Meeting C++

Bringing Clang-tidy Magic to Visual Studio C++ Developers

November 11, 2017

Victor Ciura
Technical Lead, Advanced Installer
www.advancedinstaller.com

Intro

Who Am I ?

Intro

Why Am I Here ?

Intro

Why Am I Here ?

“A 14 year old code base under active development, 2.5 million lines of C++ code,
a few brave nerds, two powerful tools and one hot summer…”

or

“How we managed to **clang-tidy** our whole code base,
while maintaining our monthly release cycle”

Context: Advanced Installer

www.advancedinstaller.com

- Powerful Windows Installer authoring tool (**IDE**)
- Helps developers and IT Pros create MSI/EXE, App-V and UWP AppX packages
- **14** year old code base, under active development (since 2003)
- **2.5** million lines of C++ code
- **134** Visual Studio projects (EXEs, DLLs, LIBs)
- Microsoft **Visual Studio 2017**
- **Monthly** release cycle (~3 week sprints)
- **Windows-only** deployment
- Strong Windows **SDK** dependencies: our code has a fairly wide Windows API surface area (because of the application domain)

This talk is NOT about

VS

- We’re a **Windows**-only dev team using Visual C++
- We’re going to continue using **both** **Visual Studio** (2017) and **Clang** tools on the side, to modernize/refactor and improve our code quality

Timeline

It all started a year ago, with `clang-format`

- September, 2016 - started thinking about adopting **clang-format** (experimenting with various configs)
- October-November 2016 - preparing for clang-format adoption:

  👉 establishing internal rules, configs, exceptions, debates, strategy...

- December 16, 2016 - the BIG reformat (formatted ***all*** code with clang-format, using our custom style)
- December 2016-present - team workflow: use **`ClangFormat VS extension`** (auto-format on **save**)

Goals

- Building on the success of **clang-format** adoption within the team, we gained courage to experiment with **clang-tidy**
- New problem: getting all our code to fully ***compile*** with Clang, using the correct project settings (synced with Visual Studio) and Windows SDK dependencies
- We found several compatibility issues between MSVC compiler (VS2017) and Clang (4.0)
- Note that we were already using MSVC **/W4** and **/WX** on all our projects

Goals

- Welcome to the land of **non-standard C++** language extensions and striving for C++ ISO conformance in our code

- We started **fixing** all non-conformant code... (some automation required, batteries not included)

- Perform large scale **refactorings** on our code with clang-tidy:
  `modernize-*, readability-*`

- Run **static analysis** on our code base to find subtle latent bugs

Fixes, fixes, fixes...

🔥 **Just a few examples:**

```
Error: delete called on non-final 'AppPathVar' that has virtual functions but
non-virtual destructor [-Werror,-Wdelete-non-virtual-dtor]

Error: 'MsiComboBoxTable::PreRowChange' hides overloaded virtual function
[-Werror,-Woverloaded-virtual]
 void PreRowChange(const IMsiRow & aRow, BitField aModifiedContext);

Error: variable 'it' is incremented both in the loop header and in the loop body
[-Werror,-Wfor-loop-analysis]
```

Fixes, fixes, fixes...

🔥 **Just a few examples:**

```
Error: FilePath.cpp:36:17: error: moving a temporary object prevents copy elision
[-Werror,-Wpessimizing-move]
  : GenericPath(move(UnboxHugePath(aPath)))

Error: moving a local object in a return statement prevents copy elision
[-Werror,-Wpessimizing-move]
  return move(replacedConnString);
```

Fixes, fixes, fixes...

🔥 **Just a few examples:**

```
Error: field 'mCommandContainer' will be initialized after field 'mRepackBuildType'
[-Werror,-Wreorder]

Error: PipeServer.cpp:42:39: error: missing field 'InternalHigh' initializer
[-Werror,-Wmissing-field-initializers]
```

Fixes, fixes, fixes...

```
StringProcessing.cpp:504:9: error: no viable conversion from 'const wchar_t [6]'
to 'Facet'
  Facet facet = DEFAULT_LOCALE;
        ^       ~~~~~~~~~~~~~~

StringProcessing.cpp:344:7: note: candidate constructor (the implicit copy
constructor) not viable: no known conversion from 'const wchar_t [6]' to
'const Facet &' for 1st argument
class Facet
      ^

StringProcessing.cpp:349:3: note: candidate constructor not viable: no known
conversion from 'const wchar_t [6]' to 'const std::wstring &' for 1st argument
  Facet(const wstring & facet)
  ^
```

🔥 **Frequent offender:** **Two user-defined conversions**

Timeline

- January 12, 2017 - started playing with Clang for Windows (LLVM 3.9.1)
- January 24 - first commit, started fixing the Clang errors/warnings

  (Note: we were already on **`MSVC /W4 /WX`**)
- February 3 - created a clang++ compilation **`.bat`** file (crude automation attempt)
- March 7 - upgraded the clang++ batch file to a **PowerShell** script (**`clang-build.ps1`**)
- March 13 - our PS script also gains the ability to run clang-tidy checks
- March - first experiments with **clang-tidy** on our source code (just some core libraries)

Timeline

- April 11 🎉 - able to compile our **whole** codebase with Clang 3.9.1 (*some default warnings disabled*)

  ~ **3 months** since we started

- April 12 - created a **Jenkins** job for Clang build (every SCM change is compiled with Clang)
- May - great improvements to our PowerShell script:

  PCH, parallel compilation, project filters, SDK versions, etc.

- June - more experiments with **clang-tidy** on our source code (better coverage)
- June 16 - upgraded from VS2015 to **VS2017** (we also needed to update our Clang PS script)

Timeline

- July 3 - started work on a custom clang-based refactoring tool (`libTooling`)
- July 10 - fixed new Clang 4 issues and upgraded to **4.0.1**
- July - started to tackle Clang `-Wall` warnings in our code
- August - made extensive code transformations with our custom `libTooling` helpers
- August 24 🎉 - our whole codebase compiles with Clang `-Wall`
- August - started work on our "**Clang Power Tools**" extension for Visual Studio
- August 25 - first refactorings with **clang-tidy**:

  `modernize-use-nullptr, modernize-loop-convert`
- Aug-Sep - multiple code transformations with **clang-tidy**:

  `modernize-*, readability-*, misc-*,...`

Timeline

- September - started to fix **`-Wextra`** warnings (*in progress…*)
- September 11 - upgraded to LLVM **5.0** (fixed new warnings) **`[-Wunused-lambda-capture]`**
- September 11 - **open-sourced** our “***Clang Power Tools***” project
- September 26 - published our “Clang Power Tools” extension to **Visual Studio Marketplace**
- September 27 - introduced the project to the C++ community at **CppCon**
- **November 11** - here we are 😀

clang-tidy

Large scale refactorings we performed:

- `modernize-use-nullptr`
- `modernize-loop-convert`
- `modernize-use-override`
- `readability-redundant-string-cstr`
- `modernize-use-emplace`
- `modernize-use-auto`
- `modernize-make-shared & modernize-make-unique`
- `modernize-use-equals-default & modernize-use-equals-delete`

clang-tidy

Large scale refactorings we performed:

- `modernize-use-default-member-init`
- `readability-redundant-member-init`
- `modernize-pass-by-value`
- `modernize-return-braced-init-list`
- `modernize-use-using`
- `cppcoreguidelines-pro-type-member-init`
- `readability-redundant-string-init & misc-string-constructor`
- `misc-suspicious-string-compare & misc-string-compare`
- `misc-inefficient-algorithm`
- `cppcoreguidelines-*`

clang-tidy

🔥 **Issues we found:**

```
[readability-redundant-string-cstr]

// mChRequest is a 1KB buffer, we don't want to send it whole.
// So copy it as a C string, until we reach a null char.
ret += mChRequest.c_str();
```

clang-tidy

🔥 **Issues we found:**

```
[modernize-make-shared, modernize-make-unique]

-   requestData.reset(new BYTE[reqLength]);

+   requestData = std::make_unique<BYTE>();
```

clang-tidy

🔥 **Issues we found:**

```
[modernize-use-auto]

=> error: unused typedef 'BrowseIterator' [-Werror,-Wunused-local-typedef]

   typedef vector<BrowseSQLServerInfo>::iterator BrowseIterator;
```

clang-tidy

🔥 **Issues we found:**

```
[modernize-loop-convert]

=> unused values (orphan) [-Werror,-Wunused-value]

   vector<ModuleInfo>::iterator first = Modules_.begin();
   vector<ModuleInfo>::iterator last  = Modules_.end();
or:
   size_t count = Data_.size();

   for (auto & module : Modules_)
   {
     ...
   }
```

clang-tidy

🔥 **Issues we found:**

```
[modernize-use-using]  => errors & incomplete

- typedef int (WINAPI * InitExtractionFcn)(ExtractInfo *);
+ using InitExtractionFcn =
          int (*)(ExtractInfo *) __attribute__((stdcall)) (ExtractInfo *);

=> using InitExtractionFcn = int (WINAPI *)(ExtractInfo *);
```

🔥 Issues we found:

```
[modernize-use-using]  => errors & incomplete

template<typename KeyType>
class Row
{
  - typedef KeyType  KeyT;      <= substitutes concrete key type (template argument)
  + using KeyT = basic_string<wchar_t, char_traits<wchar_t>, allocator<wchar_t> >;
...
  KeyType mID;
};

// purpose of type alias being to access that template type from a derived class:
typename Row::KeyT

Concrete type used in code: Row<wstring>
```

How Did We Achieve All That ?

TOOLS

💪 Power Team 🤓

`PowerShell scripts`	Gabriel Diaconița
`Clang Power Tools VS Extension`	Ionuț Enache Alexandru Dragomir
`LibTooling`	Mihai Udrea
`Fixing Clang errors/warnings in our code`	Myself & many others...

We started simple...

compile.bat

```
SET INCLUDE="..\..;C:\Program Files (x86)\Microsoft Visual Studio
14.0\VC\include;C:\Program Files (x86)\Microsoft Visual Studio
14.0\VC\atlmfc\include;C:\Program Files (x86)\Windows
Kits\10\Include\10.0.10240.0\ucrt;C:\Program Files (x86)\Windows
Kits\8.1\Include\um;C:\Program Files (x86)\Windows Kits\8.1\Include\shared;"

setlocal EnableDelayedExpansion

For /R . %%G IN (*.cpp) do (
clang++ "%%G" -std=c++14 -fsyntax-only -Werror -Wmicrosoft
-Wno-invalid-token-paste -Wno-unused-variable -Wno-unused-value -fms-extensions
-fdelayed-template-parsing -fms-compatibility -D_ATL_NO_HOSTING
-DUNICODE -D_UNICODE -DWIN32 -D_DEBUG -DDEBUG

IF !errorlevel! NEQ 0 goto exit
)
```

We started simple...

tidy.bat

```
SET INCLUDE="..\..;C:\Program Files (x86)\Microsoft Visual Studio
14.0\VC\include;C:\Program Files (x86)\Microsoft Visual Studio
14.0\VC\atlmfc\include;C:\Program Files (x86)\Windows
Kits\10\Include\10.0.10240.0\ucrt;C:\Program Files (x86)\Windows
Kits\8.1\Include\um;C:\Program Files (x86)\Windows Kits\8.1\Include\shared;"

clang-tidy %1 -checks=-*,modernize-* -fix -- -std=c++14 -Werror
-Wno-invalid-token-paste -Wmicrosoft -fms-extensions -fdelayed-template-parsing
-fms-compatibility -D_ATL_NO_HOSTING -DUNICODE -D_UNICODE
-DWIN32 -D_DEBUG -DDEBUG

clang-format -style=file -i %1
```

But soon came...

Clang PowerShell Script

Meeting C++

- way more complicated (over 1,500 lines)
- very configurable (many parameters)
- supports both clang compile and tidy workflows
- works directly on Visual Studio **.vcxproj** files (or MSBuild projects)

  ℹ️ **no** roundtrip transformation through Clang JSON compilation database)
- supports parallel compilation
- constructs Clang PCH from VS project <stdafx.h>
- automatically extracts all necessary settings from VS projects:

  👉 preprocessor definitions, platform toolset, SDK version, include directories, PCH, etc.

`clang-build.ps1`

Using The PowerShell Script

-dir	Source directory to process for VS project files
-proj	List of projects to compile
-proj-ignore	List of projects to ignore
-file	What cpp(s) to compile from the found projects
-file-ignore	List of files to ignore
-parallel	Run clang++ in parallel mode, on all logical CPU cores
-continue	Continue project compilation even when errors occur
-clang-flags	Flags passed to clang++ driver
-tidy	Run specified clang-tidy checks
-tidy-fix	Run specified clang-tidy checks with auto-fix
...	

clang-build.ps1

Using The PowerShell Script

🤓 You can run `clang-build.ps1` directly, by specifying all required parameters (low-level control over details)

or

💡 You can use a `bootstrapper` PS script (eg. `sample-clang-build.ps1`), that pre-loads some of the constant configurations specific for your team/project.

```
sample-clang-build.ps1 ==> clang-build.ps1
```

Using The PowerShell Script

```
PS> .\sample-clang-build.ps1 -parallel
```

➡ Runs clang **compile** on all projects in current directory

```
PS> .\sample-clang-build.ps1 -parallel -proj-ignore foo,bar
```

➡ Runs clang **compile** on all projects in current directory, except 'foo' and 'bar'

```
PS> .\sample-clang-build.ps1 -proj foo,bar -file-ignore meow -tidy-fix "-*,modernize-*"
```

➡ Runs **clang-tidy**, using all *modernize* checks, on all CPPs not containing 'meow' in their name, from the projects 'foo' and 'bar'.

