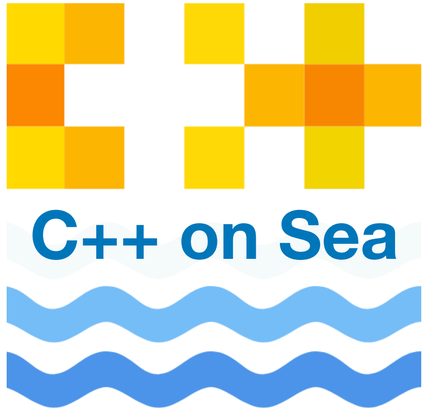



# Better Tools in Your Clang Toolbox

**February, 2019**


@ciura_victor

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

# *Abstract*

Clang-tidy is the go to assistant for most C++ programmers looking to improve their code. If you set out to modernize your aging code base and find hidden bugs along the way, clang-tidy is your friend. Last year, we brought all the clang-tidy magic to Visual Studio C++ developers with a Visual Studio extension called “Clang Power Tools”. After successful usage within our team, we decided to open-source the project and make Clang Power Tools available for free in the Visual Studio Marketplace. This helped tens of thousands of developers leverage its powers to improve their projects, regardless of their compiler of choice for building their applications.

Clang-tidy comes packed with over 250 built-in checks for best practice, potential risks and static analysis. Most of them are extremely valuable in real world code, but we found several cases where we needed to run specific checks for our project.

This session will focus on extending clang-tidy with custom checks. If you ever wanted a tidy check specific to a particular need of your codebase, you will now get a crash course in writing your own check from scratch. This talk will also share some of the things we learned while developing these tools and using them at scale on our projects and within the codebases of our community users.

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

# Better Tools in Your Clang Toolbox

## Vignette in 3 parts

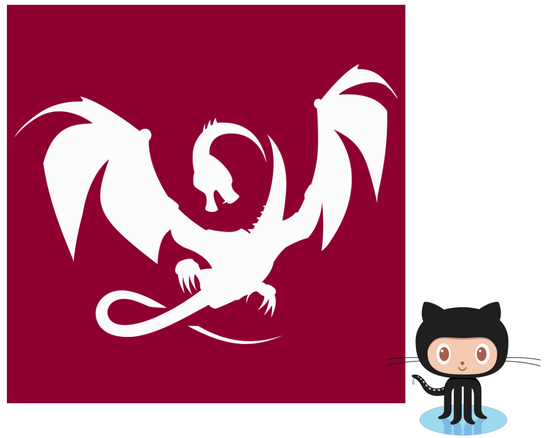
The Tools
Massaging The Code
Take Control

@ciura_victor

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

# Who Am I?

Advanced Installer



Clang Power Tools

@ciura_victor

# Part I
# The Tools

+

->

LLVM
clang-tidy
clang++
clang-format

Visual Studio
2015/2017/2019

Clang Power Tools

www.clangpowertools.com

**FREE / Open source**

# Clang Power Tools

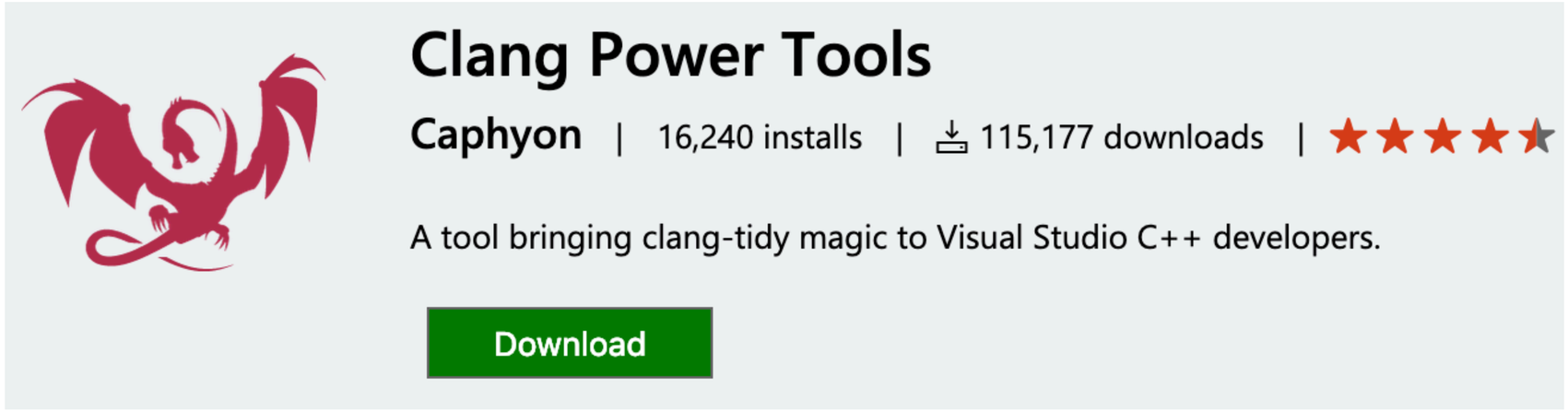
- free open-source Visual Studio extension
- helping developers leverage Clang/LLVM tools (`clang++`, `clang-tidy` and `clang-format`)
- perform various code transformations and fixes like **modernizing** code to C++ 11/14/17
- finding subtle latent **bugs** with its static analyzer and C++ Core Guidelines checks

# Lunch ~1 Year Ago: September 2017


CppCon 2017: Victor Ciura “Bringing Clang-tidy Magic to Visual Studio C++ Developers”

https://www.youtube.com/watch?v=WI-9ozmxXbo

http://sched.co/BgsQ




- **16,000** users 🎉
- **115K** installs
- **40+** releases
- 173 reported issues fixed
- 29 Git forks
- 175+ stars/followers (GitHub)
- 56+ external PRs

**1 Year and counting...**

# 1 Year and counting...

**A big Thank You to** `Gabriel` **&** `Ionuț`

**for all the great work they put into this project**

**and to all our community contributors**

# 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

# Clang PowerShell Script

- 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 `configuration file` (eg. `cpt.config`),
that pre-loads some of the constant configurations specific for your team/project
=> *source control*

# Using The PowerShell Script

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

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

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

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

```
PS> .\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'.

# cpt.config

