Выдержка из [справки](https://github.com/microsoft/vscode-cmake-tools/blob/376f82f28b43f9fe6f313e1856e081504723c368/docs/configure.md#the-cmake-tools-configure-step) по использованию расширения CMake Tools.

**CMake kits**.

Кит (набор) определяет независимую от проекта и независимую от конфигурации информацию о том, как собирать код. Кит может включать:

* Набор компиляторов: они заблокированы для определенных версий, так что вы можете быстро и легко переключать версию компилятора.
* Установка Visual Studio: построение для Visual Studio вовлекает в этот процесс больше, чем просто поиск необходимых исполняемых файлов компилятора. Visual C++ требует установки определенных переменных окружения, которые говорят ему как найти и скомпоновать заголовочные файлы и библиотеки инструментария Visual C++.
* Toolchain-файл: Низкоуровневый способ проинструктировать CMake как компилировать и компоновать цель. CMake Tools обрабатывает toolchain-файлы с помощью китов.

Киты в основном не зависят от генератора CMake (генератор CMake записывает входные файлы для нативной системы сборки). Киты Visual Studio имеют предпочтительный генератор, который будет использоваться в качестве запасного варианта, чтобы обеспечить использование соответствующего генератора MSBuild и .sln для компилятора Visual C++.

Примечания:

* Если вы используете систему сборки Ninja, не волнуйтесь о CMake генераторах Visual Studio. CMake Tools предпочтет использовать Ninja, если таковой присутствует, если только не будет явно задано другое.
* Если вы измените активный кит в то время, когда проект уже был сконфигурирован, конфигурация проекта будет сгенерирована заново с выбранным китом.
* Использование кита рекомендуется, но не является обязательным. Если вы не используете кит, CMake попытается выполнить его собственное автоматическое обнаружение.

*Как находятся и определяются киты*.

Когда расширение CMake Tools запускается в первый раз, он выполнит сканирование китов, чтобы найти доступные toolchains. При этом заполняется начальный список китов путем просмотра директорий, в которые обычно устанавливаются компиляторы, и использует vswhere чтобы найти установки Visual Studio.

*User-local киты*.

User-local кит это кит, который доступен для отдельного пользователя, для всех проектов, открытых с CMake Tools. Список user-local китов хранится в файле cmake-kits.json (NB: на самом деле – cmake-tools-kits.json, в каталоге пользователя C:\Users\<name>\AppData\Local\CMakeTools), который вы можете редактировать с помощью команды Edit CMake Kits из палитры команд.

Вы можете редактировать этот файл вручную чтобы определить новые глобальные киты, однако содержимое этого файла будет автоматически обновлено CMake Tools в процессе выполнения процедуры scan for kits.

Совет: определите новый кит с вашими собственными настройками вместо того, чтобы изменять киты, которые создает расширение CMake Tools, так чтобы ваши изменения не были переписаны во время следующего сканирования китов.

NB: на моей рабочей машине cmake-kits.json выглядит следующим образом:



*Project-local киты*.

Стандартные user-local киты доступны для всех проектов, которые используют CMake Tools. Для определения project-local кита, создайте .vscode/cmake-kits.json файл в директории проекта. Вы управляете содержимым этого файла вручную, но CMake Tools автоматически перезагрузит и обновит его, когда увидит, что этот файл добавлен, удален, или изменен. При смене китов вы можете выбрать как локальные для пользователя, так и локальные для проекта киты.

Примером использования локального для проекта кита является ситуация, когда проект определяет свои собственные файлы цепочки инструментов CMake (NB: когда мы хотим, чтобы этот проект явно указал с каким toolchain он хочет, чтобы его собирали). Такой кит может быть создан, чтобы указать факт необходимости его загрузки. Вы фиксируете его в хранилище вместе с исходниками и тем самым шарите его между другими разработчиками, достигая более легкой коллаборации благодаря именованному toolchain.

*Сканирование китов*.