Bootstrapper PS script sample-clang-build.ps1

```
param( [alias("proj")]         [Parameter(Mandatory=$false)][string[]] $aVcxprojToCompile
     , [alias("proj-ignore")] [Parameter(Mandatory=$false)][string[]] $aVcxprojToIgnore
     , [alias("file")]        [Parameter(Mandatory=$false)][string]   $aCppToCompile
     , [alias("file-ignore")] [Parameter(Mandatory=$false)][string]   $aCppToIgnore
     , [alias("parallel")]    [Parameter(Mandatory=$false)][switch]   $aUseParallelCompile
     , [alias("tidy")]        [Parameter(Mandatory=$false)][string]   $aTidyFlags
     , [alias("tidy-fix")]    [Parameter(Mandatory=$false)][string]   $aTidyFixFlags
     )

Set-Variable -name kClangCompileFlags                                       -Option Constant `
                                             -value @( "-std=c++14"
                                                     , "-Wall"
                                                     , "-fms-compatibility-version=19.10"
                                                     , "-Wmicrosoft"
                                                     , "-Wno-invalid-token-paste"
                                                     , "-Wno-unknown-pragmas"
                                                     , "-Wno-unused-value"
                                                     )
Set-Variable -name kVisualStudioVersion -value "2017"                        -Option Constant
Set-Variable -name kVisualStudioSku     -value "Professional"                -Option Constant
```

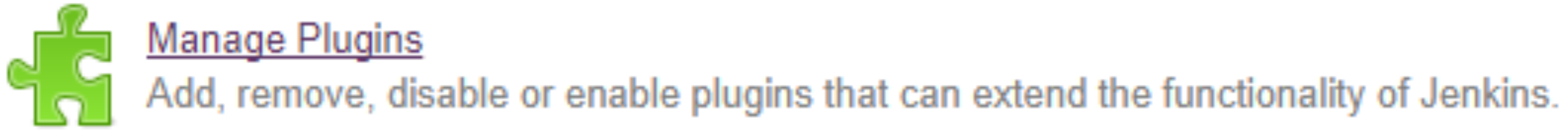
Using The PowerShell Script

+

Jenkins CI Configuration

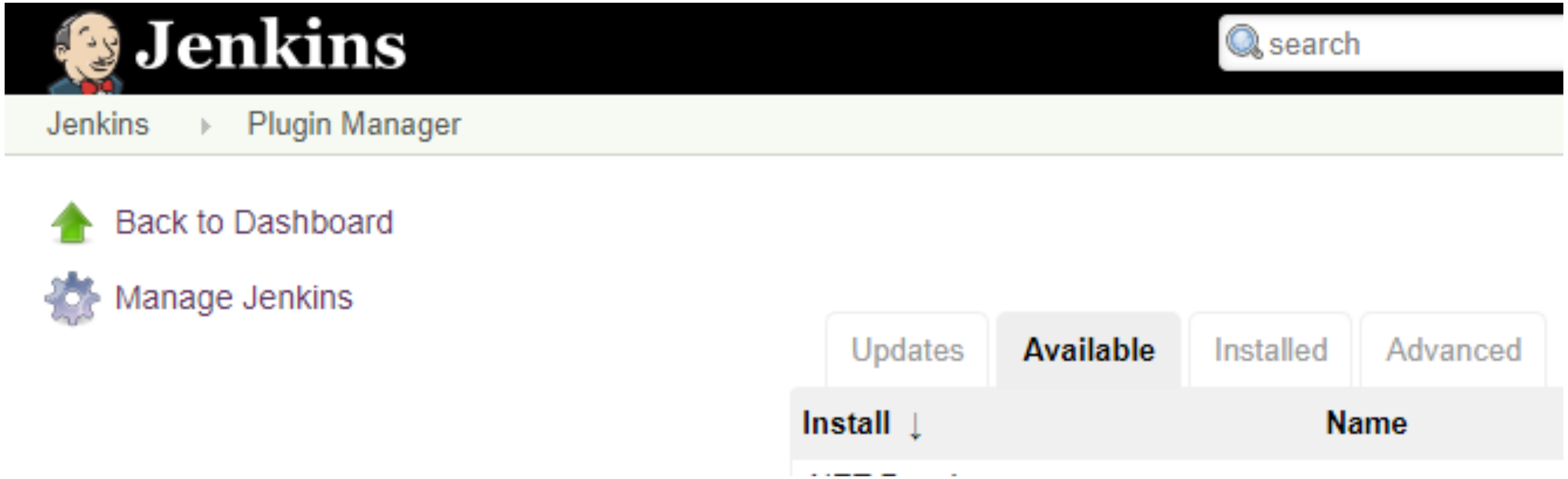
Jenkins CI Configuration

Install PowerShell plugin (available from Jenkins gallery)



Manage Plugins

Add, remove, disable or enable plugins that can extend the functionality of Jenkins.



Jenkins

search

Jenkins › Plugin Manager

Back to Dashboard

Manage Jenkins

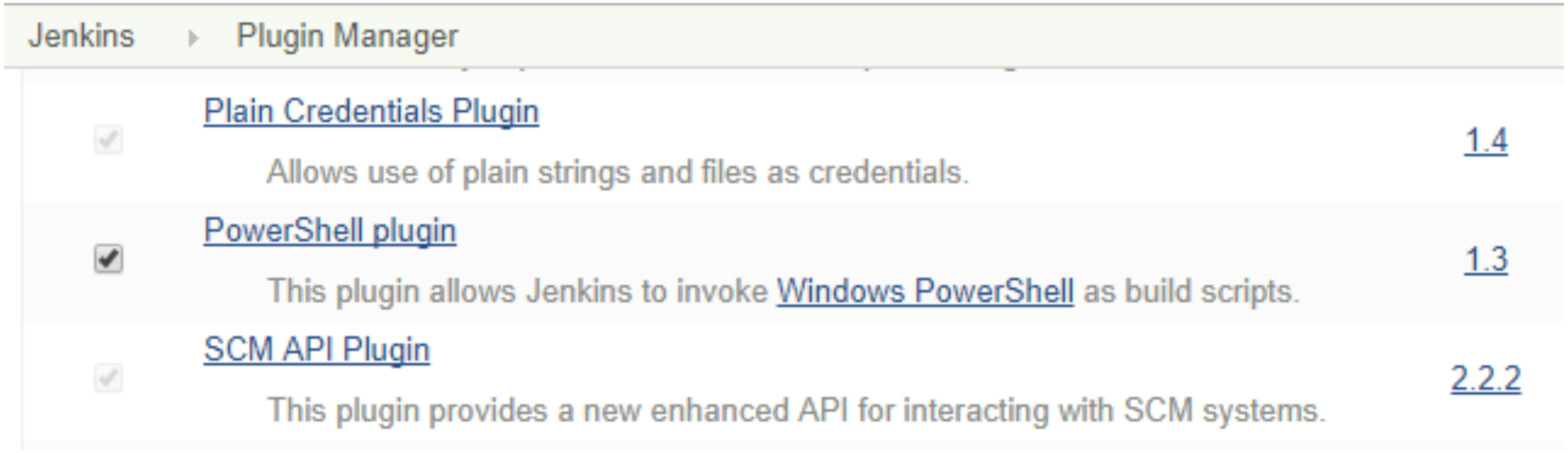
Updates | Available | Installed | Advanced

Install ↓	Name

https://wiki.jenkins.io/display/JENKINS/PowerShell+Plugin

Jenkins CI Configuration

Install PowerShell plugin



Jenkins ▸ Plugin Manager

✓	Plain Credentials Plugin Allows use of plain strings and files as credentials.	1.4
✓	PowerShell plugin This plugin allows Jenkins to invoke Windows PowerShell as build scripts.	1.3
✓	SCM API Plugin This plugin provides a new enhanced API for interacting with SCM systems.	2.2.2

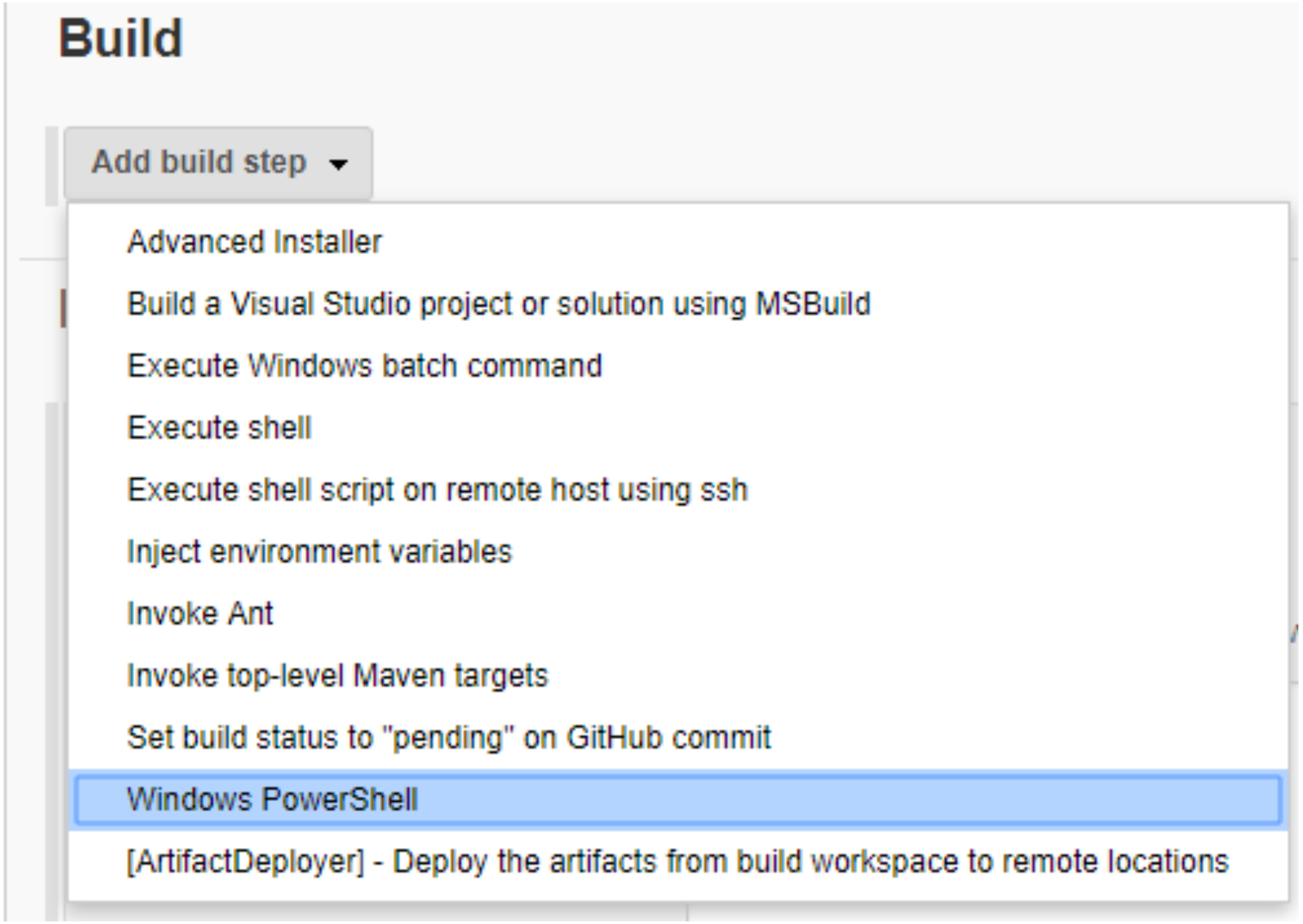
https://wiki.jenkins.io/display/JENKINS/PowerShell+Plugin

Jenkins CI Configuration

- Create a **new job** just for clang builds

or

- Attach a **new build step** on an existing job



Build

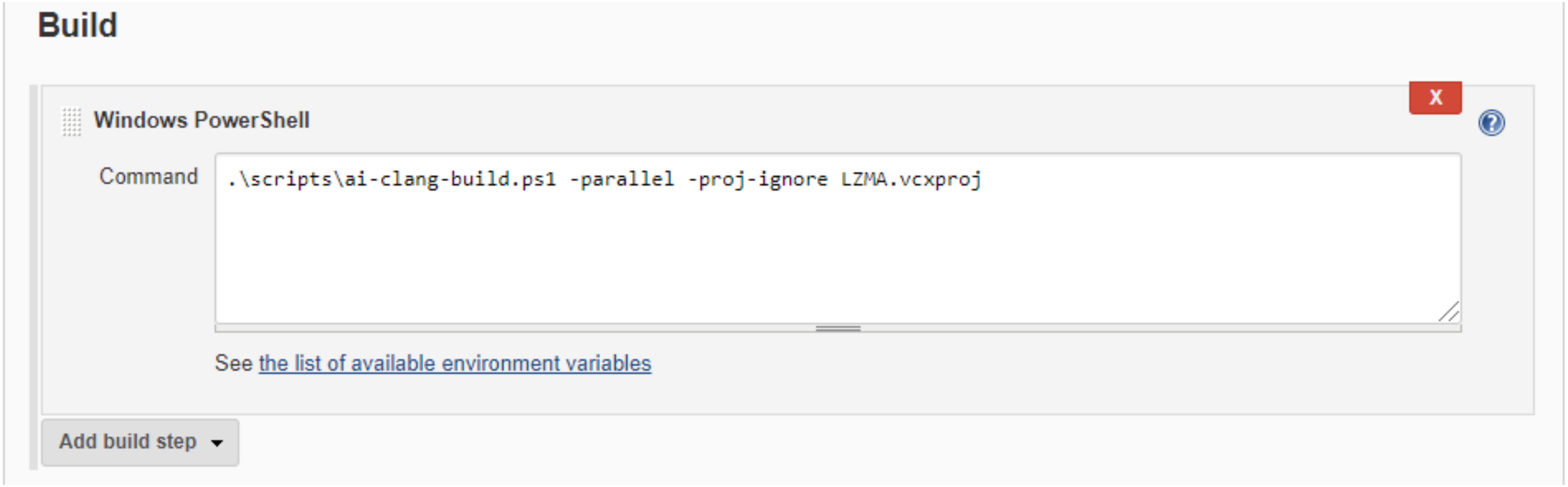
Add build step

Advanced Installer
Build a Visual Studio project or solution using MSBuild
Execute Windows batch command
Execute shell
Execute shell script on remote host using ssh
Inject environment variables
Invoke Ant
Invoke top-level Maven targets
Set build status to "pending" on GitHub commit
Windows PowerShell
[ArtifactDeployer] - Deploy the artifacts from build workspace to remote locations

Jenkins CI Configuration

Reference PowerShell script from the job working directory.

