Better Tools in Your Clang Toolbox

September, 2018

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”.

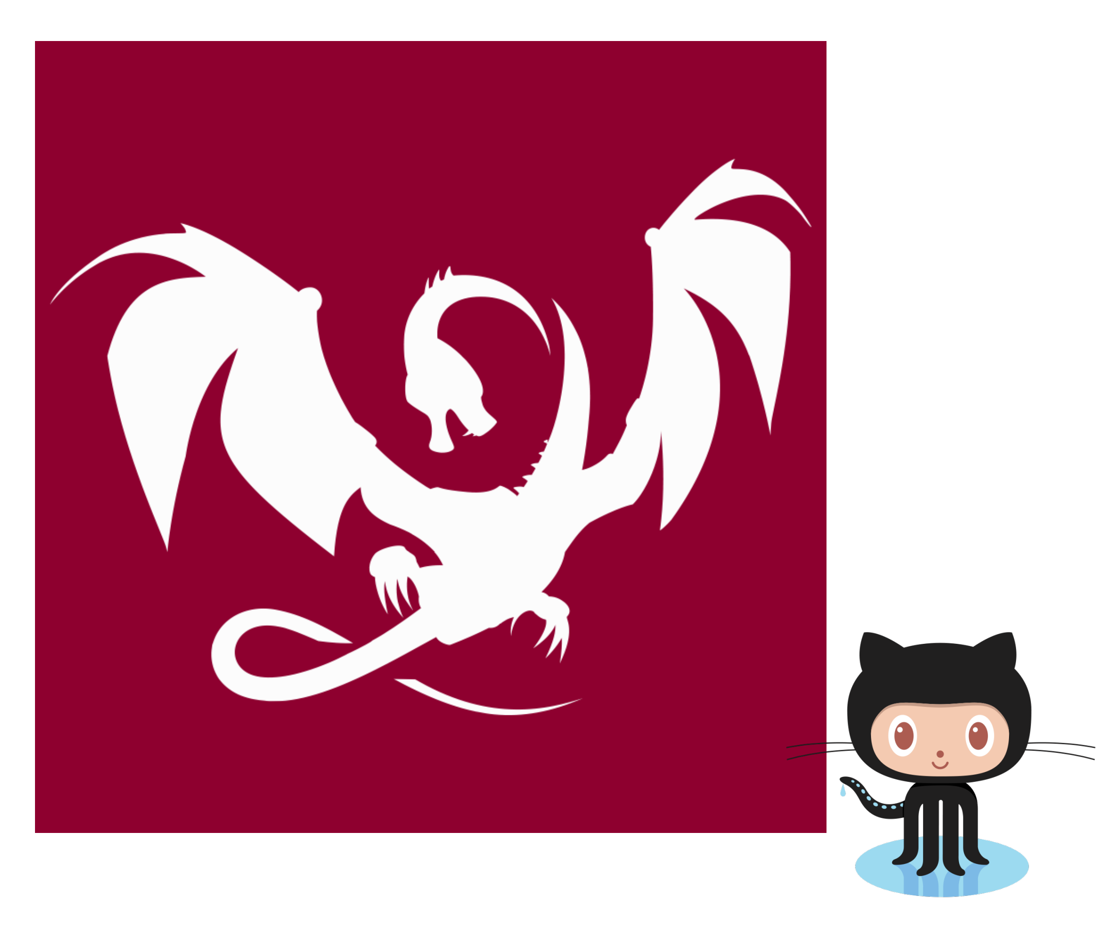
After successful usage within our team, we decided to open-source the project and make Clang Power Tools available for free in the Visual Studio Marketplace. This helped tens of thousands of developers leverage its powers to improve their projects, regardless of their compiler of choice for building 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 talk will 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.

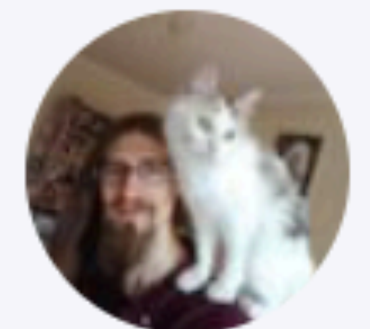
http://clangpowertools.com
https://github.com/Caphyon/clang-power-tools

Who Am I ?

