Developer Documentation

- Building Pragma
- Binary Modules

Building Pragma

You can also find these build instructions on the Pragma repository on GitHub.

Build Requirements

- ~50 GiB of disk space
- CMake 3.21.4 or newer
- Python 3.9.5 or newer

Windows

- Visual Studio 2022 or newer

Linux

- clang-14 or newer (Pragma is *not* compatible with gcc!)

Build Instructions

Launch a command-line interface and clone the pragma repository (with submodules) into a directory of your choice:

```
git clone https://github.com/Silverlan/pragma.git --recurse-submodules
```

After that you can simply navigate to the pragma directory and run the python build-script, which will automatically download all dependencies, configure CMake, and build and install the project (this will take several hours):

```
python build_scripts/build.py --with-pfm --with-all-pfm-modules --with-vr
```

> If you don't need the filmmaker, you can omit the `--with-pfm --with-all-pfm-modules` arguments, which will significantly reduce the build time and the required amount of disk space.

Linux

> Before running the build script, you will have to install the following packages:

```
# Required for the build script
sudo apt-get install python3

# Required for Pragma core
sudo apt install build-essential
sudo add-apt-repository ppa:savoury1/llvm-defaults-14
sudo apt update
sudo apt install clang-14
sudo apt install libstdc++-12-dev
sudo apt install libstdc++6
sudo apt-get install patchelf

# Required for Vulkan
sudo apt-get -qq install -y libwayland-dev libxrandr-dev

sudo apt-get install libxcb-keysyms1-dev
sudo apt-get install xcb libxcb-xkb-dev x11-xkb-utils libx11-xcb-dev libxkbcommon-x11-dev

# Required for GLFW
sudo apt install xorg-dev

# Required for OIDN
sudo apt install git-lfs

# Required for Cycles
sudo apt-get install subversion

# Required for Curl
sudo apt-get install libssl-dev
sudo apt install libssh2-1

# Required for OIIO
sudo apt-get install python3-distutils
```

Once the build script has been completed, you should find the build files in `pragma/build`, and the install files in `pragma/build/install`. The `install` directory should contain everything you need to run Pragma.

If you make any code changes to the core engine code, you can build the `pragma-install` target to build them. This will also re-install the binaries.

If you make any code changes to a module, you will have to build the module build target first, and then build `pragma-install` afterwards.

Build Customization

Running the build-script with the arguments above will build and install Pragma and the Pragma Filmmaker with all dependencies. Alternatively you can also configure the build to your liking with the following parameters:

Parameter	Description	Default
`--help`	Display this help	
`--generator <generator>`	The generator to use.	`Visual Studio 17 2022` (On Windows) `Unix Makefiles` (On Linux)
`--c-compiler`	[Linux only] The C-compiler to use.	`clang-14`
`--cxx-compiler`	[Linux only] The C++-compiler to use.	`clang++-14`
`--with-essential-client-modules <1/0>`	Include essential modules required to run Pragma.	1
`--with-common-modules <1/0>`	Include non-essential but commonly used modules (e.g. audio and physics modules).	1
`--with-pfm <1/0>`	Include the Pragma Filmmaker.	0
`--with-core-pfm-modules <1/0>`	Include essential PFM modules.	1

`--with-all-pfm-modules <1/0>`	Include non-essential PFM modules (e.g. chromium and cycles).	0
`--with-vr <1/0>`	Include Virtual Reality support.	0
`--build <1/0>`	Build Pragma after configurating and generating build files.	1
`--build-config <config>`	The build configuration to use.	`RelWithDebInfo`
`--build-directory <path>`	Directory to write the build files to. Can be relative or absolute.	`build`
`--deps-directory <path>`	Directory to write the dependency files to. Can be relative or absolute.	`deps`
`--install-directory <path>`	Installation directory. Can be relative (to build directory) or absolute.	`install`
`--verbose <1/0>`	Print additional debug information.	0
`--module <moduleName>:<gitUrl>`	Custom modules to install. Use this argument multiple times to use multiple modules.	

Example for using the `--module` parameter:

```
--module pr_physx:"https://github.com/Silverlan/pr_physx.git"
```

If you want to create your own binary module, please check out the article on binary modules.

Binary Modules

Binary modules allow you to change or extend the behavior of Pragma without having to change the core source code. Binary modules are written in C++ and loaded during runtime using Lua (with some special exceptions). Some example modules are:

- pr_chromium: Adds an integrated Chromium-based Web Browser to Pragma
- pr_openvr: Adds virtual reality support to Pragma
- pr_bullet: Adds support for the Bullet physics engine to Pragma
- pr_physx: Adds support for the PhysX physics engine to Pragma
- pr_curl: Adds support for the curl library to Pragma
- pr_sqlite: Adds SQLite support to Pragma

Installing Modules

If the binary module is available on GitHub and has at least one release, you can use the following console command in Pragma to install the module directly:

```
install_module <GitHubUser>/<RepoName>
```

For instance, if you want to install the pr_curl module, simply run `install_module Silverlan/pr_curl` in the console, which will automatically download the module and install it.