```
<cpt-config>
  <clang-flags>  "-Werror"
                ,"-Wall"
                ,"-fms-compatibility-version=19.10"
                ,"-Wmicrosoft"
                ,"-Wno-invalid-token-paste"
                ,"-Wno-unknown-pragmas"
                ,"-Wno-unused-value"
  </clang-flags>
  <header-filter>'.*'</header-filter>
  <parallel/>
  <vs-sku>'Professional'</vs-sku>
  <file-ignore>  'htmlayoutsdk\\include\\behaviors'
                ,'vsphere\\vim25\\core'
  </file-ignore>
  <proj-ignore>  'SciLexer'
                ,'tools\\msix-psf'
  </proj-ignore>
</cpt-config>
```

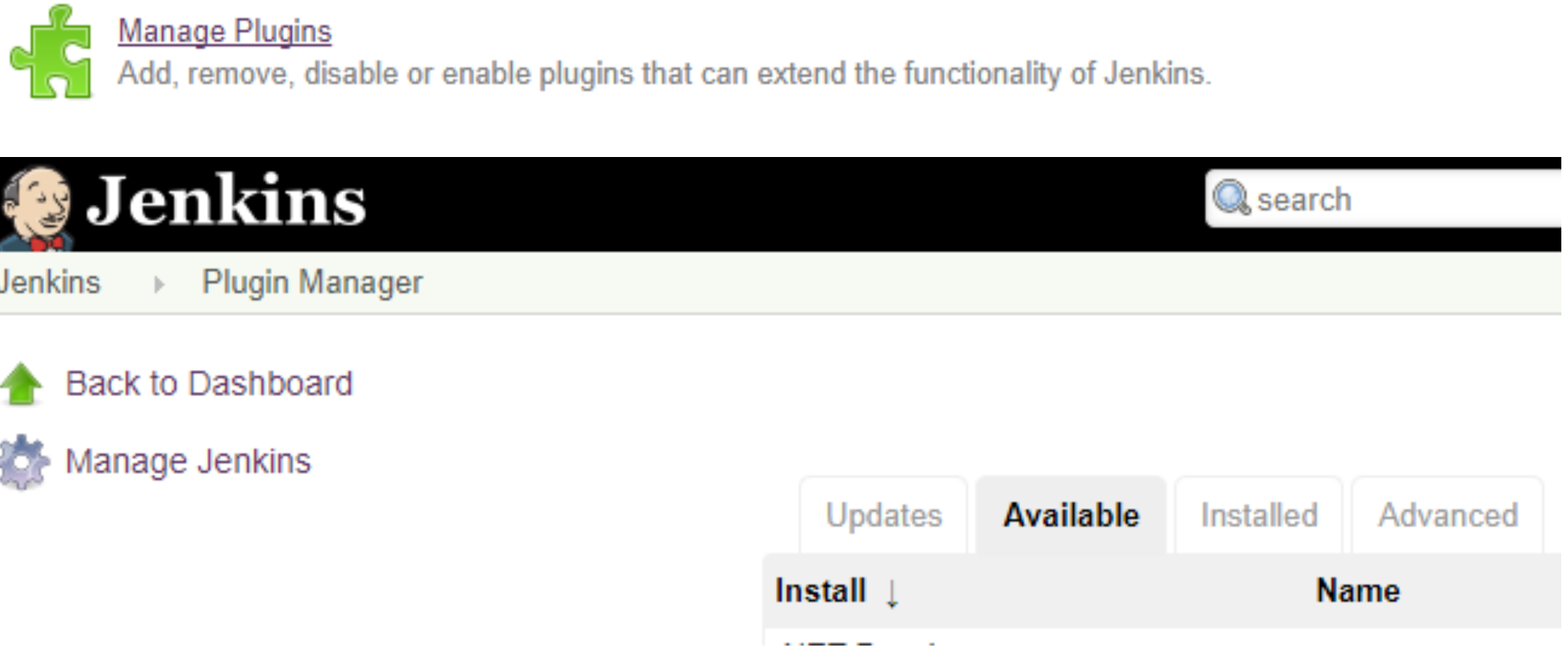
# Using The PowerShell Script

+

# Jenkins CI Configuration

# Jenkins CI Configuration

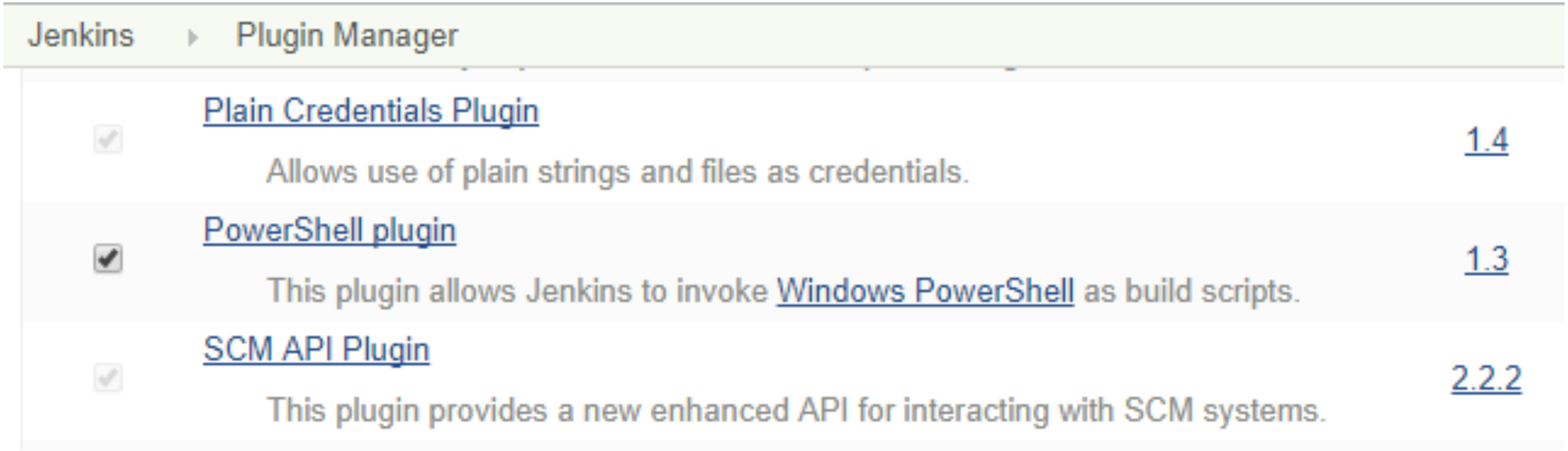
**Install PowerShell plugin (available from Jenkins gallery)**




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

# Jenkins CI Configuration

**Install PowerShell plugin**



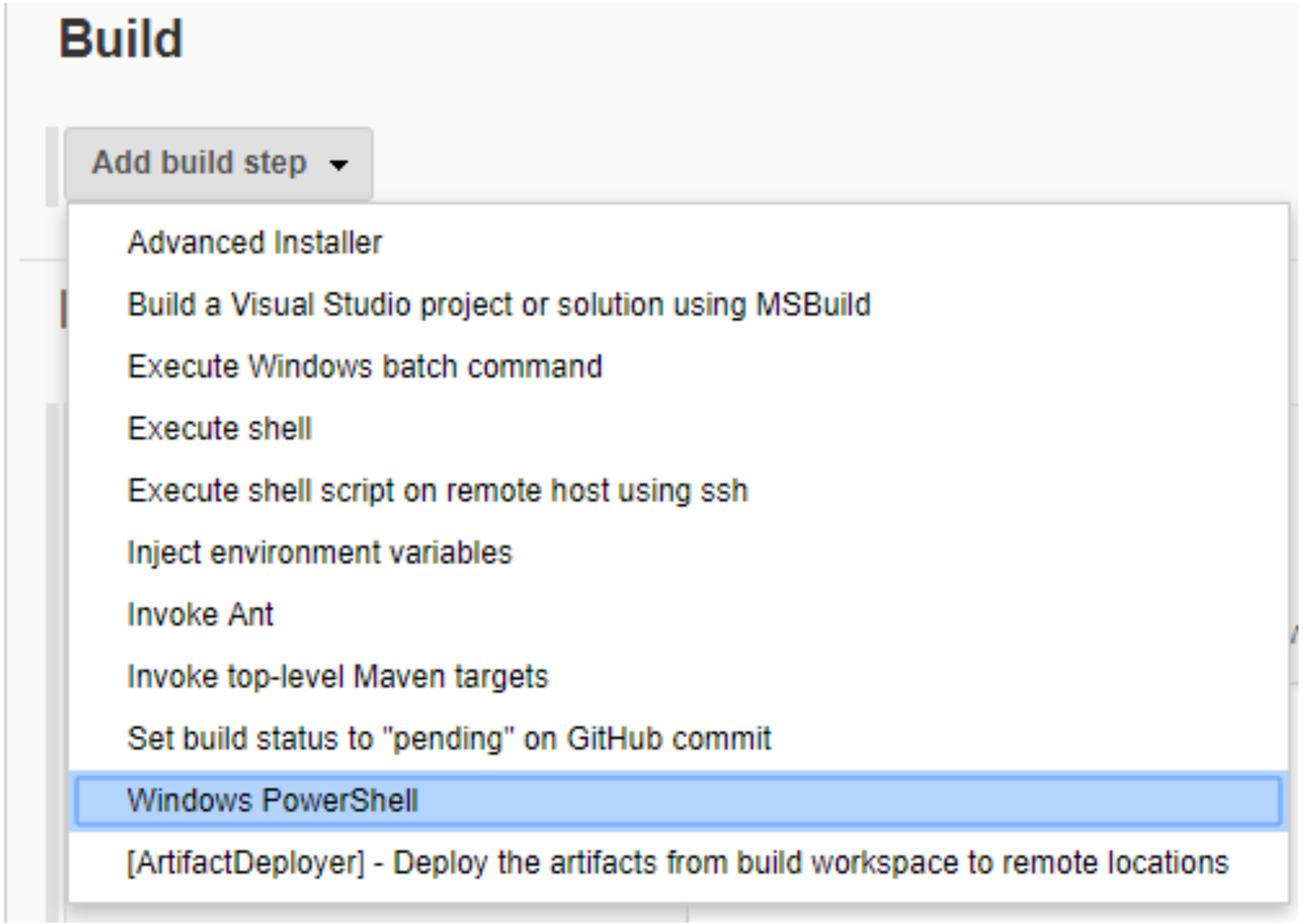

https://wiki.jenkins.io/display/JENKINS/PowerShell+Plugin