Advanced Installer



Clang Power Tools



Simon Brand @TartanLlama · Jul 6

So this happened

#CppCon

PLAN TO GO TO CPPCON

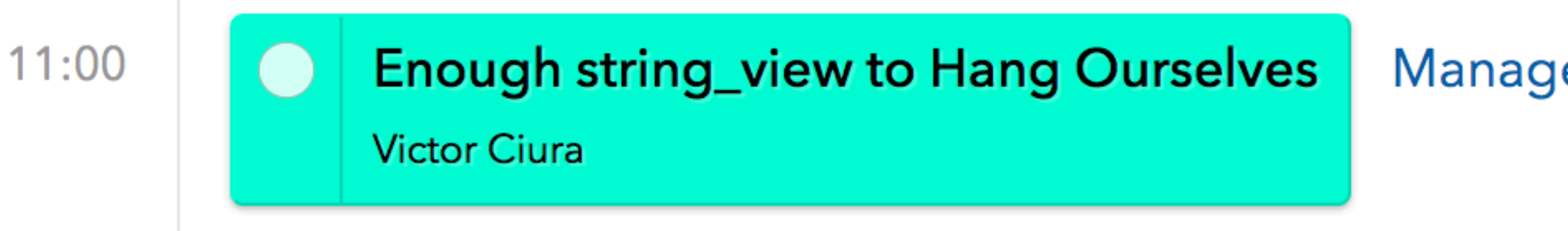
SUBMIT FOUR TALKS TO BE SAFE

ALL FOUR ARE ACCEPTED

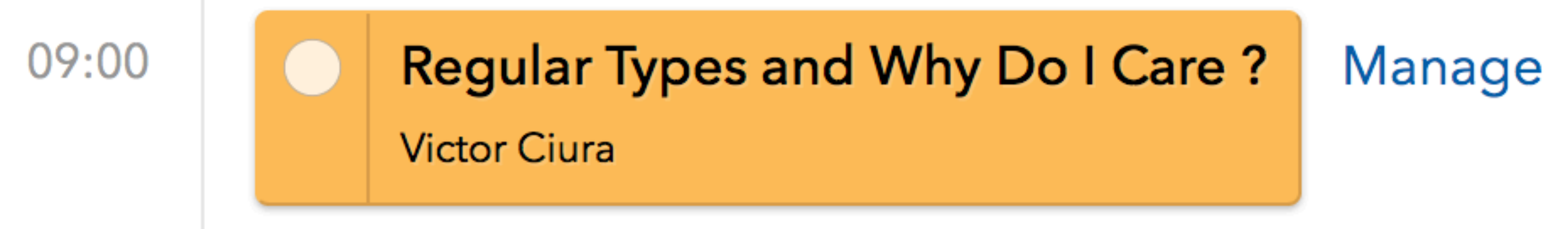
ALL FOUR ARE ACCEPTED

imgflip.com

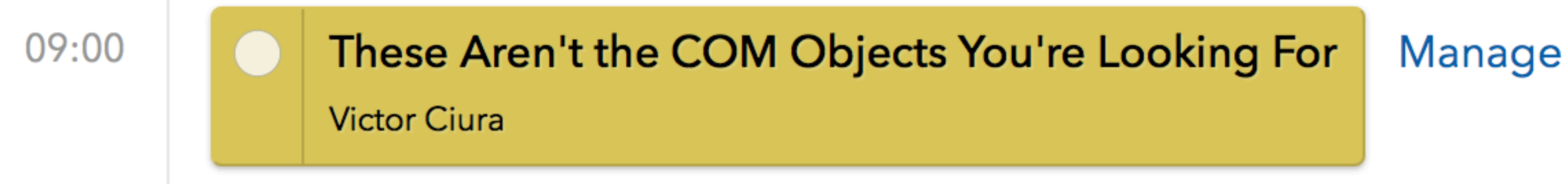
Thaaaats



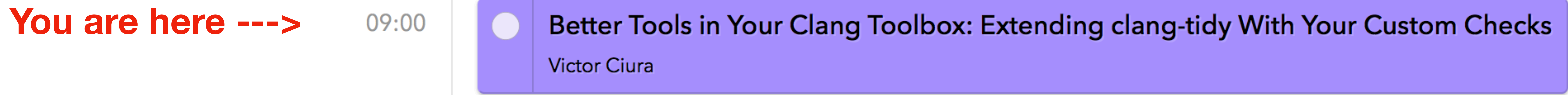
Tuesday, September 25



Wednesday, September 26



Thursday, September 27

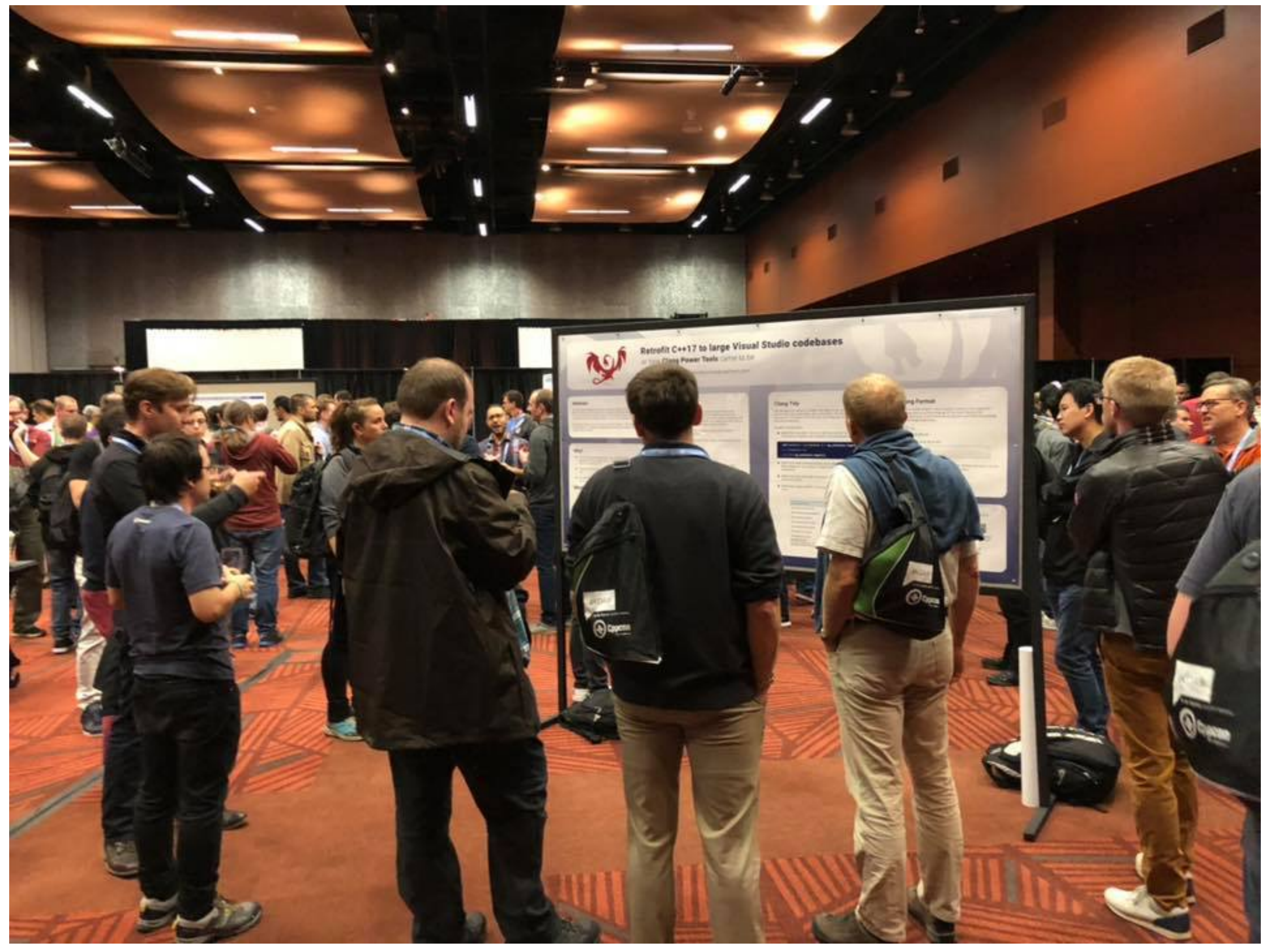


Better Tools in Your Clang Toolbox

Vignette in 3 parts

Part I
The Tools

CppCon 2018 Welcome Reception



Retrofit C++17 to large Visual Studio codebases

or how **Clang Power Tools** came to be

Gabriel Diaconița | gabriel.diaconita@caphyon.com

Caphyon, Romania

Abstract

You work hard to bring great products to market. Newly written C++17 code is easier to read, less prone to bugs, safer and more performant. However, not even great software products are perfect, they're being continuously developed. What about existing Visual C++ code in your product codebase?

Clang Power Tools is a free, open-source tool that brings C++17 to your existing Visual C++ codebase, effortlessly modernizing thousands of files and millions of lines of code.

Why?

- The LLVM Compiler Infrastructure offers excellent tools for automatically modernizing large codebases of C++ code, bringing these shiny modern features to production code.
- Visual Studio developers on Windows are not given easy access to these tools.
- Clang Power Tools brings clang-format, clang-compile and clang-tidy directly into Visual Studio.

What do I need?

1. Download and install Clang for Windows (LLVM pre-built binary, v4.0-7.0)
2. Clang Power Tools directly from the directly from Visual Studio Marketplace.

   Got a build automation server?
   Clang Power Tools integrates seamlessly with popular automation servers.
   Keep builds fast by compiling only translation units affected by the modified source / header files.

   Configure once, run everywhere.
   cpt.config versioned configuration files bring tailored Clang Power Tools settings to your entire team.

Clang Compile

Clang compile is the first step you take towards modernization. It will gradually lead you to make the code standard-compliant and weed out potential issues, such as double implicit conversions, integer promotions, and more. After your code compiles successfully with Clang you can then move to more advanced tools, like Tidy, LibTooling and Static Analyzer.

It also serves as gateway towards compliance with the latest Visual C++ compilation mode `/permissive-`.

Clang Power Tools can automatically clang compile every source file directly after compiling with MSVC.

General

Compile flags	ste;- Wno-unknown-pragmas;-Wno-i
File to ignore	
Project to ignore	
Treat additional includes as	system include directories

- [ ] Treat warnings as errors
- [ ] Continue on error
- [x] Clang compile after MSVC compile
- [ ] Verbose mode

Clang compile after MSVC compile
Automatically run Clang compile on the current source file, after successful MSVC compilation.

Clang Tidy

After getting your codebase to compile with clang, you can use clang-tidy to perform large scale refactorings, called modernizers. This will retrofit important C++17 features to your old code, increasing ease of reading, safety, and code performance.

Examples of modernizers:

- `modernize-use-auto` : will insert the 'auto' type specifier for variable declarations to improve code readability and maintainability

```
std::vector<int>::iterator I = my_container.begin();
// transforms to:
auto I = my_container.begin();
```

- `modernize-make-shared/unique`: inserts `make_shared` and `make_unique` for smart pointer exception safety and memory optimization
- `modernize-use-override` : inserts 'override' keyword to virtual functions overrides
- `modernize-loop-convert` : converts raw loops to range-based for loops

modernize

- [x] modernize-use-auto
- [x] modernize-use-bool-literals
- [x] modernize-use-default-member-init
- [x] modernize-use-emplace
- [x] modernize-use-equals-default
- [x] modernize-use-equals-delete
- [x] modernize-use-noexcept
- [x] modernize-use-nullptr

modernize-use-auto
This check is responsible for using the auto type specifier for variable declarations to improve code readability and maintainability. For example:

Clang Format

Even after modernizing the code, keeping it consistent with a particular formatting style is a serious challenge for any team. Eliminate the styling-related part of code reviews by having your source files automatically formatted using clang-format.

After selecting a formatting style (LLVM, Google, Mozilla and more), you can have source files auto-formatted at each Save operation.

Format On Save

- [x] Enable

Eager to see Clang Power Tools in action? Make sure you check out the following links:

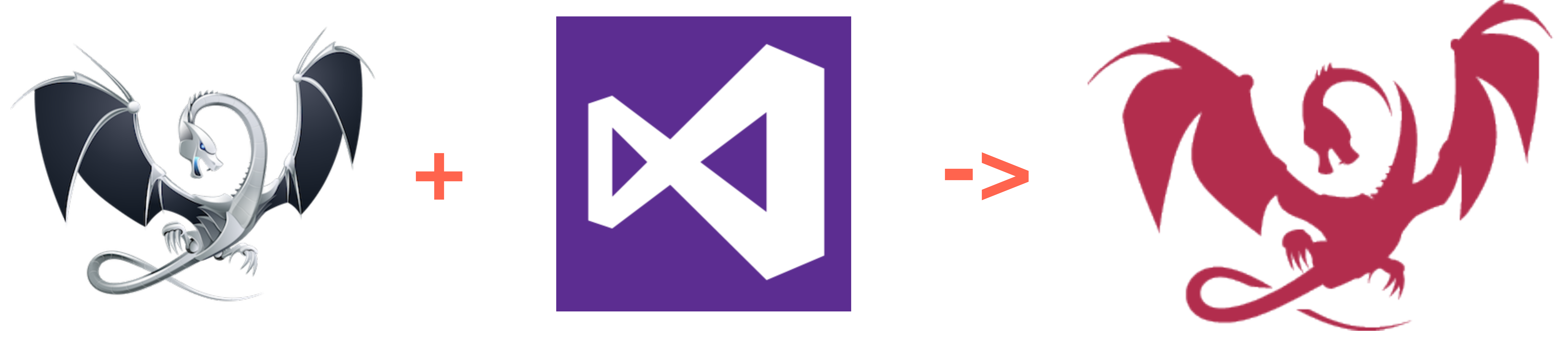
www.clangpowertools.com
The project's official site

Get Clang Power Tools now from Visual Studio Marketplace

github.com/Caphyon/clang-power-tools

Clang Power Tools was created with ❤ by some very awesome people at CAPHYON

https://caphyon.com



LLVM
clang-tidy
clang++
clang-format

Visual Studio
2015/2017

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

Clang Power Tools

September, 27

Exactly 1 Year Ago: `September 27`

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

https://www.youtube.com/watch?v=Wl-9ozmxXbo http://sched.co/BgsQ

1 Year and counting...

- **80,000** installs 🎉
- **35+** releases
- 173 reported issues fixed
- 22 Git forks
- 150+ stars/followers (GitHub)
- 50+ external PRs

Not bad for a "hobby" project 🤓

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'.