Both the bootstrapper PS script (eg. **`ai-clang-build.ps1`**) and the main PS script (**`clang-build.ps1`**) should be in the same directory.



Build

Windows PowerShell

Command

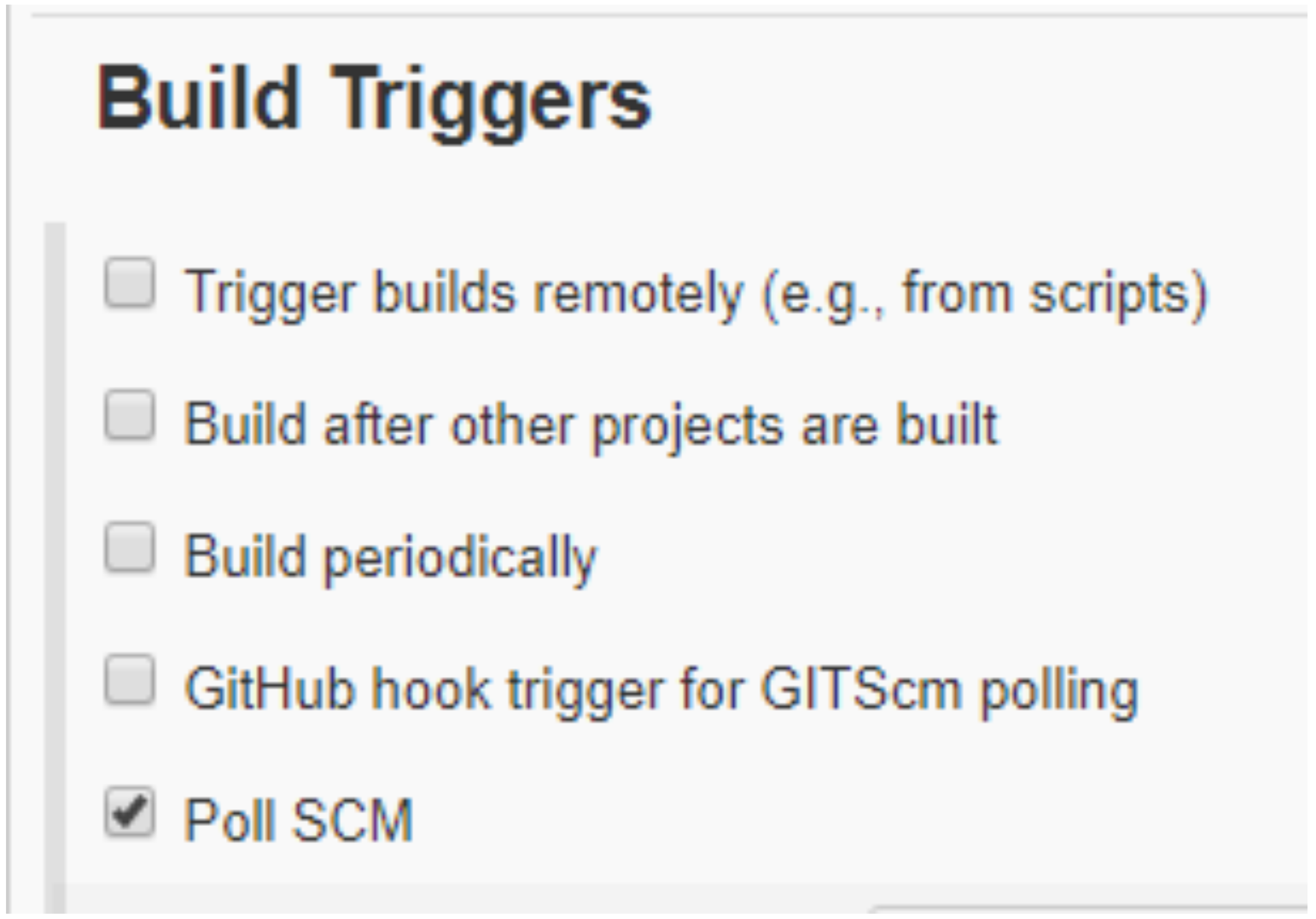
```
.\scripts\ai-clang-build.ps1 -parallel -proj-ignore LZMA.vcxproj
```

See the list of available environment variables

Meeting C++

Jenkins CI Configuration

💡 If you configured Clang build as a new Jenkins job, a good workflow is to track and build any SCM changes:

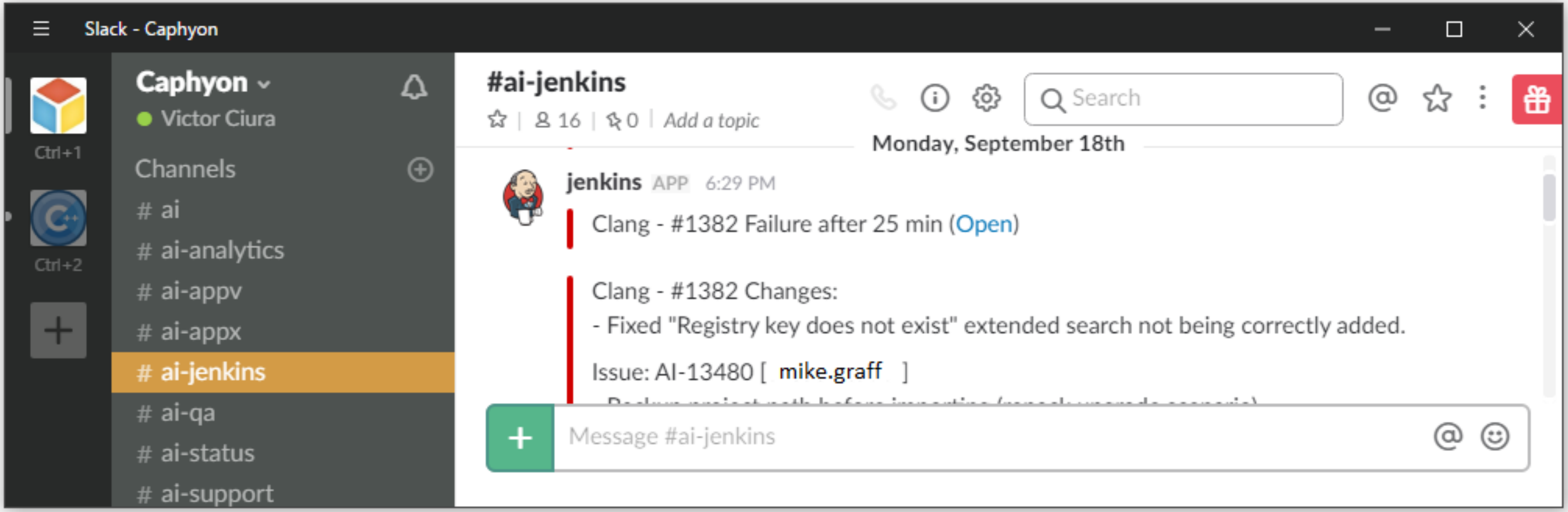


Build Triggers

- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build after other projects are built
- [ ] Build periodically
- [ ] GitHub hook trigger for GITScm polling
- [✓] Poll SCM

Jenkins CI Workflow

🔥 **When Clang build is broken...**



Monday, September 18th

jenkins APP 6:29 PM

Clang - #1382 Failure after 25 min (Open)

Clang - #1382 Changes:
- Fixed "Registry key does not exist" extended search not being correctly added.

Issue: AI-13480 [mike.graff]

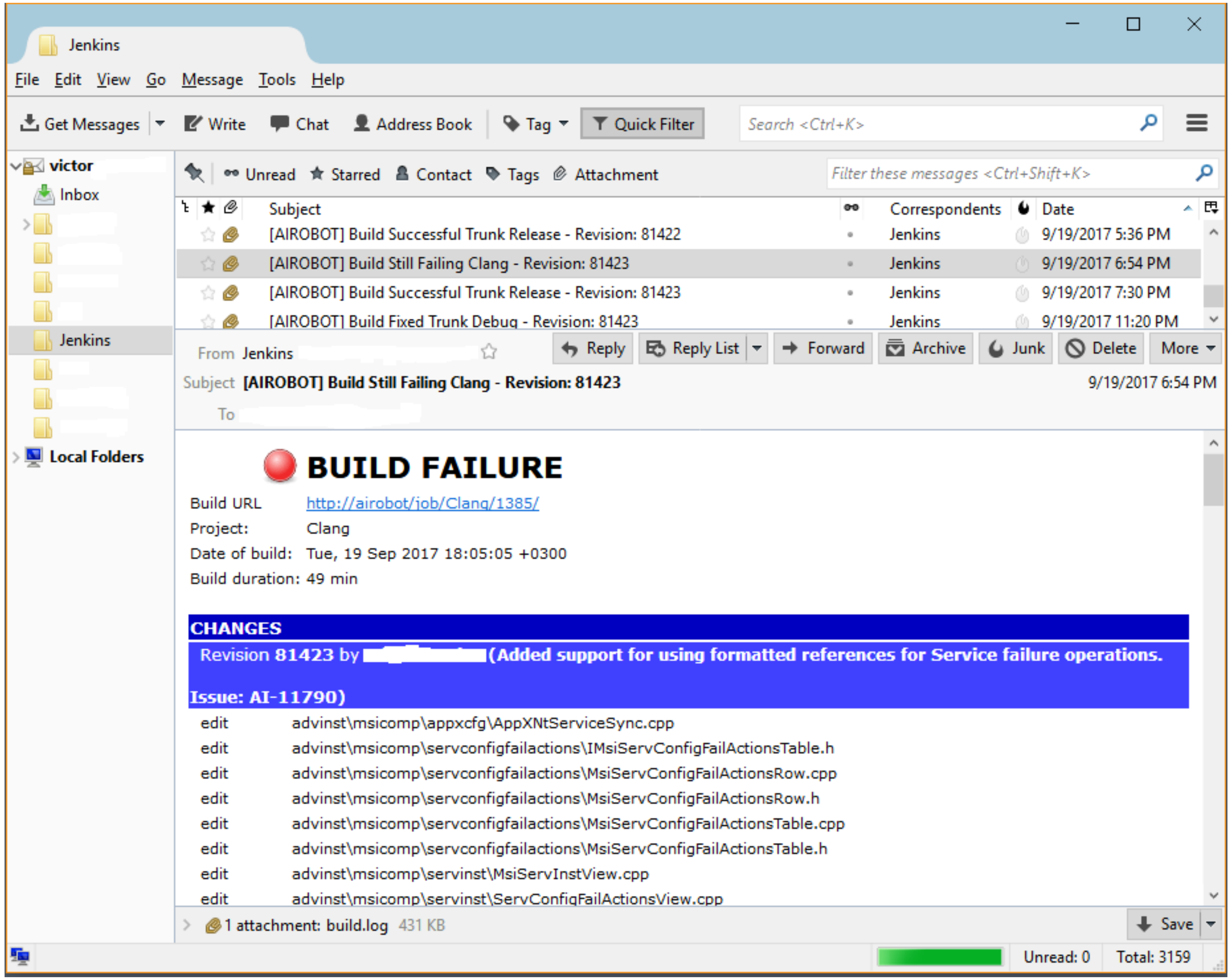
Slack bot alert ➡ #ai-jenkins

Meeting C++

Jenkins CI Workflow

🔥 When Clang build is broken...

Team devs email alert ➡



Subject	Correspondents	Date
[AIROBOT] Build Successful Trunk Release - Revision: 81422	Jenkins	9/19/2017 5:36 PM
[AIROBOT] Build Still Failing Clang - Revision: 81423	Jenkins	9/19/2017 6:54 PM
[AIROBOT] Build Successful Trunk Release - Revision: 81423	Jenkins	9/19/2017 7:30 PM
[AIROBOT] Build Fixed Trunk Debug - Revision: 81423	Jenkins	9/19/2017 11:20 PM

From Jenkins [redacted]
Subject [AIROBOT] Build Still Failing Clang - Revision: 81423 9/19/2017 6:54 PM
To [redacted]

BUILD FAILURE

Build URL http://airobot/job/Clang/1385/
Project: Clang
Date of build: Tue, 19 Sep 2017 18:05:05 +0300
Build duration: 49 min

CHANGES

Revision 81423 by [redacted] (Added support for using formatted references for Service failure operations.

