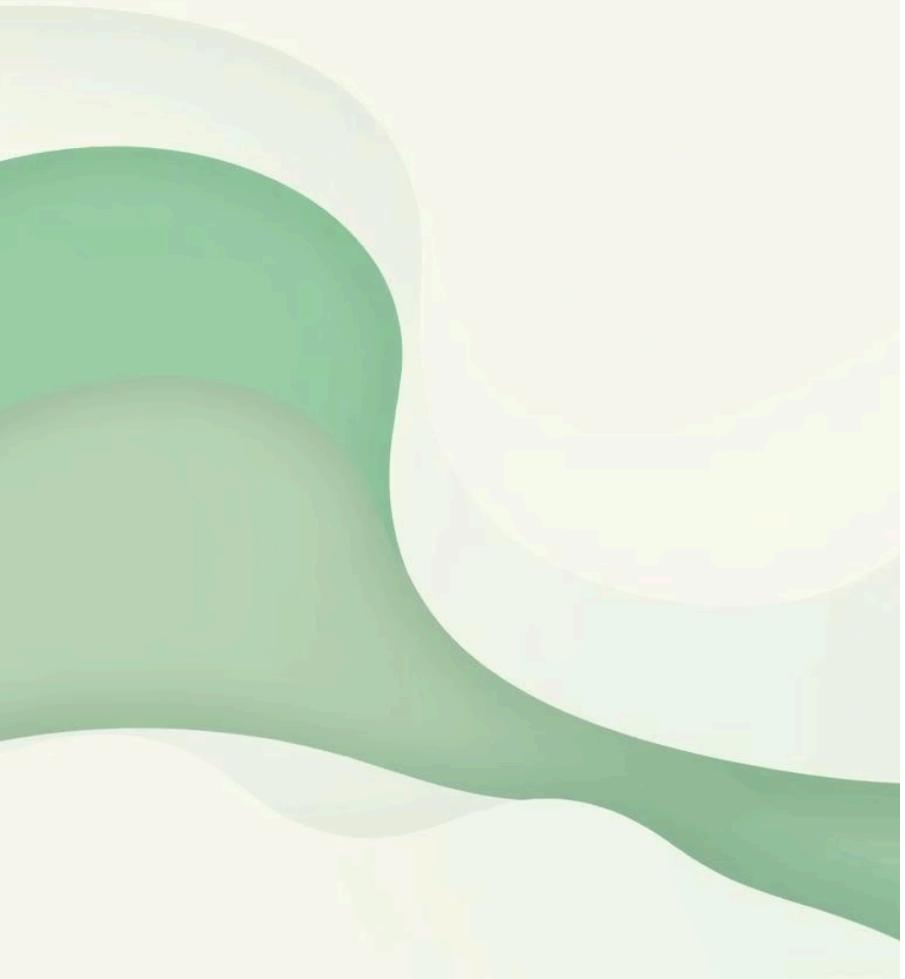

Setup Documentation

This document describes how to set up and build from scratch on Windows and Linux using Clang and the Vulkan SDK. It consolidates the steps shown in the tutorial videos into a clear technical reference.

1. Prerequisites Required Software

Git

Clang / LLVM

LLVM Builds (**Windows installer**)

On Linux: install via your package manager
(sudo apt-get install clang, pacman -S clang, etc.)

Visual Studio Code

Download Here

Install the C/C++ extension and optionally:

- Shader language support
- To-do highlight
- Themes (e.g., Gruvbox)

Vulkan SDK

Download Vulkan SDK

On Windows: run the installer and ensure
"Add LLVM to system path for all users" is selected.

On Linux:

- Download and extract the tarball
- Edit the shell configuration files for all users:

```
sudo nano /etc/zsh/zshrc
sudo nano /etc/bash.bashrc
```

Add at the end of each file:

```
# Vulkan SDK environment for all users

SETUP_ENV="Path_to_your_extracted_Vulkan_SDK/setup-env.sh"

if [ -z "$VULKAN_SDK" ]; then
    if [ -f "$SETUP_ENV" ]; then
        source "$SETUP_ENV"
        echo "Vulkan SDK environment loaded successfully."
    else
        echo "Warning: $SETUP_ENV not found!"
    fi
fi
```

Replace Path_to_your_extracted_Vulkan_SDK with the full path to your Vulkan SDK folder.

This ensures VULKAN_SDK is set for all users automatically.

Ensure VULKAN_SDK environment variable is set.

Additional Linux Dependencies

On Ubuntu/Debian-based distributions:

```
sudo apt-get install libx11-dev libxkbcommon-x11-dev libx11-xcb-dev
```

Other distros: install equivalent X11/XCB development libraries.