# Jenkins CI Configuration

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

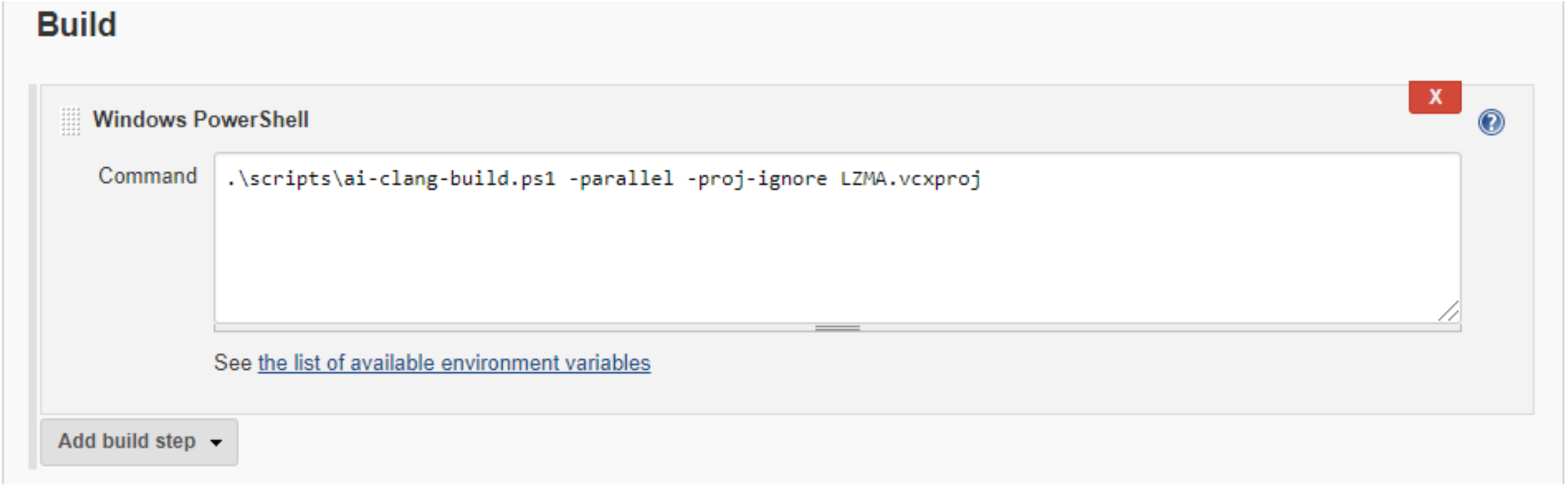
or

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



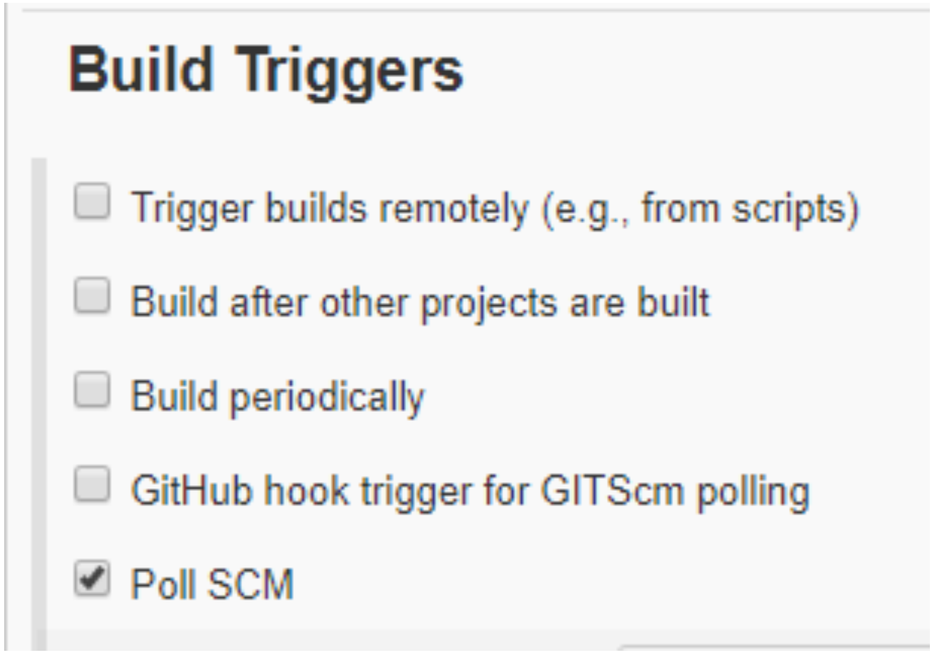
# Jenkins CI Configuration

**Reference PowerShell script from the job working directory:** `clang-build.ps1`



# Jenkins CI Configuration

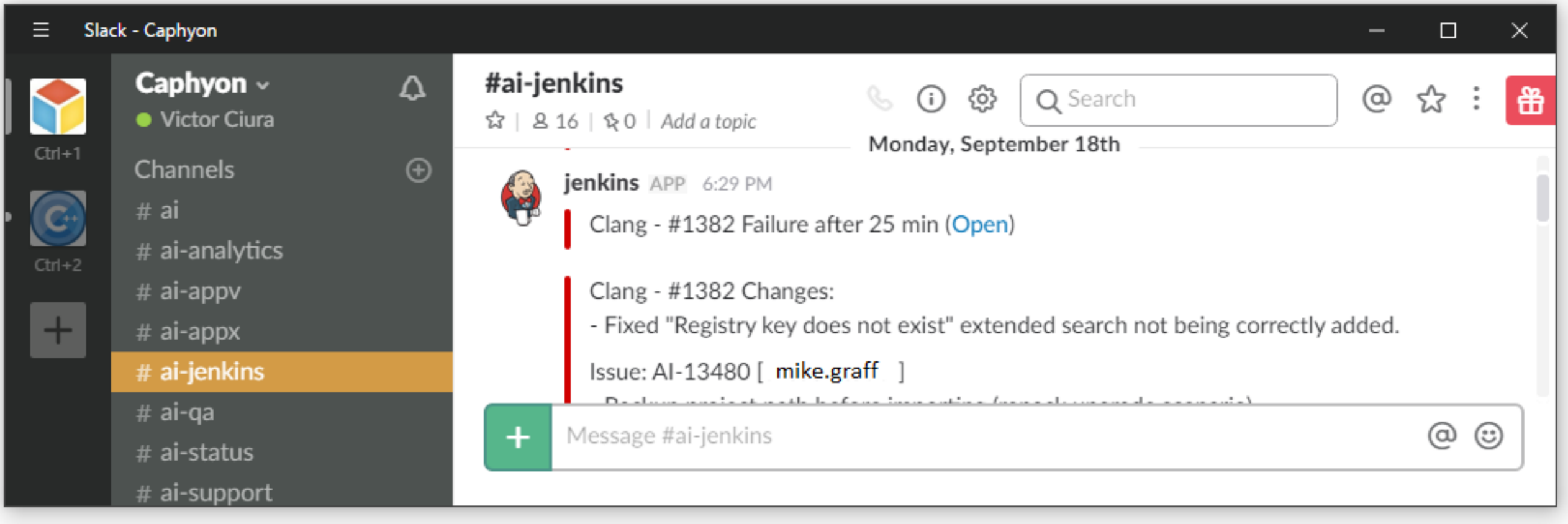
💡 **If you configured Clang build as a new Jenkins job, a good workflow is to track and build any SCM changes:**




👉

# 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

File Edit View Go Message Tools Help

Get Messages | Write | Chat | Address Book | Tag | Quick Filter | Search <Ctrl+K>

victor
- Inbox
- Jenkins
- Local Folders

Unread | Starred | Contact | Tags | Attachment | Filter these messages <Ctrl+Shift+K>

| 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 | Reply | Reply List | Forward | Archive | Junk | Delete | More

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

To

**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

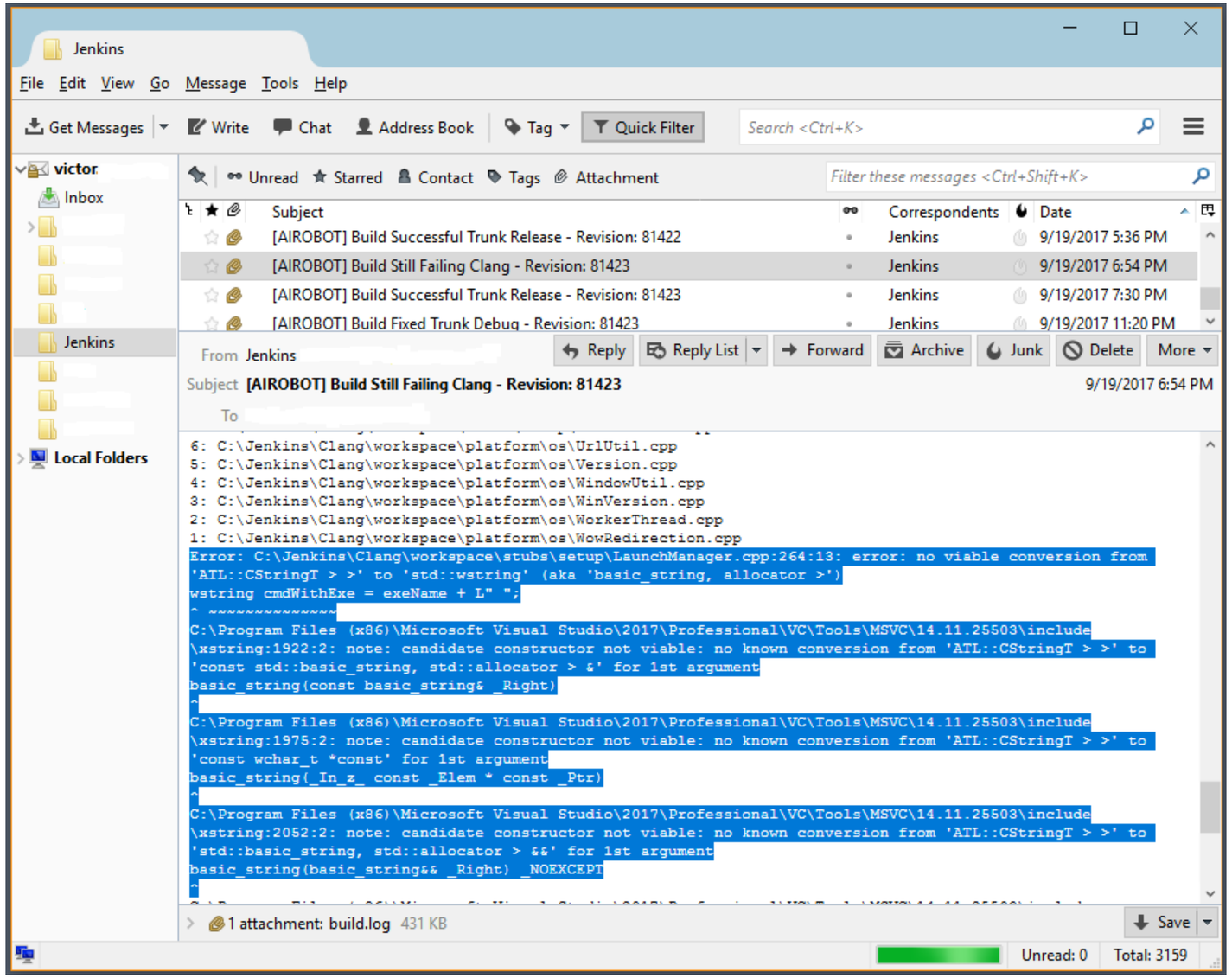