Issue: AI-11790)

```
edit    advinst\msicomp\appxcfg\AppXNtServiceSync.cpp
edit    advinst\msicomp\servconfigfailactions\IMsiServConfigFailActionsTable.h
edit    advinst\msicomp\servconfigfailactions\MsiServConfigFailActionsRow.cpp
edit    advinst\msicomp\servconfigfailactions\MsiServConfigFailActionsRow.h
edit    advinst\msicomp\servconfigfailactions\MsiServConfigFailActionsTable.cpp
edit    advinst\msicomp\servconfigfailactions\MsiServConfigFailActionsTable.h
edit    advinst\msicomp\servinst\MsiServInstView.cpp
edit    advinst\msicomp\servinst\ServConfigFailActionsView.cpp
```

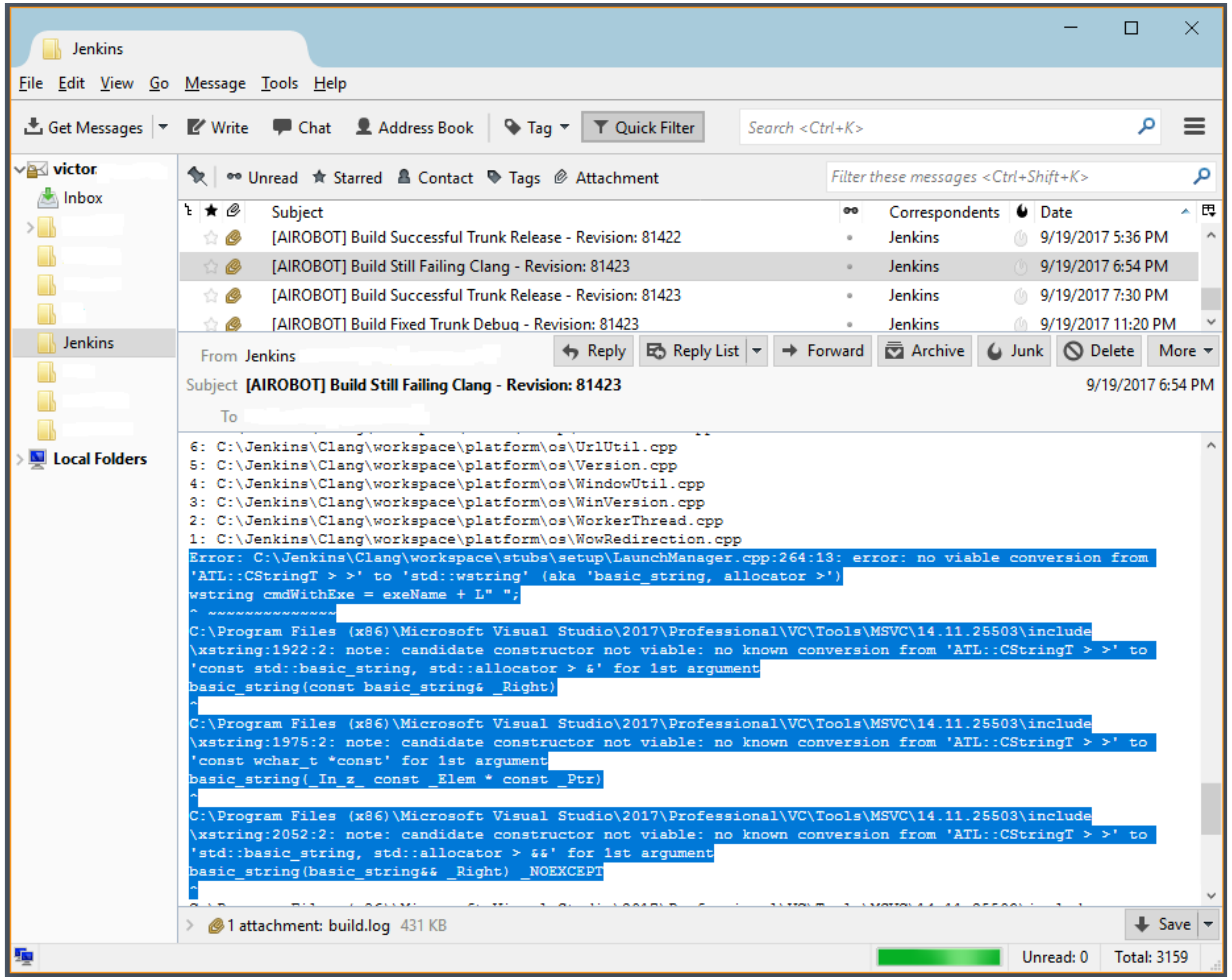
1 attachment: build.log 431 KB

Unread: 0 Total: 3159

Jenkins CI Workflow

🔥 **When Clang build is broken...**

Team devs email alert ➡



Subject	Correspondents	Date
[AIROBOT] Build Successful Trunk Release - Revision: 81422	Jenkins	9/19/2017 5:36 PM
[AIROBOT] Build Still Failing Clang - Revision: 81423	Jenkins	9/19/2017 6:54 PM
[AIROBOT] Build Successful Trunk Release - Revision: 81423	Jenkins	9/19/2017 7:30 PM
[AIROBOT] Build Fixed Trunk Debug - Revision: 81423	Jenkins	9/19/2017 11:20 PM

From Jenkins [redacted]
Subject **[AIROBOT] Build Still Failing Clang - Revision: 81423** 9/19/2017 6:54 PM
To [redacted]

```
6: C:\Jenkins\Clang\workspace\platform\os\UrlUtil.cpp
5: C:\Jenkins\Clang\workspace\platform\os\Version.cpp
4: C:\Jenkins\Clang\workspace\platform\os\WindowUtil.cpp
3: C:\Jenkins\Clang\workspace\platform\os\WinVersion.cpp
2: C:\Jenkins\Clang\workspace\platform\os\WorkerThread.cpp
1: C:\Jenkins\Clang\workspace\platform\os\WowRedirection.cpp
Error: C:\Jenkins\Clang\workspace\stubs\setup\LaunchManager.cpp:264:13: error: no viable conversion from
'ATL::CStringT > >' to 'std::wstring' (aka 'basic_string, allocator >')
wstring cmdWithExe = exeName + L" ";
^ ~~~~~~~~~~~~~~~
C:\Program Files (x86)\Microsoft Visual Studio\2017\Professional\VC\Tools\MSVC\14.11.25503\include
\xstring:1922:2: note: candidate constructor not viable: no known conversion from 'ATL::CStringT > >' to
'const std::basic_string, std::allocator > &' for 1st argument
basic_string(const basic_string& _Right)
^
C:\Program Files (x86)\Microsoft Visual Studio\2017\Professional\VC\Tools\MSVC\14.11.25503\include
\xstring:1975:2: note: candidate constructor not viable: no known conversion from 'ATL::CStringT > >' to
'const wchar_t *const' for 1st argument
basic_string(_In_z_ const _Elem * const _Ptr)
^
C:\Program Files (x86)\Microsoft Visual Studio\2017\Professional\VC\Tools\MSVC\14.11.25503\include
\xstring:2052:2: note: candidate constructor not viable: no known conversion from 'ATL::CStringT > >' to
'std::basic_string, std::allocator > &&' for 1st argument
basic_string(basic_string&& _Right) _NOEXCEPT
^
```

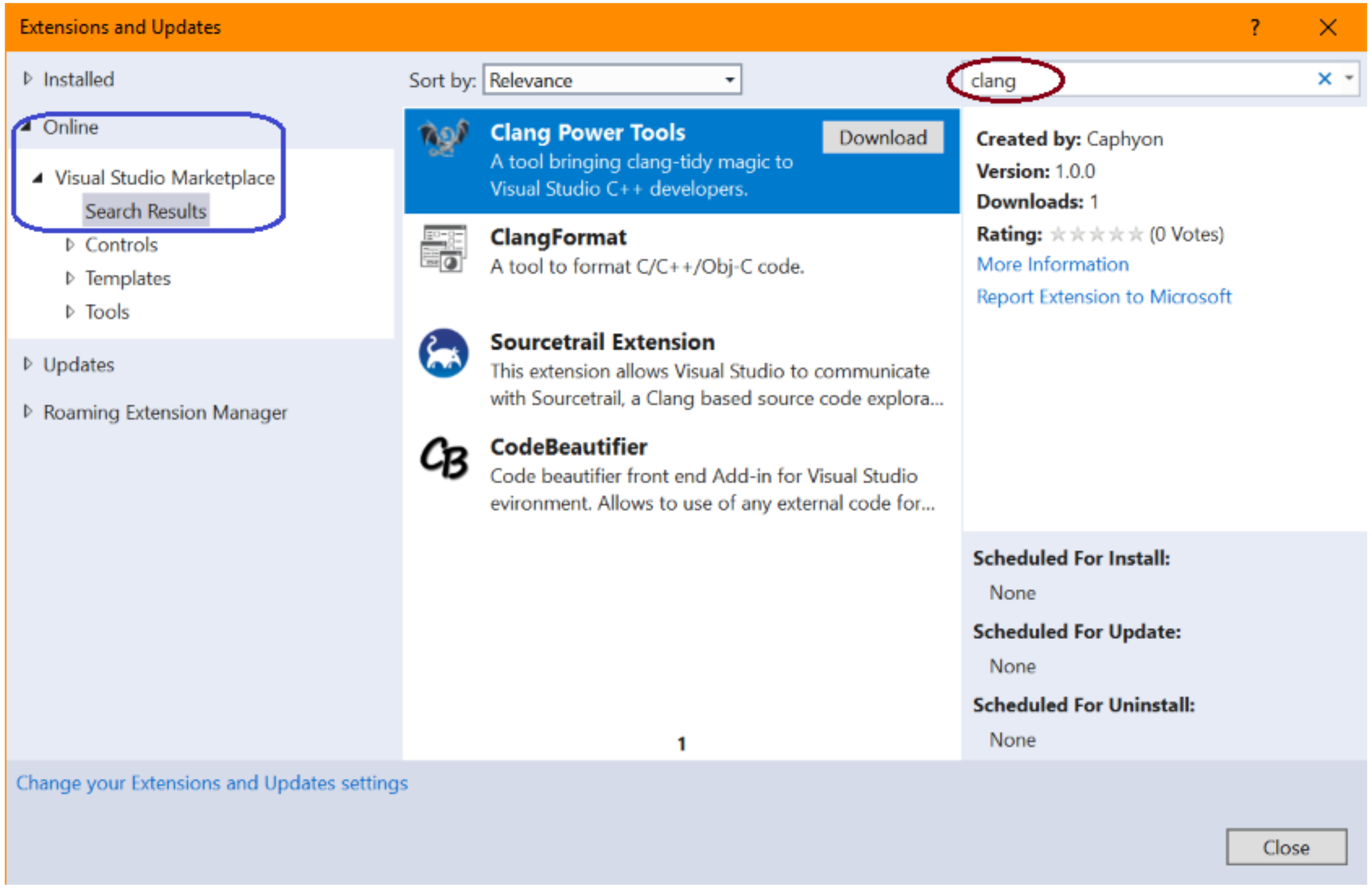
1 attachment: build.log 431 KB

Unread: 0 Total: 3159

What About Developer Workflow?

Install The "Clang Power Tools" Visual Studio Extension

[Tools]
⬇
Extensions and updates


Extensions and Updates

- Installed
- Online
  - Visual Studio Marketplace
    - Search Results
  - Controls
  - Templates
  - Tools
- Updates
- Roaming Extension Manager

Sort by: Relevance

clang

Clang Power Tools Download
A tool bringing clang-tidy magic to Visual Studio C++ developers.

ClangFormat
A tool to format C/C++/Obj-C code.

Sourcetrail Extension
This extension allows Visual Studio to communicate with Sourcetrail, a Clang based source code explora...

CodeBeautifier
Code beautifier front end Add-in for Visual Studio evironment. Allows to use of any external code for...

