cppcon the c++ conference

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

September 27, 2017


CAPHYON

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

Intro

Who Am I ?

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)

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"

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, here, at CppCon...

- September, 2016 - started thinking about adopting **clang-format** (experimenting with various configs)
- September 21, 2016 - **CppCon**: *"clang-format Birds of a Feather"*
- October-November 2016 - preparing for clang-format adoption (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**)

https://marketplace.visualstudio.com/items?itemName=LLVMExtensions.ClangFormat

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
- 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]

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

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);

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

Fixes, fixes, fixes...

Frequent offender:

```
Error: 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:344:7: note: candidate constructor (the implicit move constructor) not viable:
no known conversion from 'const wchar_t [6]' to '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 &' (aka 'const basic_string<wchar_t, char_traits<wchar_t>,
allocator<wchar_t> > &') for 1st argument
  Facet(const wstring & facet)
  ^
```

Timeline

- January 12, 2017 - started playing with Clang for Windows (LLVM 3.9.1)
- January 24, 2017 - first commit, started fixing the Clang errors/warnings (Note: we were already on **`MSVC /W4 /WX`**)
- February 3, 2017 - created a clang++ compilation **`.bat`** file (crude automation attempt)
- March 7, 2017 - upgraded the clang++ batch file to a **PowerShell** script (**`clang-build.ps1`**)
- March 13, 2017 - our PS script also gains the ability to run clang-tidy checks
- March 2017 - first experiments with **clang-tidy** on our source code (just some core libraries)
- April 11, 2017 🎉 - able to compile our **whole** codebase with Clang 3.9.1 (*some default warnings disabled*) **~ 3 months**
- April 12, 2017 - created a **Jenkins** job for Clang build (every SCM change is compiled with Clang)
- May 2017 - great improvements to our PowerShell script (PCH, parallel compilation, project filters, SDK versions)
- June 2017 - more experiments with **clang-tidy** on our source code (better coverage)
- June 16, 2017 - upgraded from VS2015 to **VS2017** (we also needed to update our Clang PS script)

Timeline

- July 3, 2017 - started work on a custom clang-based refactoring tool (**`libTooling`**)
- July 10, 2017 - fixed new Clang 4 issues and upgraded to **4.0.1**
- July 2017 - started to tackle Clang **`-Wall`** warnings in our code
- August 2017 - made extensive code transformations with our custom **`libTooling`** helpers
- August 24, 2017 🎉 - our whole codebase compiles with Clang **`-Wall`**
- August 2017 - started work on our "**Clang Power Tools**" extension for Visual Studio
- August 25, 2017 - first refactorings with **clang-tidy**: **`modernize-use-nullptr, modernize-loop-convert`**
- Aug-Sep 2017 - multiple code transformations with **clang-tidy**: **`modernize-*, readability-*, misc-*,...`**
- September 2017 - started to fix **`-Wextra`** warnings (*in progress...*)
- September 11, 2017 - upgraded to LLVM **5.0** (fixed new warnings) **`[-Wunused-lambda-capture]`**
- September 11, 2017 - **open-sourced** "Clang Power Tools" extension for VS and PowerShell script "**`clang-build.ps1`**"
- September 26, 2017 - published our "Clang Power Tools" extension to **Visual Studio Marketplace**
- **September 27**, 2017 - 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`

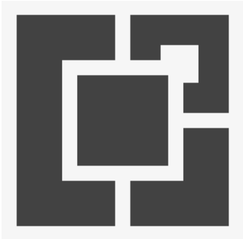
clang-tidy

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-*`

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
)
```

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

- over 1,000 lines
- very configurable (many parameters)
- supports both clang compile and tidy workflows
- works directly on Visual Studio .vcxproj files (or MSBuild projects)

  (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
-include-dirs	Directories to be used for includes
-parallel	Run clang++ in parallel mode, on all logical CPU cores (incompatible with **-tidy-fix**)
-continue	Continue project compilation even when errors occur
-clang-flags	Flags passed to clang++
-literal	Disable name regex matching for projects and source files
-tidy	Run specified clang-tidy checks
-tidy-fix	Run specified clang-tidy checks with auto-fix
-vs-ver	Visual Studio Edition (eg. "2017")
-vs-sku	Visual Studio SKU (eg. "Community", "Professional")
-win-sdk-ver	Default Windows SDK Version (if projects are missing Windows Target Platform)

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 for your team.

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

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

➡ Runs clang-tidy, using all *modernize* checks, on all CPPs containing 'meow' in their name, from the projects containing 'foo' or 'bar' in their names.

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

➡ Runs clang compile on all projects in current directory, except '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]   $aCppToCompile
     , [alias("parallel")]     [Parameter(Mandatory=$false)][switch]   $aUseParallelCompile
     , [alias("continue")]     [Parameter(Mandatory=$false)][switch]   $aContinueOnError
     , [alias("literal")]      [Parameter(Mandatory=$false)][switch]   $aDisableNameRegexMatching
     , [alias("tidy")]         [Parameter(Mandatory=$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 kIncludeDirectories  -value @( "third-party",
                                                , "third-party\WTL\Include"
                                                ) -Option Constant

Set-Variable -name kVisualStudioVersion -value "2017"                          -Option Constant
Set-Variable -name kVisualStudioSku     -value "Professional"                  -Option Constant
```

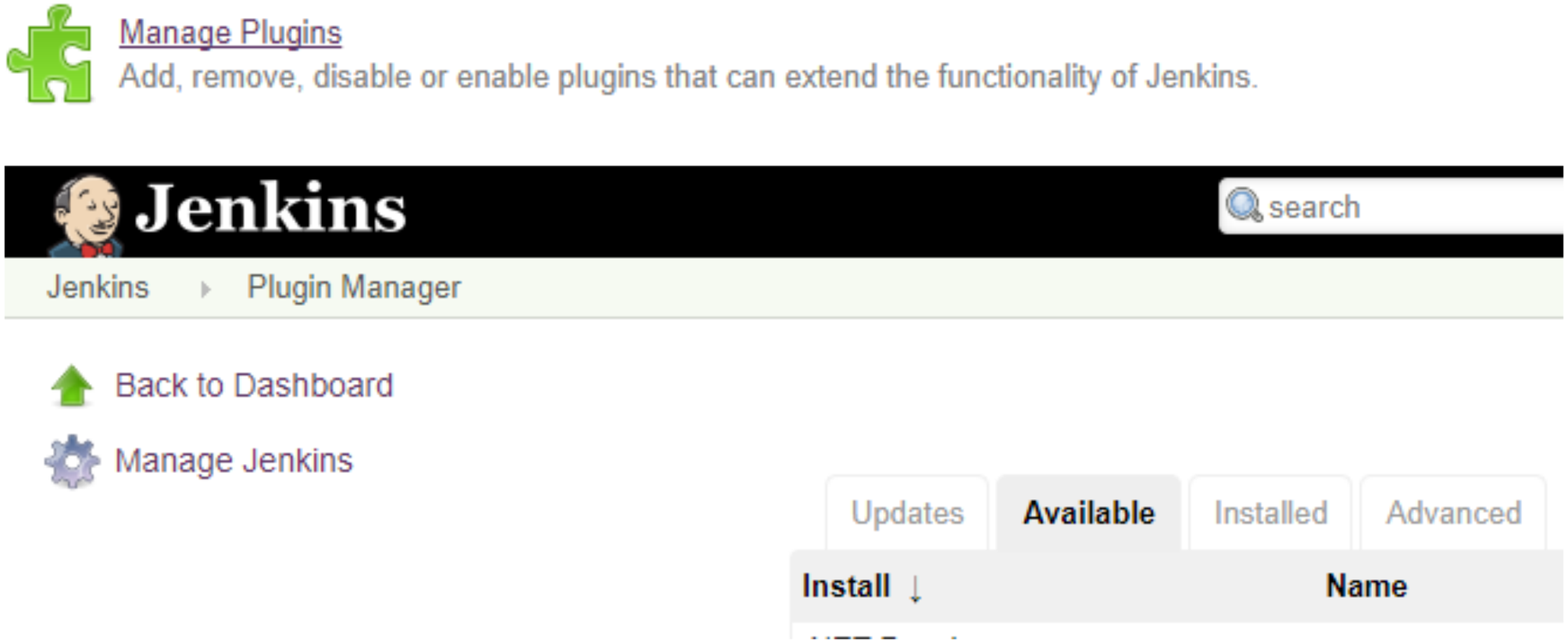
Using The PowerShell Script

+

Jenkins CI Configuration

Jenkins CI Configuration

Install PowerShell plugin (available from Jenkins gallery)