PS script config

cpt.config

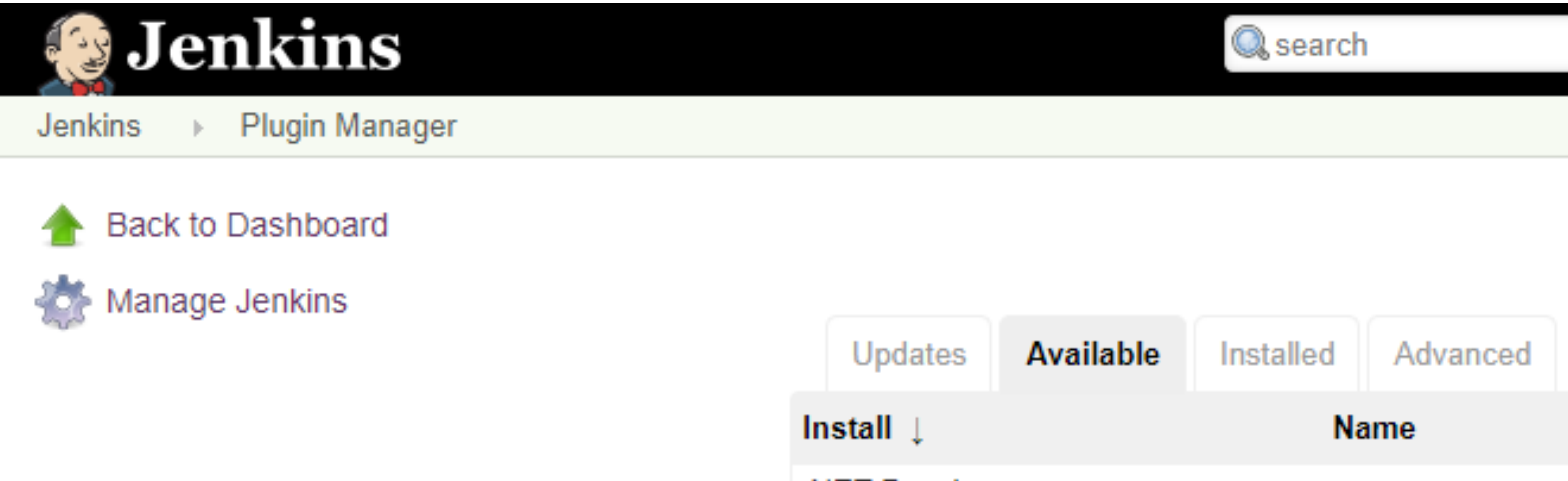
```
<cpt-config>
  <clang-flags>  "-Werror"
               , "-Wall"
               , "-fms-compatibility-version=19.10"
               , "-Wmicrosoft"
               , "-Wno-invalid-token-paste"
               , "-Wno-unknown-pragmas"
               , "-Wno-unused-value"
  </clang-flags>
  <header-filter>'.*'</header-filter>
  <parallel/>
  <treat-sai/>
  <vs-sku>'Professional'</vs-sku>
</cpt-config>
```

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.



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

Jenkins CI Configuration

Install PowerShell plugin

Jenkins ▸ Plugin Manager

	Plugin	Version
✓	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: `clang-build.ps1`

Build

Windows PowerShell

Command

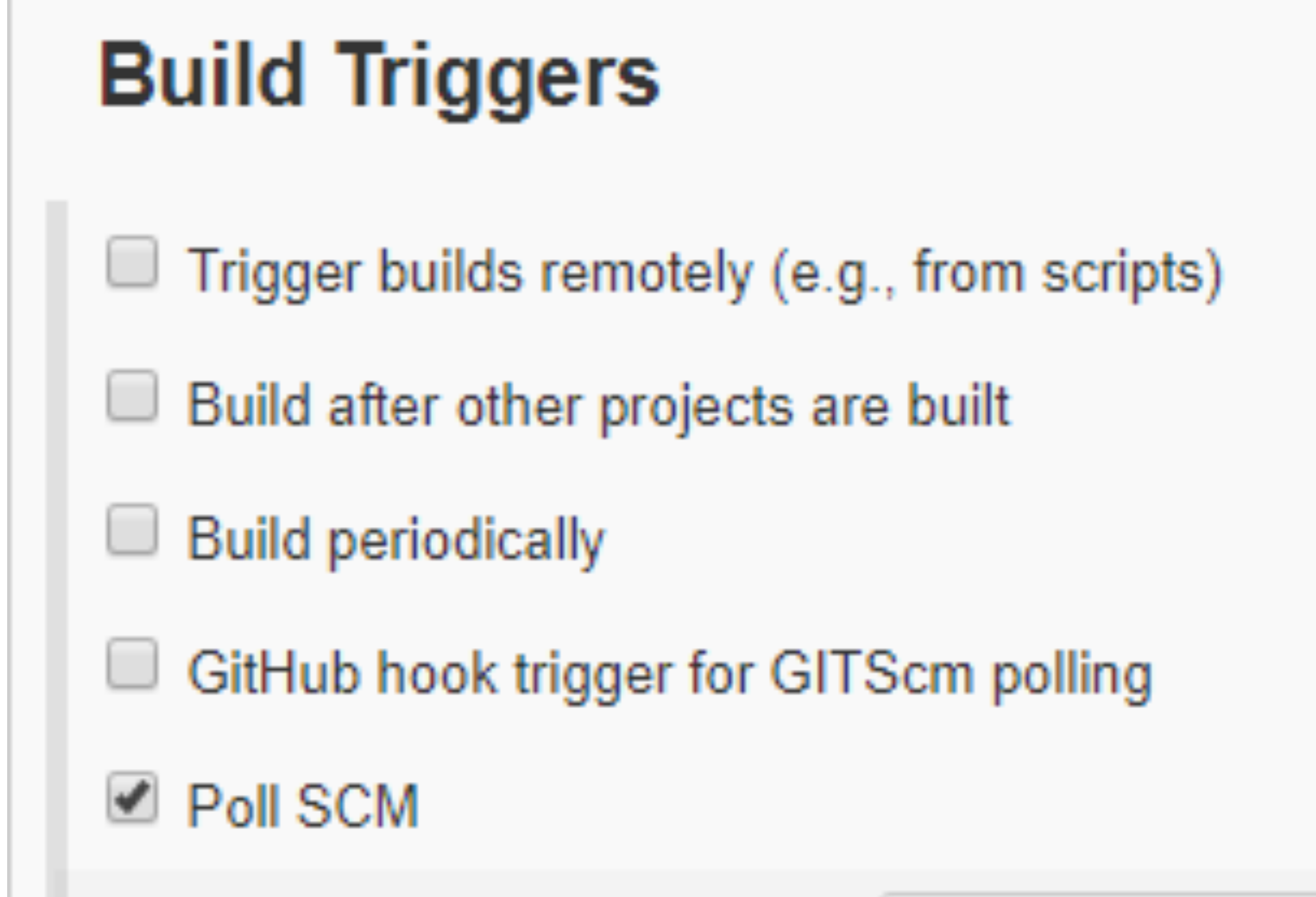
```
.\scripts\ai-clang-build.ps1 -parallel -proj-ignore LZMA.vcxproj
```

See the list of available environment variables

Add build step

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 - Caphyon

Caphyon
Victor Ciura

Channels
ai
ai-analytics
ai-appv
ai-appx
ai-jenkins
ai-qa
ai-status
ai-support

Ctrl+1
Ctrl+2

#ai-jenkins
16 | 0 | Add a topic
Search

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]

Message #ai-jenkins

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 (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 ➡