1

Created by: Caphyon
Version: 1.0.0
Downloads: 1
Rating: ☆☆☆☆☆ (0 Votes)
More Information
Report Extension to Microsoft

Scheduled For Install:
None
Scheduled For Update:
None
Scheduled For Uninstall:
None

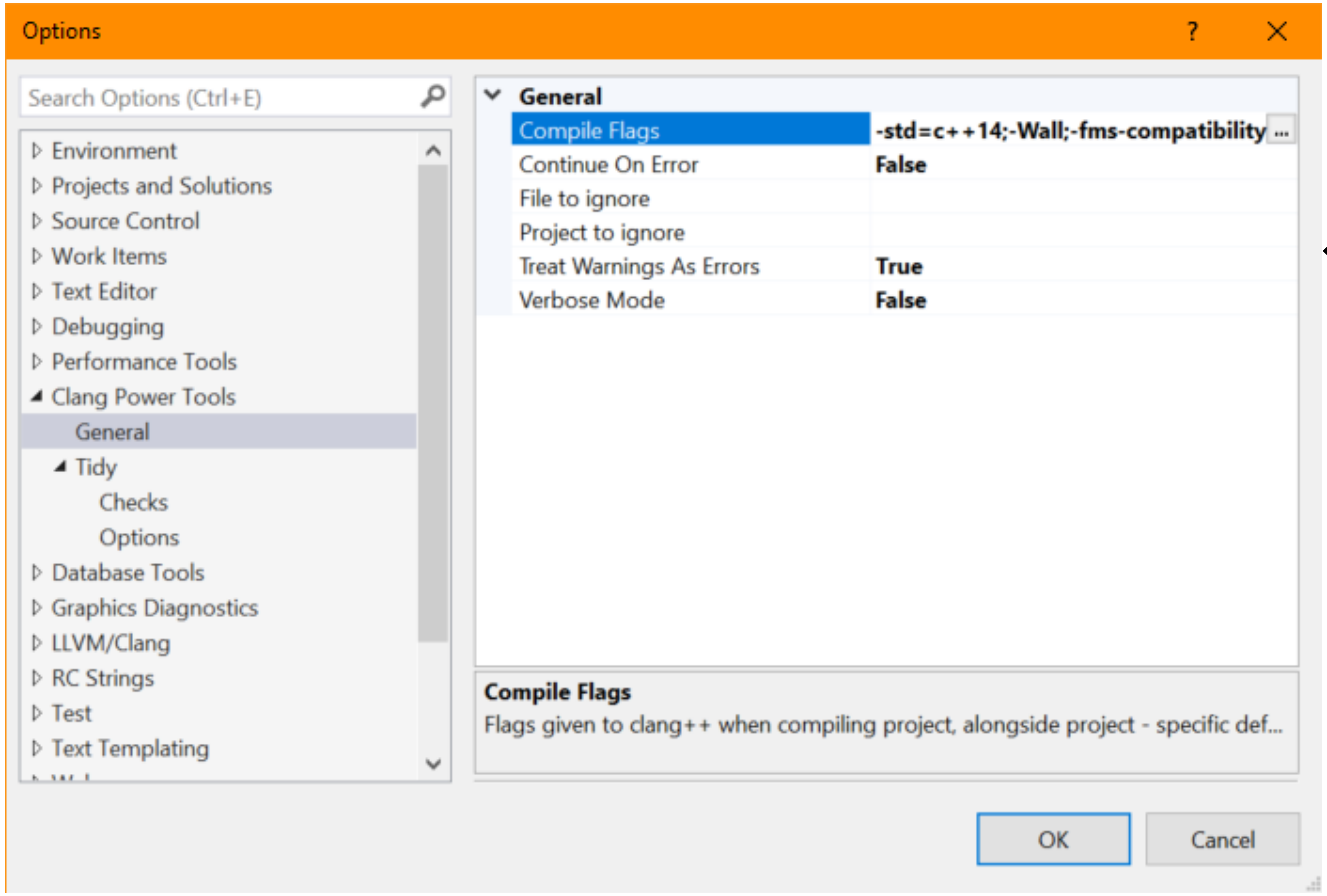
Change your Extensions and Updates settings

Close

Requires "Clang for Windows" (LLVM pre-built binary) to be installed. http://releases.llvm.org/5.0.0/LLVM-5.0.0-win64.exe

Configure The "Clang Power Tools" Visual Studio Extension

[Tools]
⬇
Options...
⬇
clang



General

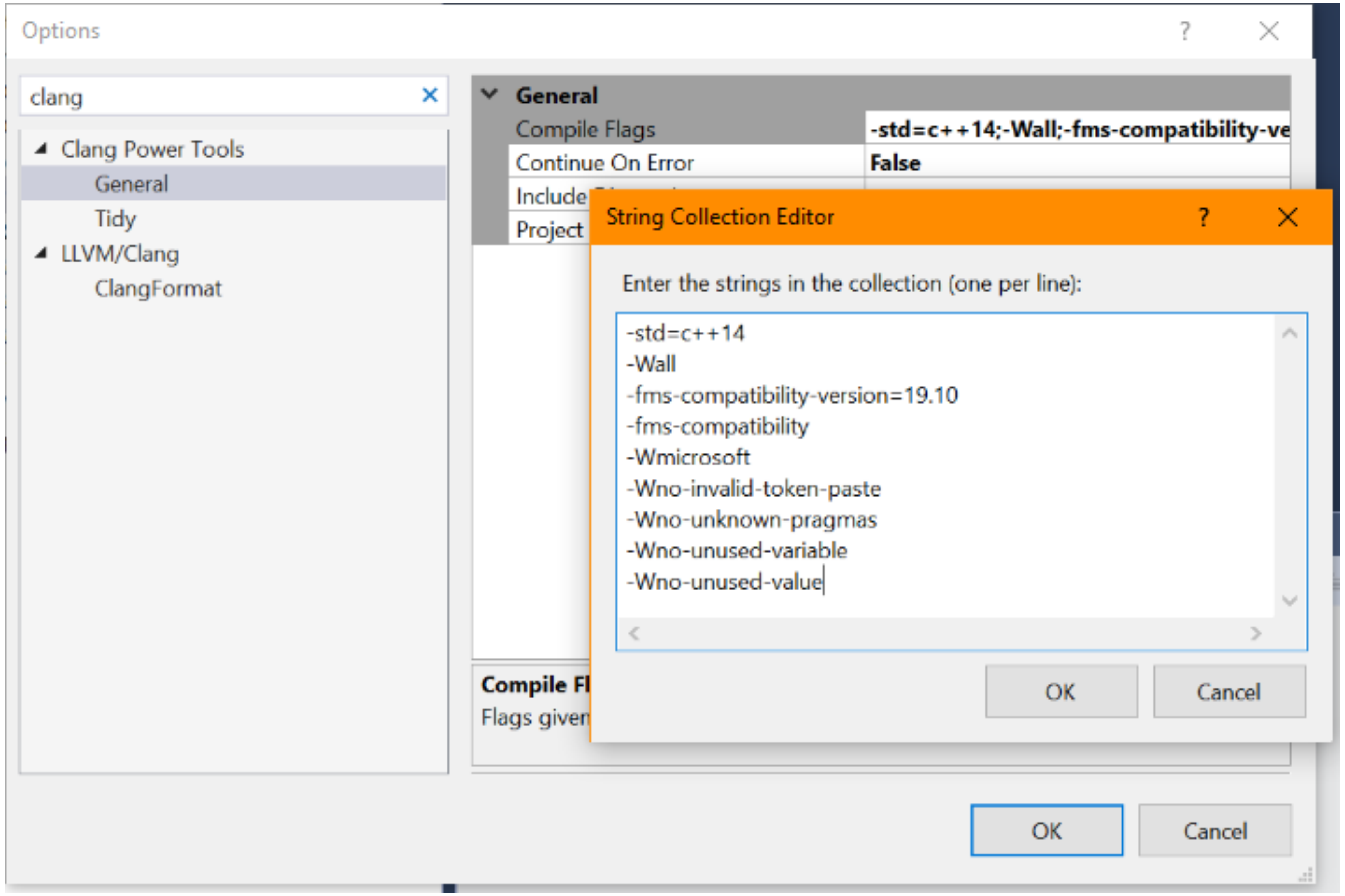
Compile Flags	-std=c++14;-Wall;-fms-compatibility
Continue On Error	False
File to ignore	
Project to ignore	
Treat Warnings As Errors	True
Verbose Mode	False

Compile Flags
Flags given to clang++ when compiling project, alongside project - specific def...

⬅ Compilation settings

Configure The "Clang Power Tools" Visual Studio Extension

[Tools]
⬇
Options...
⬇
clang



Compile Flags	-std=c++14;-Wall;-fms-compatibility-ve
Continue On Error	False

Enter the strings in the collection (one per line):

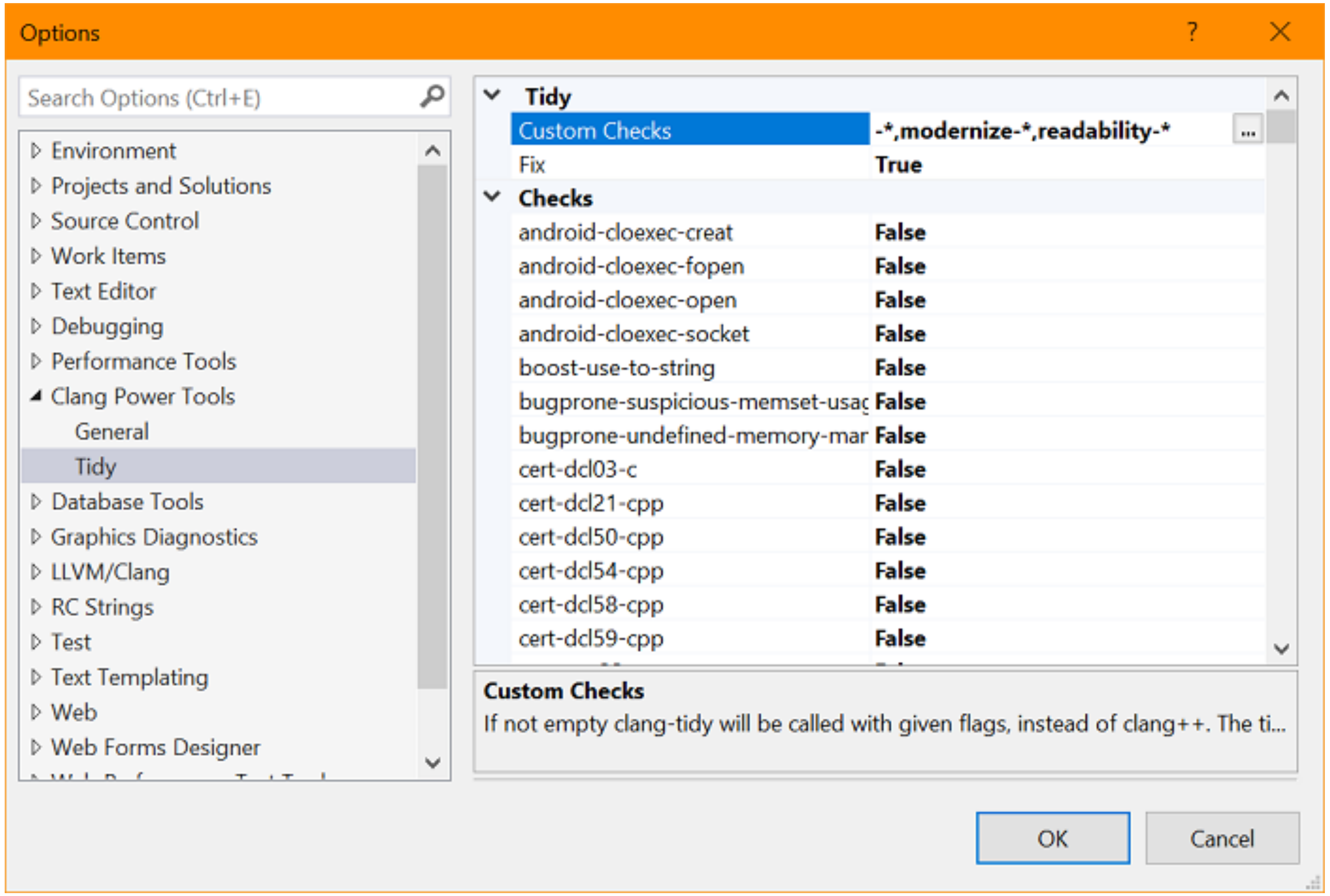
```
-std=c++14
-Wall
-fms-compatibility-version=19.10
-fms-compatibility
-Wmicrosoft
-Wno-invalid-token-paste
-Wno-unknown-pragmas
-Wno-unused-variable
-Wno-unused-value
```

