**15 December 2024 at 16:28**

**You said:**

I amtrying to build python lz4 on windows 11 from source (for python 3.13 as pre-built wheel does not exist). This package is required as dependency for dask[complete] package.  
  
I have changed setup.cfg to include:  
  
[build] compiler = mingw32  
  
at the end. But still encounter these errors:  
  
Building wheels for collected packages: lz4 Building wheel for lz4 (pyproject.toml) ... error error: subprocess-exited-with-error × Building wheel for lz4 (pyproject.toml) did not run successfully. │ exit code: 1 ╰─> [25 lines of output] D:\Programs\Msys2\tmp\pip-build-env-qunz488p\overlay\Lib\site-packages\setuptools\\_distutils\dist.py:261: UserWarning: Unknown distribution option: 'tests\_require' warnings.warn(msg) running bdist\_wheel running build running build\_py creating build\lib.win-amd64-cpython-313\lz4 copying lz4\version.py -> build\lib.win-amd64-cpython-313\lz4 copying lz4\\_\_init\_\_.py -> build\lib.win-amd64-cpython-313\lz4 creating build\lib.win-amd64-cpython-313\lz4\block copying lz4\block\\_\_init\_\_.py -> build\lib.win-amd64-cpython-313\lz4\block creating build\lib.win-amd64-cpython-313\lz4\frame copying lz4\frame\\_\_init\_\_.py -> build\lib.win-amd64-cpython-313\lz4\frame running build\_ext building 'lz4.\_version' extension creating build\temp.win-amd64-cpython-313\Release\lz4 gcc -shared -O -Wall -ID:\Programs\Msys2\home\j1304\venv313\include -ID:\Programs\Python313\include -ID:\Programs\Python313\Include -c lz4/\_version.c -o build\temp.win-amd64-cpython-313\Release\lz4\\_version.o /Ot /Wall /wd4711 /wd4820 gcc.EXE: warning: /Ot: linker input file unused because linking not done gcc.EXE: error: /Ot: linker input file not found: No such file or directory gcc.EXE: warning: /Wall: linker input file unused because linking not done gcc.EXE: error: /Wall: linker input file not found: No such file or directory gcc.EXE: warning: /wd4711: linker input file unused because linking not done gcc.EXE: error: /wd4711: linker input file not found: No such file or directory gcc.EXE: warning: /wd4820: linker input file unused because linking not done gcc.EXE: error: /wd4820: linker input file not found: No such file or directory error: command 'D:\\Programs\\Msys2\\ucrt64\\bin\\gcc.EXE' failed with exit code 1 [end of output] note: This error originates from a subprocess, and is likely not a problem with pip. ERROR: Failed building wheel for lz4 Failed to build lz4 ERROR: ERROR: Failed to build installable wheels for some pyproject.toml based projects (lz4)

**Copilot said:**

It looks like you're encountering issues with building the lz4 package from source on Windows 11 for Python 3.13. The errors you're seeing are related to the GCC compiler not recognizing certain flags or files.

Here are a few steps you can try to resolve this issue:

1. **Ensure MSYS2 and MinGW are properly installed and updated**:
   * Open the MSYS2 terminal and update the package database and core system packages:
   * pacman -Syu
   * Install the necessary development tools:
   * pacman -S base-devel mingw-w64-x86\_64-toolchain
2. **Set the correct environment variables**:
   * Make sure your environment variables are set correctly to use the MinGW compiler. You can add the following to your .bashrc or .bash\_profile:
   * export PATH="/mingw64/bin:$PATH"
3. **Modify the setup.cfg file**:
   * Ensure that the setup.cfg file is correctly configured to use the MinGW compiler. It should look something like this:
   * [build]
   * compiler = mingw32
4. **Install the lz4 package using pip with the correct compiler flags**:
   * You might need to specify additional compiler flags to ensure the build process uses the correct paths and options. Run the following command in your MSYS2 terminal:
   * CFLAGS="-O2 -Wall" pip install lz4
5. **Check for any missing dependencies**:
   * Make sure all dependencies required for building lz4 are installed. Sometimes, additional libraries or headers might be needed.

If these steps don't resolve the issue, you might want to check the [GitHub issues page for the python-lz4 project](https://github.com/python-lz4/python-lz4/issues/290) for any similar issues or additional troubleshooting steps

1

2

3

.

Let me know if you need further assistance!