Обновите user-local киты, запустив Scan for Kits из палитры команд vscode. Следующий процесс используется для поиска доступных китов:

1. Поиск компиляторов по переменной PATH.
   1. CMake Tools использует переменную среды PATH для получения списка директорий, в которых компиляторы могут быть найдены.
   2. CMake Tools ищет gcc или clang бинарники в путях PATH и получает информацию о версии от каждого исполняемого файла, который он найдет (NB: ключ --version аналогично тому, как мы делали это ранее при проверке корректности установки инструментов gcc, make, cmake). Для gcc, если соответствующий исполняемый файл g++ находится в том же каталоге, он добавляется в кит как соответствующий компилятор C++. Тоже самое относится к бинарному файлу clang++ в каталоге исполняемого файла clang.

Примечание: CMake Tools автоматически обнаруживают только Clang и GCC. Если вам нужно автоматическое определение дополнительных инструментов, откройте [github issue](https://github.com/microsoft/vscode-cmake-tools/issues) с информацией об именах двоичных файлов компилятора и о том, как анализировать информацию об их версии.

1. Поиск установок Visual Studio <…>.
2. Сохранение результатов в файл user-local kits.
   1. User-local kit файл cmake-kits.json обновляется новой информацией о китах.

Предупреждение: имя каждого кита создается на основе информации о компиляторе кита и его версии. Наборы с таким же названием будут перезаписаны. Чтобы предотвратить перезапись пользовательских китов, дайте им уникальные имена. CMake Tools не будет удалять записи из cmake-kits.json. Он только добавляет и обновляет существующие.

*Опции китов*.

CMake определяет различные опции, которые могут быть указаны для каждого кита в их определении, в файле cmake-kits.json, и эти параметры можно смешивать и сопоставлять по мере необходимости. Например, один кит может запросить окружение среды Visual Studio, в то же время указав clang-cl в качестве компилятора.

См. Configure CMake для большей информации о том, как киты применяются во время конфигурации.

*Спецификация компилятора*.

Для указания компилятора, список путей к нему должен быть прописан в файле cmake-kits.json. Наиболее распространенными языками, с которыми работает CMake, являются C и CXX, и CMake Tools имеют встроенную поддержку для их поиска. Однако, можно указать любой язык:

{

"name": "My Compiler Kit",

"compilers": {

"C": "/usr/bin/gcc",

"CXX": "/usr/bin/g++",

"Fortran": "/usr/bin/gfortran"

}

}

*Спецификация toolchain*.

CMake Tools не определяет цепочки инструментов автоматически, но вы можете указать CMake toolchain файл, например так:

{

"name": "Emscripten",

"toolchainFile": "/path/to/emscripten/toolchain.cmake"

}

NB: можно указать ранее созданный mingw64.cmake, и тогда не придется подключать его в CMakeLists.txt.

CMake Tools передаст этот путь в переменную CMAKE-TOOLCHAIN\_FILE в процессе конфигурации.

*Общие опции*.

Могут быть указаны следующие дополнительные опции:

preferredGenerator – генератор CMake, который следует использовать с этим китом, если он не установлен по умолчанию. CMake Tools по-прежнему будет искать в cmake.preferredGenerators из settings.json, но вернется к этому параметру, если генератор из пользовательских настроек недоступен.

cmakeSettings – это JSON объект, переданный как список настроек кэша при запуске CMake configure. Не используйте это для настроек и параметров, специфичных для проекта. Вместо этого, используйте для этой цели settings.json. Этот параметр наиболее полезен, когда toolchain файл учитывает дополнительные опции, которые можно передать в качестве в качестве переменных кэша.

environmentVariables – это JSON объект пар ключ-значение, указывающий дополнительные переменные среды, которые должны быть определены при использовании этого комплекта.

**Процесс конфигурации CMake**.

В терминах CMake, *конфигурация* означает детектирование требований и генерация файлов для сборки, которые произведут конечные скомпилированные артефакты.