⬅ clang++ flags

Configure The "Clang Power Tools" Visual Studio Extension

Meeting C++

[Tools]
⬇
Options...
⬇
clang



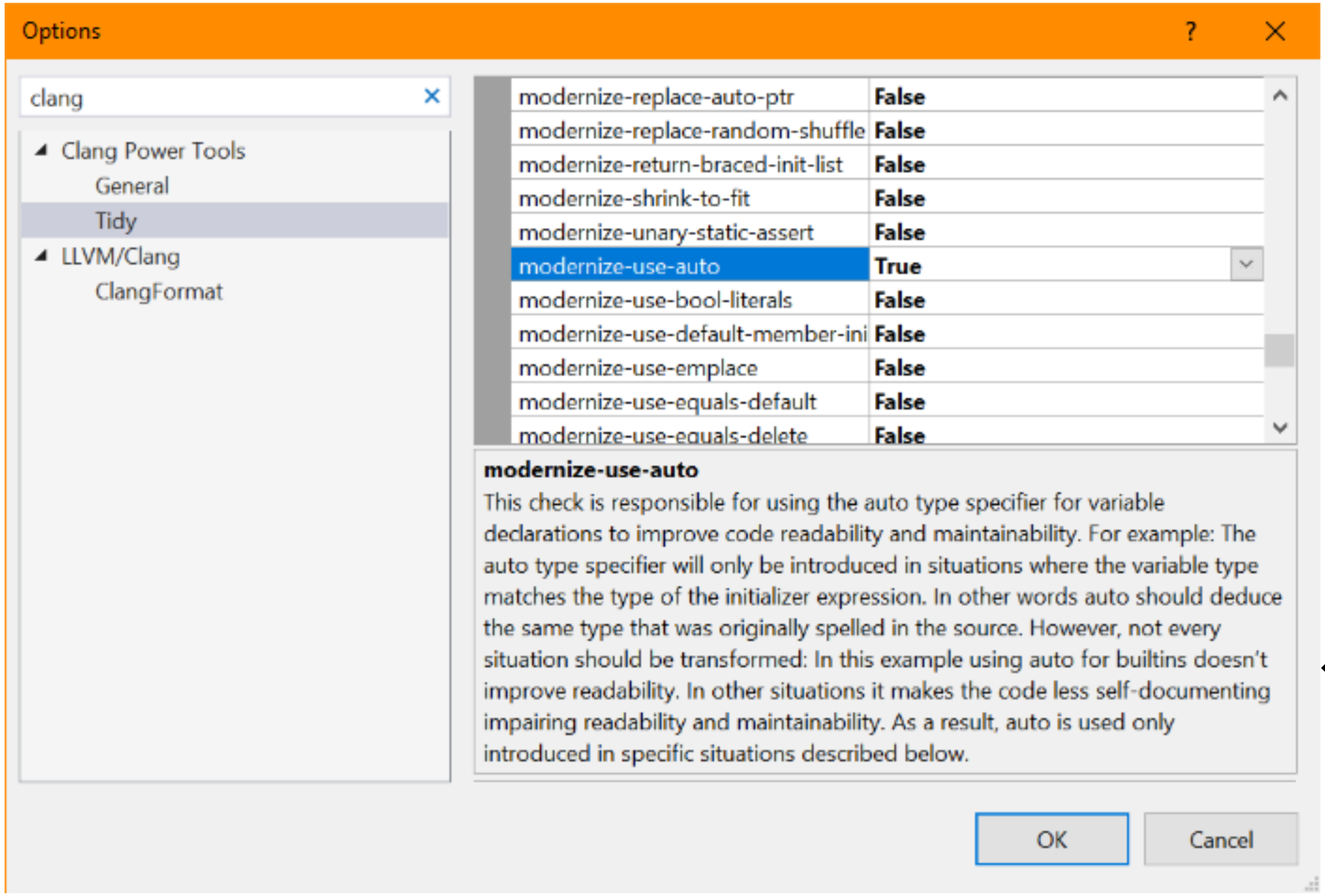
Tidy	
Custom Checks	-*,modernize-*,readability-*
Fix	True
Checks	
android-cloexec-creat	False
android-cloexec-fopen	False
android-cloexec-open	False
android-cloexec-socket	False
boost-use-to-string	False
bugprone-suspicious-memset-usaç	False
bugprone-undefined-memory-mar	False
cert-dcl03-c	False
cert-dcl21-cpp	False
cert-dcl50-cpp	False
cert-dcl54-cpp	False
cert-dcl58-cpp	False
cert-dcl59-cpp	False

Custom Checks
If not empty clang-tidy will be called with given flags, instead of clang++. The ti...

⬅ clang-tidy settings

Configure The "Clang Power Tools" Visual Studio Extension

[Tools]
⬇
Options...
⬇
clang



modernize-replace-auto-ptr	False
modernize-replace-random-shuffle	False
modernize-return-braced-init-list	False
modernize-shrink-to-fit	False
modernize-unary-static-assert	False
modernize-use-auto	True
modernize-use-bool-literals	False
modernize-use-default-member-ini	False
modernize-use-emplace	False
modernize-use-equals-default	False
modernize-use-equals-delete	False

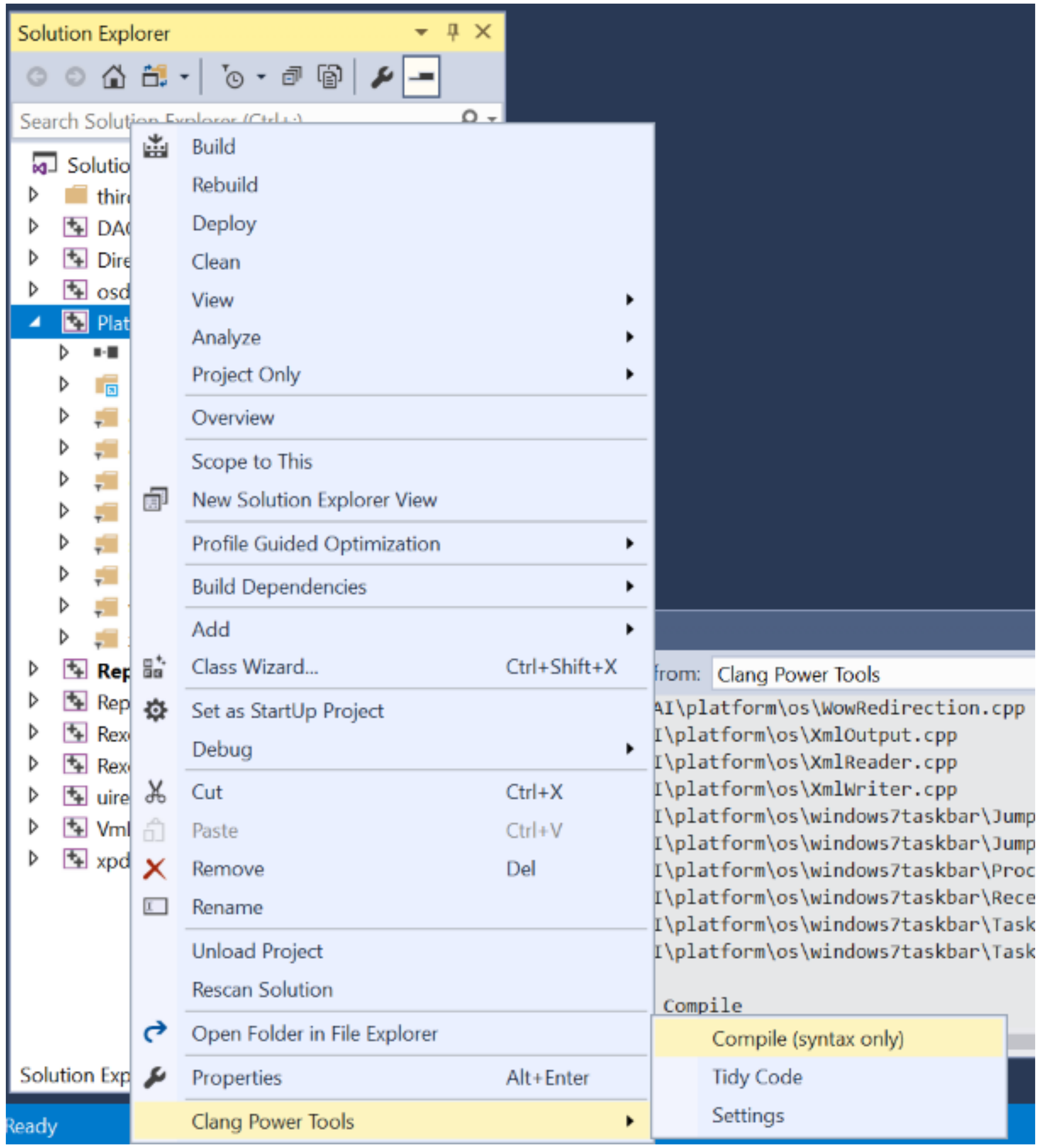
modernize-use-auto
This check is responsible for using the auto type specifier for variable declarations to improve code readability and maintainability. For example: The auto type specifier will only be introduced in situations where the variable type matches the type of the initializer expression. In other words auto should deduce the same type that was originally spelled in the source. However, not every situation should be transformed: In this example using auto for builtins doesn't improve readability. In other situations it makes the code less self-documenting impairing readability and maintainability. As a result, auto is used only introduced in specific situations described below.

⬅ clang-tidy checks

⬅ inline documentation

Using The "Clang Power Tools" Visual Studio Extension

Run Clang Power Tools on a whole project or solution ➡

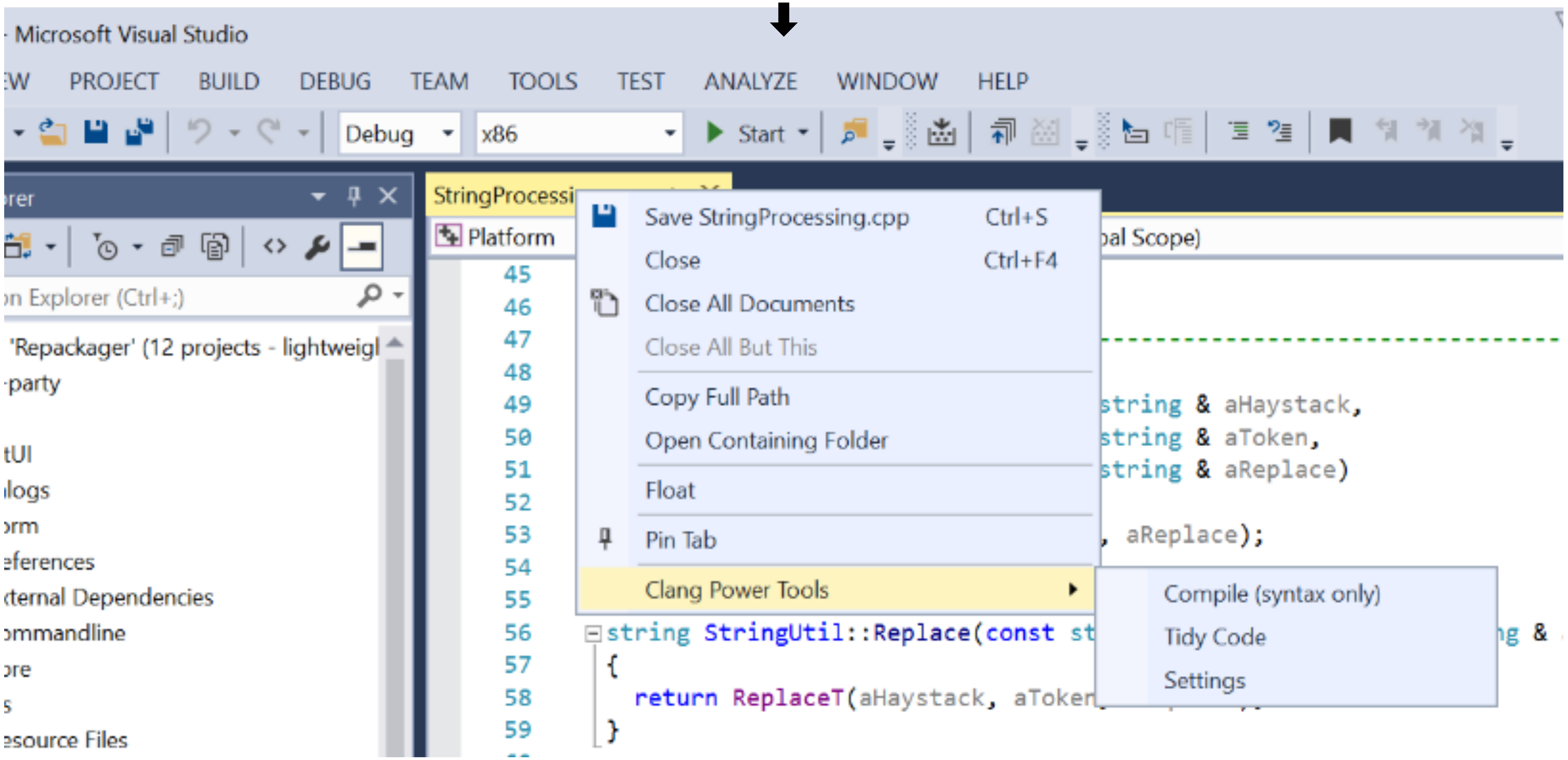


```
AI\platform\os\WowRedirection.cpp
I\platform\os\XmlOutput.cpp
I\platform\os\XmlReader.cpp
I\platform\os\XmlWriter.cpp
I\platform\os\windows7taskbar\Jump
I\platform\os\windows7taskbar\Jump
I\platform\os\windows7taskbar\Proc
I\platform\os\windows7taskbar\Rece
I\platform\os\windows7taskbar\Task
I\platform\os\windows7taskbar\Task

Compile
```

⬅ Compile or Tidy code

Using The "Clang Power Tools" Visual Studio Extension

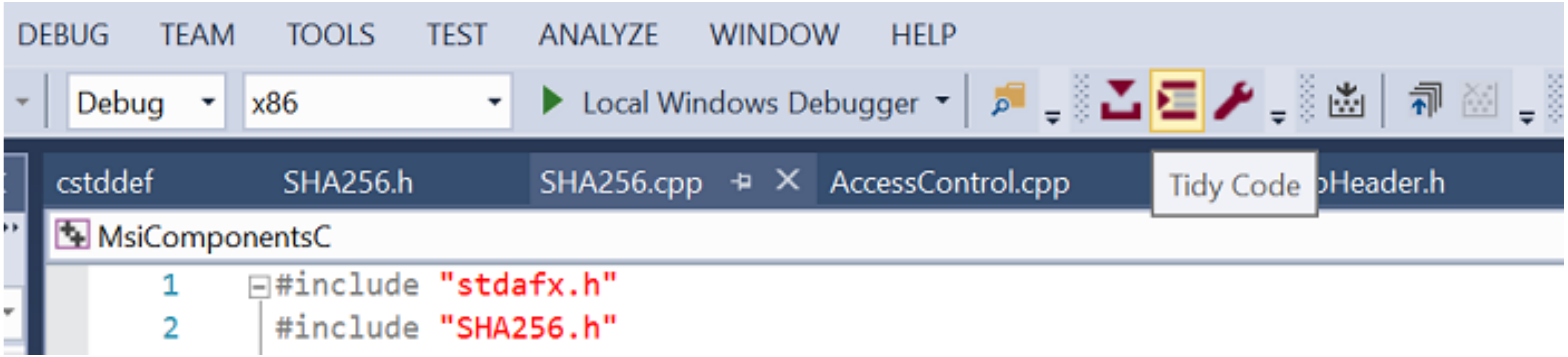
Run Clang Power Tools on an open source file



```
string & aHaystack,
string & aToken,
string & aReplace)

, aReplace);

string StringUtil::Replace(const st                ng &
{
  return ReplaceT(aHaystack, aToken
}
```