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

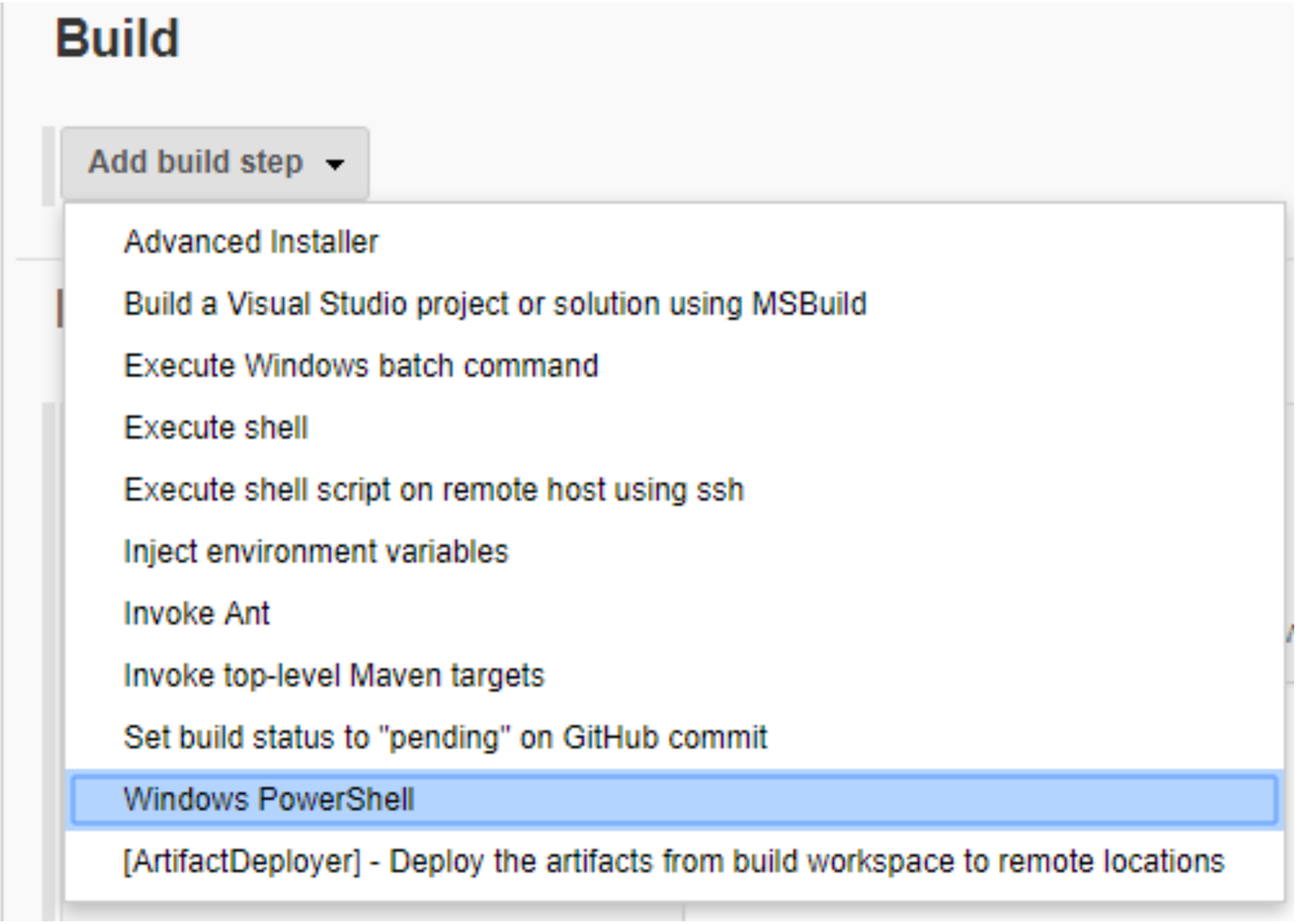
https://wiki.jenkins.io/display/JENKINS/PowerShell+Plugin

Jenkins CI Configuration

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

or

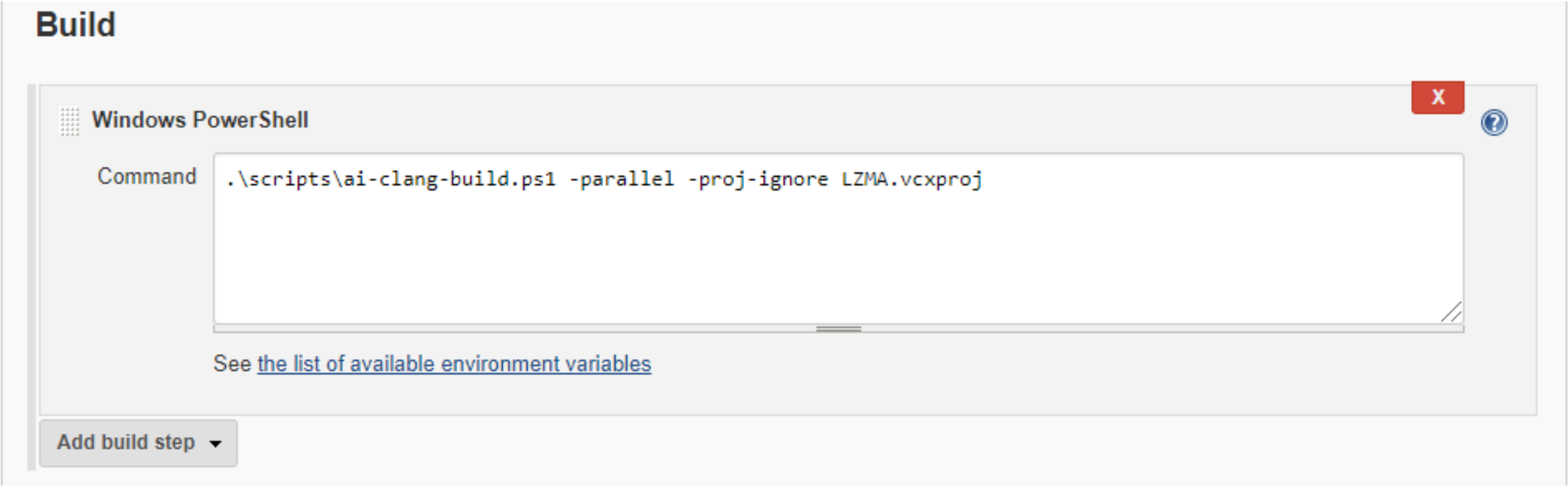
- Attach a **new build step** on an existing job