Следующие концепты помогут вам понять, как CMake Tools взаимодействует с процессом конфигурации CMake:

* CMake Cache – это список пар ключ-значение, которые сохраняются между запусками процесса конфигурации. Он содержит следующее:
  + Значения, которые дорого определить, например, поддерживает ли компилятор –flag или #include.
  + Значения, которые редко изменяются, например, путь к заголовочному файлу или библиотеке.
  + Значения, предоставляющие разработчику контроль, например, BUILD\_TESTING, чтобы определить, следует ли создавать тестовые библиотеки/исполняемые файлы.
* Аргументы инициализатора кэша – это аргументы, передаваемые программе CMake, которая устанавливает значения в кэше перед тем, как запустить какой-либо скрипт CMakeLists.txt. Это позволяет вам управлять настройками сборки. В командной строке CMake, эти установки появляются с ключом –D.
* До тех пор, пока значения не будут переписаны или удалены, они сохраняются в CMake Cache между вызовами CMake.
* CMake НЕ выполняется непосредственно процедуру сборки, она ложится на инструменты сборки, установленные в вашей системе. А результат конфигурации проекта зависит от генератора CMake. Генератор говорит CMake какой тип инструмента будет использован для компиляции и генерации результатов сборки.

Независимо от того, какой генератор выбран, CMake Tools всегда поддерживают сборку из vscode. Если вы выполняете сборку из vscode, мы рекомендуем вам использовать инструмент сборки Ninja.

**Шаг конфигурации проекта с помощью CMake Tools**.

CMake Tools управляет программой CMake через cmake-file-api, который предоставляет информацию о проекте через файл на диске.

Когда CMake Tools запускает процесс конфигурации, он принимает во-внимание следующее:

1. *Активный кит*.

CMake киты предоставляют информацию о доступных toolchains в вашей системе, которые могут быть использованы CMake для сборки вашего проекта.

* Для toolchain CMake Tools устанавливает переменную кэша CMAKE\_TOOLCHAIN\_FILE, которая указывает путь к файлу, указанному в ките.
* Для компилятора CMake Tools устанавливает переменную кэша CMAKE\_<LANG>\_COMPILER, которая указывает путь к каждому <LANG> из кита.
* Для Visual Studio CMake Tools устанавливает переменные окружения, необходимые для использования выбранной установки Visual Studio, и задают для CC и CXX значение cl.exe, чтобы CMake определял компилятор Visual C++ в качестве основного компилятора, даже если другие компиляторы, такие как GCC, присутствуют в $PATH.

Каждый кит может также определить дополнительные настройки переменных кэша, требуемые для работы этого кита. Набор может также определять preferredGenerator.

См. раздел CMake kits для большей информации о том, как работают наборы.

См. раздел Kit options для большей информации о различных типах наборов.

1. *Какой генератор использовать*.

CMake Tools попытается не разрешить программе CMake неявно решать какой генератор использовать. Вместо этого, он попытается определить предпочитаемый генератор на основе различных источников и остановится, когда найдёт валидный генератор. Источниками, которые он проверяет, являются следующие:

* Конфигурационная установка cmake.generator.
* Конфигурационная установка cmake.preferredGenerators. Каждый элемент в этом списке проверяется на валидность, и если он соответствует, то он и выбирается. Список имеет разумное значение по умолчанию, которое подходит для большинства сред.
* Атрибут кита preferredGenerator. Автоматически генерируемые Visual Studio киты устанавливают этот атрибут в значение Visual Studio generator, соответствующее их версии.
* Если не подобран никакой генератор, CMake Tools выдает ошибку.

1. *Опции конфигурации*.

CMake Tools имеет множество мест, где можно определить параметры конфигурации. Они ищутся по порядку и объединяются вместе. Когда ключи имеют одинаковое имя, используется самое последнее значение, найденное во время поиска. Порядок поиска параметров следующий:

* 1. Опция cmake.configureSettings из settings.json.
  2. значение settings из активного variant options.
  3. BUILD\_SHARED\_LIBS устанавливается на основе variant options.
  4. CMAKE\_BUILD\_TYPE устанавливается на основе variant options.
  5. CMAKE\_INSTALL\_PREFIX устанавливается на основе cmake.installPrefix.
  6. CMAKE\_TOOLCHAIN\_FILE устанавливается на основе toolchain.
  7. Атрибут cmakeSettings активного кита.