1 attachment: build.log 431 KB | Save

Unread: 0 | Total: 3159

# Jenkins CI Workflow

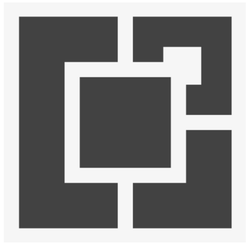
🔥 **When Clang build is broken...**

**Team devs email alert ➡**



# What About Developer Workflow?

+

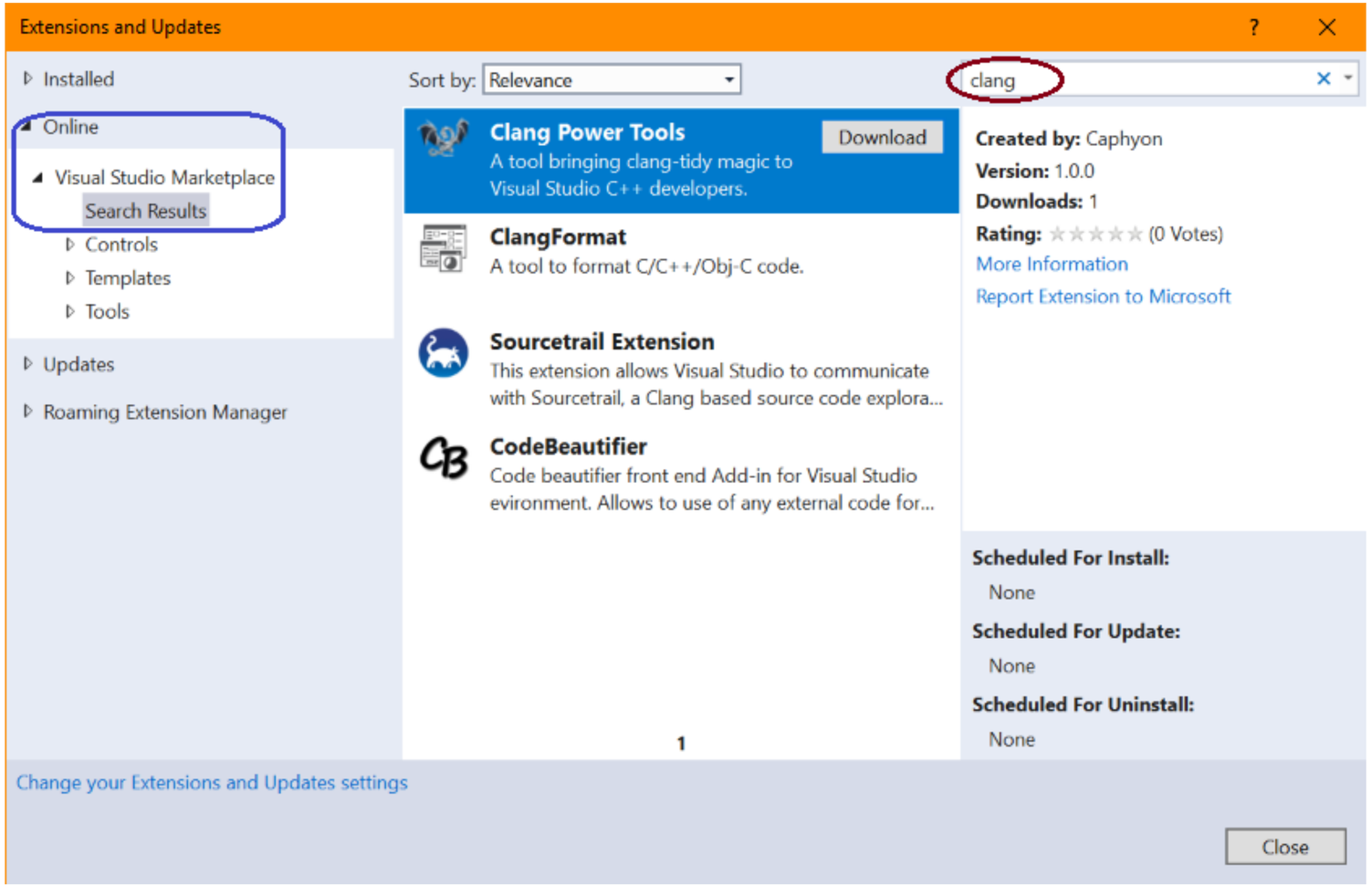


# Install The "Clang Power Tools" Visual Studio Extension

**[Tools]**

⬇

**Extensions and updates**




Requires **LLVM for Windows** to be installed.

http://releases.llvm.org/7.0.1/LLVM-7.0.1-win64.exe

# Configure The "Clang Power Tools" Visual Studio Extension

[Tools]

⬇

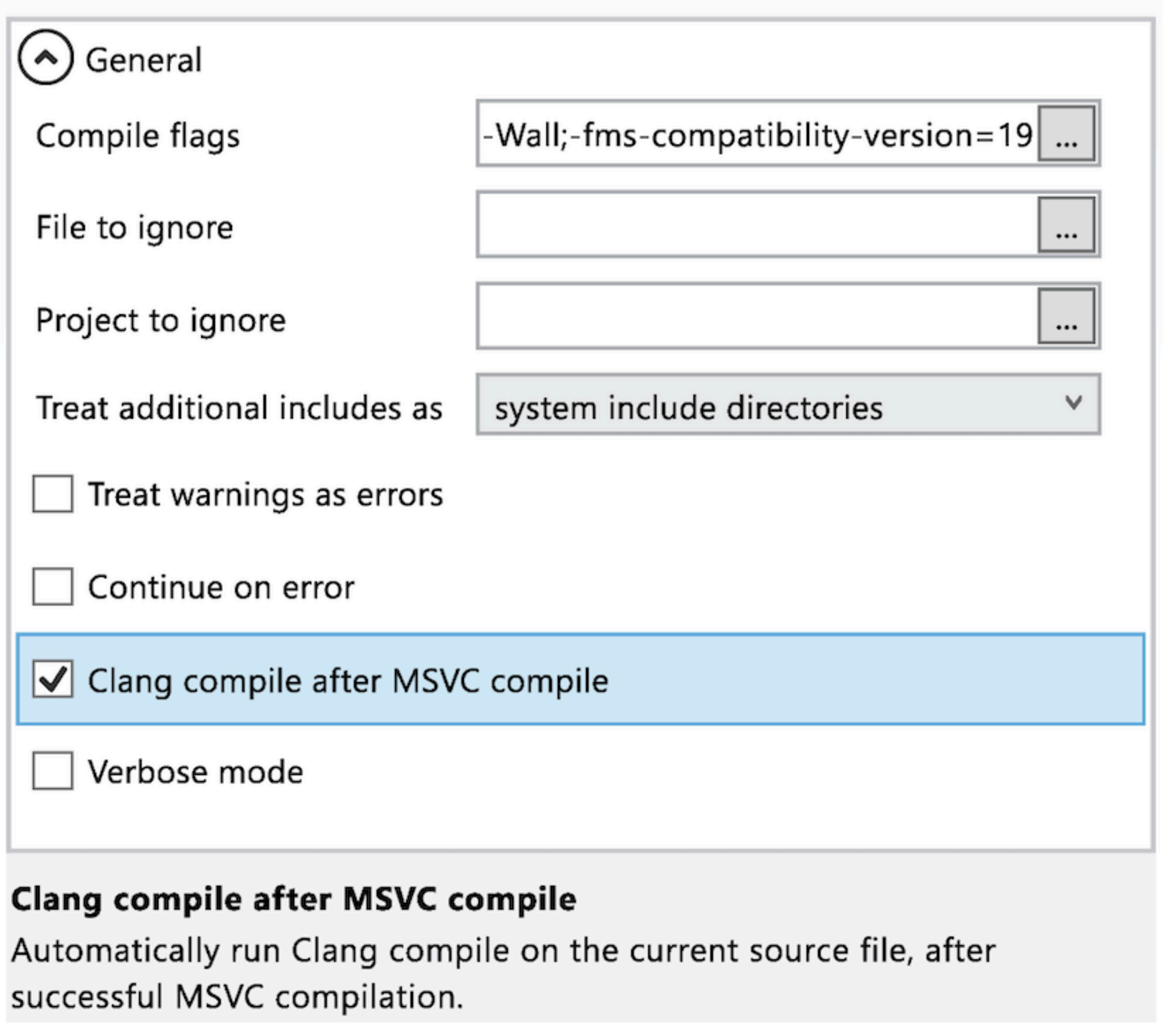
⚙ Options...

⬇

Clang Power Tools

⬇

General




⬅ Compilation settings

# Configure The "Clang Power Tools" Visual Studio Extension

[Tools]

⬇

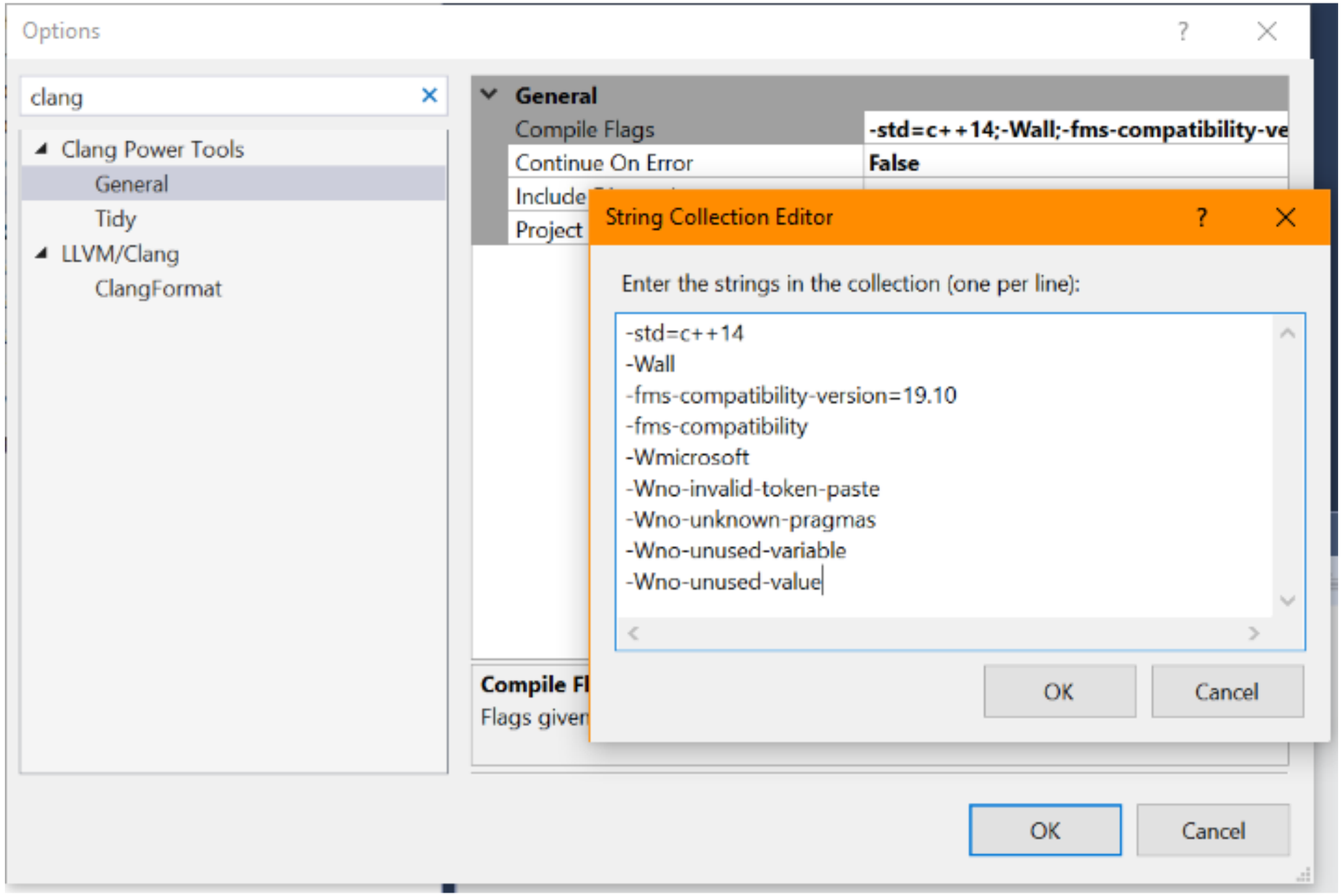
Options...

⬇

Clang Power Tools

⬇

General




⬅ clang++ flags

# Configure The "Clang Power Tools" Visual Studio Extension

[Tools]

⬇

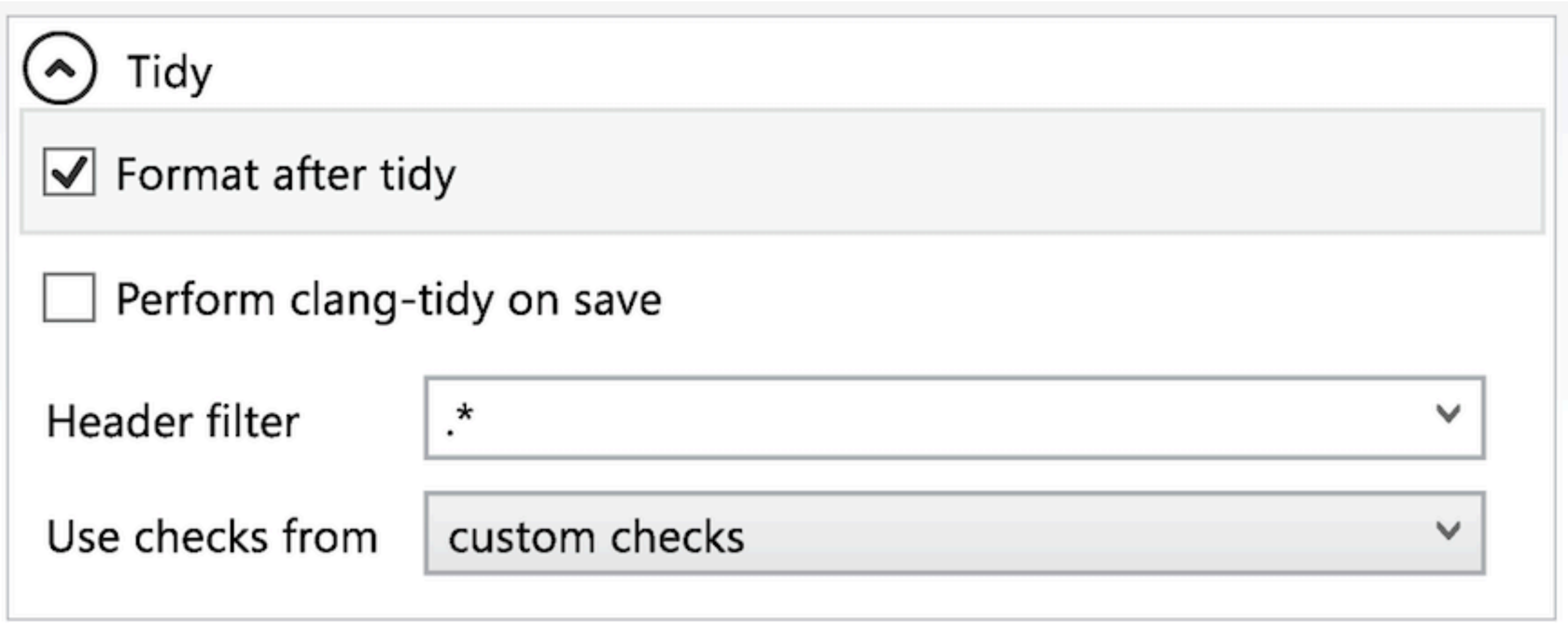
Options...

⬇

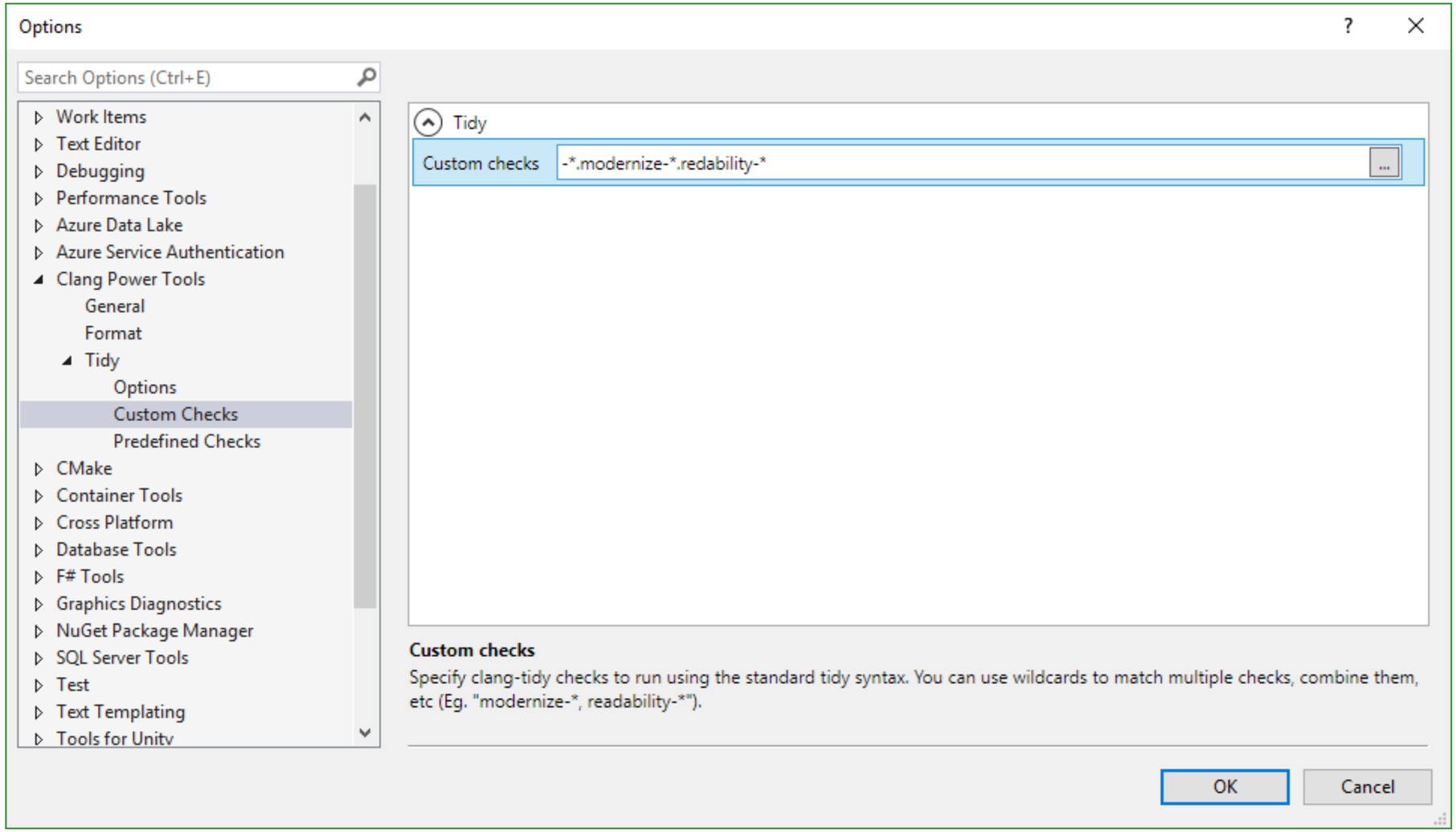
Clang Power Tools

⬇

Tidy