Jenkins CI Configuration

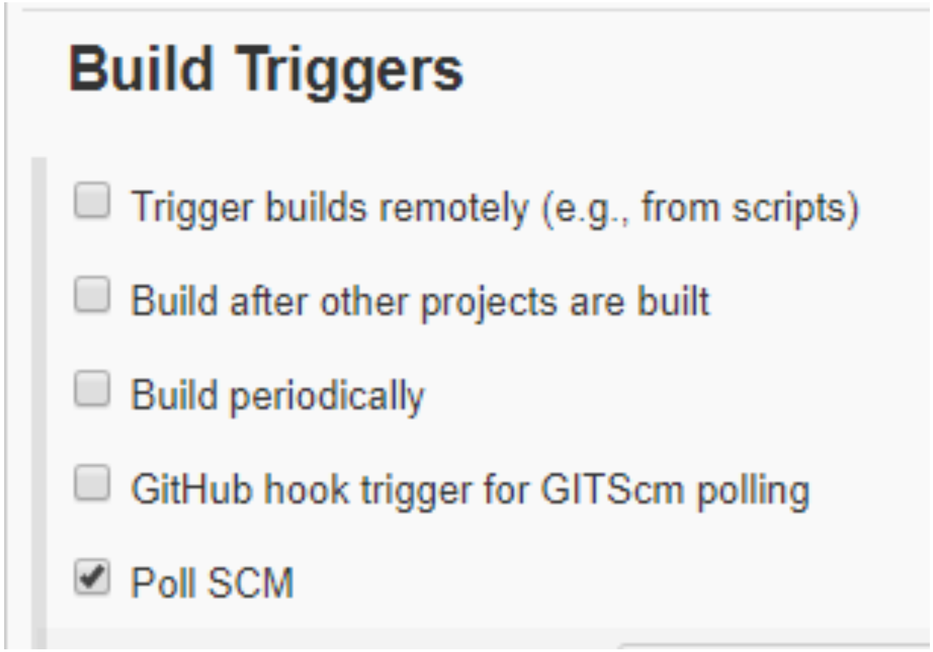
Reference PowerShell script from the job working directory.