Кроме этого, cmake.configureArgs передаются перед любым из вышеперечисленных.

1. *Окружение конфигурации*.

CMake Tools устанавливает переменные среды для дочернего процесса, который он запускает для CMake. Как и параметры конфигурации, значения объединяются из разных источников, причем более поздние источники имеют приоритет. Источниками являются:

1. Переменные среды, требуемые активным комплектом.
2. Значение cmake.environment.
3. Значение cmake.configureEnvironment.
4. Переменные среды, требуемые активным variant’ом.

Все вышеперечисленное учитывается при конфигурации. После завершения CMake Tools загружает информацию о проекте из CMake и создает диагностику на основе выходных данных CMake. Теперь вы готовы к сборке с помощью CMake Tools.

**Очистка конфигурации**.

Чтобы заставить Cmake Tools выполнить очистку конфигурации, запустите команду CMake: Delete Cache and Reconfigure из палитры команд. Очистка конфигурации удаляет файл CMakeCache.txt и каталог CMakeFiles из каталога build/. Это сбрасывает все состояния по умолчанию CMake.

Очистка конфигурации требуется для определенных изменений в системе сборки, например, при изменении активного кита, но также может быть удобна для сброса, если вы изменили параметры конфигурации вне CMake Tools.

CMake Tools автоматически выполнит очистку конфигурации, если вы измените активный кит.

[**Сборка с помощью CMake Tools**](https://github.com/microsoft/vscode-cmake-tools/blob/main/docs/build.md).

Как только вы сконфигурировали ваш проект, вы можете запустить команду CMake: Build. Большую часть времени вы потратили на конфигурацию проекта. Процесс сборки намного проще.

Вы можете запустить процесс сборки:

1. Выполнить команду CMake: Build из палитры команд.
2. Нажать горячую клавишу на клавиатуре (по-умолчанию – F7).
3. Выбрав кнопку «Build» в строке статуса vscode.

Когда начинается сборка, открывается панель вывода CMake Tools для отображения вывода инструмента сборки. Кнопка «Build» изменится на кнопку «Stop» и может отображать прогресс запущенной сборки. Нажатие кнопки «Stop» отменяет текущую сборку. Запуск сборки во время выполнения другой сборки отменяет текущую сборку и запустит новую.

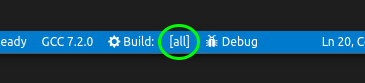
Примечание: Индикатор выполнения отображается только в том случае, если инструмент сборки выдает выходные строки, которые можно проанализировать, чтобы получить процент хода сборки. Сюда входят генераторы Ninja и Makefile. MSBuild не выдает эту информацию, поэтому индикатор выполнения не отображается.

Результаты сборки записываются в каталог, указанный в cmake.buildDirectory. По умолчанию это подкаталог каталога проекта, поэтому результаты сборки видны в проводнике проекта. Точные пути к файлам будут зависеть от конфигурации вашего проекта CMake.

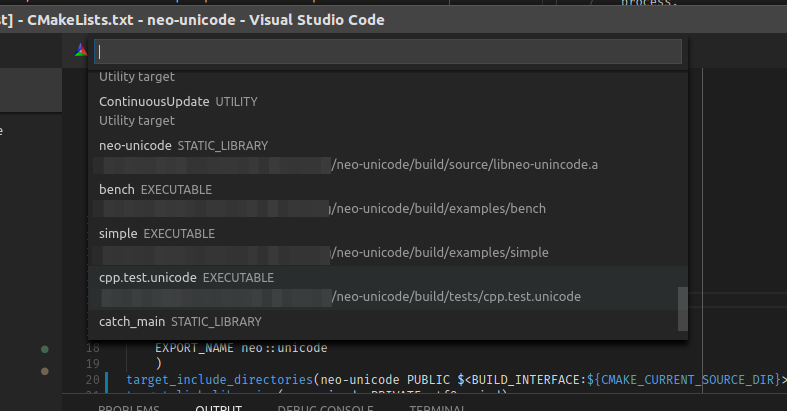
**Построение цели по умолчанию**.