Using The "Clang Power Tools" Visual Studio Extension

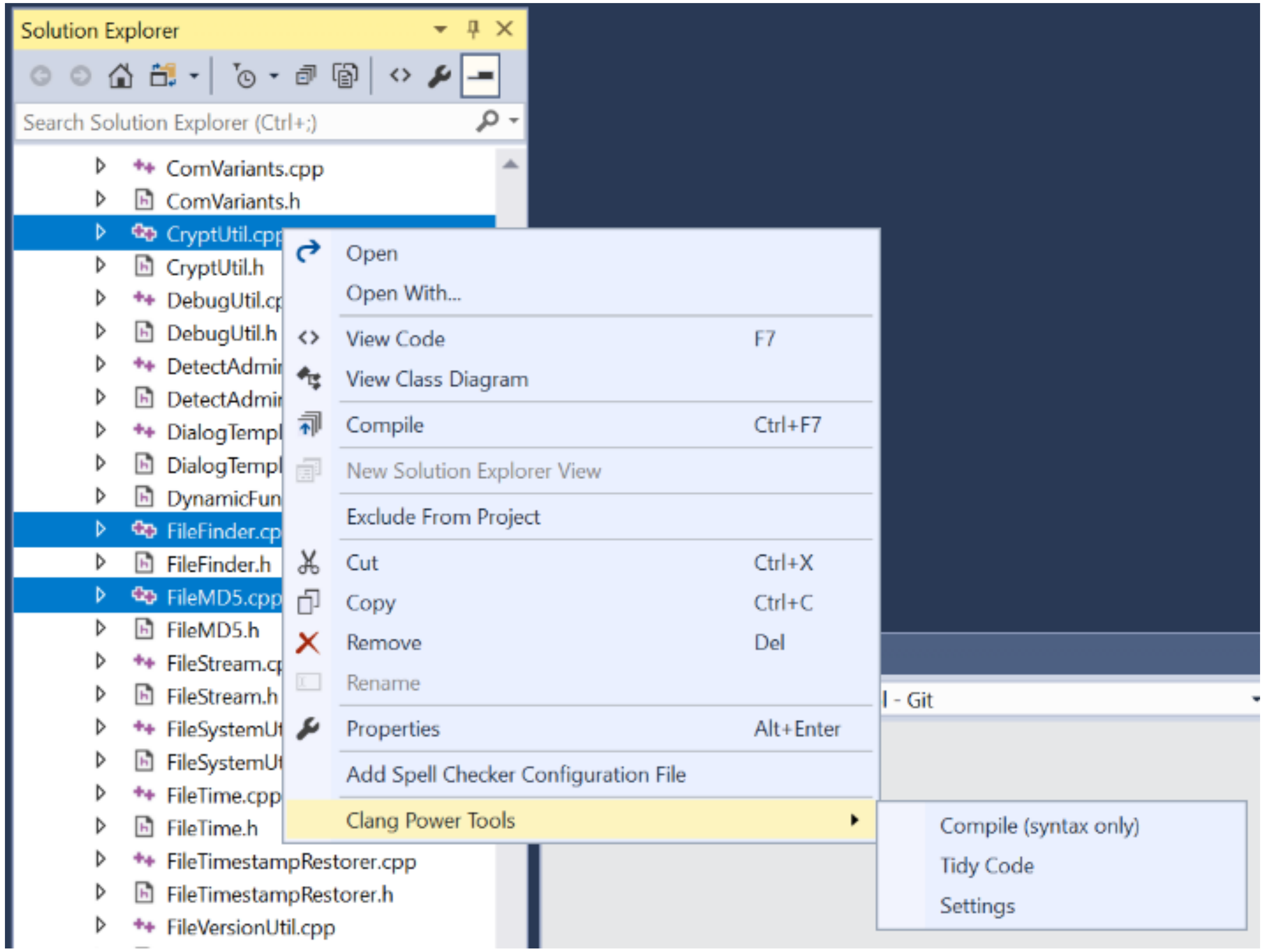
Handy Toolbar



```
#include "stdafx.h"
#include "SHA256.h"
```

Using The "Clang Power Tools" Visual Studio Extension

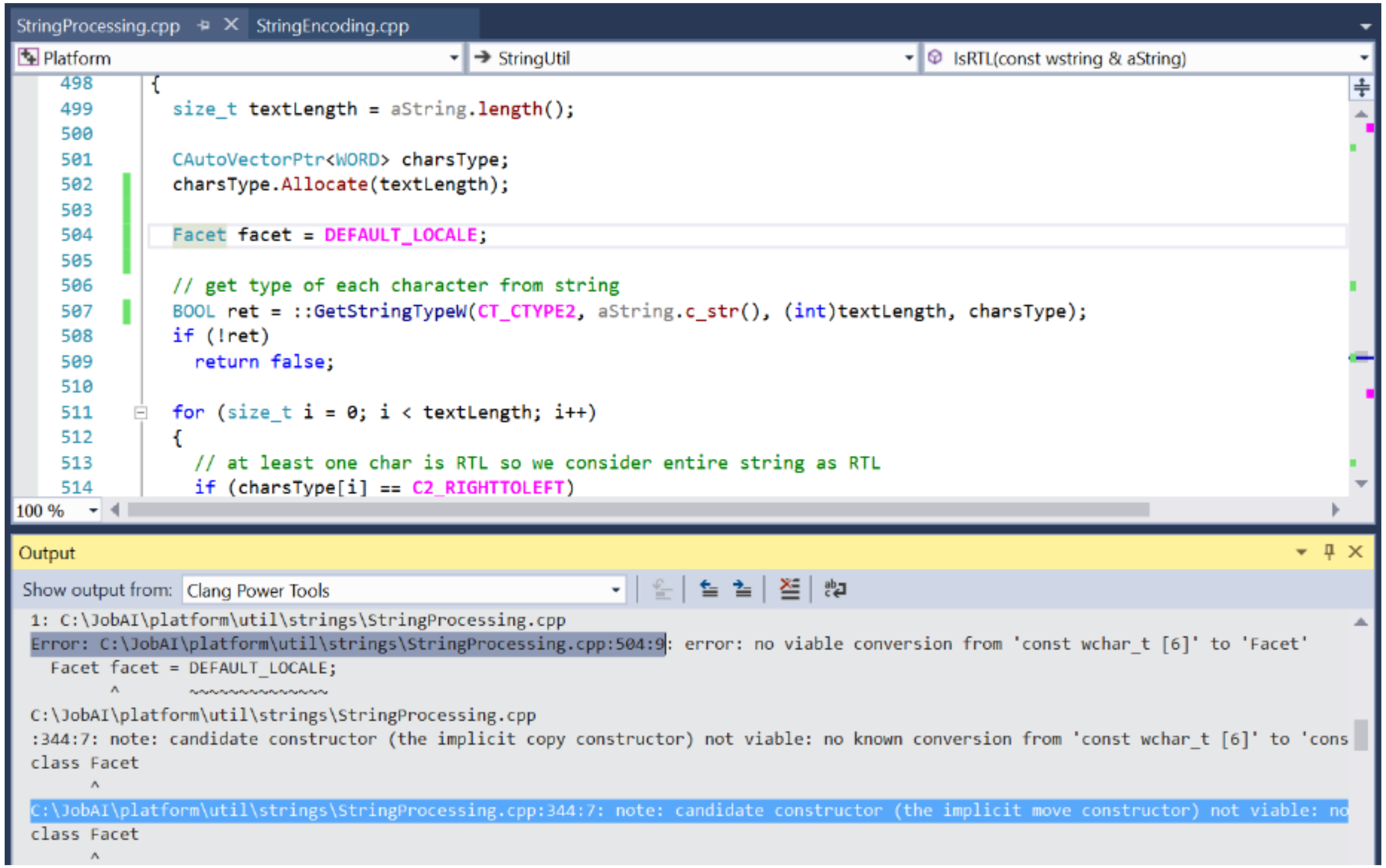
Run Clang Power Tools on selected files ➡



⬅ Compile or Tidy code

Meeting C++

Using The "Clang Power Tools" Visual Studio Extension

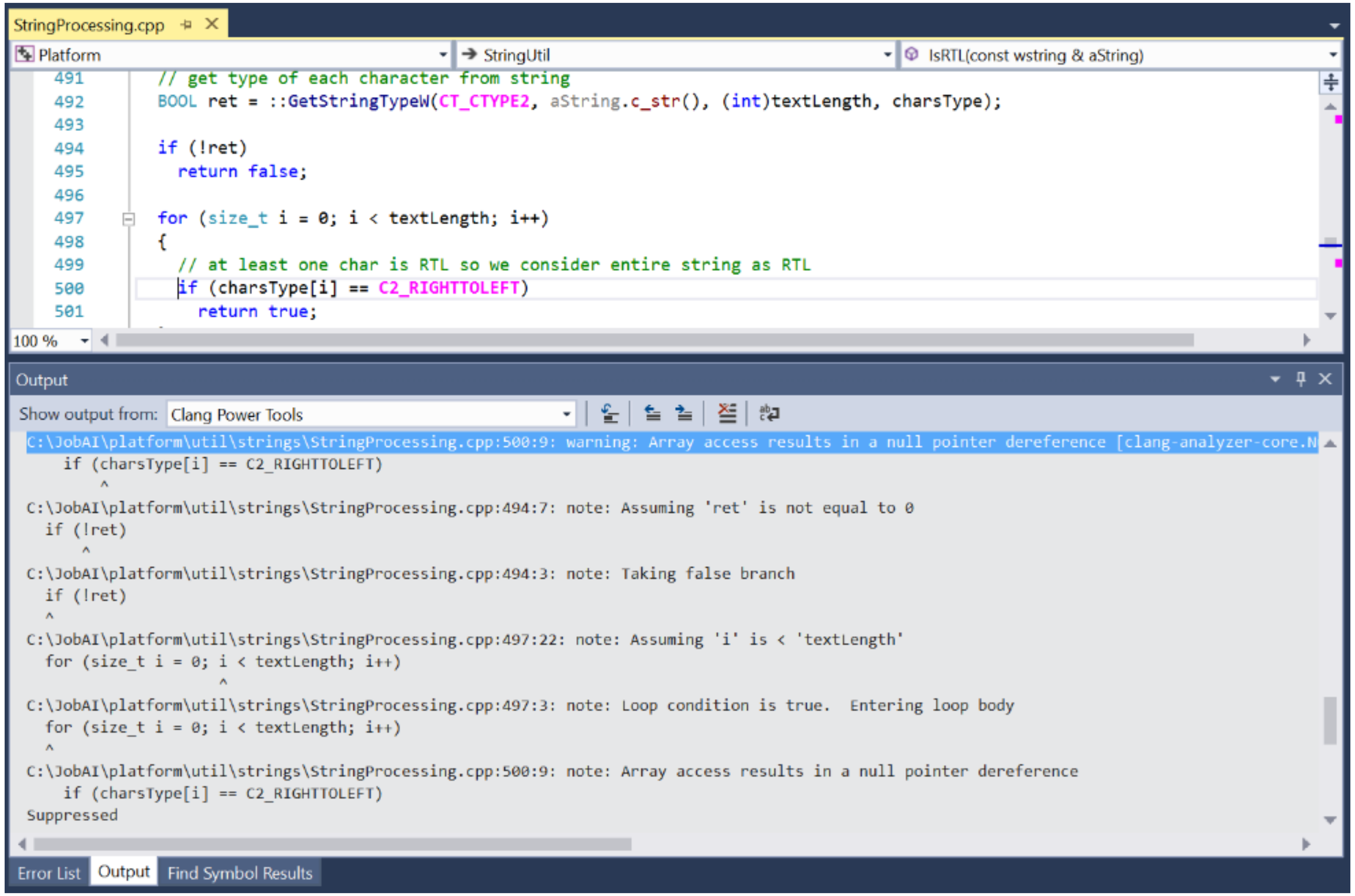


```
{
  size_t textLength = aString.length();

  CAutoVectorPtr<WORD> charsType;
  charsType.Allocate(textLength);

  Facet facet = DEFAULT_LOCALE;

  // get type of each character from string
  BOOL ret = ::GetStringTypeW(CT_CTYPE2, aString.c_str(), (int)textLength, charsType);
  if (!ret)
    return false;

  for (size_t i = 0; i < textLength; i++)
  {
    // at least one char is RTL so we consider entire string as RTL
    if (charsType[i] == C2_RIGHTTOLEFT)
```

```
1: C:\JobAI\platform\util\strings\StringProcessing.cpp
Error: C:\JobAI\platform\util\strings\StringProcessing.cpp:504:9: error: no viable conversion from 'const wchar_t [6]' to 'Facet'
  Facet facet = DEFAULT_LOCALE;
        ^       ~~~~~~~~~~~~~~
C:\JobAI\platform\util\strings\StringProcessing.cpp
:344:7: note: candidate constructor (the implicit copy constructor) not viable: no known conversion from 'const wchar_t [6]' to 'cons
class Facet
      ^
C:\JobAI\platform\util\strings\StringProcessing.cpp:344:7: note: candidate constructor (the implicit move constructor) not viable: no
class Facet
      ^
```