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

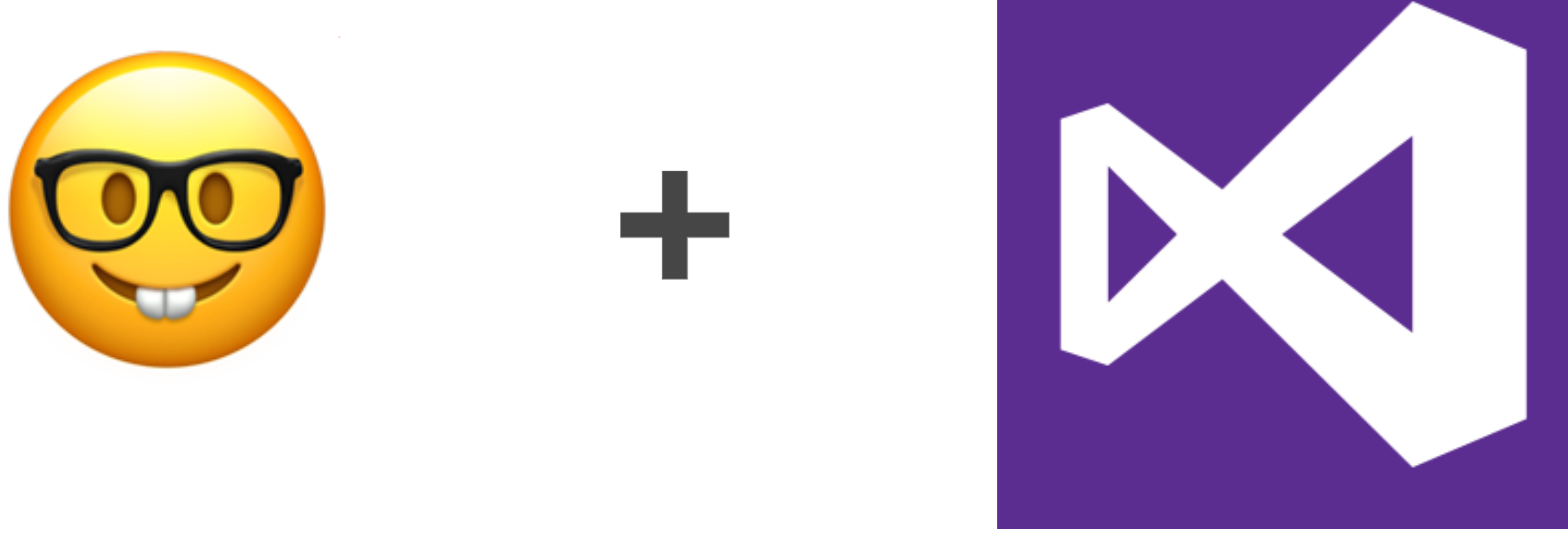
To

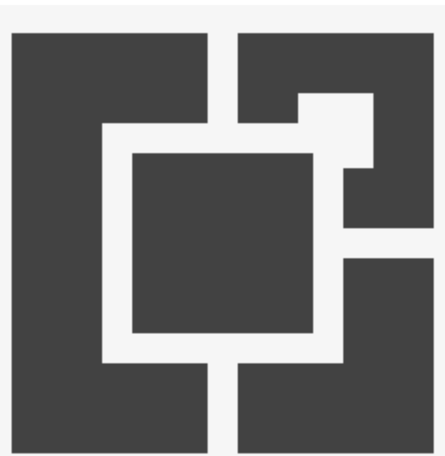
```
6: C:\Jenkins\Clang\workspace\platform\os\UrlUtil.cpp
5: C:\Jenkins\Clang\workspace\platform\os\Version.cpp
4: C:\Jenkins\Clang\workspace\platform\os\WindowUtil.cpp
3: C:\Jenkins\Clang\workspace\platform\os\WinVersion.cpp
2: C:\Jenkins\Clang\workspace\platform\os\WorkerThread.cpp
1: C:\Jenkins\Clang\workspace\platform\os\WowRedirection.cpp
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 | Save

Unread: 0 | Total: 3159

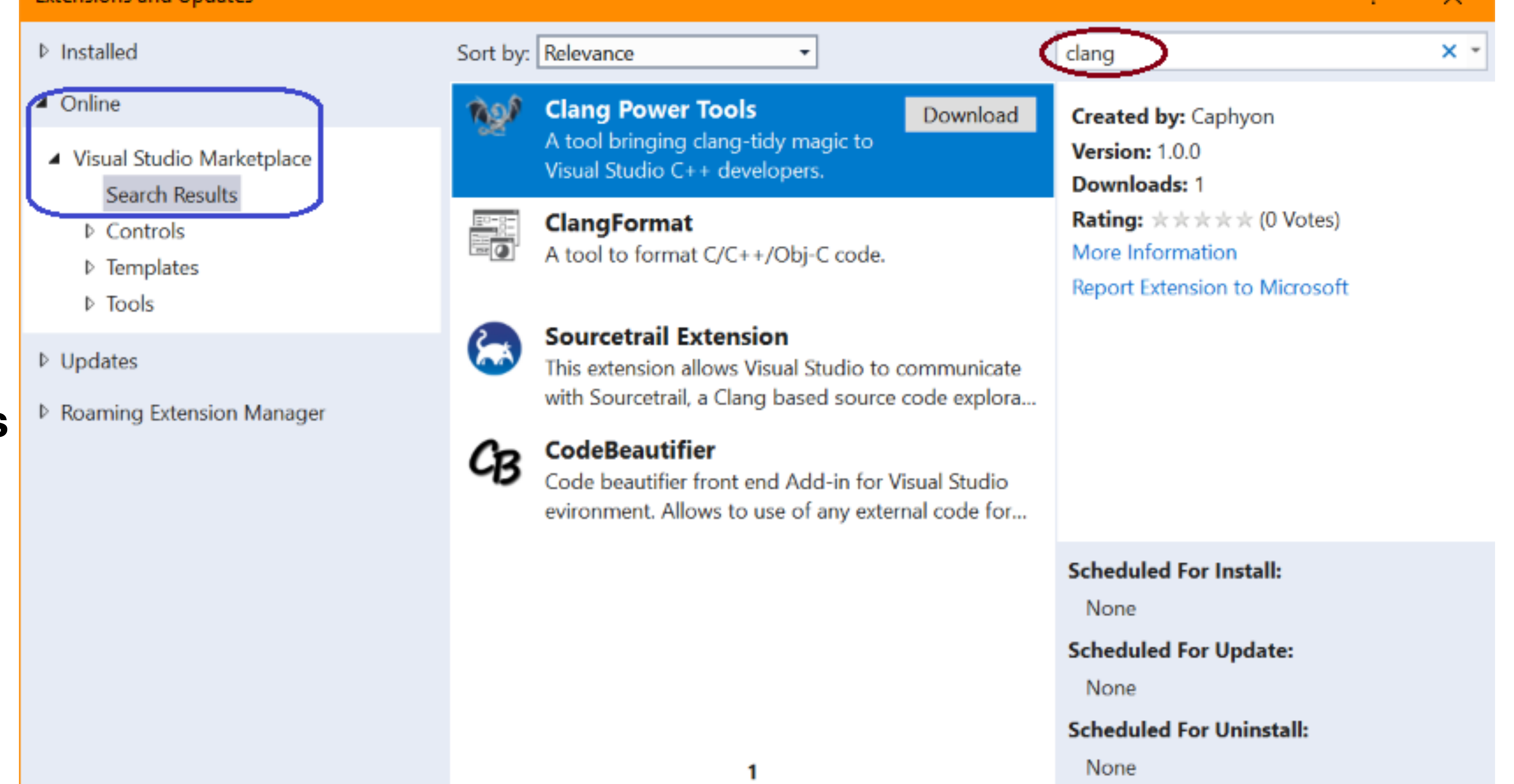
What About Developer Workflow?



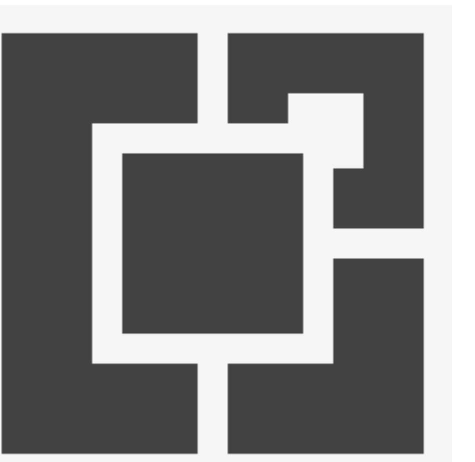


Install The "Clang Power Tools" Visual Studio Extension

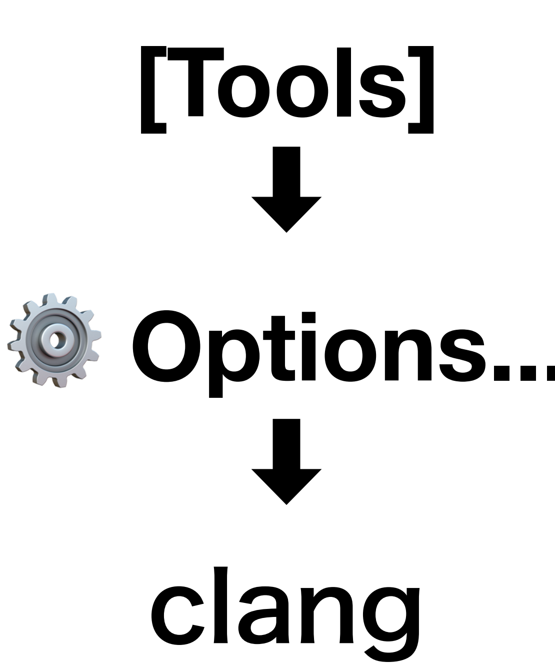
[Tools]
⬇