CMake Tools сохраняют «default target» для процесса сборки. Целью по умолчанию является цель «all» (называемая ALL\_BUILD в некоторых генераторах), которая строит все цели, которые CMake назначил для сборки по умолчанию.

Имя цели по умолчанию показано в статусной строке справа от кнопки «Build», в квадратных скобках.



При выборе этой кнопки отображается список быстрого выбора для всех целей, о которых знает CMake Tools и которые могут быть собраны, а также полный путь к результату сборки, который будет создан этой целью:



**Сборка одиночной цели**.

Вы можете собрать одну цель, не изменяя текущую цель сборки из палитры команд VS Code, выполнив команду CMake: Build a target или нажав сочетание клавиш (по умолчанию – Shift+F7). CMake соберёт любые зависимые цели, даже если они не выбраны напрямую.

**Создание задачи сборки**.

Вы можете также определить задачу сборки, выполнив команду Tasks: Configure task из палитры команд.

При выборе задачи для «CMake: Build», следующая задача будет создана в файле ${workspaceFolder}/.vscode/tasks.json.

{

"version": "2.0.0",

"tasks": [

{

"type": "cmake",

"label": "CMake: build",

"command": "build",

"targets": [

"ALL\_BUILD"

],

"group": "build",

"problemMatcher": [],

"detail": "CMake template build task"

}

]

}

Целью по умолчанию является цели «all» (называемая ALL\_BUILD в некоторых генераторах). Вы можете изменить значение свойства «targets» значением единственной цели или списка целей, которые могут быть построены. Вот пример двух кастомизированных задач сборки, в которых изменено имя задачи, список целей и описание задач:

{

"version": "2.0.0",

"tasks": [

{

"type": "cmake",

"label": "Sample CMake build task with single target",

"command": "build",

"targets": [

"prj1"

],

"group": "build",

"problemMatcher": [],

"detail": "Build task to build prj1"

},

{

"type": "cmake",

"label": "Sample CMake build task with multiple targets",

"command": "build",

"targets": [

"prj1",

"prj2"

],

"group": "build",

"problemMatcher": [],

"detail": "Build task to build prj1 and prj2"

}

]

}

Теперь вы можете запустить сборку, выполнив команду «Tasks: Run Build Task» из палитры команд vscode. Если в «tasks.json» определены другие задачи, средство выбора покажет вам список задачи на выбор.

**Как CMake Tools выполняет сборку**.

**Ключи сборки**.

CMake Tools запускает сборку, передавая ключ --build в CMake. Этот флаг используется как инструмент вызова сборки, не зависящий от генератора. CMake Tools также передаёт параметр --config <build-type> на основе текущего типа сборки из активного variant. Это инструктирует мульти-конфигурационные инструменты сборки о том, какую конфигурацию они должны создать.

Для продвинутых сценариев дополнительные флаги для --build можно устанавливать с помощью cmake.buildArgs, а дополнительные флаги для непосредственно инструмента сборки можно установить с помощью cmake.buildToolsArgs.

**Окружение сборки**.

Переменные среды наследуются от вызывающего процесса vscode с дополнительными переменными из настроек cmake.buildEnvironment и cmake.environment.

Если вы используете кит Visual Studio, CMake Tools запускает сборку с соответствующими переменными среды, установленными для сборки с выбранной версией Visual Studio, во многом аналогично тому, как они задают эти переменные среды на этапе настройки CMake Tools.

**Очистка сборки**.

Чтобы очистить сборку, из палитры команд vscode выполните команду CMake: Clean. Или запустите команду CMake: Clean rebuild, чтобы быстро удалить все результаты сборки и запустить сборку с нуля.

[**Настройки CMake**](https://github.com/microsoft/vscode-cmake-tools/blob/main/docs/cmake-settings.md#cmake-settings).