1After upgrading Python from 3.12.5 to 3.13: Missing \_version ... - GitHub2Support free threaded builds · Issue #292 · python-lz4/python-lz43Failed building wheel for lz4 · Issue #43 · steeve/python-lz4

Edit in Pages

Copy

1 of 30 responsesAI-generated content may be incorrect

**15 December 2024 at 16:31**

**You said:**

What are options here if gcc does not have these switches:  
  
gcc -shared -O -Wall -ID:\Programs\Msys2\home\j1304\venv313\include -ID:\Programs\Python313\include -ID:\Programs\Python313\Include -c lz4/\_version.c -o build\temp.win-amd64-cpython-313\Release\lz4\\_version.o /Ot /Wall /wd4711 /wd4820 gcc.EXE: warning: /Ot: linker input file unused because linking not done gcc.EXE: error: /Ot: linker input file not found: No such file or directory gcc.EXE: warning: /Wall: linker input file unused because linking not done gcc.EXE: error: /Wall: linker input file not found: No such file or directory gcc.EXE: warning: /wd4711: linker input file unused because linking not done gcc.EXE: error: /wd4711: linker input file not found: No such file or directory gcc.EXE: warning: /wd4820: linker input file unused because linking not done gcc.EXE: error: /wd4820: linker input file not found: No such file or directory  
Are these specific to visual studio? check. What are alternatives for gcc?

**Copilot said:**

The errors you're encountering suggest that the /Ot, /Wall, /wd4711, and /wd4820 switches are not recognized by GCC. These switches are specific to the Microsoft Visual Studio (MSVC) compiler, not GCC. Here are some alternatives and steps you can take:

1. **Use MSVC Compiler**:
   * Since the flags are specific to MSVC, you can try using the Microsoft Visual C++ Build Tools. You can install them from the Visual Studio Build Tools.
   * Modify your setup.cfg to use the MSVC compiler:
   * [build]
   * compiler = msvc
2. **Remove Unsupported Flags**:
   * If you prefer to stick with GCC, you can remove or replace the unsupported flags in your build configuration. For example, replace /Ot with -O2 and /Wall with -Wall. The /wd4711 and /wd4820 flags can be omitted as they are specific to MSVC warnings.
3. **Alternative Compilers**:
   * **MinGW**: A popular alternative that provides a port of GCC for Windows. It includes the necessary tools to build native Windows applications

1

.

* + **LLVM/Clang**: Another powerful compiler that can be used as an alternative to GCC. It is compatible with many GCC flags and provides excellent diagnostics

1

.

* + **TDM-GCC**: A GCC distribution for Windows that includes additional libraries and tools

1

.

1. **Install and Configure MinGW**:
   * If you choose MinGW, ensure it is properly installed and configured:
   * pacman -S mingw-w64-x86\_64-gcc
   * Update your setup.cfg to use MinGW:
   * [build]
   * compiler = mingw32
2. **Check for Dependencies**:
   * Ensure all necessary dependencies and development tools are installed. Sometimes, additional libraries or headers might be required for the build process.

By following these steps, you should be able to resolve the issues with building the lz4 package from source on Windows 11. Let me know if you need further assistance!

1

: [GNU Compiler Collection Alternatives](https://alternativeto.net/software/gnu-compiler-collection/)

1GNU Compiler Collection Alternatives: Top 10 Compilers & Similar Apps

Edit in Pages

Copy

2 of 30 responsesAI-generated content may be incorrect

**15 December 2024 at 16:31**

**You said:**

I already did this:  
  
[build] compiler = mingw32  
  
You fogot. I mentioned in the very beginning.

**Copilot said:**

You're right, my apologies for missing that detail. Since you've already set the compiler to mingw32 in your setup.cfg, let's try a few more steps to resolve the issue:

1. **Remove Unsupported Flags**:
   * The errors indicate that GCC does not recognize the /Ot, /Wall, /wd4711, and /wd4820 flags. These are MSVC-specific flags. You should remove these flags from the build configuration. Instead, use GCC-compatible flags like -O2 and -Wall.
2. **Modify the Build Command**:
   * You can manually specify the correct flags when running the build command. For example:
   * CFLAGS="-O2 -Wall" python setup.py build
3. **Ensure Correct Path Configuration**:
   * Make sure your environment variables are correctly set to use the MinGW compiler. Add the MinGW bin directory to your PATH:
   * export PATH="/mingw64/bin:$PATH"
4. **Install Required Packages**:
   * Ensure all necessary development tools and libraries are installed. You can install them using MSYS2:
   * pacman -S base-devel mingw-w64-x86\_64-toolchain
5. **Check for Missing Dependencies**:
   * Sometimes, additional libraries or headers might be required. Ensure all dependencies are installed.
6. **Use a Virtual Environment**:
   * Create a virtual environment and install the package within it to avoid conflicts with other installed packages:
   * python -m venv myenv
   * source myenv/bin/activate
   * pip install lz4

If these steps don't resolve the issue, you might want to check the GitHub issues page for the python-lz4 project for any similar issues or additional troubleshooting steps.