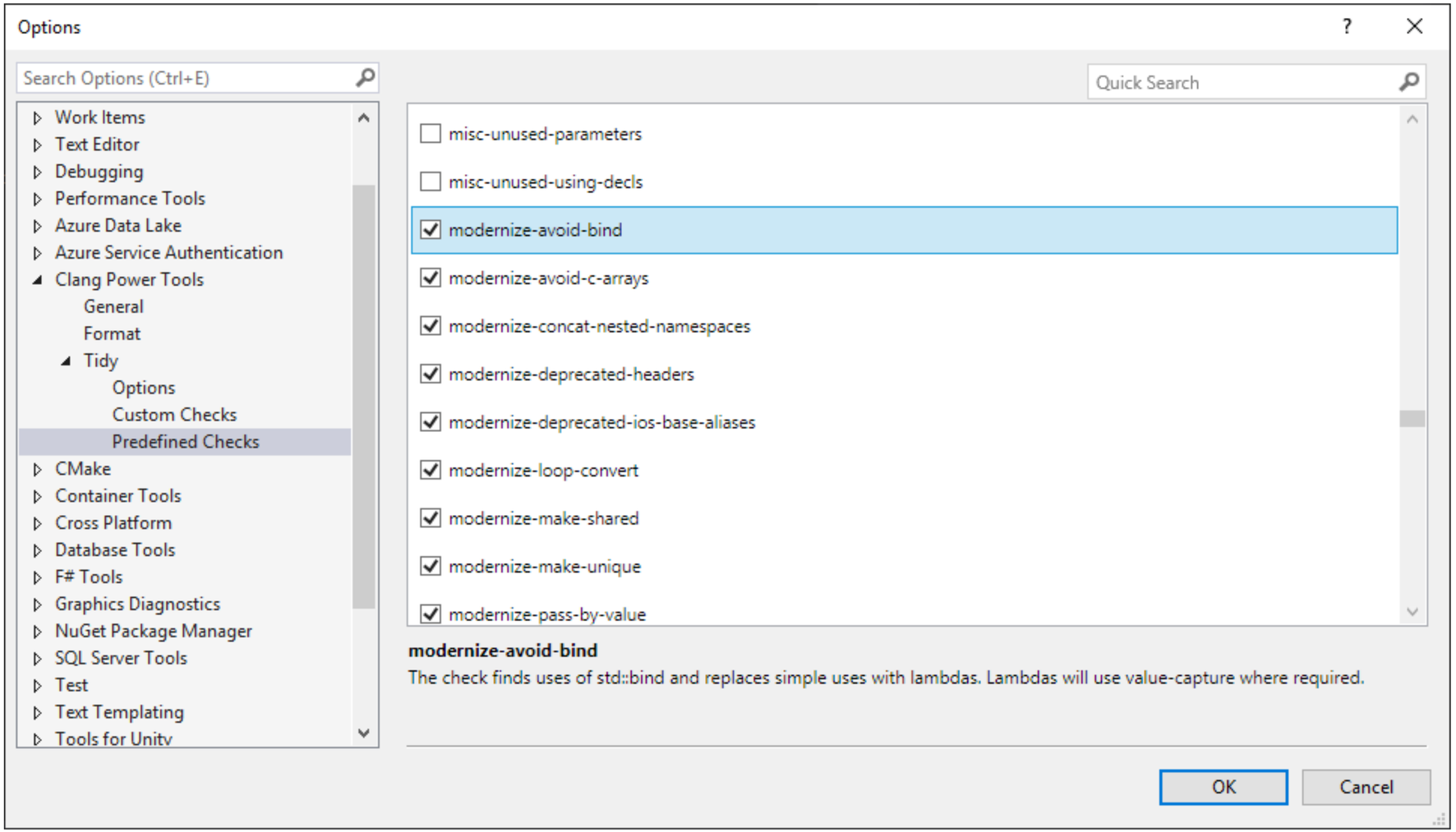


# Configure The "Clang Power Tools" Visual Studio Extension



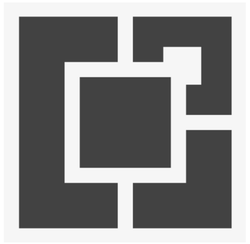

⬅ wildcard match

# Configure The "Clang Power Tools" Visual Studio Extension

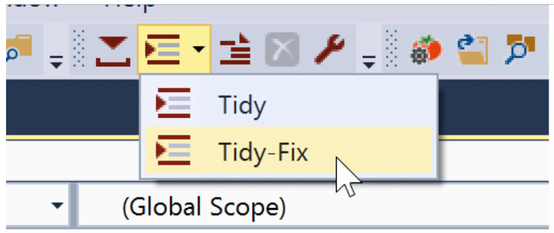



⬅ clang-tidy checks

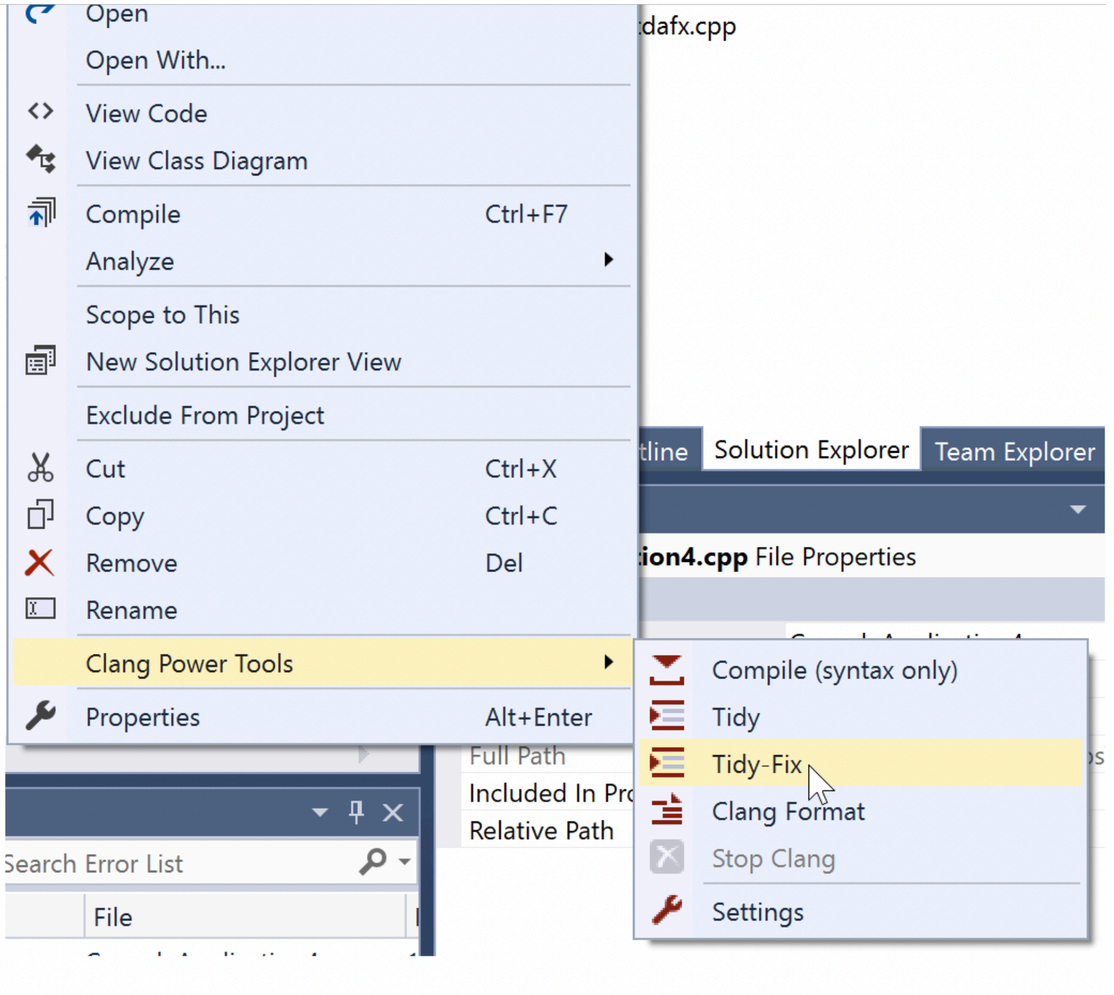
⬅ inline documentation

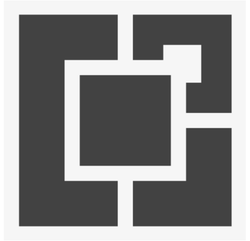


# Using The "Clang Power Tools" Visual Studio Extension



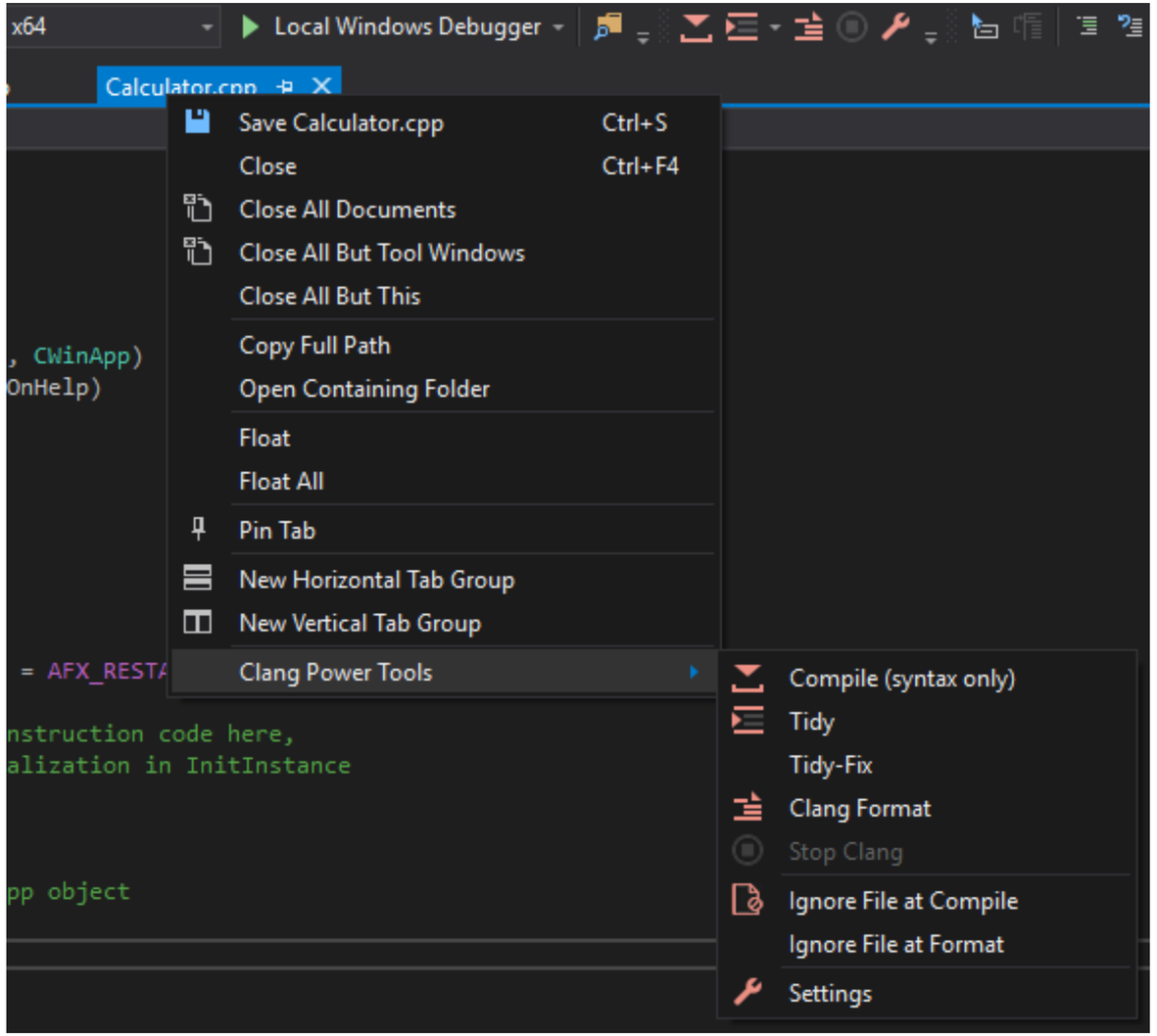

**Run Clang Power Tools on a whole *project* or *solution***

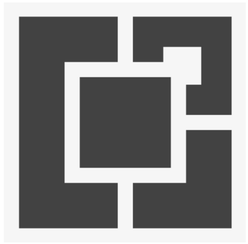




# Using The "Clang Power Tools" Visual Studio Extension

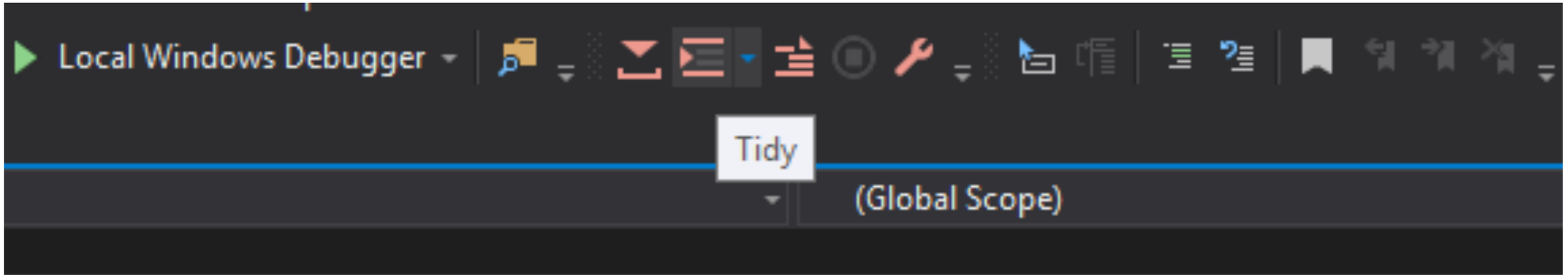
**Options on an open source file (tab) ➡**

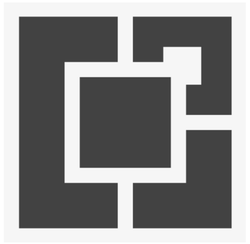




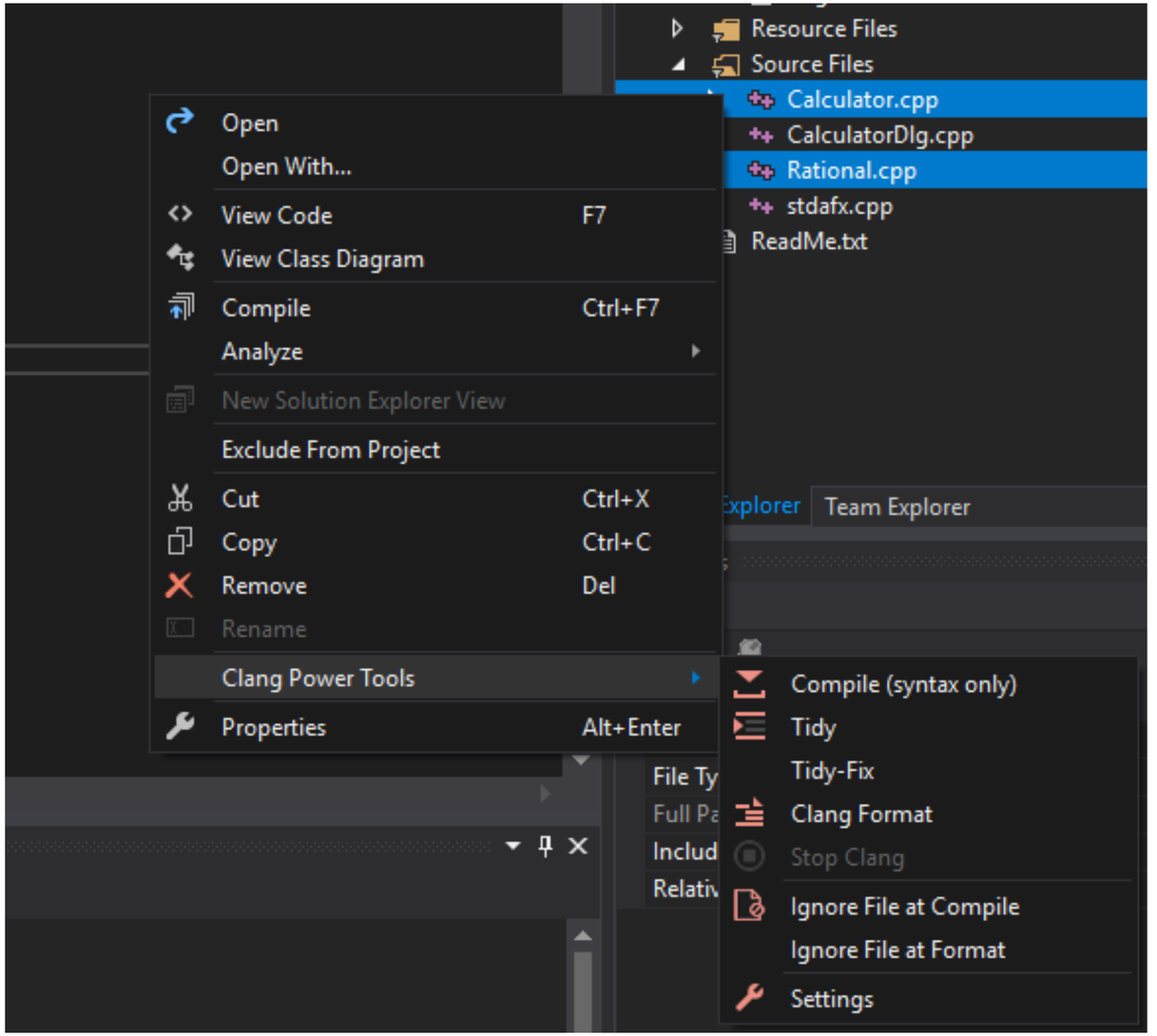
# Using The "Clang Power Tools" Visual Studio Extension

Handy Toolbar



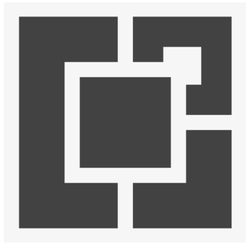


# Using The "Clang Power Tools" Visual Studio Extension




← 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)
```

