JamPlus manual: Quick Start Guide

Archived: 2026-04-05 17:35:12 UTC

Overview

JamPlus works out of the box with C/C++ compilation for Visual C++ compilers, GCC, MinGW, and others. It can also generate workspaces for Visual Studio 20xx, Visual C++ 6, Xcode, and CodeBlocks.

When the Jam executable is run, it reads a file called `Jamfile.jam` in the current working directory. The Jamfile contains a description of how to build the current project. In the tutorials below, we'll talk about how to properly set up a Jamfile build description.

Tutorial 0: Use Jam to Build Simple Applications

Building helloworld

For this tutorial, we are going to make a basic **helloworld** application.

First, we need to create a `helloworld.c` file. It will do nothing more than print **Hello, world!** to the user.

#include <stdio.h>

int main()

{

printf("Hello, world!\n");

return 0;

}

Straight out of the box, you can run `jam` within the directory of `helloworld.c`, and Jam will build a `helloworld` executable using all of the `*.c`, `*.cpp`, `*.m`, and `*.mm` files in the directory.

helloworld.exe (or the equivalent) is generated. It might not be named helloworld.exe, because

the first source file title in the directory is used as the executable name.

jam

Remove the executable and all intermediate files.

```
jam clean

# helloworld.exe (or the equivalent) is generated. It might not be named helloworld.exe, because

# the first source file title in the directory is used as the executable name. To override this,

# add helloworld to the command-line:

jam helloworld

# Remove the helloworld executable and all intermediate files.

jam clean:helloworld
```

While this is a convenient Jam feature, generally a more complex build is needed. We'll make one of those in Tutorial 1: Hello World!.

Tutorial 1: Hello World!

Initial Setup

For this tutorial, we are going to make a basic **helloworld** application.

First, we need to create a `helloworld.c` file. It will do nothing more than print **Hello, world!** to the user.

```
#include <stdio.h>

int main()

{

printf("Hello, world!\n");

return 0;

}
```

In the same directory as `helloworld.c`, we also need to create `Jamfile.jam` with the description of how to build the basic **helloworld** application.

```
# This file is Jamfile.jam.

#

# Please note that Jam tokens are whitespace separated. Even the semicolon
```

delineating the end of statements has whitespace before it.

C.Application helloworld : helloworld.c ;

Compiling the Tutorial

Compiling the **helloworld** application is simple. Assuming the Jam executable is in your `PATH`, run the following from the directory containing your `Jamfile.jam` and `main.c`:

That's it. Jam will detect your OS and then the appropriate compilers for your platform. On Windows, Jam looks for Visual Studio 2015, then 2013, 2012, 2010, 2008, and so on until Visual C++ 6. If it can't find any of those, it looks for a MinGW installation in the `c:/Program Files/mingw-x64/` and `c:/mingw/` directories. On Mac OS X, Jam looks for clang or gcc. When a compiler is found, the **helloworld** application's Release configuration is built. The resultant filename will be `.build/win64-release/TOP/helloworld/helloworld.exe` on Windows and in a similar location but with an executable name of `helloworld` on other platforms.

Building the **helloworld** application Debug configuration is performed with a command line flag. In this case, the built executable will be `.build/win64-debug/TOP/helloworld/helloworld.exe` on Windows and in a similar location but with an executable name removing the `.exe` on other platforms.

or, preferred:

If you would like to run with another compiler by default, such as MinGW, an additional command line flag must be provided.

Cleaning Up the Tutorial Files

Compiler intermediates and outputs are, by default, stored within a `.build/` directory. These files can be cleaned up using the `clean` target.

rem Cleans the default platform's Release build.

jam clean

rem Cleans the default platform's Release build, too.

jam CONFIG=release clean

rem Cleans the Debug build.

jam CONFIG=debug clean

rem Cleans the Debug build.

jam C.TOOLCHAIN=-/debug clean

Building an IDE Workspace

One of the exciting features of JamPlus is its ability to build IDE workspaces for Visual Studio or Xcode or others. It does so with a set of scripts that are present in the JamPlus `bin/` directory. These scripts create an out-of-source build tree optimized for best performance within JamPlus. By being out-of-source, the source tree remains pristine, and all compiler intermediates are stored in a separate location.

For the **helloworld** application, let's build a Visual Studio 2015 solution on Windows or an Xcode workspace on macOS.

It may be preferable to output to a different directory than `.build/`.

jam --workspace --gen=vs2015 Jamfile.jam build

After running `jam -workspace`, the new out-of-source work area will have been created. Inside the generated `build/_workspaces_/IDE/` directory, you'll find a `helloworld.sln`. Open this file in Visual Studio to proceed.

When Visual Studio has loaded the `helloworld.sln`, you'll find it contains the **helloworld** project within the Solution Explorer. If the **helloworld** project is not already the active project, right click on it and choose **Set as StartUp Project**.

On Mac OS X, you'll find the appropriate .xcodeproj as `build/_workspaces_/xcode/helloworld.xcworkspace`. Open this project file in Xcode, and choose the appropriate scheme (such as `helloworld-release`).

At this point, build as you normally would. Change between the Debug and Release configurations if you'd like. In fact, you can even debug as if the project were manually created!