Let me know if you need further assistance!

Edit in Pages

Copy

3 of 30 responsesAI-generated content may be incorrect

**15 December 2024 at 16:42**

**You said:**

Apart from setup.cfg can tweaking toml file or setup.py can resolve. Can we point out in either of these files to mingw64. There must be something in these file which detect os as windows and compiling with VS switches instead of mingw32?

**Copilot said:**

Yes, you can tweak the pyproject.toml or setup.py files to ensure the build process uses mingw64 instead of defaulting to Visual Studio (MSVC) switches. Here are some steps you can take:

**Modifying pyproject.toml**

1. **Add a [tool.setuptools] section**:
   * You can specify build options directly in the pyproject.toml file. Here’s an example of how you might configure it:
   * [build-system]
   * requires = ["setuptools", "wheel"]
   * build-backend = "setuptools.build\_meta"
   * [tool.setuptools]
   * compiler = "mingw32"

**Modifying setup.py**

1. **Override the build\_ext command**:
   * You can customize the setup.py to ensure it uses the correct compiler and flags. Here’s an example:
   * from setuptools import setup, Extension
   * from setuptools.command.build\_ext import build\_ext
   * import os
   * class CustomBuildExt(build\_ext):
   * def build\_extensions(self):
   * if os.name == 'nt':
   * for ext in self.extensions:
   * ext.extra\_compile\_args = ['-O2', '-Wall']
   * ext.extra\_link\_args = []
   * build\_ext.build\_extensions(self)
   * setup(
   * name='lz4',
   * version='1.0',
   * ext\_modules=[Extension('lz4.\_version', ['lz4/\_version.c'])],
   * cmdclass={'build\_ext': CustomBuildExt},
   * )

**Ensuring Correct Compiler Usage**

1. **Set Environment Variables**:
   * Make sure your environment variables are set to use mingw64. You can add the following to your .bashrc or .bash\_profile:
   * export PATH="/mingw64/bin:$PATH"
2. **Install Required Packages**:
   * Ensure all necessary development tools and libraries are installed using MSYS2:
   * pacman -S base-devel mingw-w64-x86\_64-toolchain

By making these changes, you should be able to ensure that the build process uses mingw64 instead of defaulting to MSVC switches. Let me know if you need further assistance!

Edit in Pages

Copy

4 of 30 responsesAI-generated content may be incorrect

**15 December 2024 at 16:45**

**You said:**

See I have this section in setup.py:  
  
if compiler == 'msvc': extension\_kwargs['extra\_compile\_args'] = [ '/Ot', '/Wall', '/wd4711', '/wd4820', ] elif compiler in ('unix', 'mingw32'): if liblz4\_found is True and use\_system\_liblz4 is True: extension\_kwargs = pkgconfig\_parse('liblz4') else: extension\_kwargs['extra\_compile\_args'] = [ '-O3', '-Wall', '-Wundef' ] else: print('Unrecognized compiler: {0}'.format(compiler)) sys.exit(1)  
I fail to undetstanf why did not recognize build command on setup.cfg?

**Copilot said:**

It looks like the setup.py script is correctly set up to handle different compilers, but the build process might not be recognizing the setup.cfg configuration. Here are a few things to check and try:

1. **Ensure setup.cfg is in the Correct Directory**:
   * Make sure that setup.cfg is in the same directory as setup.py. The build process needs to find setup.cfg in the root of your project directory.
2. **Check the setup.cfg Syntax**:
   * Verify that the syntax in setup.cfg is correct. It should look something like this:
   * [build]
   * compiler = mingw32
3. **Explicitly Specify the Compiler in setup.py**:
   * You can add a check in setup.py to explicitly set the compiler if it is not being picked up from setup.cfg. Here’s an example:
   * from setuptools import setup, Extension
   * import os
   * compiler = os.environ.get('COMPILER', 'mingw32')
   * if compiler == 'msvc':
   * extra\_compile\_args = ['/Ot', '/Wall', '/wd4711', '/wd4820']
   * elif compiler in ('unix', 'mingw32'):
   * extra\_compile\_args = ['-O3', '-Wall', '-Wundef']
   * else:
   * print('Unrecognized compiler: {0}'.format(compiler))
   * sys.exit(1)
   * setup(
   * name='lz4',
   * version='1.0',
   * ext\_modules=[Extension('lz4.\_version', ['lz4/\_version.c'], extra\_compile\_args=extra\_compile\_args)],
   * )
4. **Set Environment Variable**:
   * You can set an environment variable to ensure the correct compiler is used:
   * export COMPILER=mingw32
5. **Use pyproject.toml**:
   * If setup.cfg is not being recognized, you can try using pyproject.toml to specify the build system and compiler:
   * [build-system]
   * requires = ["setuptools", "wheel"]
   * build-backend = "setuptools.build\_meta"
   * [tool.setuptools]
   * compiler = "mingw32"
6. **Check for Overriding Settings**:
   * Ensure there are no other configuration files or environment settings that might be overriding the setup.cfg settings.

By following these steps, you should be able to ensure that the build process recognizes and uses the mingw32 compiler as specified. Let me know if you need further assistance!

Edit in Pages

Copy

5 of 30 responsesAI-generated content may be incorrect

**15 December 2024 at 17:38**

**You said:**

I saw again the errors:  
  
gcc -shared -O -Wall -ID:\Programs\Msys2\home\j1304\venv313\include -ID:\Programs\Python313\include -ID:\Programs\Python313\Include -c lz4/\_version.c -o build\temp.win-amd64-cpython-313\Release\lz4\\_version.o /Ot /Wall /wd4711 /wd4820 gcc.EXE: warning: /Ot: linker input file unused because linking not done gcc.EXE: error: /Ot: linker input file not found: No such file or directory gcc.EXE: warning: /Wall: linker input file unused because linking not done gcc.EXE: error: /Wall: linker input file not found: No such file or directory gcc.EXE: warning: /wd4711: linker input file unused because linking not done gcc.EXE: error: /wd4711: linker input file not found: No such file or directory gcc.EXE: warning: /wd4820: linker input file unused because linking not done gcc.EXE: error: /wd4820: linker input file not found: No such file or directory error: command 'D:\\Programs\\Msys2\\ucrt64\\bin\\gcc.EXE' failed with exit code 1  
  
Since gcc is run proves that mingw32 is recognized. Whether Msys gcc have these switches missing which may be available in linux:  
gcc -shared -O -Wall -ID:\Programs\Msys2\home\j1304\venv313\include -ID:\Programs\Python313\include -ID:\Programs\Python313\Include -c lz4/\_version.c -o build\temp.win-amd64-cpython-313\Release\lz4\\_version.o /Ot /Wall /wd4711 /wd4820 gcc.EXE: warning: /Ot: linker input file unused because linking not done gcc.EXE: error: /Ot: linker input file not found: No such file or directory gcc.EXE: warning: /Wall: linker input file unused because linking not done gcc.EXE: error: /Wall: linker input file not found: No such file or directory gcc.EXE: warning: /wd4711: linker input file unused because linking not done gcc.EXE: error: /wd4711: linker input file not found: No such file or directory gcc.EXE: warning: /wd4820: linker input file unused because linking not done gcc.EXE: error: /wd4820: linker input file not found: No such file or directory error: command 'D:\\Programs\\Msys2\\ucrt64\\bin\\gcc.EXE' failed with exit code 1

**Copilot said:**

It seems that the GCC compiler is being invoked correctly, but the issue lies with the MSVC-specific flags (/Ot, /Wall, /wd4711, /wd4820) that are not recognized by GCC. These flags are indeed specific to the Microsoft Visual Studio compiler and are not available in GCC, whether on Windows or Linux.