Extensions and updates

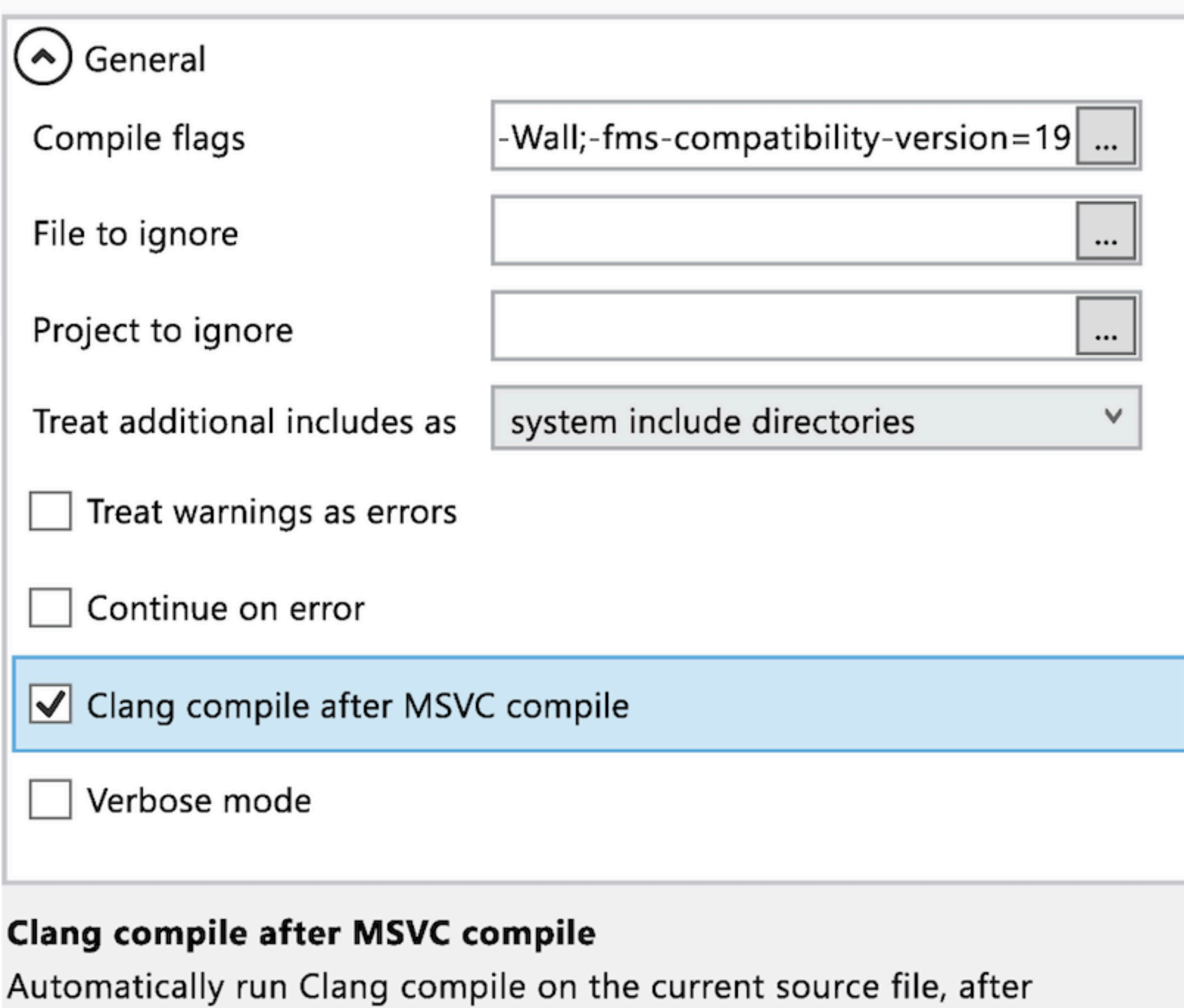


Requires "LLVM for Windows" (pre-built binary) to be installed. http://releases.llvm.org/6.0.0/LLVM-6.0.0-win64.exe

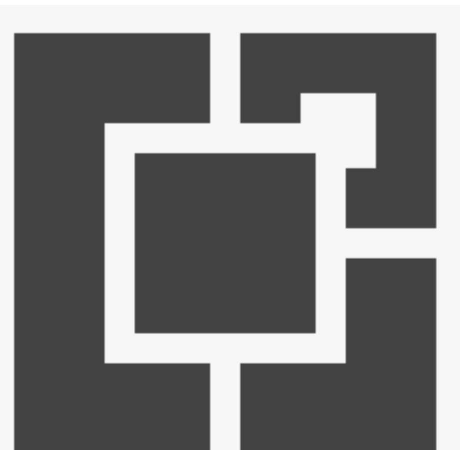


Configure The "Clang Power Tools" Visual Studio Extension





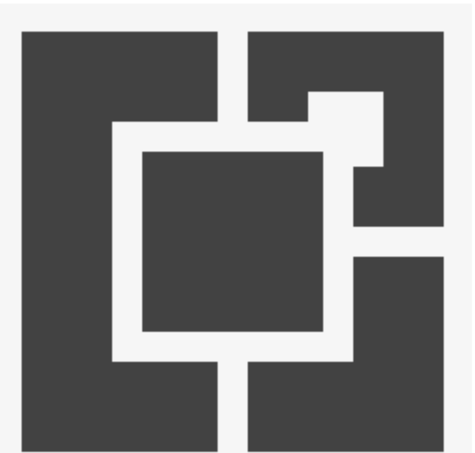
⬅ **Compilation settings**



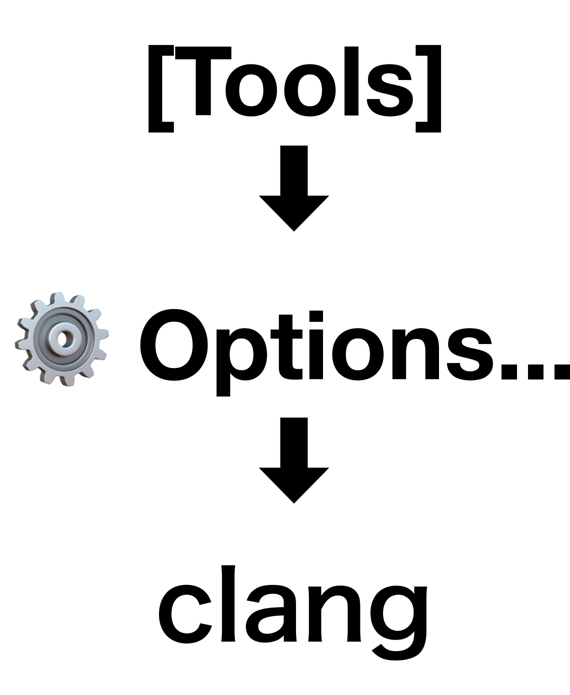
Configure The "Clang Power Tools" Visual Studio Extension

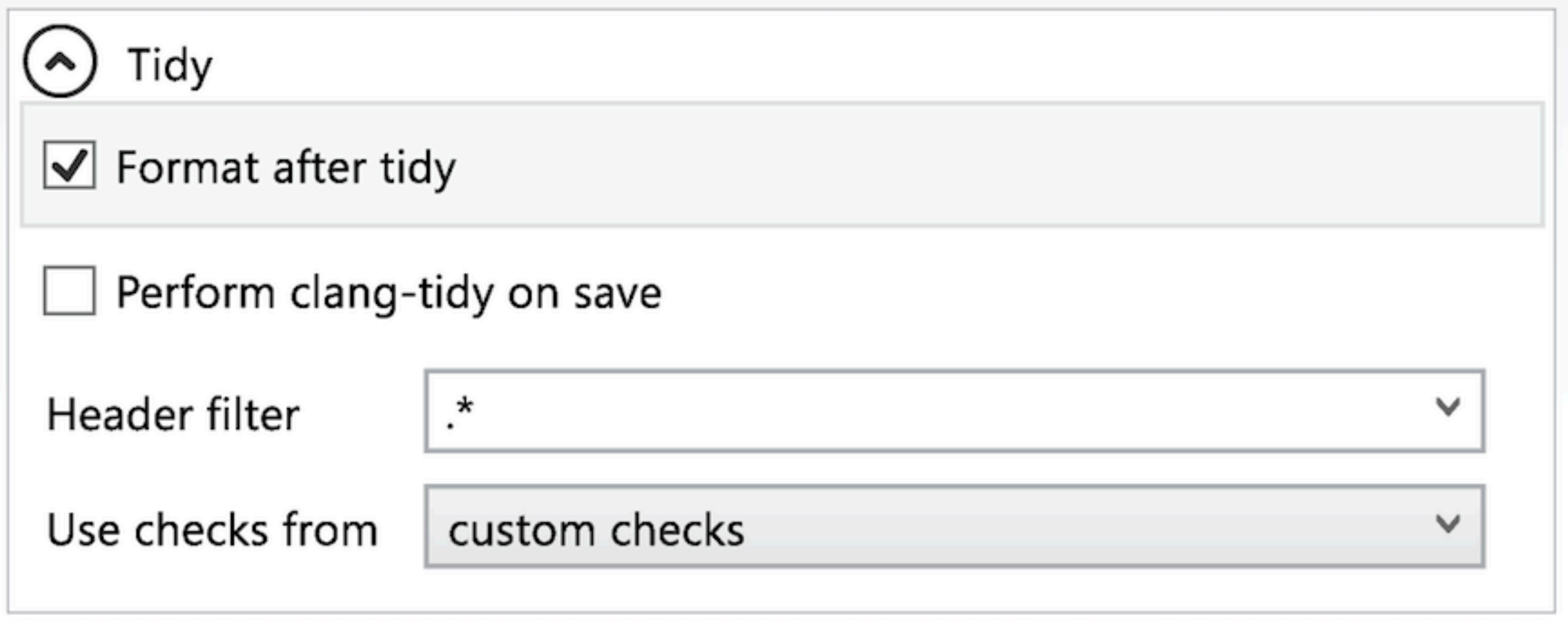
[Tools]
⬇
Options...
⬇
clang

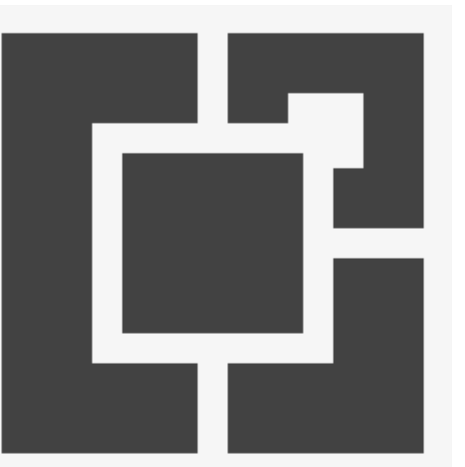
⬅ clang++ flags



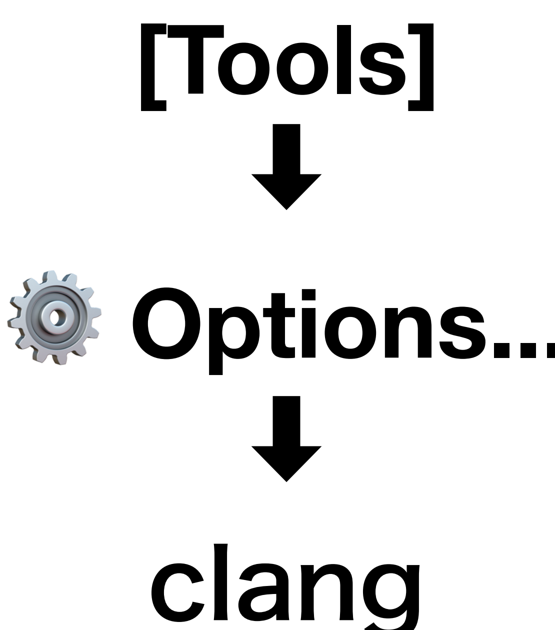
Configure The "Clang Power Tools" Visual Studio Extension

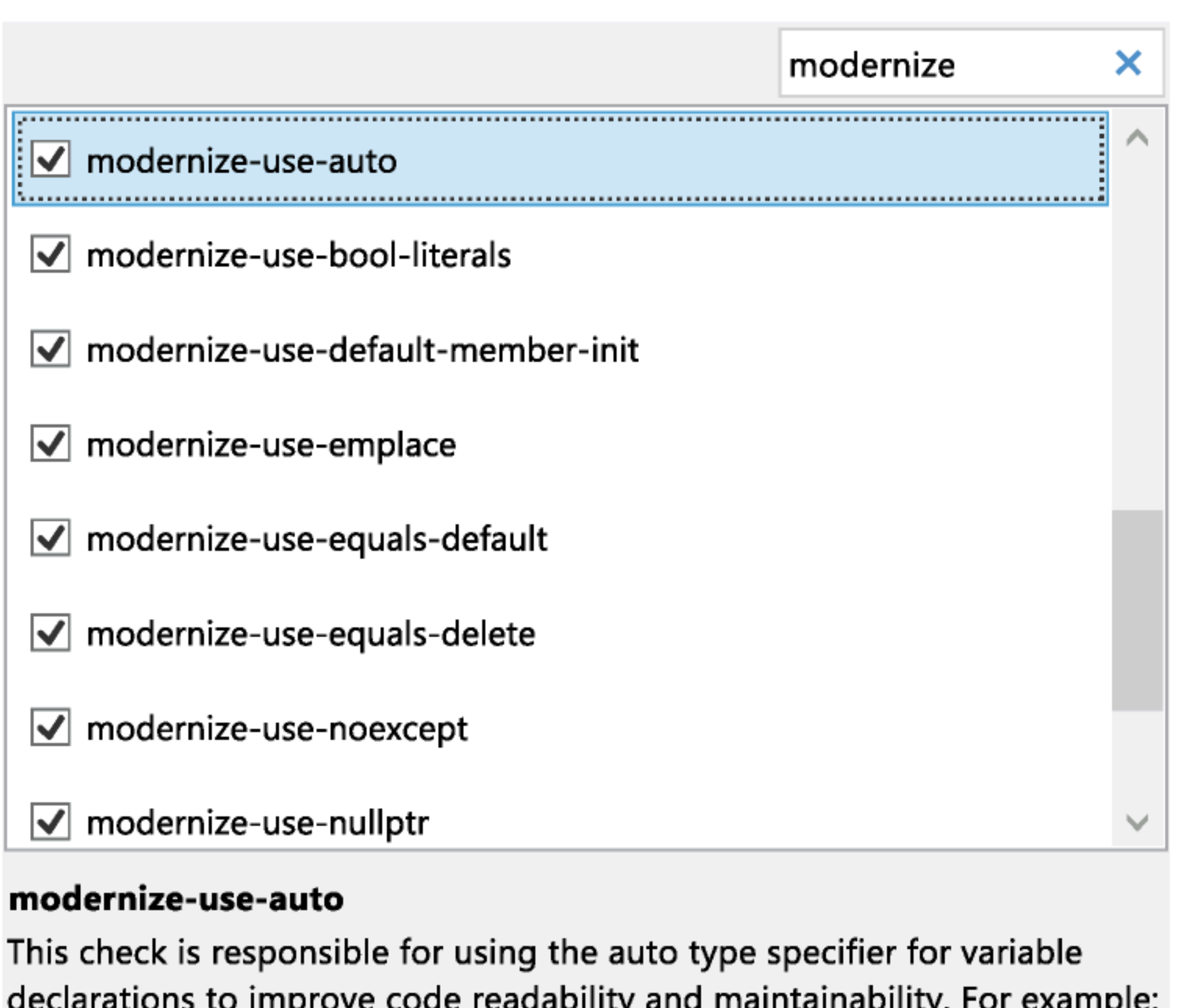






Configure The "Clang Power Tools" Visual Studio Extension