Alternatively, if a binary module comes with prebuilt binaries, you can simply extract the archive to your Pragma installation directory, the file structure in the archive should ensure that its placed in the correct location. Simple modules that consist of a single library file can usually be found in the "modules" directory, while more complex modules (like chromium) reside in their own sub-directory within "modules".

The module can then be loaded with a simple Lua-script:

```
local moduleName = "chromium/pr_chromium"
local result = engine.load_library(moduleName)
if(result ~= true) then
    console.print_warning("Failed to load \"" .. moduleName .. "\" module: " .. result)
```

```
    return
end
```

The module name is the path to the binary file, without the "modules/" prefix and without the file extension. On Linux the module file also has the "lib" prefix, which has to be omitted as well. For instance, if the module filename is "modules/curl/libpr_curl.so", then the module name should be "curl/pr_curl".

Building Modules

If you want to build a module manually, you'll have to first build Pragma. You can then add the module to the build instructions by using one of the following methods:

Method 1) (Recommended)

Open `pragma/build_scripts/user_modules.py` in a text-editor and follow the instructions to add your module.

After that, simply re-run the build script.

> Tip: You can use the --rerun option to make the build script re-use the options used in the previous build.

Method 2)

You can also add the following option when running the build script:

```
--module <name>:<gitUrl>
```

For instance, if you want to add the `pr_chromium` module to the build:

```
--module pr_chromium:https://github.com/Silverlan/pr_chromium
```

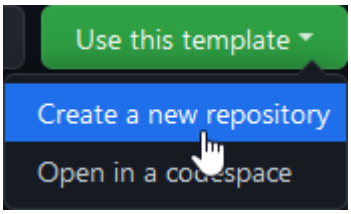
(You can use the `--module` argument multiple times if you want to add more than one module.)

Once the module has been added to the build, you can simply build the `pragma-install` target. It will build the module and install it automatically. You should then be able to find the module within the "modules" directory of the pragma installation folder.

Custom Modules

Setting up your own custom module is very simple and only takes a few minutes using the Pragma module template repository on GitHub, which also includes workflows for automated builds and releases.

To do so, go to the template repository and create a new repository from it:



You can choose whatever name you want for the repository, but for consistency it is recommended that it should have the same name as the module, which means:

- It should start with the "pr_" prefix
- It should be all lowercase
- It should only contain letters and underscores

Some examples are: "pr_chromium", "pr_bullet", "pr_audio_fmod", etc.

Next click "Create repository from template" to generate the repository.

The generated repository now contains a bunch of template files, which still need to be initialized. To do so, open the file "template_pragma_module.json" on GitHub and edit the values within:



- **name**: A pretty name, which will appear in the generated readme.

- **module_name**: The internal name of the module (i.e. the name of the CMake target and the binaries). This name should always start with the prefix `pr_` and always be lowercase. It should match the repository name if possible.
- **install_directory**: The directory where the module should be installed to, relative to the Pragma installation. Please choose this value carefully:
  - If you know your module is going to be a single binary with no additional files, it's recommended to leave this value at the default ("modules/").
  - If your module requires additional files, it's recommended to change this value to a sub-directory, e.g. "modules/<moduleName>/".
  - If your module requires Lua-scripts or asset files, it's recommended to change this value to an addon path, e.g. "addons/<addonName>/modules/<moduleName>/".
- **release_directory**: The directory that will be used for the GitHub releases. This value usually depends on the *install_directory*:
  - If the *install_directory* is the default ("modules/"), leave this value empty. In this case only the binary file will be added to the GitHub release.
  - If the *install_directory* is a sub-directory in "modules/", set this value to the same sub-directory (i.e. the same value as *install_directory*).
  - If the *install_directory* points to an addon, set this value to the addon path (e.g. "addons/<addonName>/").

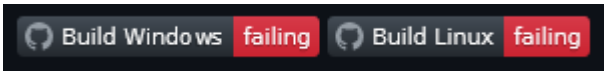
Example:

```
{
    "name": "Chromium",
    "module_name": "pr_chromium",
    "install_directory": "addons/chromium_browser/modules/chromium/",
    "release_directory": "addons/chromium_browser/"
}
```

When the module has been built, the binary will be located in "pragma/addons/chromium_browser/modules/chromium/pr_chromium.dll" (or "libpr_chromium.so" for Linux).

The GitHub release will contain all files that are within "pragma/addons/chromium_browser/". Users will be able to simply extract the release over their Pragma installation to install the module.