To resolve this, you need to ensure that the build process uses GCC-compatible flags. Here’s how you can modify your setup.py to handle this correctly:

**Modify setup.py**

1. **Check for Compiler and Set Flags**:
   * Update your setup.py to ensure it uses the correct flags for GCC. Here’s an example:
   * from setuptools import setup, Extension
   * from setuptools.command.build\_ext import build\_ext
   * import os
   * class CustomBuildExt(build\_ext):
   * def build\_extensions(self):
   * compiler = self.compiler.compiler\_type
   * if compiler == 'msvc':
   * for ext in self.extensions:
   * ext.extra\_compile\_args = ['/Ot', '/Wall', '/wd4711', '/wd4820']
   * elif compiler in ('unix', 'mingw32'):
   * for ext in self.extensions:
   * ext.extra\_compile\_args = ['-O3', '-Wall', '-Wundef']
   * else:
   * print('Unrecognized compiler: {0}'.format(compiler))
   * sys.exit(1)
   * build\_ext.build\_extensions(self)
   * setup(
   * name='lz4',
   * version='1.0',
   * ext\_modules=[Extension('lz4.\_version', ['lz4/\_version.c'])],
   * cmdclass={'build\_ext': CustomBuildExt},
   * )

**Ensure Correct Environment**

1. **Set Environment Variables**:
   * Make sure your environment variables are set to use mingw64. Add the following to your .bashrc or .bash\_profile:
   * export PATH="/mingw64/bin:$PATH"
2. **Install Required Packages**:
   * Ensure all necessary development tools and libraries are installed using MSYS2:
   * pacman -S base-devel mingw-w64-x86\_64-toolchain

**Run the Build Command**

1. **Build the Package**:
   * Run the build command with the correct flags:
   * CFLAGS="-O3 -Wall -Wundef" python setup.py build

By making these changes, you should be able to ensure that the build process uses the correct flags for GCC and avoids the MSVC-specific flags that are causing the errors. Let me know if you need further assistance!

Edit in Pages

Copy

6 of 30 responsesAI-generated content may be incorrect

**15 December 2024 at 17:47**

**You said:**

I want to know which packages depend on lz4. Although I recognized error and could install correctly - lz4. dask still builds with old lz4 and fails at that.

**Copilot said:**

It's great to hear you managed to install lz4 correctly! To identify which packages depend on lz4, you can use the following methods:

1. **Check Dependencies Using pipdeptree**:
   * pipdeptree is a tool that shows the dependency tree of installed Python packages. You can install it using:
   * pip install pipdeptree
   * Then, run:
   * pipdeptree | grep lz4
   * This will list all packages that depend on lz4.
2. **Check dask Dependencies**:
   * Since dask[complete] is the package you're working with, you can check its dependencies directly. Run:
   * pip show dask[complete]
   * This will display information about dask and its dependencies, including lz4.
3. **Manual Inspection**:
   * You can manually inspect the requirements.txt or setup.py files of the packages you're using to see if they list lz4 as a dependency.
4. **Using pip check**:
   * This command checks for broken dependencies and can help identify if lz4 is causing issues:
   * pip check

If dask is still building with an old version of lz4, you might need to ensure that the correct version of lz4 is being used. You can specify the version explicitly when installing dask:

pip install lz4==<desired\_version>

pip install dask[complete]

This should help ensure that dask uses the correct version of lz4. Let me know if you need further assistance!

Edit in Pages

Copy

7 of 30 responsesAI-generated content may be incorrect

**15 December 2024 at 17:48**

**You said:**

$ pipdeptree | grep lz4 lz4==0.1.dev1190+g546d6d5.d20241215

**Copilot said:**

It looks like you have a development version of lz4 installed. To ensure that dask and other packages use the correct version of lz4, you might need to reinstall lz4 and dask with the appropriate versions.

Here are the steps you can follow:

1. **Uninstall the Current lz4**:
2. pip uninstall lz4
3. **Install the Stable Version of lz4**:
4. pip install lz4
5. **Reinstall dask[complete]**:
6. pip install --upgrade --force-reinstall dask[complete]
7. **Verify the Installation**:
   * Check the installed version of lz4 to ensure it's the correct one:
   * pip show lz4

By following these steps, you should be able to ensure that dask and other packages use the correct version of lz4. Let me know if you encounter any issues or need further assistance!

Edit in Pages

Copy

8 of 30 responsesAI-generated content may be incorrect

**15 December 2024 at 18:00**

**You said:**

bang on! I got source from pypi for lz4, uninstalled compiled wheel from github, rebuilt wheel using mingw32 compiler from source downloaded from pypi. Finally successfully installed dask[complete].

**Copilot said:**

That's fantastic news! 🎉 It sounds like you navigated through the challenges and successfully installed dask[complete] with the correct version of lz4. Great job on troubleshooting and resolving the issue!

If you have any more questions or need further assistance with anything else, feel free to ask. Happy coding! 😊

Edit in Pages

Copy

9 of 30 responsesAI-generated content may be incorrect