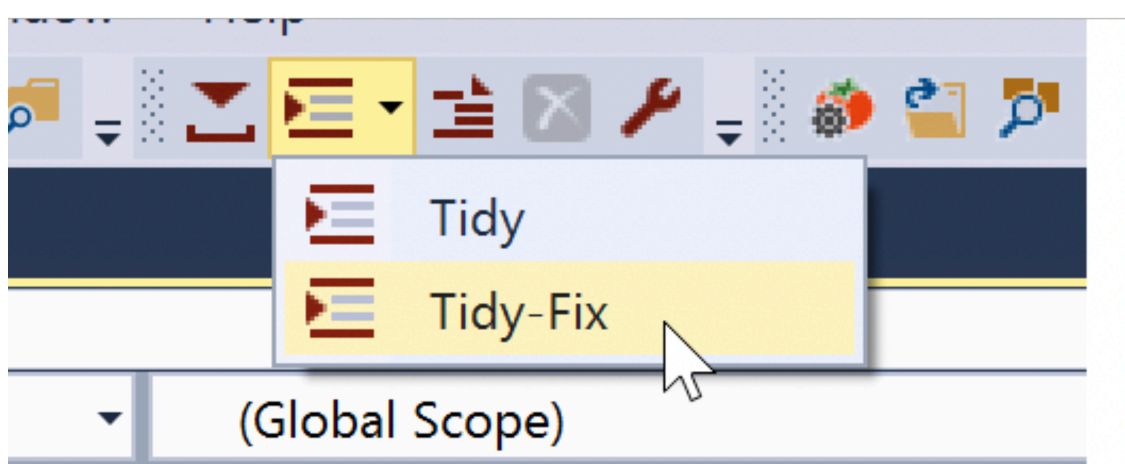
[Tools]
⬇

Options...
⬇
clang

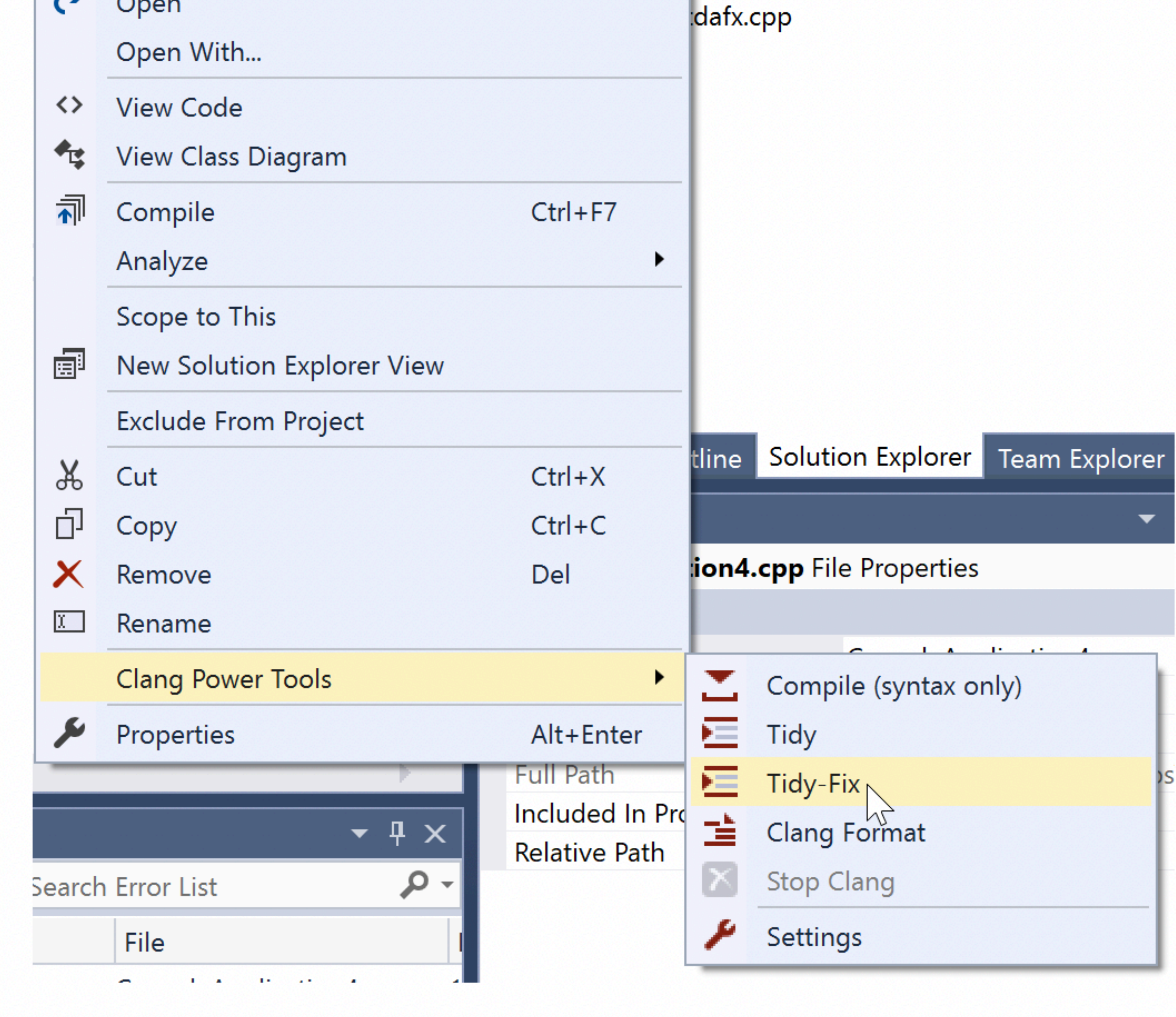


⬅ clang-tidy checks

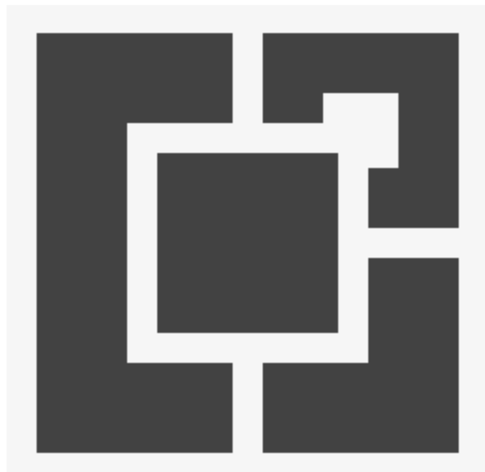
⬅ inline documentation

Using The "Clang Power Tools" Visual Studio Extension





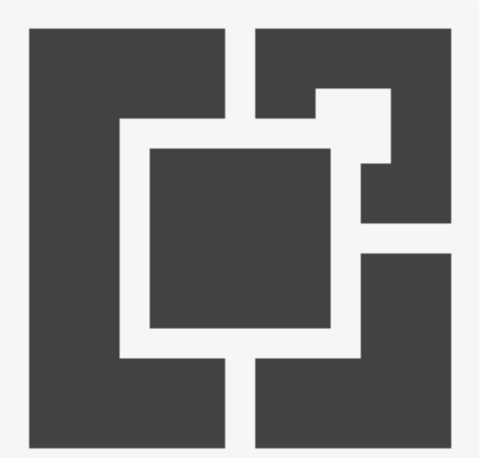
Run Clang Power Tools on a whole project or solution



Using The "Clang Power Tools" Visual Studio Extension

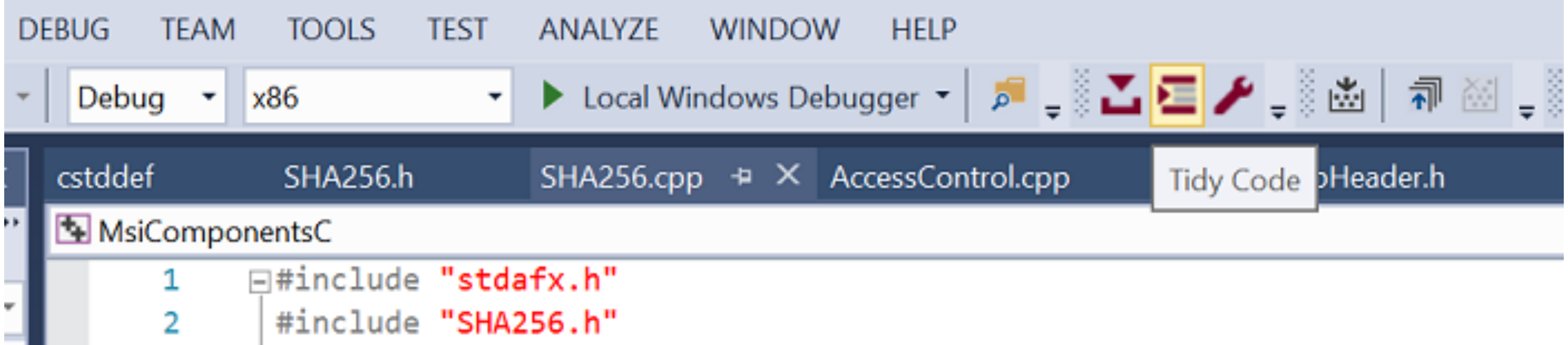
Run Clang Power Tools on an open source file

Save StringProcessing.cpp Ctrl+S
Close Ctrl+F4
Close All Documents
Close All But This
Copy Full Path
Open Containing Folder
Float
Pin Tab
Clang Power Tools
Compile (syntax only)
Tidy Code
Settings



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 ➡

Solution Explorer

Search Solution Explorer (Ctrl+;)