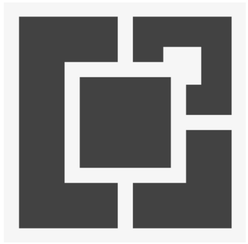
100 %

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]`

# Why Do I Care ?

15 year old code base under active development
3 million lines of C++ code
a few brave nerds…

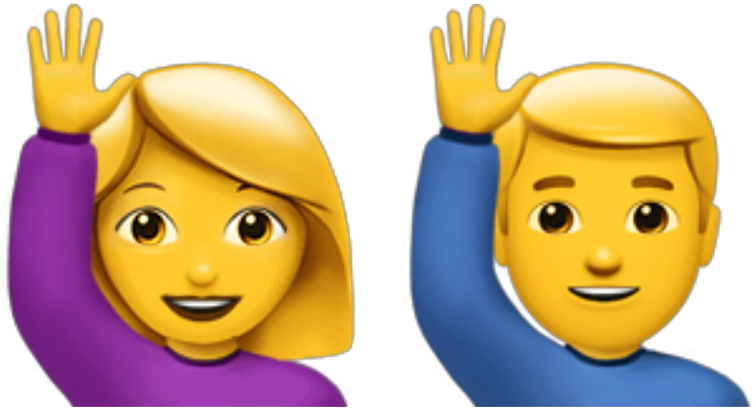
or

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

## *Mandatory Slide*

**Gauging the audience...**

C++98/03 C++11 C++14 C++17



# Part II
# Massaging The Code

# Why do we need this ?

**ISO C++ standard conformance**

**Finding bugs**

# ISO C++ standard conformance

**MSVC** `/permissive-`

**Problem:  older Windows SDKs**

https://docs.microsoft.com/en-us/cpp/build/reference/permissive-standards-conformance?view=vs-2017

# ISO C++ standard conformance

Latest **MSVC** STL

**Compiles/requires with Clang 7**

https://docs.microsoft.com/en-us/cpp/build/reference/permissive-standards-conformance?view=vs-2017

# Goals

- It all started with **clang-format**
- 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 and Clang
- 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)
- Perform large scale **refactorings** on our code with clang-tidy:
  `modernize-*, readability-*`
- Run **static analysis** on our code base to find subtle latent bugs
- Switch to the new MSVC compiler: `/permissive-`

# 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 needed**

# Fixes, fixes, fixes...

```
Error: destructor called on non-final 'InternalMessageGenerator' that has virtual
functions but non-virtual destructor [-Werror,-Wdelete-non-virtual-dtor]
                _Getptr()->~_Ty();
                ^


MessageCenter.cpp:49:29: note: in instantiation of function template
specialization 'std::make_shared<InternalMessageGenerator>' requested here
    mInternalMsgGenerator = make_shared<InternalMessageGenerator>(...);
                            ^
C:\Program Files (x86)\Microsoft Visual
Studio\2017\Professional\VC\Tools\MSVC\14.14.26428\include\memory:1783:15: note:
qualify call to silence this warning
                _Getptr()->~_Ty();
```

**Frequent offender**

# Fixes, fixes, fixes...

```
Error: delete called on 'NetFirewall::INetFirewallMgr' that is abstract but has
non-virtual destructor [-Werror,-Wdelete-non-virtual-dtor]
                delete _Ptr;
                ^
C:\Program Files (x86)\Microsoft Visual
Studio\2017\Professional\VC\Tools\MSVC\14.14.26428\include\memory:2267:4: note:
in instantiation of member function
'std::default_delete<NetFirewall::INetFirewallMgr>::operator()' requested here
                        this->get_deleter()(get());
                        ^
NetFirewallMgrFactory.cpp:21:44: note: in instantiation of member function
'std::unique_ptr<NetFirewall::INetFirewallMgr,
std::default_delete<NetFirewall::INetFirewallMgr> >::~unique_ptr' requested here
  unique_ptr<NetFirewall::INetFirewallMgr> fwMgr;
```

**Frequent offender**

# Fixes, fixes, fixes...

```
FormattedLexer.cpp(2982): error [-Werror,-Wenum-compare-switch]:

comparison of two values with different enumeration types in switch statement
'FormattedLexer::CharType' and 'FormattedLexer::CharSubType'
  case REGULAR:
       ^~~~~~~
```

**Frequent offender**

# Iterative Conformance

```
-fno-delayed-template-parsing
-Werror=microsoft
-Werror=typename-missing
-Wno-xyz-warn
      eg. -Wno-microsoft-sealed
```

# Iterative Conformance

## The long road to MSVC `/permissive-`

**Problems:**

**fix issues in your code**

**deal with older Windows SDKs**

(eg. targeting Win7, WinXP)

# MSVC /permissive-

## Fix issues in your code

**Tips:**

- **lots of issues related to TPL two-phase lookup**
- **include headers required by your template inline code**
- **fix issues related to dependent types**
- **do not assume STL headers include each other; be explicit**

# MSVC `/permissive-`

## Deal with older Windows SDKs

(eg. targeting Win7, WinXP)


**Tips:**

- **forward declare `struct IUnknown` before including Win SDK headers**

  (related to TPL two-phase lookup)

# MSVC /permissive-

## Deal with older Windows SDKs

(eg. targeting Win7, WinXP)

**Tips:**

- **use `/Zc:strictStrings-` for SDK headers (your PCH)**

Off by default; the `/permissive-` implicitly sets this option.

When set, the compiler requires strict const-qualification conformance for pointers initialized by using string literals.

https://docs.microsoft.com/en-us/cpp/build/reference/zc-strictstrings-disable-string-literal-type-conversion?view=vs-2017

# cpt.config

```
<cpt-config>
  <clang-flags>   "-Werror"
                , "-Wall"
                , "-fms-compatibility-version=19.10"
                , "-Wmicrosoft"
                , "-Wno-invalid-token-paste"
                , "-Wno-unknown-pragmas"
                , "-Wno-unused-value"
  </clang-flags>
  <header-filter>'.*'</header-filter>
  <parallel/>
  <vs-sku>'Professional'</vs-sku>
  <file-ignore>   'htmlayoutsdk\\include\\behaviors'
                , 'vsphere\\vim25\\core'
  </file-ignore>
  <proj-ignore>   'SciLexer'
                , 'tools\\msix-psf'
  </proj-ignore>
</cpt-config>
```

# clang-tidy

clang-tidy

Extra Clang Tools 7 documentation

CLANG-TIDY - CLANG-TIDY CHECKS

## Clang-Tidy Checks

- abseil-string-find-startswith
  - Options
- android-cloexec-accept
- android-cloexec-accept4
- android-cloexec-creat
- android-cloexec-dup
- android-cloexec-epoll-create
- android-cloexec-epoll-create1
- android-cloexec-fopen
- android-cloexec-inotify-init
- android-cloexec-inotify-init1
- android-cloexec-memfd-create
- android-cloexec-open
- android-cloexec-socket
- android-comparison-in-temp-failure-retry
- boost-use-to-string
- bugprone-argument-comment
  - Options
- bugprone-assert-side-effect
  - Options
- bugprone-bool-pointer-implicit-conversion
- bugprone-copy-constructor-init

**over 250 checks**

https://clang.llvm.org/extra/clang-tidy/checks/list.html

# 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 *);
```

# clang-tidy

🔥 **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>
```

# Beyond clang-tidy

- 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...