2. Repository Setup

Create a project folder:

```
git clone https://github.com/nourOkaram/Kaffi
```

Simplify folder structure if needed (move files up one level).

Ensure you have a code/ folder containing the repository.

3. Project Structure/

```
├─── engine/
│    ├─── src/
│    │    ├─── defines.h
│    │    ├─── test.h
│    │    └─── test.c
│    └─── build.bat | build.sh
├─── testbed/
│    ├─── src/
│    │    └─── main.c
│    └─── build.bat | build.sh
├─── bin/ # Build outputs
├─── Doxy/ # Doxygen generated documentation
├─── Docs
│    ├─── Introducing Kaffi.pdf
│    └─── Setup.pdf
├─── buildall.bat | buildall.sh
├─── Doxyfile
└─── .vscode/
   ├─── c_cpp_properties.json
   ├─── keybindings.json
   ├─── launch.json
   ├─── tasks.json
   ├─── settings.json
   └─── extensions.json
```

4. Build Scripts Windows – engine/build.bat

Uses clang to compile engine sources into engine.dll and engine.lib.

Links against:

- user32.lib
- vulkan-1.lib

Defines:

- _DEBUG, KEXPORT, _CRT_SECURE_NO_WARNINGS.

Linux – engine/build.sh

Compiles into libengine.so.

Links against:

- vulkan, xcb, x11, x11-xcb, xkbcommon.

Defines: _DEBUG, KEXPORT.

⚠ ⚠️ **Note:** Make build scripts executable on Linux:

```
chmod +x engine/build.sh testbed/build.sh build-all.sh
```

Testbed – testbed/build.bat | build.sh

Builds the testbed application.

Links against engine.lib (Windows) or libengine.so (Linux).

Linux build script uses -Wl,-rpath,. so runtime libraries are found locally.

Build All – buildall.bat | buildall.sh

Runs both engine and testbed build scripts.

Stops execution if an error occurs.

5. Core Code engine/source/defines.h

Defines fixed-width integer types (u8, i32, f32, etc.).

Static assertions ensure type sizes match expectations.

Platform detection for Windows, Linux, Unix, Android, Apple.

KAPI macro for function exports (__declspec(dllexport) / __declspec(dllimport)).

engine/source/test.h

```
#pragma once
#include "defines.h"

KAPI void print_int(i32 i);
```

engine/source/test.c

```
#include "test.h"
#include <stdio.h>

void print_int(i32 i) { printf("The number is %i\n", i); }
```

testbed/source/main.c

```
#include <test.h>

int main(void) { print_int(42); return 0; }
```

6. VS Code Setup c_cpp_properties.json

Configures IntelliSense for Windows and Linux.

Specifies include paths (engine/source, Vulkan SDK).

Defines DEBUG, KEXPORT.

Compiler path set to Clang.

tasks.json

Defines tasks:

- Build Engine
- Build Testbed
- Build All

launch.json

- Windows: Uses cppvsdbg, runs testbed.exe.
- Linux: Uses cppdbg (GDB), runs testbed.

extensions.json

Recommends project-specific VS Code extensions.

7. Building and Running Build

Windows

```
Run build-all.bat
OR
Ctrl + Shift + B
```

Linux

```
./buildall.sh
OR
Ctrl + Shift + B
```

Run in Debug Mode F5

- On Windows: new external console opens.
- On Linux: output appears in the VS Code terminal.

8. Build Outputs

Inside bin/:

Windows:

- engine.dll → Engine library
- engine.lib → Import library
- engine.pdb → Debug symbols
- testbed.exe → Test application
- testbed.pdb → Debug symbols

Linux:

- libengine.so → Shared object library
- testbed → Test application

9. Verification

- Set a breakpoint in main.c at the print_int(42) call.
- Run debugger (F5).
- Step into (F11) to follow into the DLL/SO.
- Verify console output:

```
The number is 42
```

10. Doxygen Quick Setup

Windows

Download the installer from the official Doxygen website:

https://www.doxygen.nl/download.html

Run the installer and follow the on-screen steps. Once installed, open Command Prompt and execute the following command to generate documentation:

```
doxygen path\to\Doxyfile
```

Linux

Install Doxygen using your distribution's package manager:

```
sudo apt-get install doxygen  # Debian/Ubuntu
sudo yum install doxygen      # CentOS/Fedora
```

After installation, generate documentation by running Doxygen with your `Doxyfile`:

```
doxygen /path/to/Doxyfile
```

💡 **Note:** This quick setup focuses on downloading and running Doxygen with an existing `Doxyfile`. No additional configuration is required within Doxygen itself for basic operation.

1. Next Steps

With the project scaffolding complete for both Windows and Linux, the next videos will cover platform-agnostic low-level engine components.