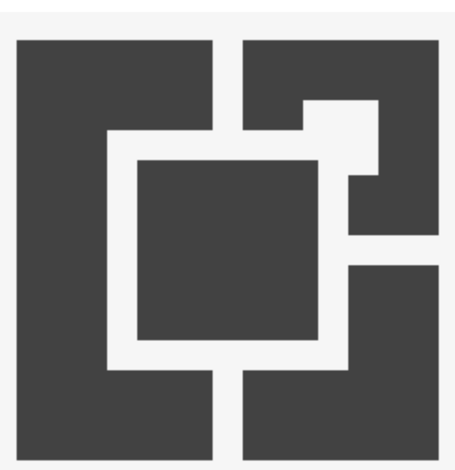
- ComVariants.cpp
- ComVariants.h
- CryptUtil.cpp
- CryptUtil.h
- DebugUtil.cp
- DebugUtil.h
- DetectAdmin
- DetectAdmin
- DialogTempl
- DialogTempl
- DynamicFun
- FileFinder.cp
- FileFinder.h
- FileMD5.cpp
- FileMD5.h
- FileStream.cp
- FileStream.h
- FileSystemUt
- FileSystemUt
- FileTime.cpp
- FileTime.h
- FileTimestampRestorer.cpp
- FileTimestampRestorer.h
- FileVersionUtil.cpp

Menu item	Shortcut
Open	
Open With...	
View Code	F7
View Class Diagram	
Compile	Ctrl+F7
New Solution Explorer View	
Exclude From Project	
Cut	Ctrl+X
Copy	Ctrl+C
Remove	Del
Rename	
Properties	Alt+Enter
Add Spell Checker Configuration File	
Clang Power Tools ▸	

- Compile (syntax only)
- Tidy Code
- Settings

l - Git

⬅ 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)
```