# [-Wunused-private-field]

Remove unused class private fields:

- references
- pointers
- PODs

**Watch out for orphan method *declarations* in classes**

# 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)
- more custom code transformations (project-specific)

# Part III
# Take Control

## More `clang-tidy` checks

https://github.com/llvm-mirror/clang-tools-extra/tree/master/clang-tidy

Checks are organized in **modules**, which can be linked into clang-tidy
with minimal or no code changes in clang-tidy

Checks can plug into the analysis on the **preprocessor** level using **PPCallbacks**
or on the AST level using **AST Matchers**

Checks can **report** issues in a similar way to how Clang diagnostics work.
A **fix-it** hint can be attached to a diagnostic message

# Tools

- `add_new_check.py` - automate the process of adding a new check

  (creates check, update the CMake file and creates test)
- `rename_check.py` - renames an existing check
- `clang-query` - interactive prototyping of AST matchers and exploration of the Clang AST
- `clang-check -ast-dump` - provides a convenient way to dump the AST

```
clang-tidy/                             # Clang-tidy core.
|-- ClangTidy.h                         # Interfaces for users and checks.
|-- ClangTidyModule.h                   # Interface for clang-tidy modules.
|-- ClangTidyModuleRegistry.h           # Interface for registering of modules.
    ...
|-- mymod/                              # My Own clang-tidy module.
|-+
  |-- MyModTidyModule.cpp
  |-- MyModTidyModule.h
        ...

|-- tool/                               # Sources of the clang-tidy binary.
        ...
test/clang-tidy/                        # Integration tests.
    ...
unittests/clang-tidy/                   # Unit tests.
|-- ClangTidyTest.h
|-- MyModModuleTest.cpp
```

# Setup

```
# download the sources
git clone http://llvm.org/git/llvm.git
cd llvm/tools/
git clone http://llvm.org/git/clang.git
cd clang/tools/
git clone http://llvm.org/git/clang-tools-extra.git extra

# build everything
cd ../../../
mkdir build && cd build/
cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo ..
make check-clang-tools
```

# Init

We will add our check to the [**readability**] category/module

```
add_new_check.py readability pretty-func
```

This will create:

```
/readability/PrettyFuncCheck.h
/readability/PrettyFuncCheck.cpp
```

=> include it in:

```
/readability/ReadabilityTidyModule.cpp
```

```
#include "../ClangTidy.h"

namespace clang {
namespace tidy {
namespace readability {

class PrettyFuncCheck : public ClangTidyCheck
{
public:
  PrettyFuncCheck(StringRef Name, ClangTidyContext * Context)
    : ClangTidyCheck(Name, Context) {}

  void registerMatchers(ast_matchers::MatchFinder * Finder) override;
  void check(const ast_matchers::MatchFinder::MatchResult & Result) override;
};

} // namespace readability
} // namespace tidy
} // namespace clang
```

# `ClangTidyCheck`

Our check needs to operate on the AST level:

- `registerMatchers()` - register clang AST matchers to filter out interesting source locations

- `check()` - provide a function which is called by the Clang whenever a match was found;

  we can perform further actions here (eg. emit diagnostics)

If we wanted to analyze code on the **preprocessor** level

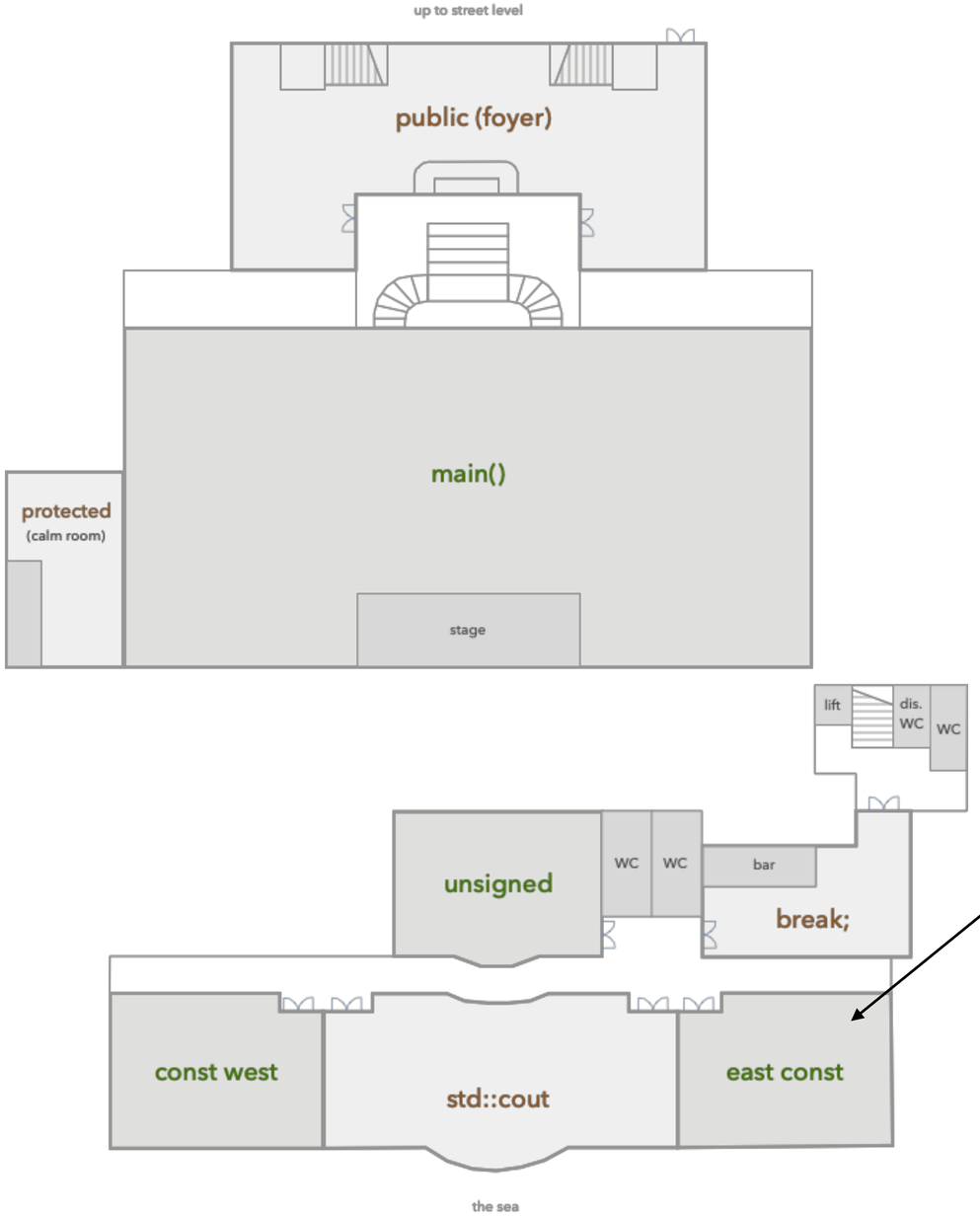
=> override `registerPPCallbacks()` method

# ClangTidyCheck

```
using namespace ast_matchers;


void PrettyFuncCheck::registerMatchers(MatchFinder * Finder)
{
  Finder->addMatcher(functionDecl().bind("needle"), this);
}
```

```
using namespace ast_matchers;


void PrettyFuncCheck::check(const MatchFinder::MatchResult & Result)
{
  const auto * MatchedDecl = Result.Nodes.getNodeAs<FunctionDecl>("needle");

  if (MatchedDecl->getName().startswith_lower("get_"))
  {
    diag(MatchedDecl->getLocation(), "function %0 needs your attention")
        << MatchedDecl
        << FixItHint::CreateInsertion(MatchedDecl->getLocation(), "Get");
  }
}
```

# Folkestone Leas Cliff Hall




you are here

```
using namespace ast_matchers;


void PrettyFuncCheck::check(MatchFinder::MatchResult const & Result)
{
  auto const * MatchedDecl = Result.Nodes.getNodeAs<FunctionDecl>("needle");

  if (MatchedDecl->getName().startswith_lower("get_"))
  {
    diag(MatchedDecl->getLocation(), "function %0 needs your attention")
        << MatchedDecl
        << FixItHint::CreateInsertion(MatchedDecl->getLocation(), "Get");
  }
}
```

# east const

# Test it...

```
clang-tidy -checks='-*,readability-pretty-func' some/file.cpp
```

# Check Options

If a check needs configuration **options**, it can access check-specific options using:

```
Options.get<Type>("SomeOption", DefaultValue)
```

# Check Options

```
class PrettyFuncCheck : public ClangTidyCheck
{
  const unsigned    Tolerance;  // option 1
  const std::string TargetFunc; // option 2
public:

  PrettyFuncCheck(StringRef Name, ClangTidyContext * Context)
    : ClangTidyCheck(Name, Context),
      Tolerance (Options.get("Tolerance", 0)),
      TargetFunc(Options.get("TargetFunc", "get_")) {}

  void storeOptions(ClangTidyOptions::OptionMap & Opts) override
  {
    Options.store(Opts, "Tolerance",  Tolerance);
    Options.store(Opts, "TargetFunc", TargetFunc);
  }
```

# .clang-tidy

```
CheckOptions:
  - key: readability-pretty-func.Tolerance          a1
    value: 123                                      b1
  - key: readability-pretty-func.TargetFunc         a2
    value: 'get_'                                   b2
```

```
clang-tidy
  -config="{CheckOptions: [{key: a1, value: b1}, {key: a2, value: b2}]}" ...
```

# Testing Our Check

Write some test units...

```
% ninja check-clang-tools
```

**or**

```
% make check-clang-tools
```

`check_clang_tidy.py`

# Debug AST Matcher

```
% clang-check -ast-dump my_source.cpp --
```

```
TranslationUnitDecl 0x2b3cd20 <<invalid sloc>> <invalid sloc>
|-TypedefDecl 0x2b3d258 <<invalid sloc>> <invalid sloc> implicit __int128_t '__int128'
|-TypedefDecl 0x2b3d2b8 <<invalid sloc>> <invalid sloc> implicit __uint128_t 'unsigned __int128'
|-TypedefDecl 0x2b3d698 <<invalid sloc>> <invalid sloc> implicit __builtin_va_list '__va_list_tag [1]'
|-CXXRecordDecl 0x2b3d6e8 </test.cpp:1:1, line:3:1> line:1:8 referenced struct A definition
| |-CXXRecordDecl 0x2b3d800 <col:1, col:8> col:8 implicit struct A
| `-CXXMethodDecl 0x2b3d8e0 <line:2:9, col:19> col:14 f 'void (void)'
|   `-CompoundStmt 0x2b3d9b8 <col:18, col:19>
`-CXXRecordDecl 0x2b3d9d0 <line:5:1, line:7:1> line:5:8 struct B definition
  |-public 'struct A'
  |-CXXRecordDecl 0x2b85050 <col:1, col:8> col:8 implicit struct B
  |-CXXMethodDecl 0x2b85100 <line:6:3, col:21> col:16 f 'void (void)' virtual
  | `-CompoundStmt 0x2b854f8 <col:20, col:21>
...
```

https://clang.llvm.org/docs/LibASTMatchersReference.html

# More custom tidy checks...

## There's never too many sanitizers 🤓

- `ParmeSan`
  Catch cheesy comments.

- `BipartiSan`
  Find code that uses two different containers in a complimentary way.