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



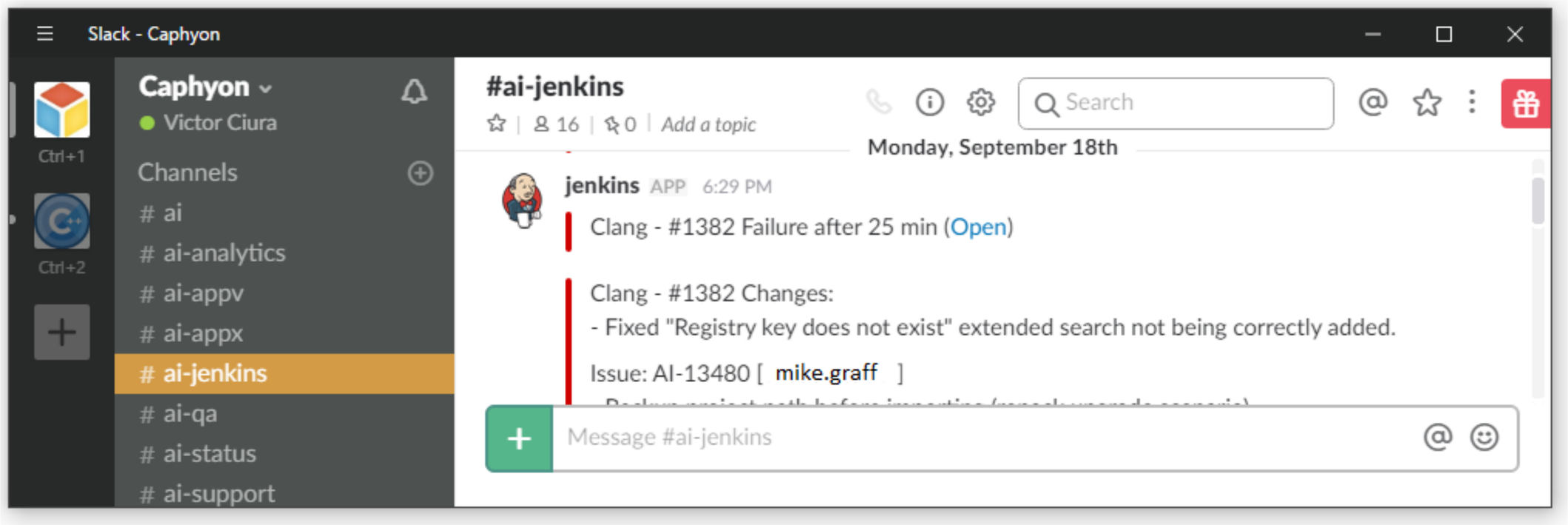
Jenkins CI Configuration

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



Jenkins CI Workflow

🔥 **When Clang build is broken...**

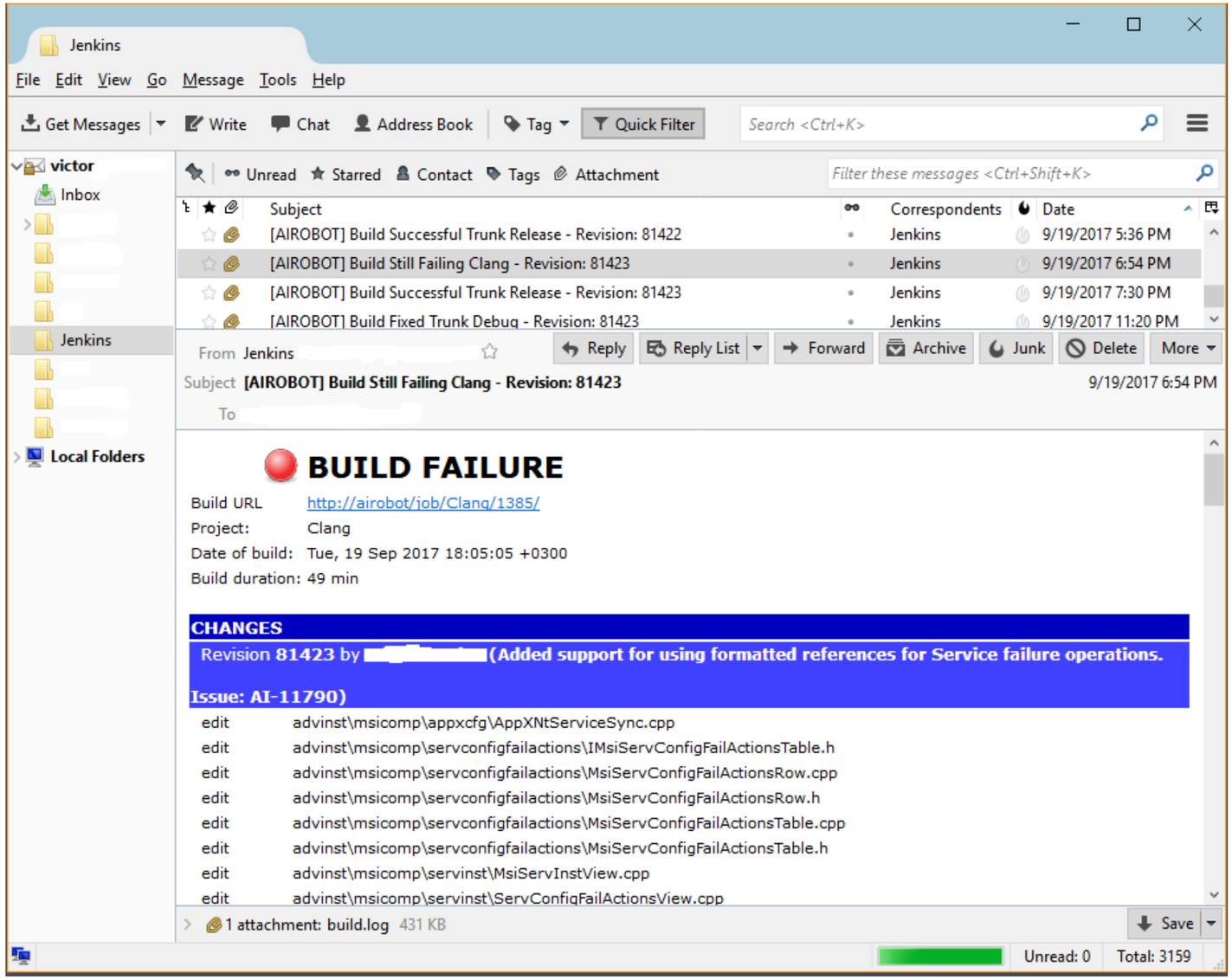


Slack bot alert ➡ #ai-jenkins

Jenkins CI Workflow

When Clang build is broken...

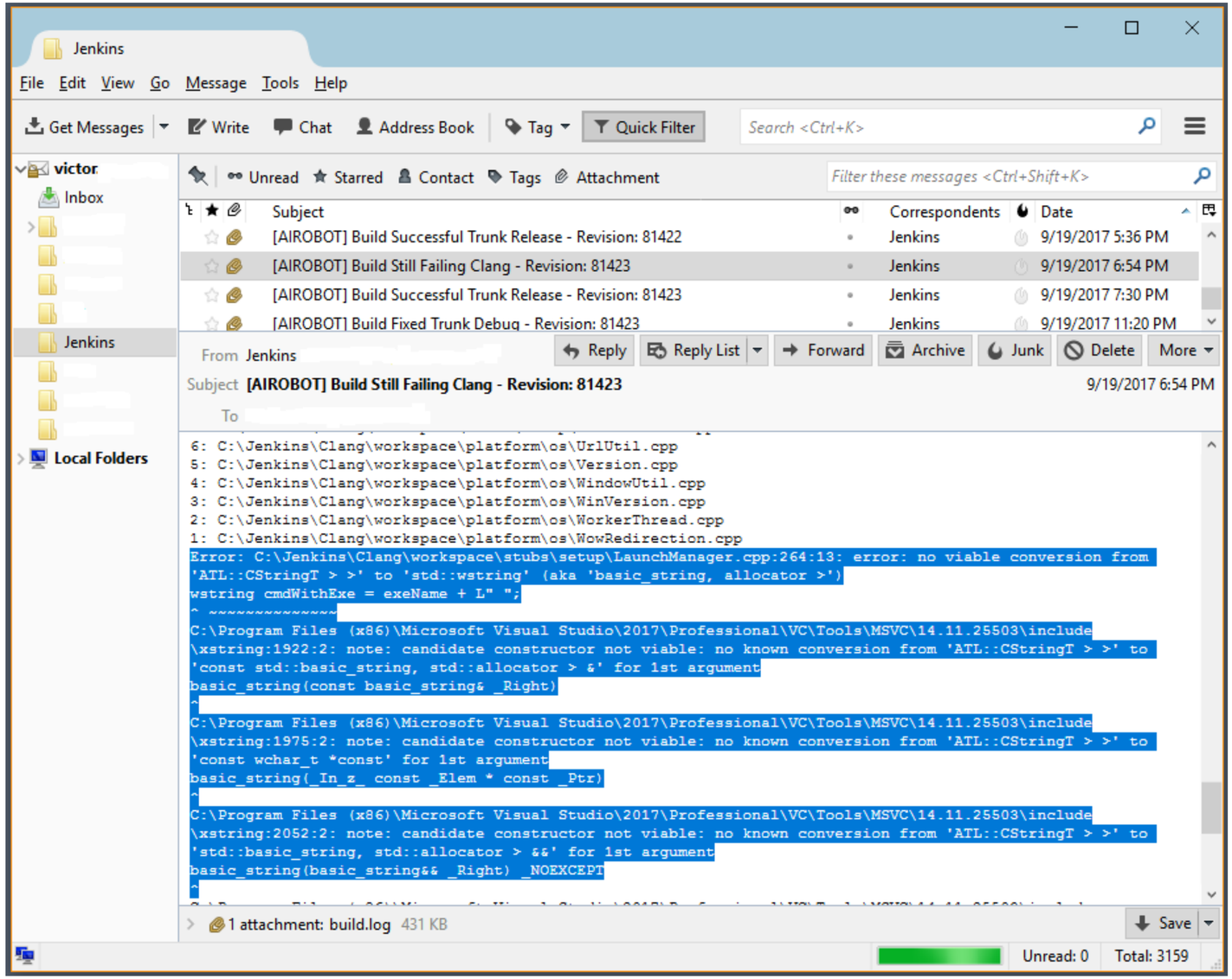
Team devs email alert ➡



Jenkins CI Workflow

When Clang build is broken...