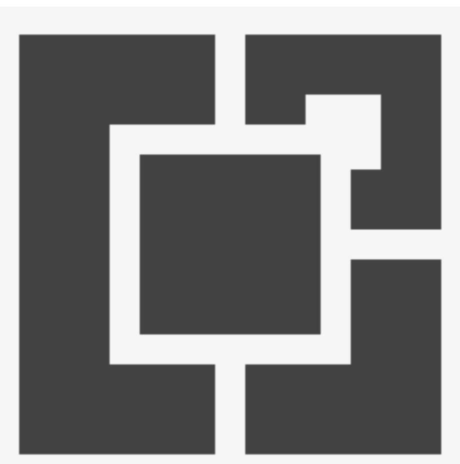
100 %

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: Windows SDK

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 (VS2017) 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

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]
```

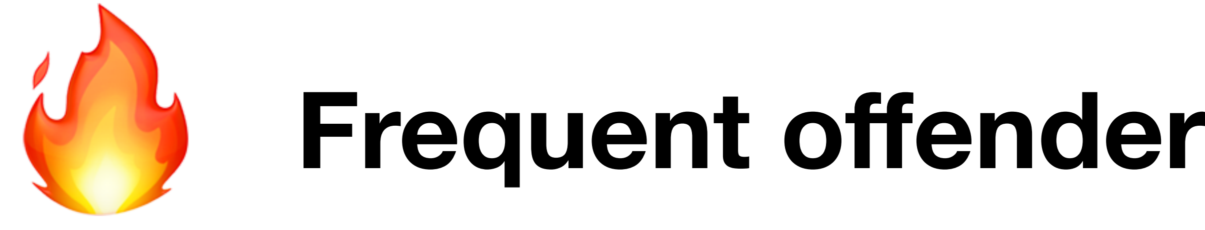
```
Error: PipeServer.cpp:42:39: error: missing field 'InternalHigh' initializer
[-Werror,-Wmissing-field-initializers]
```

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

```
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();
```



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

```
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:
       ^~~~~~~
```

Iterative Conformance

```
-fno-delayed-template-parsing

-Werror=microsoft

-Werror=typename-missing

-Wno-xyz-warn

      eg. -Wno-microsoft-sealed
```

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

clang-tidy

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...

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
```

```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

```
using namespace ast_matchers;

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

```cpp
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");
  }
}
```

Test

```
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>
...
```

Write custom checks for your needs
(project specific)

Run them regularly !

Theme Of The Week For Me:

STRINGS

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

CppCon 2018

Enough `string_view` to hang ourselves

CppCon 2018

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.

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)
}
```

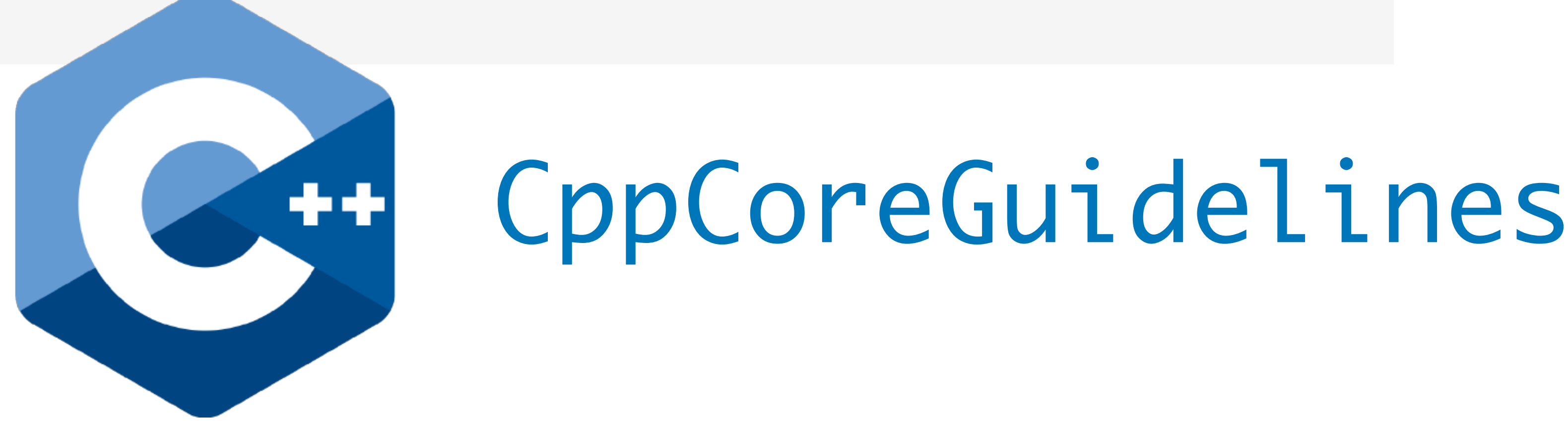
https://github.com/isocpp/CppCoreGuidelines/blob/master/docs/Lifetime.pdf

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`



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/

Today, September 27 • 14:00-15:00 http://sched.co/Gqti

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

Matthias Gehre & Gabor Horvath

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

Explore Further

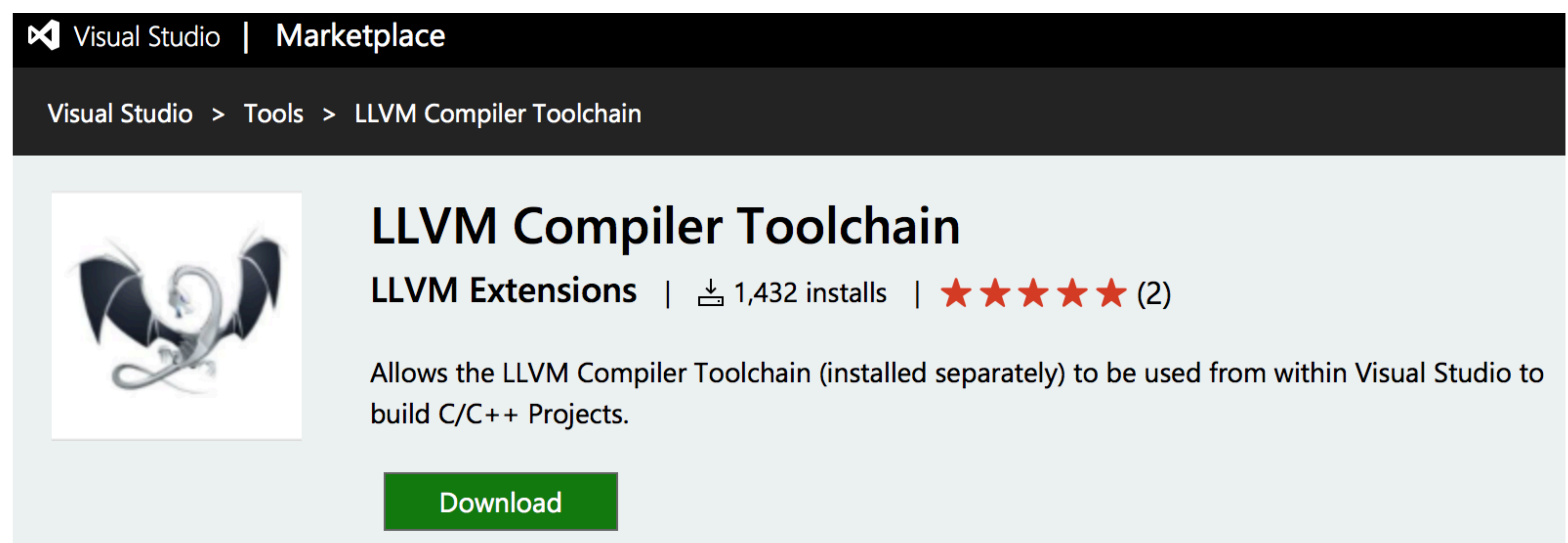
A new series of blog articles started last week on Visual C++ Team blog:

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/

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

4 of 4 talks done ✓

C++ Slack is your friend

https://cpplang.slack.com

CppLang Slack auto-invite:
https://cpplang.now.sh/

```
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