS, and Darwin, use an **#if/#endif** that checks for **__unix__** along with **__APPLE__** and **__MACH__** (see the later section on OSX and iOS).

HP-UX

Developer: Hewlett-Packard
Distributions: HP-UX
Processors: Itanium

```
#if defined(__hpux)
        /* Hewlett-Packard HP-UX. ---------------------------------- */

#endif
```

HP-UX

Macro	**GNU GCC/G++**	**HP C/aC++**
hpux	yes	
__hpux	yes	yes
unix	yes	
__unix	yes	yes
__unix__	yes	

Linux

Developer: Open source
Distributions: Centos, Debian, Fedora, OpenSUSE, RedHat, Ubuntu
Processors: x86, x86-64, POWER, etc.

```
#if defined(__linux__)
        /* Linux. -------------------------------------------------- */

#endif
```

Linux

Macro	Clang/LLVM	GNU GCC/G++	Intel ICC/ICPC	Oracle Solaris Studio	Portland PGCC/PGCPP	IBM XL C/C++
linux	yes	yes	yes	yes	yes	
__linux	yes	yes	yes	yes	yes	yes
__linux__	yes	yes	yes	yes	yes	yes
__gnu_linux	yes	yes	yes	yes		
unix	yes	yes	yes	yes	yes	
__unix	yes	yes	yes	yes	yes	yes
__unix__	yes	yes	yes	yes	yes	yes

Notes:

- Linux is available for a wide variety of processors, but it is primarily used on x86 and x86-64 processors. The above table's compilers are all for these processors. Some of them may not be available for other processors.
- There are no predefined compiler macros indicating the specific Linux distribution. At run-time you can read `/proc/version` to get the distribution name and version, or invoke `uname -a` from a `Makefile` then set your own macro. However, writing Linux distribution-specific code is rarely necessary due to high compatibility between distributions.
- Linux is POSIX compliant and defines the standard `_POSIX*` macros in `<unistd.h>`. While Linux is compliant with the latest POSIX.1-2008 specification, Linux distributions erroneously set `_POSIX_VERSION` to `200809L`, instead of leaving it set to `200112L`, as required by the POSIX specification. In practice, this is not a big issue since the existence, not value, of the `_POSIX_VERSION` macro is sufficient to detect POSIX compliance. After that, the individual `_POSIX_*` feature macros provide better information about which specific POSIX features are implemented (see later in this article for POSIX discussion).

OSX, iOS, and Darwin

Developer: Apple and open source
Distributions: OSX, iOS, Darwin
Processors: x86, x86-64, ARM

```
#if defined(__APPLE__) && defined(__MACH__)
        /* Apple OSX and iOS (Darwin). ------------------------------ */
#include <TargetConditionals.h>
#if TARGET_IPHONE_SIMULATOR == 1
        /* iOS in Xcode simulator */

#elif TARGET_OS_IPHONE == 1
        /* iOS on iPhone, iPad, etc. */

#elif TARGET_OS_MAC == 1
        /* OSX */

#endif
#endif
```

OSX and Darwin

Macro	Clang/LLVM	GNU GCC/G++	Intel ICC/ICPC	Portland PGCC/PGCPP
__APPLE__	yes	yes	yes	yes
__MACH__	yes	yes	yes	yes

iOS

Macro	Clang/LLVM	GNU GCC/G++
__APPLE__	yes	yes
__MACH__	yes	yes

Notes:

- For Apple's OSes, all compilers define `__APPLE__` and `__MACH__` macros. The `__MACH__` macro indicates the MACH kernel at the heart of OSX/iOS and partially derived from the obsolete NeXTSTEP. For rigor, *both* of these macros must be defined to detect OSX/iOS. If only `__MACH__` is defined, the OS is NeXTSTEP or one of the other OSes derived from the MACH kernel.
- All OSX compilers are available from the command-line. Apple's Xcode IDE can be configured to invoke any of them from the GUI.
- Mac OSX and iOS include BSD UNIX components originally from FreeBSD. The open source parts of Mac OSX and iOS are distributed as Darwin, without Apple's proprietary user interface and tools. Despite being UNIX-like, Mac OSX and iOS compilers *do not define* the conventional `__unix__`, `__unix`, or `unix` macros. They do define the `BSD` macro in `<sys/param.h>` (see later discussion about BSD).