Once you have edited the values in the json-file, commit your changes. This will trigger a workflow which will initialize the repository with your values, which should take about two minutes. Simply refresh the page a few times, once you can see the following icons at the top of the readme, it means the initialization was completed:



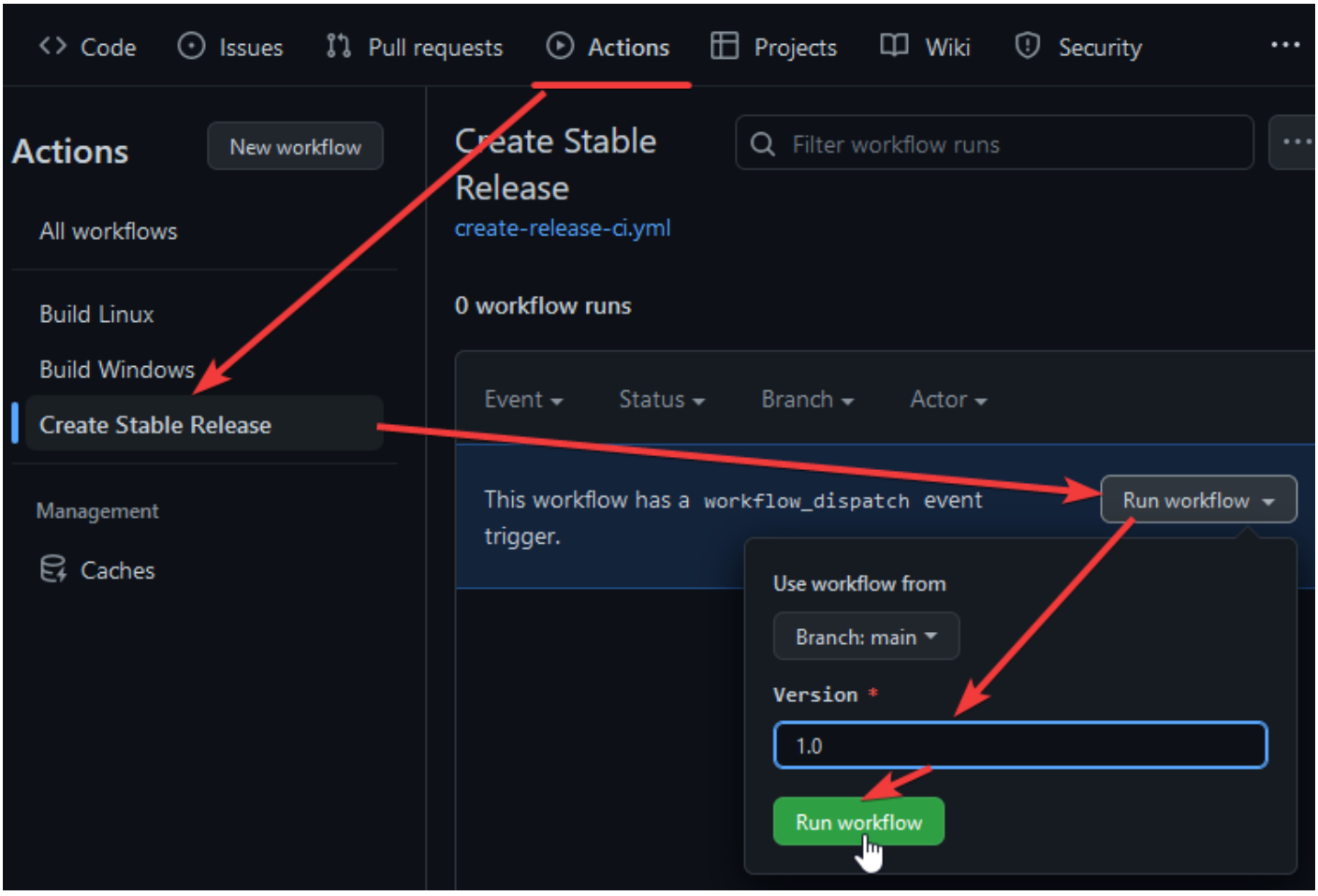
The initialization will also trigger the automated build workflows, which may take a few hours to run. Once they have completed, the two icons should turn green, and you should find the compiled binary files available for download for both Windows and Linux in the "Releases" section of the repository.

The build workflows will automatically trigger whenever you push any new commits to the repository and the release binaries will be updated every time. If there are any errors during the build process, the icons will say "failing" again, and you can check the workflow log for more information on what caused the failure.

Once you've set up the repository, you can follow the instructions in the section about building modules to add the module to the pragma build.

Stable Releases

If your module has reached a state that can be considered "stable", you can publish a stable release. To do so, wait for the regular build workflows to complete, then go to the "Actions" tab of your repository and select the "Create Stable Release" workflow, enter a version number and click "Run workflow":



This workflow should only take a minute, as it just publishes the latest binaries as a new release.

Testing

You can find a simple test Lua-script in the "examples" directory of your module. Follow the instructions in the commented section of the script to install and run it to test the module.

Advanced Building

If your module requires additional steps (such as downloading and building external dependencies) before it can be built, you can add those steps to the python script in "build_scripts/setup.py". This script will be executed automatically whenever the Pragma build script is run.

Advanced Installation

By default the module binary file will be installed to the "modules" directory of the Pragma installation. If you want to change this behavior, you can do so by editing the "CMakeInstall.txt" file.