⬅ **Clang compile error**

Using The "Clang Power Tools" Visual Studio Extension



StringProcessing.cpp

Platform | StringUtil | IsRTL(const wstring & aString)

```cpp
    // get type of each character from string
    BOOL ret = ::GetStringTypeW(CT_CTYPE2, aString.c_str(), (int)textLength, charsType);

    if (!ret)
      return false;

    for (size_t i = 0; i < textLength; i++)
    {
      // at least one char is RTL so we consider entire string as RTL
      if (charsType[i] == C2_RIGHTTOLEFT)
        return true;
```

Output

Show output from: Clang Power Tools

```
C:\JobAI\platform\util\strings\StringProcessing.cpp:500:9: warning: Array access results in a null pointer dereference [clang-analyzer-core.N
    if (charsType[i] == C2_RIGHTTOLEFT)
        ^
C:\JobAI\platform\util\strings\StringProcessing.cpp:494:7: note: Assuming 'ret' is not equal to 0
  if (!ret)
      ^
C:\JobAI\platform\util\strings\StringProcessing.cpp:494:3: note: Taking false branch
  if (!ret)
  ^
C:\JobAI\platform\util\strings\StringProcessing.cpp:497:22: note: Assuming 'i' is < 'textLength'
  for (size_t i = 0; i < textLength; i++)
                     ^
C:\JobAI\platform\util\strings\StringProcessing.cpp:497:3: note: Loop condition is true.  Entering loop body
  for (size_t i = 0; i < textLength; i++)
  ^
C:\JobAI\platform\util\strings\StringProcessing.cpp:500:9: note: Array access results in a null pointer dereference
    if (charsType[i] == C2_RIGHTTOLEFT)
Suppressed
```

Error List | Output | Find Symbol Results

⬅ **clang-tidy : analyzer report**

🤔

Eg.

`[clang-analyzer-core.NullDereference]`

Where Can I Get It ?

(Free)

Extension for Visual Studio 2015/2017 www.clangpowertools.com

Clang Power Tools [marketplace.visualstudio.com]

PowerShell scripts: `sample-clang-build.ps1 => clang-build.ps1`

https://github.com/Caphyon/clang-power-tools/blob/master/ClangPowerTools/ClangPowerTools/clang-build.ps1

https://github.com/Caphyon/clang-power-tools/blob/master/ClangPowerTools/ClangPowerTools/sample-clang-build.ps1

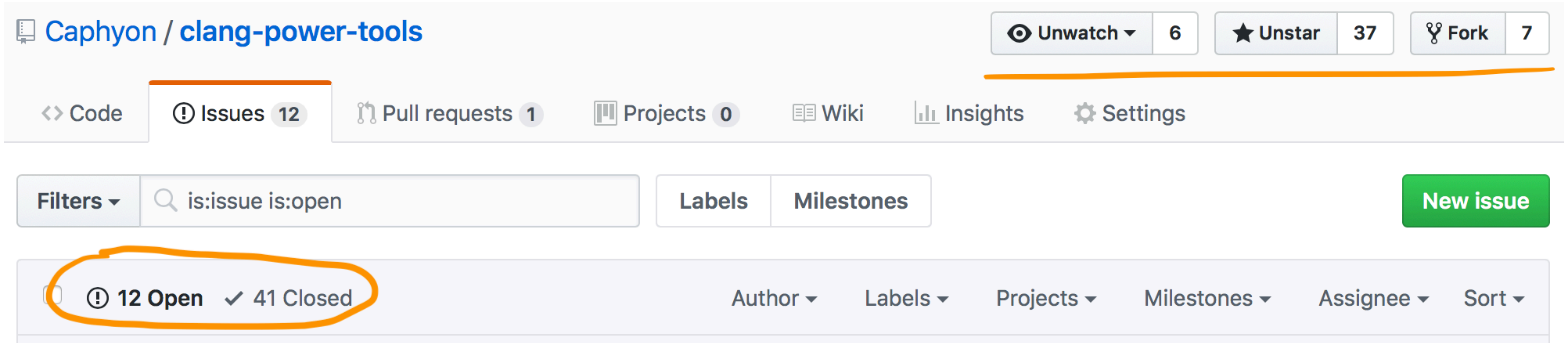
Get Involved

https://github.com/Caphyon/clang-power-tools

- submit issues/bugs
- give us feedback
- make pull requests
- suggest new features and improvements

www.clangpowertools.com

Get Involved



Caphyon / clang-power-tools

www.clangpowertools.com

Beyond clang-tidy

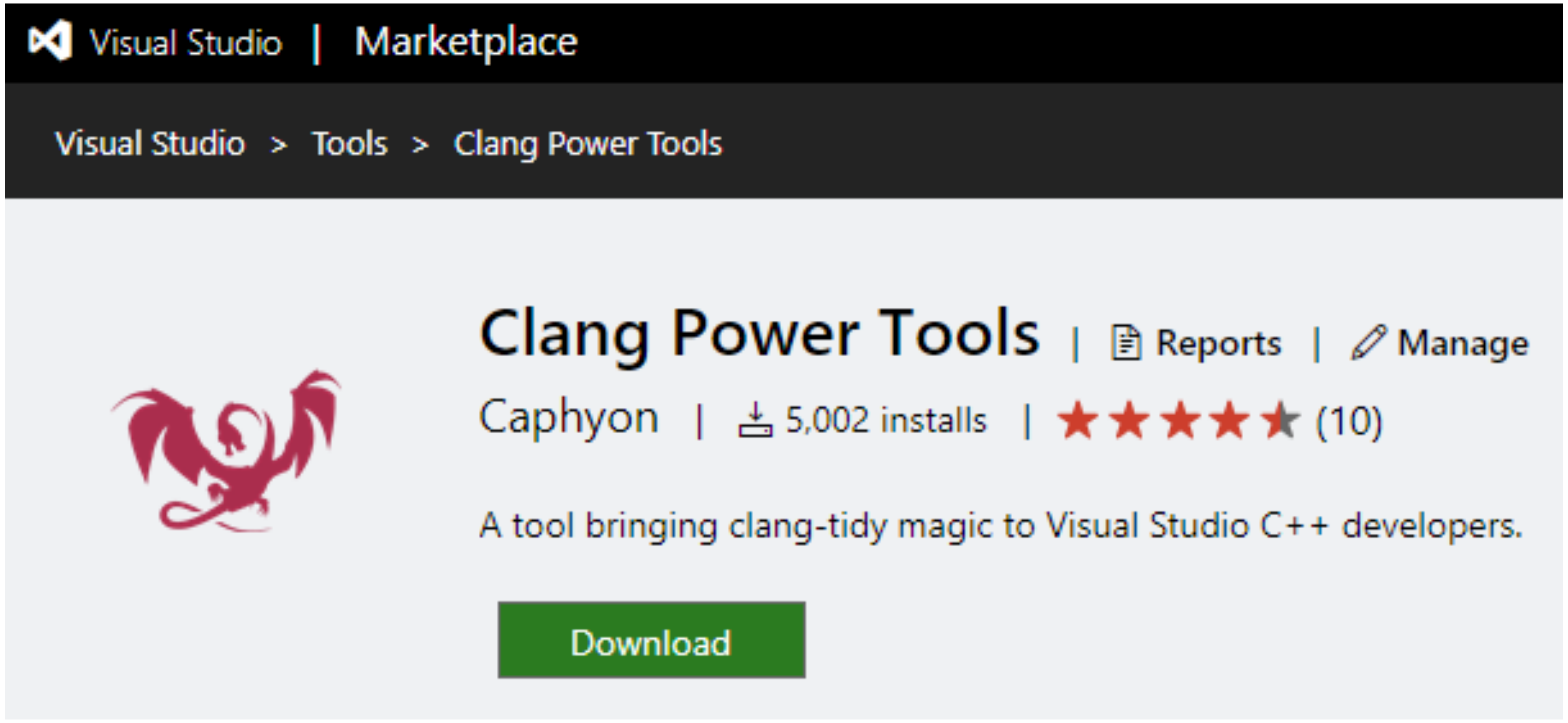
LibTooling

- **we wrote `custom tools` for our needs (project specific)**
- **fixed hundreds of member initializer lists with wrong order [-Wreorder]**
- **removed unused class private fields (references, pointers) [-Wunused-private-field]**
- **refactored some heavily used class constructors (changed mechanism for acquiring dependencies - interface refs)**
- **even more on the way...**

Roadmap

- **`-Wextra`** (a few remaining issues in our code)
- improve **Clang Power Tools** Visual Studio extension
- run more clang-tidy checks (fix more issues with **`clang-analyzer-*`**)
- re-run previous checks (for new code)
- use **`libTooling`** for more custom code transformations (project-specific)

44 days and counting...



Visual Studio | Marketplace

Visual Studio > Tools > Clang Power Tools

Clang Power Tools | Reports | Manage

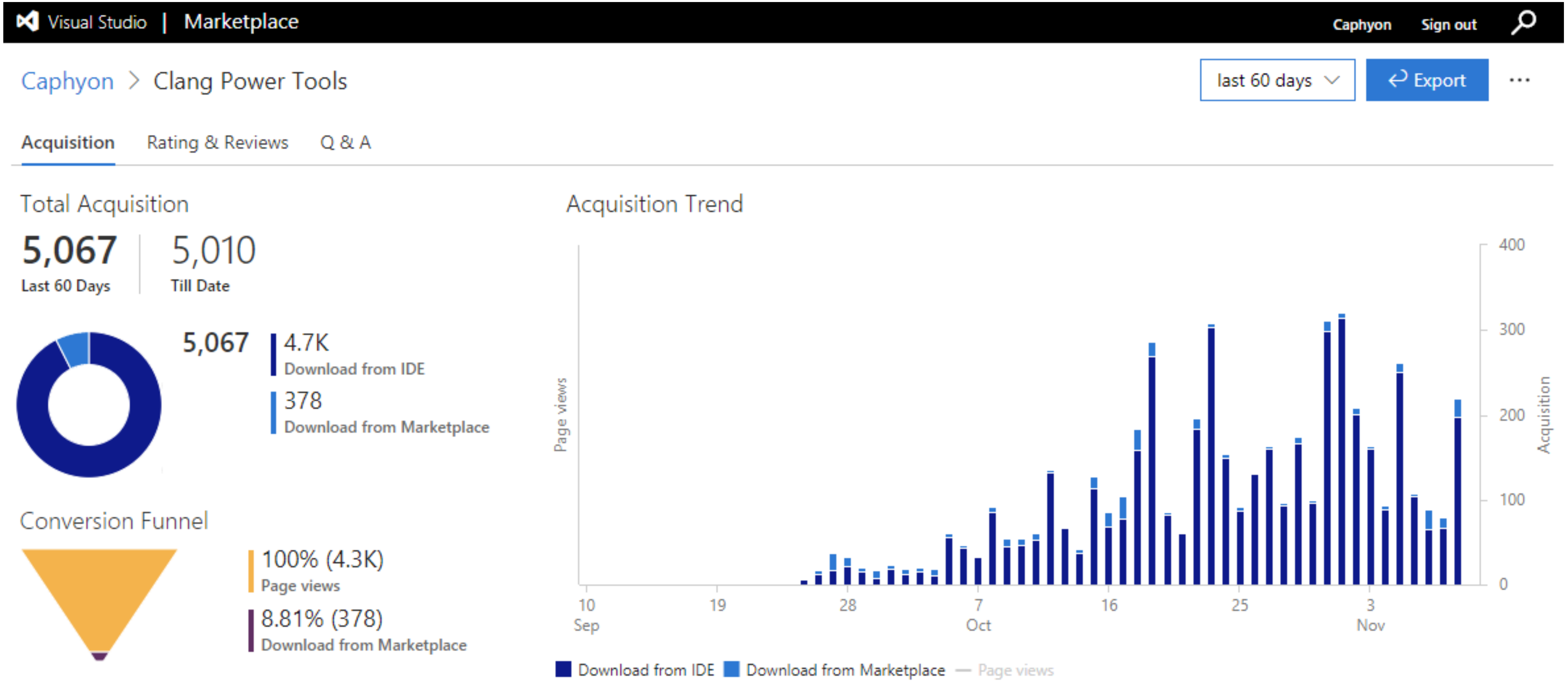
Caphyon | 5,002 installs | ★★★★★ (10)

A tool bringing clang-tidy magic to Visual Studio C++ developers.

Thank you to all early users for great feedback and bug reports !

🤗

44 days and counting...



Visual Studio | Marketplace

Caphyon Sign out

Caphyon > Clang Power Tools

last 60 days Export ...

Acquisition Rating & Reviews Q & A

Total Acquisition

5,067 Last 60 Days | 5,010 Till Date

5,067

4.7K Download from IDE

378 Download from Marketplace

Conversion Funnel

100% (4.3K) Page views

8.81% (378) Download from Marketplace

Acquisition Trend

Questions

?

www.clangpowertools.com