- Some on-line lists of compiler macros (like this one) list **__MACOSX__**. Some forum comments (like these) claim **__OSX__** exists. *These are incorrect.* There are no such macros predefined by OSX compilers, but they may be defined by specific project Makefiles and platform-detector scripts like GNU **autoconf**.

- Some lists (like this one) still include **macintosh** or **Macintosh** macros. These were only available on the obsolete Mac OS 9 discontinued back in 2002.

- Some forum advice and books (like this one) claim that the **__APPLE__** macro is only defined by Apple's own compilers. *This is incorrect.* While it's true that compilers distributed by Apple define this macro, so do OSX distributions of Intel's ICC, the old IBM XL for PowerPC, and the latest direct downloads of Clang and GCC from open source web sites.

- OSX and iOS compilers do not define macros to distinguish between OSX and iOS. However, Apple's **<TargetConditionals.h>** in each platform's SDK provides **TARGET_*** macros that indicate the OS. All of the macros exist for all platforms, but their values change between **0** and **1** flags as follows:

TargetConditionals.h

	Mac OSX	iOS	iOS Simulator
TARGET_OS_EMBEDDED	0	1	0
TARGET_OS_IPHONE	0	1	1
TARGET_OS_MAC	1	1	1
TARGET_IPHONE_SIMULATOR	0	0	1

- Note that the above macros are *not mutually exclusive*: **TARGET_OS_MAC** is set to **1** for *all* platforms, and **TARGET_OS_IPHONE** is **1** for iOS *and* the simulator. To detect OSX vs. iOS vs. the iOS simulator you have to check the macro values in a specific order (see below).

- There are no macros to distinguish at compile time between an iPhone, iPad, or other Apple devices using the same OSes.

Solaris

Developer: Oracle and open source
Distributions: Oracle Solaris, Open Indiana
Processors: x86, x86-64, SPARC

```
#if defined(__sun) && defined(__SVR4)
        /* Solaris. ------------------------------------------------- */

#endif
```

Solaris

Macro	Clang/LLVM	GNU GCC/G++	Oracle Solaris Studio
sun	yes	yes	yes
__sun	yes	yes	yes
__sun__	yes	yes	
__SunOS			yes
__svr4__	yes	yes	
__SVR4	yes	yes	yes
unix	yes	yes	yes
__unix	yes	yes	yes
__unix__	yes	yes	

Notes:

- To conform with long-standing convention, Solaris compilers define **__sun** and **sun**, despite Oracle's acquisition of Sun Microsystems in 2010. Clang and GCC also define **__sun__**, but Solaris Studio does not.

- Oracle compilers do not define an "oracle" macro or a "solaris" macro.

- Checking for **__sun** is not sufficient to identify Solaris. Compilers for the obsolete BSD-based SunOS also defined **__sun** (and Solaris Studio still defines **__SunOS**, even on System V-based Solaris). To identify Solaris specifically, the **__sun** *and* **__SVR4** macros must be defined. Also note that you need to check for upper-case **__SVR4** instead of the lower-case **__svr4** that's only defined by GCC and not by Solaris Studio.