Team devs email alert ➡

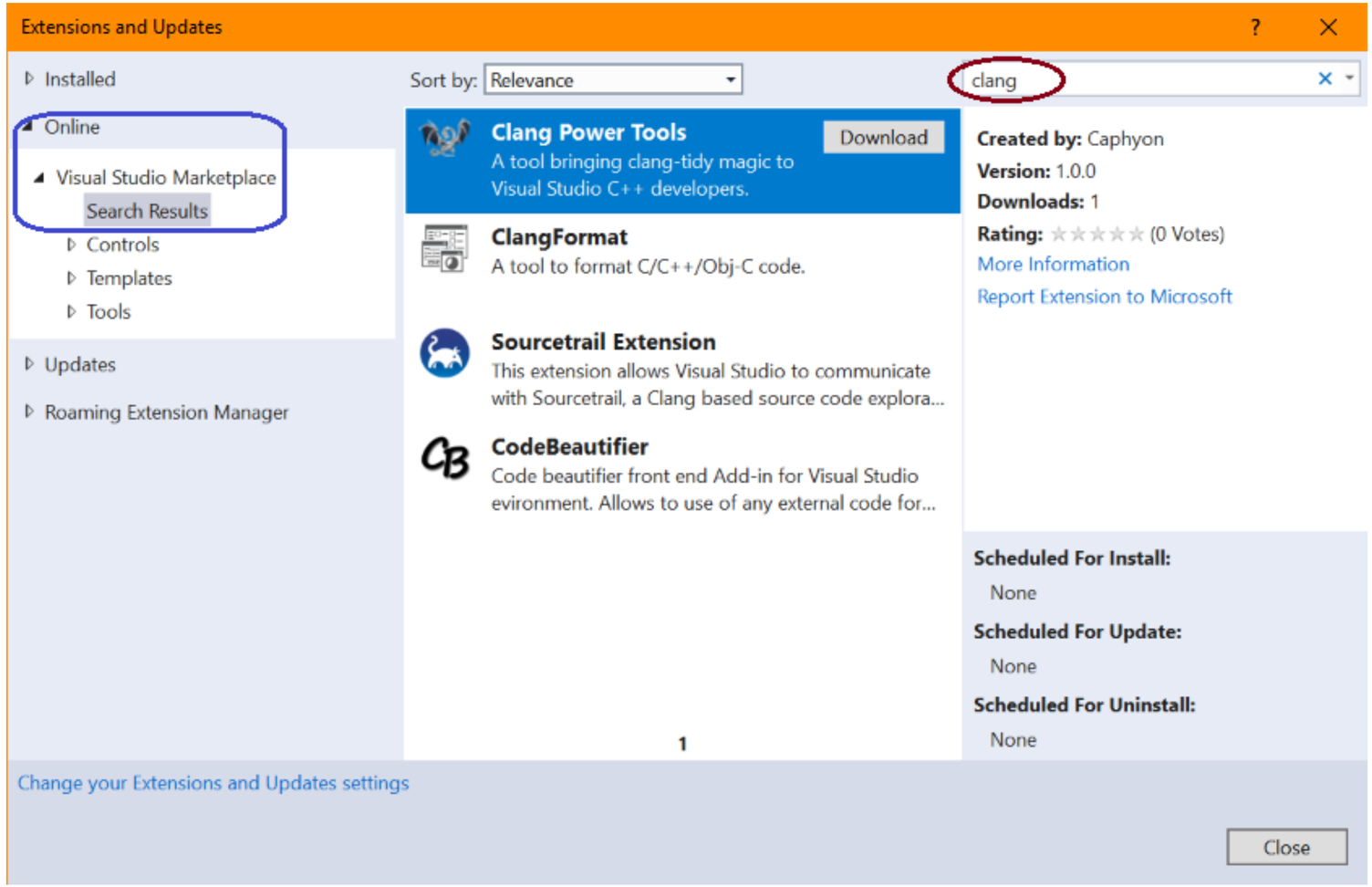


What About Developer Workflow?

