Prerequisites

In order to run code on the reference hardware, we need the ability to cross-compile for the given target architecture. ARM/Linux cross-compilers require a Linux computer in order to build the code. In this guide we will walk through how to set up Linux, cross-compilers, and F´ across the variety of computers students may be using.

> Please come to the class having worked through and succeeded at each of the steps below.

Step 0: Basic Requirements

These are the basic requirements for students' computers to work through this section and attend the class.

1. Computer running Windows 10, Mac OS X, or Ubuntu
2. Administrator access
3. 5GB of free disk space, 8 GB of RAM
4. Knowledge of the command line for your operating system (Bash, PowerShell, etc).

The following steps elaborate on the F´ installation guide with specifics for the F´ system reference used by this class. We include a breakdown of setup for each common OS type. Users may consult the troubleshooting section of the installation guide if problems arise.

Step 1: Setting Up An Ubuntu Virtual Machine and Necessary Packages

In this section we will set up Linux to run on each type of laptop. This is typically done through the use of virtual machines and/or emulation. Please follow the instructions for the operating system you run on your laptop. Our goal will be to get up and running with Ubuntu 20.04 on each type of laptop. *This does not require dual booting.*

Note: you will need one of the operating systems found here. If you run something else, try setting up an Ubuntu 20.04 virtual machine in something like VirtualBox.

Microsoft Windows 10

Windows 10 ships with a technology known as WSL. WSL allows users to run Linux virtual machines transparently within the Windows 10 operating system. In order to install WSL please run the following commands *in an Administrator PowerShell console*.

PowerShell: Install WSL with Default Ubuntu

```
wsl --install
```

To start Ubuntu under WSL, search for Ubuntu in the start menu and select the "Ubuntu 20.04 on Windows" APP. All class commands should be run these Ubuntu 20.04 terminals.

Note: full instructions and troubleshooting helpe is available in the Microsoft documentation.

Once WSL is running, you can install the base F´ required packages from the Ubuntu shell using:

```
sudo apt install build-essential git g++ gdb cmake python3 python3-venv python
```

The cross-compilers are installed using a similar command:

```
sudo apt install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu binutils-aarch64-
```

Finally, Windows users must open up a firewall port and forward that port to WSL to ensure the hardware can call back into F´ ground data system running in WSL. First we'll need to note the IP address of the WSL machine. This is done with the following command *in an administrator PowerShell*.

```
wsl hostname -I
```

> Record the output of this command for the next step. For this guide, we will use the value `127.0.0.1` .

Next, we will add a firewall rule and forward it to the WSL instance. This is done with the following commands in *an administrator PowerShell*. Remember, to chane `127.0.0.1` to the address you noted above.

> Warning: these commands work with the Windows firewall. Security and anti-virus tools can run extra firewalls. Users must allow the port `50000` or disable these extra firewalls.

PowerShell: Add and Forward External Firewall Rule

```
New-NetFirewallRule -DisplayName "fprime" -Direction inbound -Profile Any -Act
netsh interface portproxy add v4tov4 listenport=50000 listenaddress=0.0.0.0 co
```

> Remember to change `127.0.0.1` to your recorded ip address as discovered with `wsl hostname -I` Users are advised to remove this rule after the class has been completed.

Ubuntu 20.04 / Linux

Ubuntu users just need to ensure that the basic packages and cross-compilers are installed on their system. The basic packages that F´ requires are installed with:

```
sudo apt install build-essential git g++ gdb cmake python3 python3-venv python
```

The cross-compilers are installed using a similar command:

```
sudo apt install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu binutils-aarch64-
```

Mac OS X

MacOS like Linux is a unix system and thus may be used directly for most of this class. However, Mac users must install the following utilities *and ensure they are available on the command line path*.

1. Python 3
2. CMake
3. GCC/CLang typically installed with xcode-select

Installing GCC/CLang on macOS

```
xcode-select --install
```

Installing Python and running the above command to install gcc/CLang should ensure that those tools are on the path. CMake requires one additional step to ensure it is on the path:

```
sudo "/Applications/CMake.app/Contents/bin/cmake-gui" --install
```

In order to cross-compile for the ARM chips for this class, a Linux box is essential. Mac users may choose to pair with team members who have Linux/WSL setup, or may choose to follow the instructions in Appendix II to install a docker container including the necessary tools.

Step 2: Cloning The F´ System Reference

This class activity will be taught within the F´ System Reference project. The F´ System Reference can be downloaded by and initialized with the following commands.

```
cd ~
git clone https://github.com/fprime-community/fprime-system-reference
cd fprime-system-reference
git submodule update --init
```

> This class assumes we work within the home directory. Choosing differently means you **must** be prepared to adjust all commands with your chosen path.

> WSL users should clone within an Ubuntu shell. Using git on Windows is known to cause file line ending problems.

Step 3: Installing F´ Tools

Each version of F´, including the one shipped within the system reference is stamped with a known working set of F´ tools. These tools run in Python and are typically setup in a virtual environment to prevent issues with your OS.

Set Up a Virtual Environment

```
python3 -m venv ~/class-venv
```

Activate The Virtual Environment

```
. ~/class-venv/bin/activate
```

Update Python Packages

```
pip install -U setuptools setuptools_scm wheel pip
```

Install F´ Tools

```
pip install -r ~/fprime-system-reference/fprime/requirements.txt
```

The F´ virtual environment is now setup with the correct version of the tools for the class. Should the version of F´ used by the class change, just rerun the installation command from within the activated environment.

> Users **must** be within an activated virtual environment for the remainder of the class. When opening up a new shell or terminal simply repeat the activation command above. There is no need to set up nor install again, simply activate.

Step 4: Testing the Setup

> **Reminder**: F´ tools need to be run from within an activated virtual environment. WSL users must also run from within an Ubuntu shell.

In this section we will build the F´ system reference for your local computer and run it against the F´ ground data system to ensure that all tools are set up correctly. If an error is encountered, please reference Appendix III for help.

First we generate a build cache for F´. This is a required first step, but is only needed to be run once for each target architecture. Notice we perform this work within the SystemReference folder as that is our F´ deployment.

Generate an F´ Build Cache

```
cd ~/fprime-system-reference/SystemReference
fprime-util generate
```

Next the user should build F´. Building is repeated anytime a file is changed and the user wishes to rebuild the model or code.

```
cd ~/fprime-system-reference/SystemReference
fprime-util build
```

Finally, the user can run F´ and attach it to an F´ ground data system instance to see the working code. **Note:** success in this step is seeing the upper right corner of the GDS change to a green circle.

```
cd ~/fprime-system-reference/SystemReference
fprime-gds
```

You should now be prepared for the completing the remaining IMU exercise.