Windows with Cygwin (POSIX)

Developer: Open source
Distributions: Cygwin

Processors: x86

```
#if defined(__CYGWIN__) && !defined(_WIN32)
        /* Cygwin POSIX under Microsoft Windows. -------------------- */

#endif
```

Cygwin building for POSIX

Macro	Clang/LLVM	GNU GCC/G++
__CYGWIN__	yes	yes
__CYGWIN32__	yes	yes
unix	yes	yes
__unix	yes	yes
__unix__	yes	yes

Notes:

- Cygwin provides a POSIX development environment for Windows, including shells, command-line tools, and compilers. Using Cygwin's libraries, POSIX applications can be built and run under Windows without any Windows-specific code.
- Cygwin POSIX libraries are 32-bit-only, so 64-bit POSIX applications cannot be built. Some code found on-line references **__CYGWIN64__**. However, there is no 64-bit Cygwin, so this macro is never defined. It exists only in forum discussions about a possible future 64-bit Cygwin.
- Clang/LLVM and GCC both can build POSIX or Windows applications. The table above shows macros when building POSIX applications. See the Windows section later in this article for macros when building Windows applications. Comparing the macro set for both types of applications note that **__CYGWIN__** and the standard **__unix__** macros are *always* defined by GCC, even when building a Windows application. For this reason, detecting POSIX builds under Cygwin must use an **#if/#endif** that checks that **__CYGWIN__** is defined, but **_WIN32** is not.
- Checking for Cygwin POSIX builds probably isn't necessarily at all. The whole point of Cygwin is to run standard POSIX applications under Windows, so checking for Cygwin explicitly shouldn't be needed.
- Intel's compilers are not supported under Cygwin, but users have hacked running them from a Cygwin bash command line. However, the compilers still build Windows applications, not POSIX applications.
- Portland Group's compilers for Windows come with a Cygwin install that enables the compilers to be run from a bash command-line, but they still build Windows applications, not POSIX applications.

Windows, Cygwin (non-POSIX), and MinGW

Developer: Microsoft
Distributions: Windows XP, Vista, 7, 8
Processors: x86, x86-64

```
#if defined(_WIN64)
        /* Microsoft Windows (64-bit). ------------------------------ */

#elif defined(_WIN32)
        /* Microsoft Windows (32-bit). ------------------------------ */

#endif
```

Windows

	Clang/LLVM *(Windows target)*		Clang/LLVM *(MinGW target)*		GNU GCC/G++ *(Windows target)*	GNU GCC/G++ *(MinGW target)*		Intel ICC/ICPC		Portland PGCC/PGCPP		Microsoft Visual Studio	
Macro	***32-bit***	***64-bit***	***32-bit***	***64-bit***	***32-bit***	***32-bit***	***64-bit***	***32-bit***	***64-bit***	***32-bit***	***64-bit***	***32-bit***	***64-bit***
__CYGWIN__					yes								
__CYGWIN32__					yes								
__MINGW32__			yes	yes		yes	yes						
__MINGW64__				yes			yes						
unix					yes								
__unix					yes								
__unix__					yes								
WIN32			yes		yes	yes	yes						
_WIN32	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes

__WIN32		yes	yes	yes	yes		yes	yes	
__WIN32__		yes	yes	yes	yes		yes	yes	
WIN64			yes		yes				
_WIN64	yes		yes		yes	yes		yes	yes
__WIN64			yes		yes			yes	
__WIN64__			yes		yes			yes	
WINNT		yes	yes	yes	yes				
__WINNT		yes		yes	yes				
__WINNT__		yes		yes	yes				

Notes:

- Clang/LLVM and GCC under Windows run within the Cygwin POSIX environment or the MinGW minimal GNU environment. Both provide a bash shell and assorted command-line utilities. Cygwin also provides POSIX libraries while MinGW does not.
- Based on command-line options, the Clang/LLVM and GCC compilers can build Windows applications or POSIX applications that run under Windows using POSIX compatibility libraries. Predefined macros for POSIX applications are described in the previous section of this article. The table above is strictly for compiling Windows applications.
- Clang/LLVM can build Windows applications using the Windows target (e.g. "**-ccc-host-triple i386-pc-win32**") or the MinGW target (e.g. "**-ccc-host-triple i386-pc-mingw32**"). The "**-m32**" option builds 32-bit applications and "**-m64**" builds 64-bit.
- GCC under Cygwin can build Windows applications using the "**-mwin32**" command-line option. While GCC is capable of building 64-bit applications, Cygwin is 32-bit only and the version of GCC included with it only builds 32-bit applications.
- Oddly enough, GCC under Cygwin predefines UNIX macros even when building Windows applications.
- Some on-line code references **__CYGWIN64__**. Since there is no 64-bit Cygwin, this macro is never defined. It exists only in forum discussions about a possible future 64-bit Cygwin.
- GCC under MinGW can build Windows applications using the "**-mwin32**" command-line option. The "**-m32**" and "**-m64**" options build 32-bit and 64-bit applications.
- While Clang/LLVM, GCC, and Portland Group compilers define a lot of WIN32 and WIN64 macros with various numbers of underscores, the only macros that matter are those that are compatible with Microsoft's Visual Studio: **_WIN32** and **_WIN64**.
- Some on-line advice recommends checking for **_MSC_VER**. The macro is defined with the compiler version number for Clang/LLVM, ICC, and Visual Studio, but it isn't defined by GCC or Portland Group compilers.
- Some lists of predefined macros (like this one) include additional macros for discontinued products, such as **__TOS_WIN__** for IBM's XL compiler on Windows (XL is still available for AIX and Linux), and **__WINDOWS__** for the discontinued but open sourced Watcom compiler.

How to detect POSIX and UNIX

POSIX and UNIX are not operating systems. Rather they are formal or de facto standards followed to some degree by all UNIX-style OSes.

POSIX

Developer: Standard
Distributions: All current UNIX-style OSes, including BSD, Linux, OSX, and Solaris
Processors: x86, x86-64, ARM, POWER, SPARC, etc.

```
#if !defined(_WIN32) && (defined(__unix__) || defined(__unix) || (defined(__APPLE__) && defined(__MACH__)))
        /* UNIX-style OS. ------------------------------------------- */
#include <unistd.h>
#if defined(_POSIX_VERSION)
        /* POSIX compliant */

#endif
#endif
```

All UNIX-style OSes (see also UNIX below) have **<unistd.h>** that defines macros indicating the level of POSIX compliance. The **_POSIX_VERSION** macro value indicates the version of the standard with which the OS is compliant. Known values are:

- **198808L** for POSIX.1-1988

- **199009L** for POSIX.1-1990
- **199506L** for ISO POSIX.1-1996
- **200112L** for ISO POSIX.1-2001
- **200809L** for ISO POSIX.1-2008

Another way to detect ISO POSIX.1-2008 compliance is with a run-time check using **sysconf**.

```
if ( sysconf( _SC_VERSION ) >= 200809L )
{
  /* POSIX.1-2008 */

}
else
{
  /* Pre-POSIX.1-2008 */

}
```

While the **#if/#endif** and **sysconf** call above will both work to detect broad POSIX compliance, it's more useful to check for individual POSIX features flagged by macros in **<unistd.h>**. The POSIX specification has a long list of these macros, but a few of the more useful ones include:

- **_POSIX_IPV6** indicates IPv6 address support.
- **_POSIX_MAPPED_FILES** indicates memory mapping support.
- **_POSIX_SEMAPHORES** indicates semaphore support for multi-threading.
- **_POSIX_THREADS** indicates pthreads support. A number of **_POSIX_THREAD*** macros then indicate whether thread resource usage can be reported or thread scheduling priority controlled.