+

Install The "Clang Power Tools" Visual Studio Extension

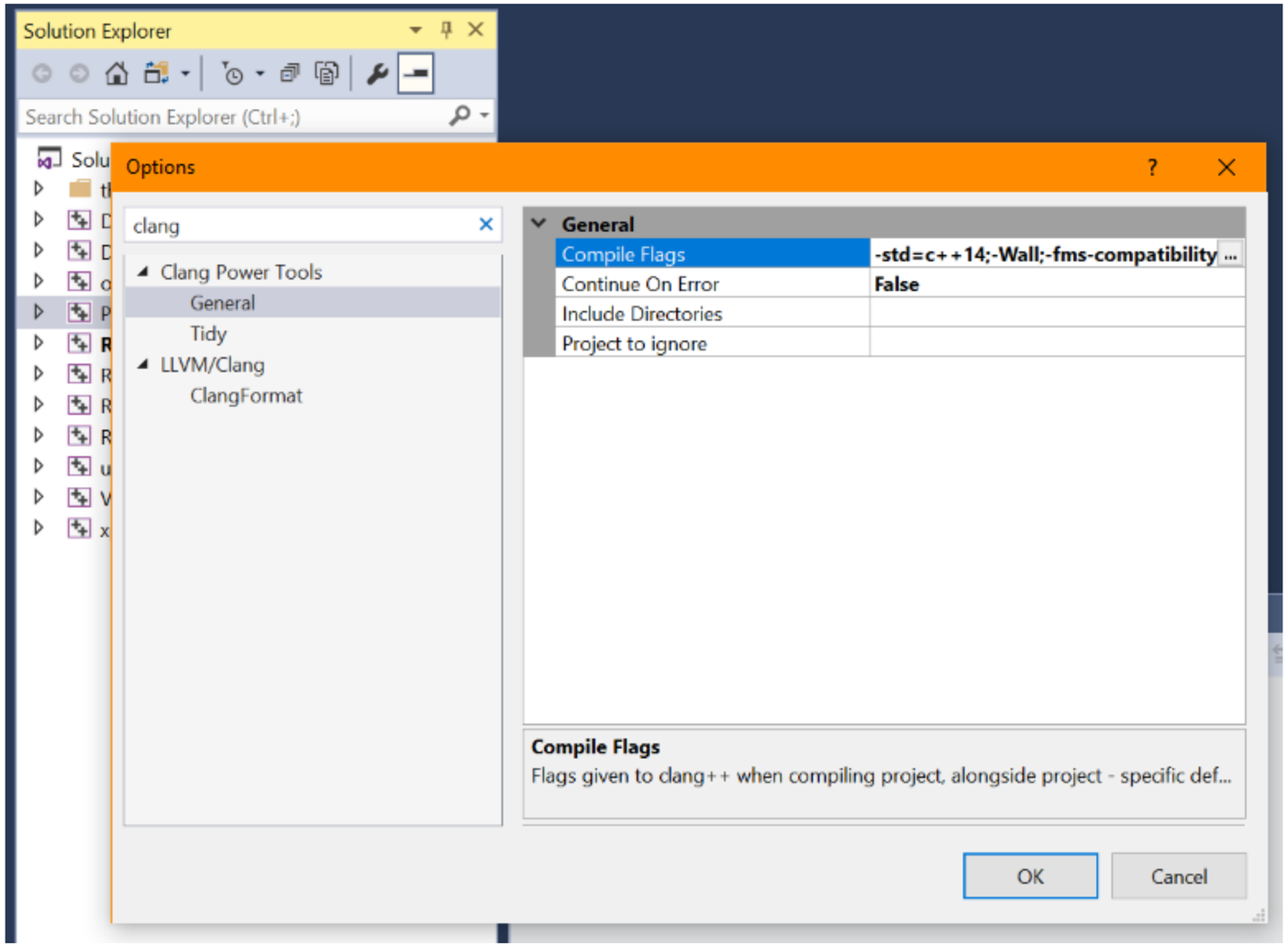
[Tools]
⬇
Extensions and updates



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

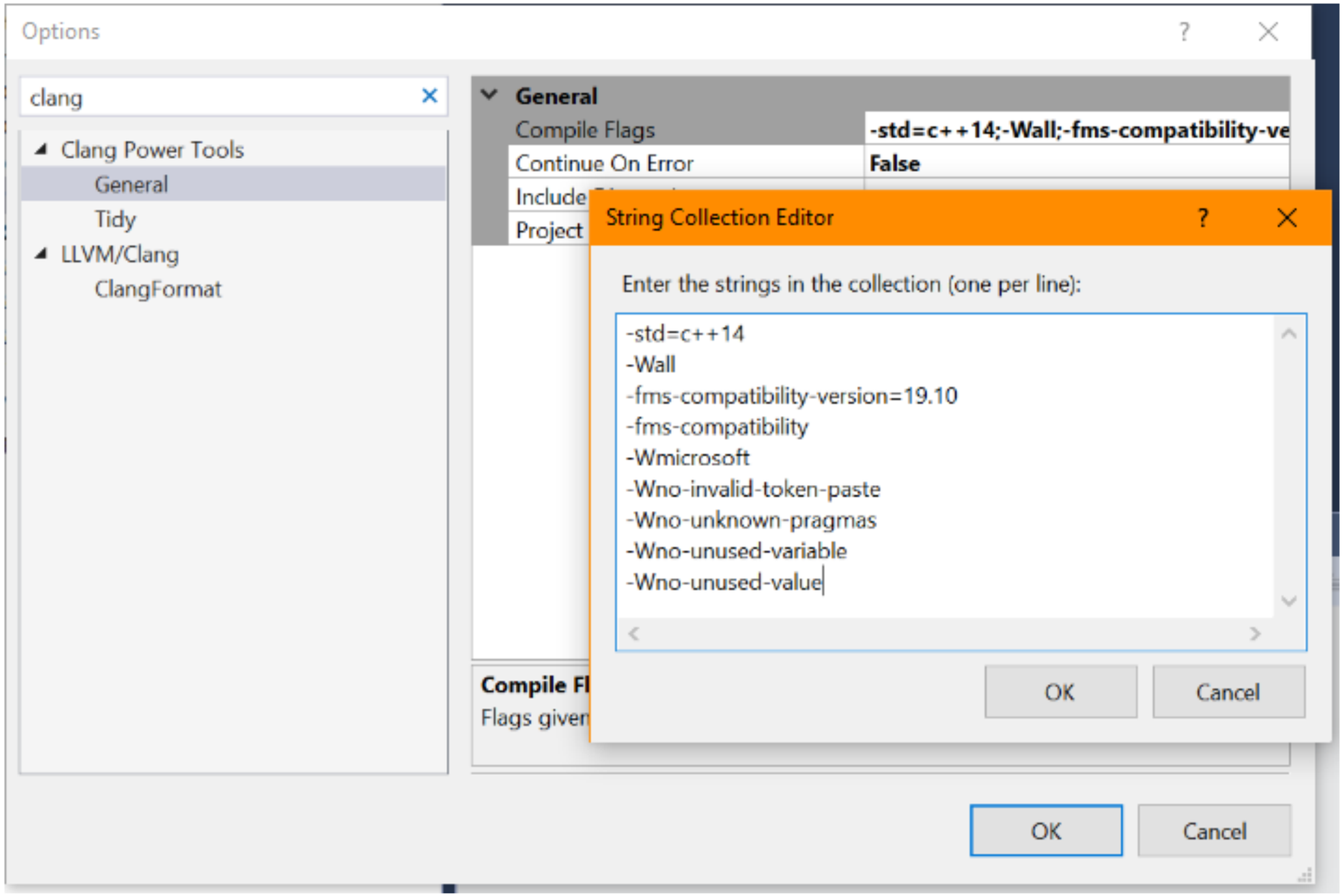
[Tools]
⬇
⚙ Options...
⬇
clang



⬅ Compilation settings

Configure The "Clang Power Tools" Visual Studio Extension

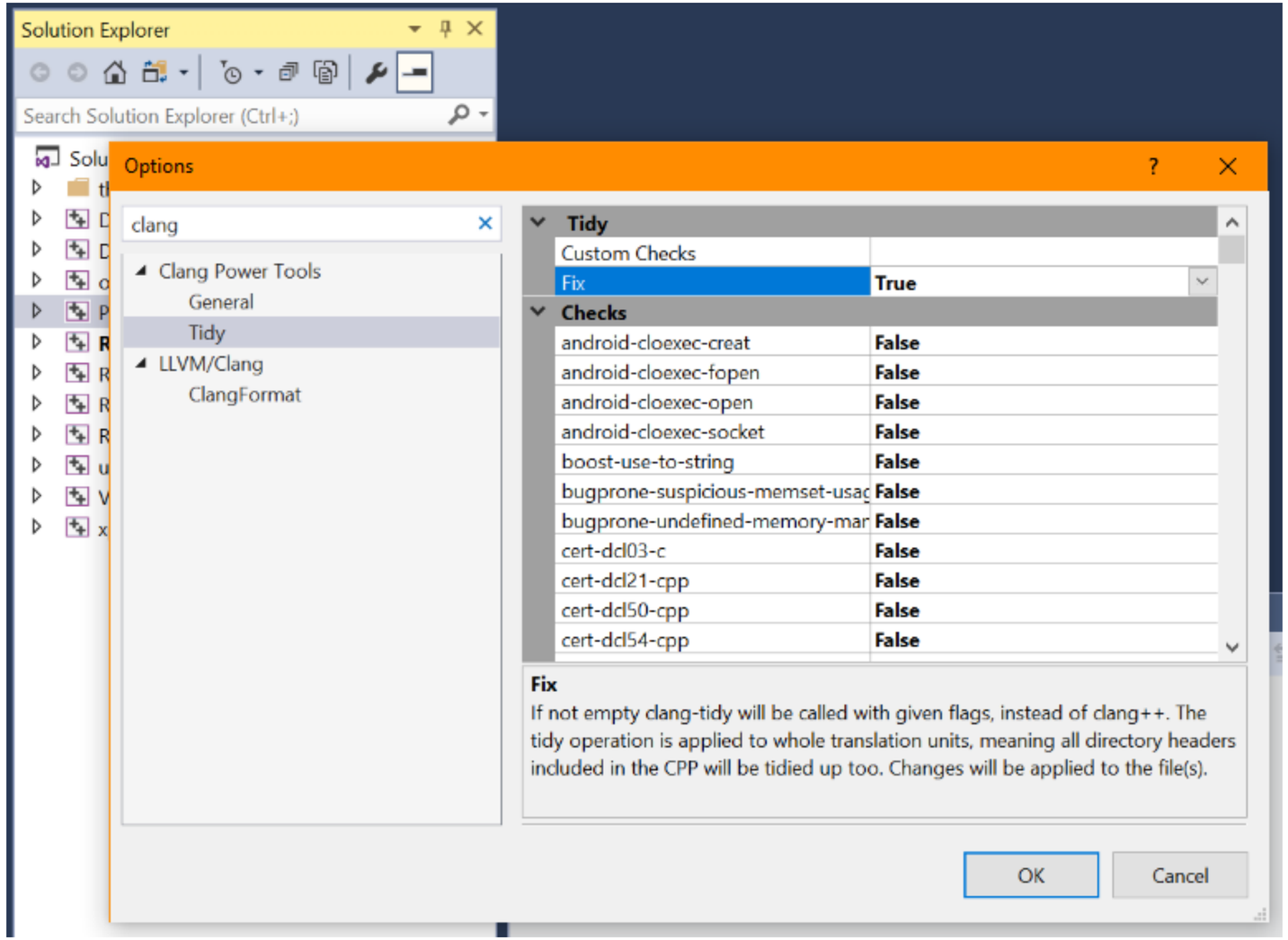
[Tools]
⬇
⚙ Options...
⬇
clang



⬅ clang++ flags

Configure The "Clang Power Tools" Visual Studio Extension

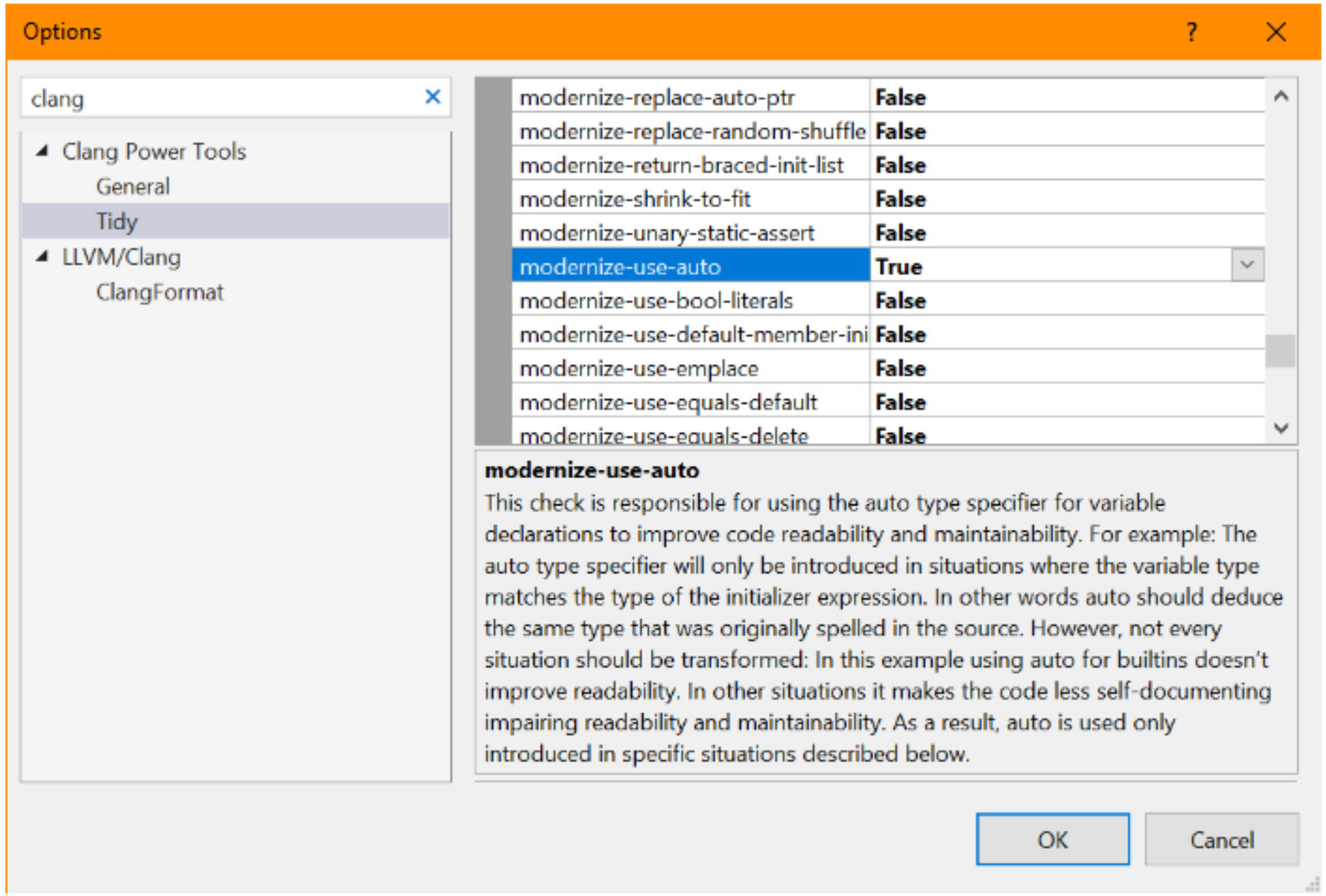
[Tools]
⬇
⚙ Options...
⬇
clang



⬅ clang-tidy settings

Configure The "Clang Power Tools" Visual Studio Extension

[Tools]
⬇
⚙ Options...
⬇
clang

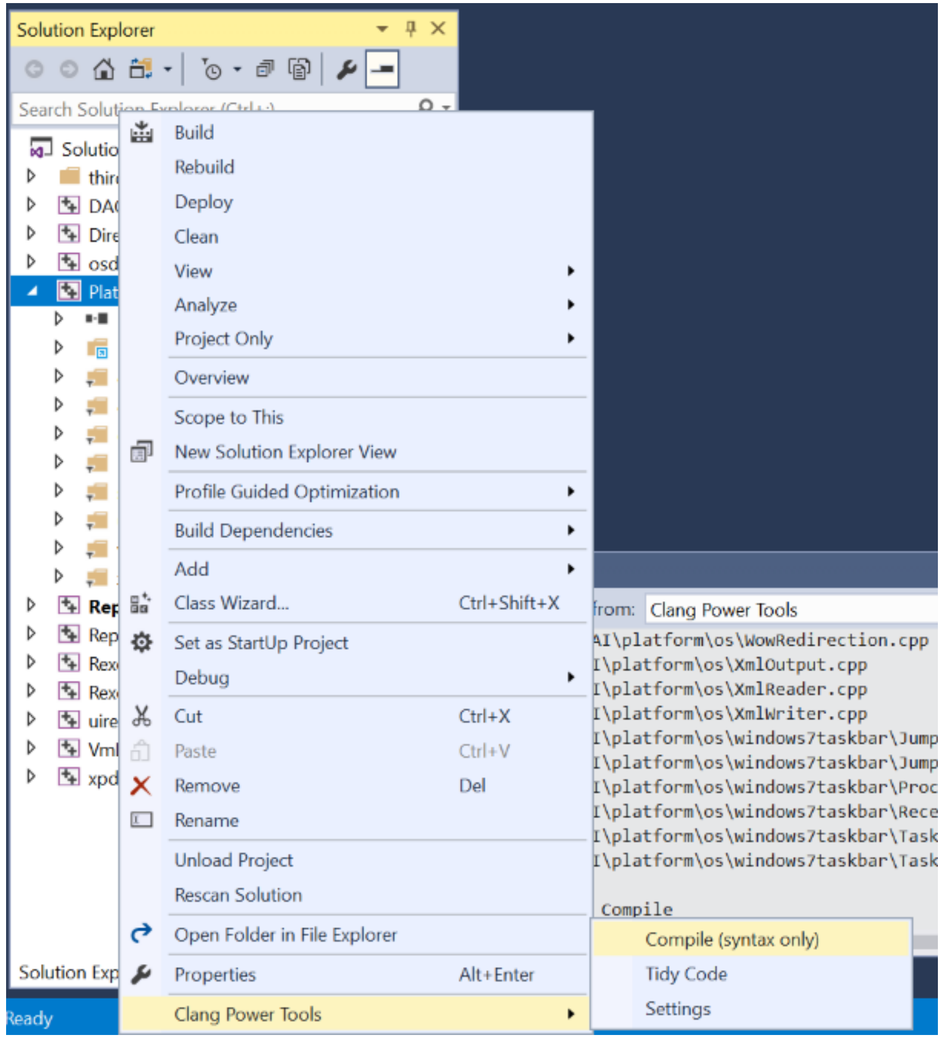


⬅ clang-tidy checks

⬅ inline documentation

Using The "Clang Power Tools" Visual Studio Extension

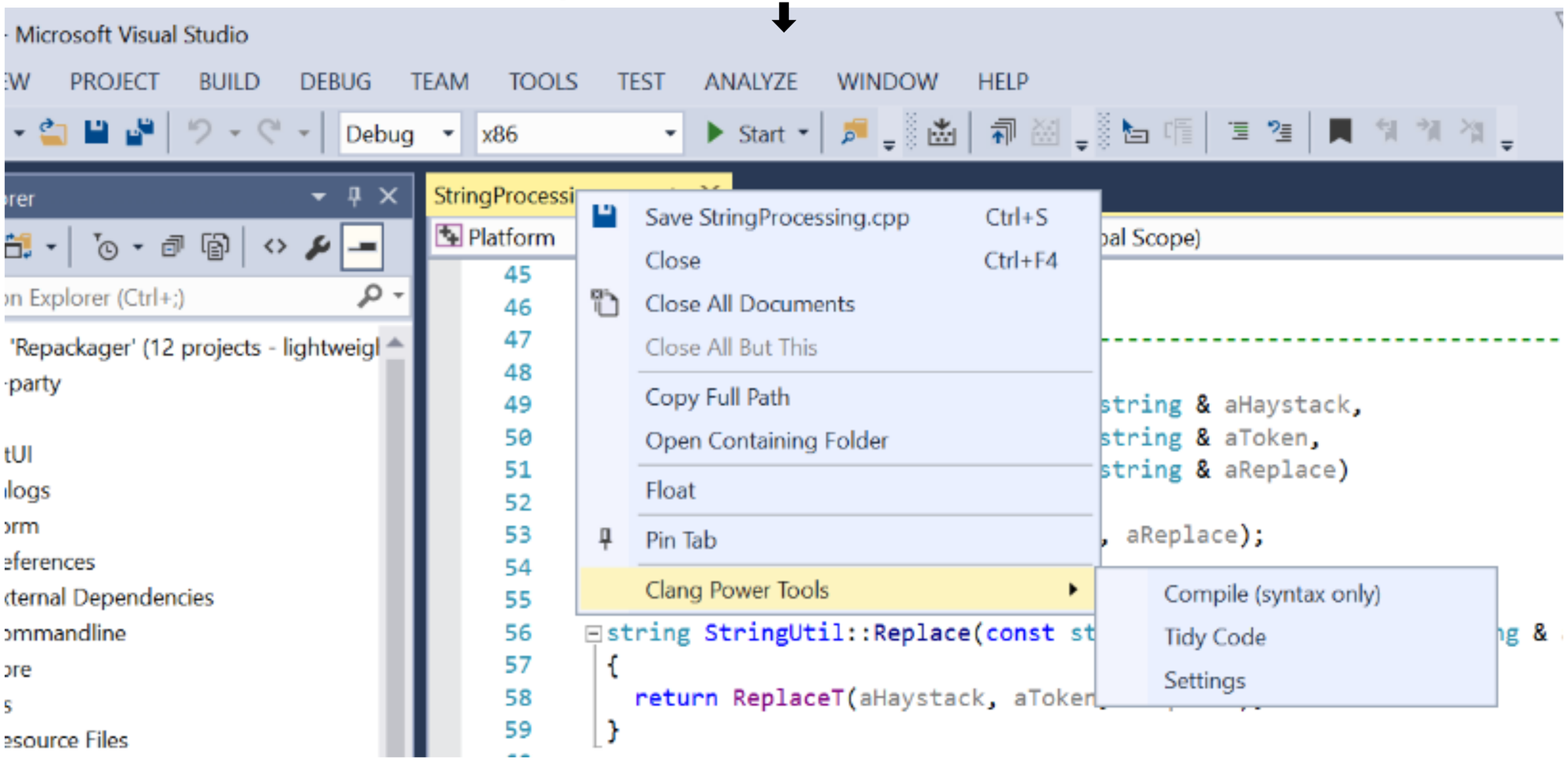
Run Clang Power Tools on a whole project or solution ➡