- `ArtiSan`
  Find code that took the writer a very long time to do and
  can be replaced with a common well tested library.

https://twitter.com/olafurw/status/1085544102870044674?s=21

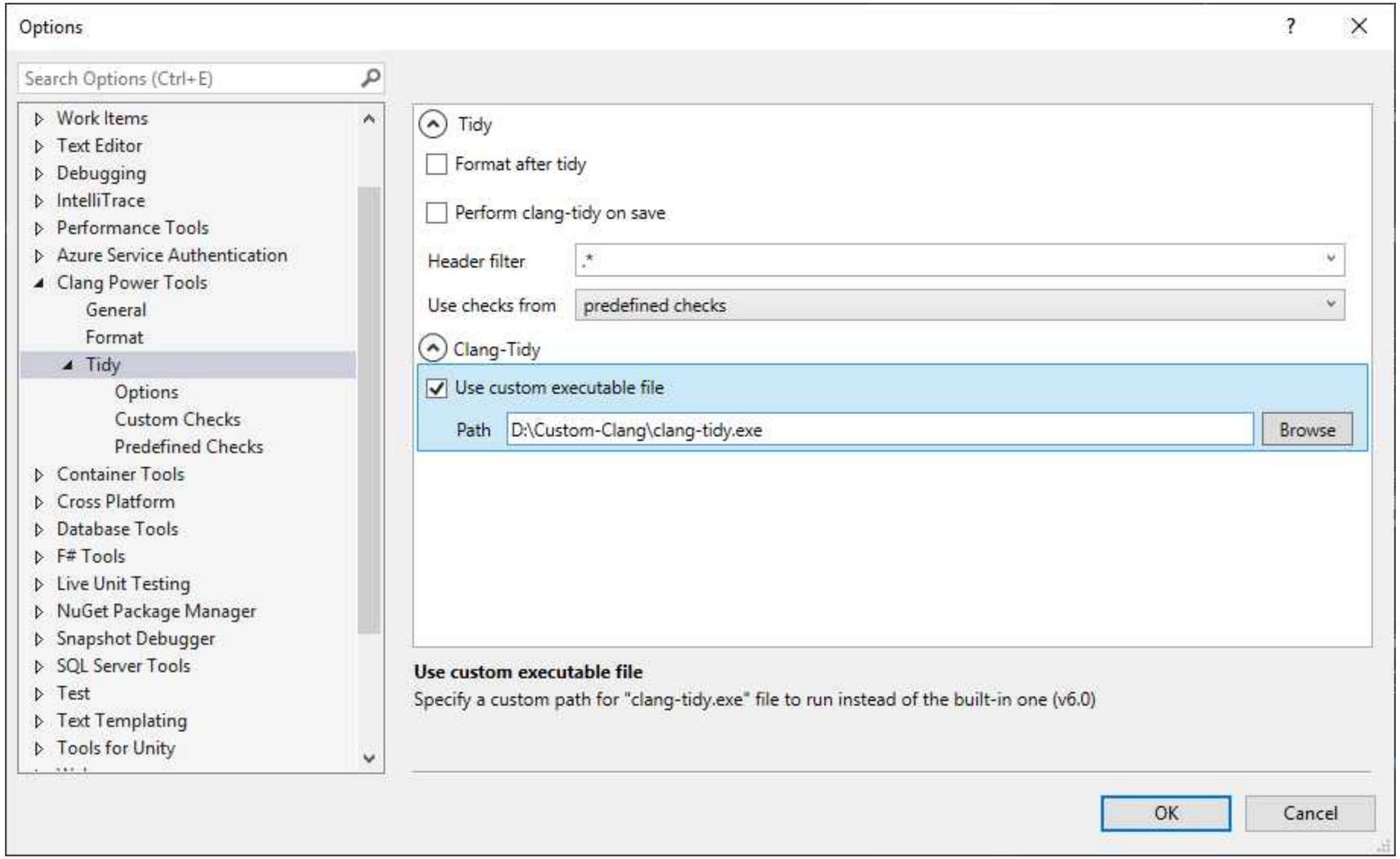
**Ólafur Waage**
@olafurw

# Write custom checks for your needs (project specific)

## Run them regularly !

# Configure The "Clang Power Tools" Visual Studio Extension

NEW




⬅ your custom clang-tidy

## 2018 Theme Of The Year For Me:

STRINGS


# Enough `string_view` to hang ourselves

**CppCon 2018**

**https://www.youtube.com/watch?v=xwP4YCP_0q0**


# These Aren't the COM Objects You're Looking For

**CppCon 2018**

Part 1 of N: Strings

https://www.youtube.com/watch?v=T_1zutIBHs0

# String related checks

clang-tidy

- abseil-string-find-startswith
- boost-use-to-string
- bugprone-string-constructor
- bugprone-string-integer-assignment
- bugprone-string-literal-with-embedded-nul
- bugprone-suspicious-string-compare
- modernize-raw-string-literal
- performance-faster-string-find
- performance-inefficient-string-concatenation
- readability-redundant-string-cstr
- readability-redundant-string-init
- readability-string-compare

https://clang.llvm.org/extra/clang-tidy/checks/list.html

# bugprone-dangling-handle

" Detect dangling references in value handles like `std::string_view`.

These dangling references can be a result of constructing handles from temporary values, where the temporary is destroyed soon after the handle is created.

https://clang.llvm.org/extra/clang-tidy/checks/bugprone-dangling-handle.html


# Lifetime profile v1.0

https://herbsutter.com/2018/09/20/lifetime-profile-v1-0-posted/

> A dangling `string_view`, which is important because it turns out to be ***easy*** to convert a `std::string` to a `string_view`, so that dangling is almost the default behavior.

CppCoreGuidelines

https://github.com/isocpp/CppCoreGuidelines/blob/master/docs/Lifetime.pdf

# Lifetime profile v1.0

https://herbsutter.com/2018/09/20/lifetime-profile-v1-0-posted/

```
void example_2_6_2_1()
{
  std::string_view s = "foo"s;      // A
  s[0];   // ERROR (lifetime.3): ‘s’ was invalidated when
          // temporary ‘"foo"s’ was destroyed (line A)
}
```

CppCoreGuidelines

https://github.com/isocpp/CppCoreGuidelines/blob/master/docs/Lifetime.pdf

# Lifetime profile v1.0

https://herbsutter.com/2018/09/20/lifetime-profile-v1-0-posted/

```
<source>:7:5: warning: passing a dangling pointer as argument [-Wlifetime]
    s[0];                          // ERROR (lifetime.3): ‘s’ was invalidated when
    ^
<source>:6:32: note: temporary was destroyed at the end of the full expression
    std::string_view s = "foo"s;                  // A
                               ^

1 warning generated.
Compiler returned: 0
```

clang -Wlifetime

CppCoreGuidelines

https://github.com/isocpp/CppCoreGuidelines/blob/master/docs/Lifetime.pdf

# Lifetime profile v1.0

https://herbsutter.com/2018/09/20/lifetime-profile-v1-0-posted/


CppCon 2018 https://www.youtube.com/watch?v=sjnp3P9x5jA

***Implementing the C++ Core Guidelines' Lifetime Safety Profile in Clang***

**Matthias Gehre & Gabor Horvath**

CppCoreGuidelines

https://github.com/isocpp/CppCoreGuidelines/blob/master/docs/Lifetime.pdf

# Explore Further

A new series of blog articles on **Visual C++ Team blog** by **Stephen Kelly**

***Exploring Clang Tooling, Part 0: Building Your Code with Clang***

https://blogs.msdn.microsoft.com/vcblog/2018/09/18/exploring-clang-tooling-part-0-building-your-code-with-clang/

***Exploring Clang Tooling, Part 1: Extending Clang-Tidy***

https://blogs.msdn.microsoft.com/vcblog/2018/10/19/exploring-clang-tooling-part-1-extending-clang-tidy/

***Exploring Clang Tooling, Part 2: Examining the Clang AST with clang-query***

https://blogs.msdn.microsoft.com/vcblog/2018/10/23/exploring-clang-tooling-part-2-examining-the-clang-ast-with-clang-query/

# Explore Further

A new series of blog articles on **Visual C++ Team blog** by **Stephen Kelly**

***Exploring Clang Tooling, Part 3: Rewriting Code with clang-tidy***

https://blogs.msdn.microsoft.com/vcblog/2018/11/06/exploring-clang-tooling-part-3-rewriting-code-with-clang-tidy/

***Exploring Clang Tooling: Using Build Tools with clang-tidy***

https://blogs.msdn.microsoft.com/vcblog/2018/11/27/exploring-clang-tooling-using-build-tools-with-clang-tidy/

# Explore Further

More blog articles by **Stephen Kelly**

***Future Developments in clang-query***

https://steveire.wordpress.com/2018/11/11/future-developments-in-clang-query/

***Composing AST Matchers in clang-tidy***

https://steveire.wordpress.com/2018/11/20/composing-ast-matchers-in-clang-tidy/

# Explore Further


***Refactor your codebase with Clang tooling***

**Stephen Kelly**

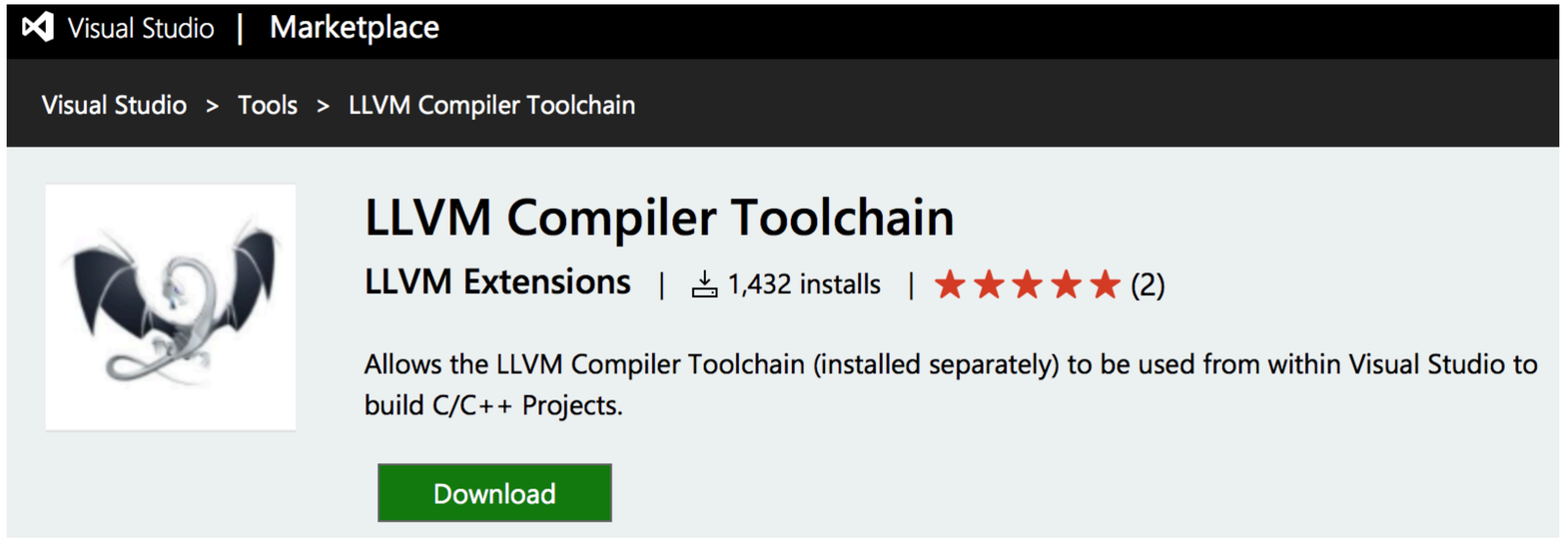
code::dive 2018

https://www.youtube.com/watch?v=_T-5pWQVxeE

https://steveire.wordpress.com/2019/01/02/refactor-with-clang-tooling-at-codedive-2018/

# Explore Further

A new Visual Studio extension is available in the **Marketplace**:




**FREE**

https://marketplace.visualstudio.com/items?itemName=LLVMExtensions.llvm-toolchain

# Better Tools in Your Clang Toolbox

www.clangpowertools.com

facebook.com/ClangPowerTools

@ClangPowerTools

# CppCast

```
auto CppCast = pod_cast<C++>("http://cppcast.com");
```

**Rob Irving** **@robwirving**

**Jason Turner** **@lefticus**

# http://cpp.chat

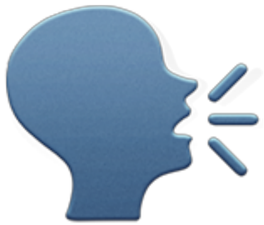
https://www.youtube.com/channel/UCsefcSZGxO9lTBqFbsV3sJg/

https://overcast.fm/itunes1378325120/cpp-chat

**Jon Kalb** @_JonKalb

**Phil Nash** @phil_nash

# Questions



@ciura_victor