UNIX

Developer: De facto standard
Distributions: All current UNIX-style OSes, including BSD, Linux, OSX, and Solaris
Processors: x86, x86-64, ARM, POWER, SPARC, etc.

```
#if !defined(_WIN32) && (defined(__unix__) || defined(__unix) || (defined(__APPLE__) && defined(__MACH__)))
        /* UNIX-style OS. ------------------------------------------- */

#endif
```

UNIX (Clang/LLVM compilers)

Macro	Cygwin (POSIX)	DragonFly BSD	FreeBSD	iOS	Linux	NetBSD	OpenBSD	OSX	Solaris
unix	yes	yes	yes		yes		yes		yes
__unix	yes	yes	yes		yes		yes		yes
__unix__	yes	yes	yes		yes	yes	yes		yes

UNIX (GCC compilers)

Macro	AIX	Cygwin (POSIX)	FreeBSD	iOS	HP-UX	Linux	NetBSD	OpenBSD	OSX	Solaris
unix		yes	yes		yes	yes				yes
__unix		yes	yes		yes	yes				yes
__unix__	yes	yes	yes		yes	yes	yes	yes		yes

UNIX (Other compilers)

	AIX	HP-UX	Linux			OSX			Solaris
Macro	IBM XL C/C++	HP C/aC++	Intel ICC/ICPC	Oracle Solaris Studio	Portland PGCC/PGCPP	IBM XL C/C++	Intel ICC/ICPC	Portland PGCC/PGCPP	Oracle Solaris Studio
unix			yes	yes	yes				yes
__unix	yes	yes	yes	yes	yes	yes			yes
__unix__	yes		yes	yes	yes	yes			

Notes:

- There is no single UNIX macro defined by all compilers on all UNIX-style OSes. An **#if/#endif** that checks multiple macros is required.
- Compilers for Apple's OSX and iOS don't define *any* UNIX macros. An **#if/#endif** that checks **__APPLE__** and **__MACH__** is required (see the earlier section OSX and iOS).
- GCC under Cygwin defines UNIX macros *even when building Windows applications*. An **#if/#endif** that excludes **_WIN32** is required to detect UNIX builds on Cygwin (see the earlier section on Windows).

Other ways to detect the operating system type

On UNIX-style OSes a common way to detect OS features is to use GNU's `autoconf`. This tool builds configuration shell scripts that automatically check for OS and compiler features, build Makefiles, and set compiler flags. For code that only targets UNIX-style OSes, this works well. But `autoconf` doesn't work for code that must compile on non-UNIX-style OSes (e.g. Windows) or within an IDE. And it's way overkill for many projects where a simple `#if/#endif` set will do.

From the command line there are several ways to detect the OS. On UNIX-style OSes, the `uname` command reports the OS name. On Windows, the `ver` and `winver` commands report the OS name. On Linux, the `/proc/version` virtual file reports the Linux kernel version. But using any of these to automatically configure code requires scripts and Makefiles. And those have the same problems as `autoconf`.

The most elegant solution is to eschew all detection scripts and simply use the above predefined macros *already available on every OS* and designed specifically for use in `#if/#endif` sets for OS-specific code. Don't reinvent the wheel.

Further reading

Related articles at NadeauSoftware.com

C/C++ tip: How to list compiler predefined macros explains how to get a compiler's macros by using command-line options and other methods.

C/C++ tip: How to detect the compiler name and version using compiler predefined macros provides `#if/#endif` sets for detecting common compilers.

C/C++ tip: How to detect the processor type using compiler predefined macros provides `#if/#endif` sets for detecting desktop and server processors using compiler macros.

Web articles

Operating Systems at Sourceforge.net provides a list of OSes and their compiler predefined macros. However the list is cluttered with obsolete OSes (OS/2? Palm OS?), doesn't note macros shared across many OSes (like `__unix__`), and doesn't include discussion.

Pre-defined C/C++ Compiler Macros at beefchunk.com has a list of OSes and predefined macros. Like the Sourceforge list, the information is a bit cluttered with obsolete OSes (DG/UX? Unicos?) and lacks discussion.

Operating System Resources at Apache.org has a good list of OSes, links to vendor documentation, and some tables of compiler predefined macros.

Comments