⬅ **Compile or Tidy code**

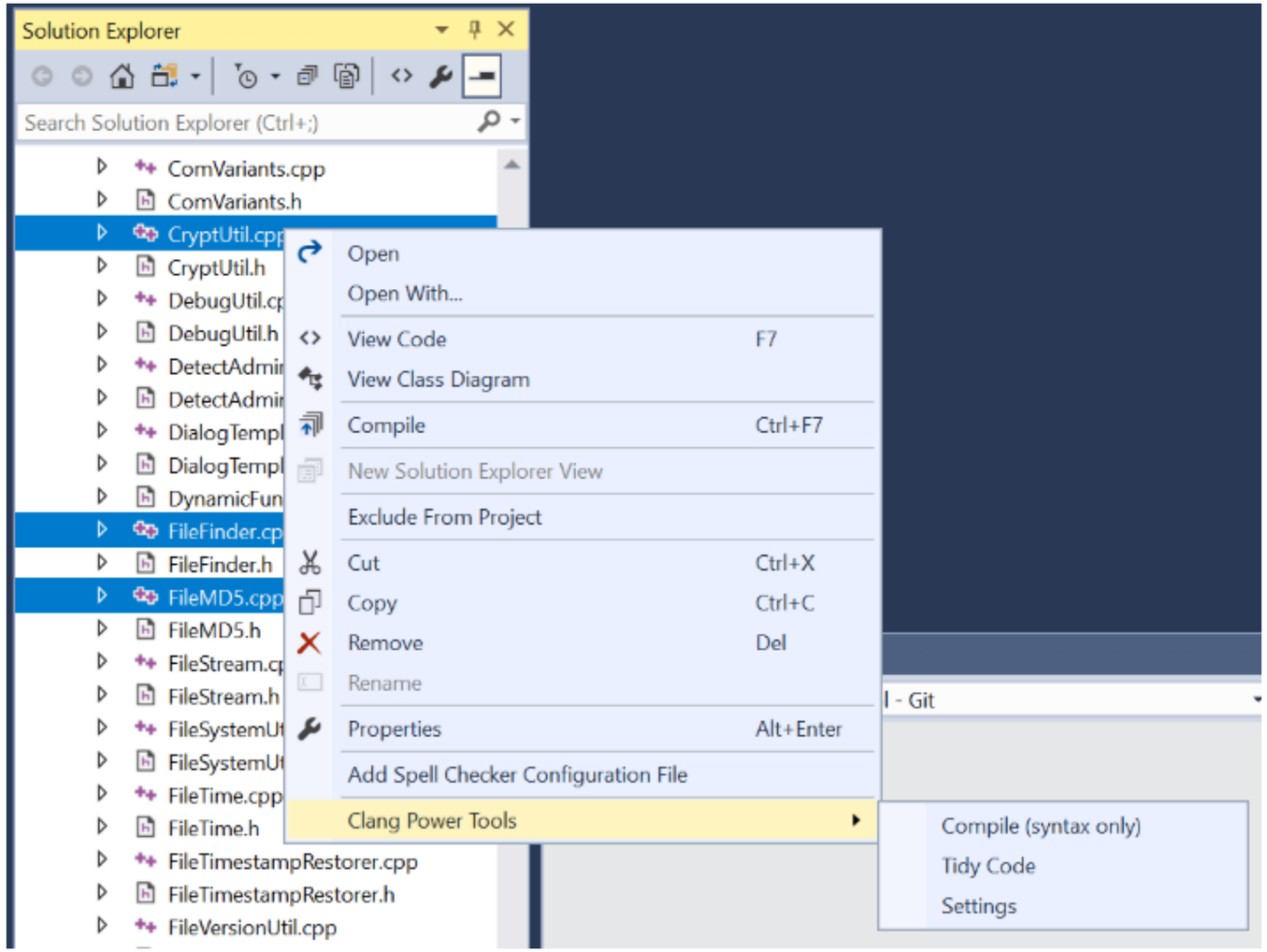
Using The "Clang Power Tools" Visual Studio Extension

Run Clang Power Tools on an open source file



Using The "Clang Power Tools" Visual Studio Extension

Run Clang Power Tools on selected files ➡



⬅ **Compile or Tidy code**

Using The "Clang Power Tools" Visual Studio Extension

StringProcessing.cpp | StringEncoding.cpp

Platform | StringUtil | IsRTL(const wstring & aString)

```
498  {
499    size_t textLength = aString.length();
500
501    CAutoVectorPtr<WORD> charsType;
502    charsType.Allocate(textLength);
503
504    Facet facet = DEFAULT_LOCALE;
505
506    // get type of each character from string
507    BOOL ret = ::GetStringTypeW(CT_CTYPE2, aString.c_str(), (int)textLength, charsType);
508    if (!ret)
509      return false;
510
511    for (size_t i = 0; i < textLength; i++)
512    {
513      // at least one char is RTL so we consider entire string as RTL
514      if (charsType[i] == C2_RIGHTTOLEFT)
```

100 %

Output

Show output from: Clang Power Tools

```
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)

```
491   // get type of each character from string
492   BOOL ret = ::GetStringTypeW(CT_CTYPE2, aString.c_str(), (int)textLength, charsType);
493
494   if (!ret)
495     return false;
496
497   for (size_t i = 0; i < textLength; i++)
498   {
499     // at least one char is RTL so we consider entire string as RTL
500     if (charsType[i] == C2_RIGHTTOLEFT)
501       return true;
```

100 %

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 ?

Extension for Visual Studio 2015/2017

Clang Power Tools (Free) marketplace.visualstudio.com

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

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

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)

Questions

?