ESA - ComMoDE (Building instructions)

LINUX Systems - building and installation

In order to build the core-API model library and integration test you need:

- Compliant distribution: **Ubuntu** LTS >=20.04 tested, **RedHat** stream based (official RHEL>= 8 tested. Not explicitly tested for Fedora >=28 and freeware LTS like CentOS stream, Rocky Linux and AlmaLinux but likely to work and perform as official RHELs). Tested explicitly on older CentOS7 (see dedicated section in this document). Other Linux distro are not officially supported. Please consider to update your current system before any other operation. Ubuntu users:

  ```
  >>sudo apt update
  >>sudo apt upgrade
  ```

  RedHat/CentOS/Fedora or similar via yum:

  ```
  >>sudo yum install epel-release
  >>sudo yum update
  ```

- Compliant environment for building C/C++ code, including GNU-GCC/G++ compiler (gcc version >= 4.8.3) and make. Ubuntu users can retrieve it with apt:

  ```
  >>sudo apt install build-essential
  ```

  RedHat/CentOS/Fedora or similar via yum:

  ```
  >>sudo yum groupinstall "Development Tools"
  ```

- python3.x distribution (version >= 3.8.x, , <= 3.10.x, https://www.python.org/). Most common recent Linux-OS should already have a compliant python3 version installed by default or reachable via apt/yum. Ubuntu :

  ```
  >>sudo apt install python3-dev
  ```

  RedHat/CentOS/Fedora or similar (Centos7 users please read the appendix **Install ComMoDE compliant python3 on Centos7** reported later):

  ```
  >>sudo yum install python3-devel
  ```

- python distribution must have a valid **pip** and **venv**(virtualenv) already installed. See https://packaging.python.org/en/latest/guides/installing-using-pip-and-virtual-environments/ .Ubuntu users can retrieve them with apt using python3-pip and python3-venv
- wget utility (available with apt/yum)
- cmake (version >= 3.12)(https://cmake.org/download/, or via apt/yum)
- doxygen + graphviz (version >= 1.8.5) (https://www.doxygen.nl/download.html or apt/yum installable)

Procedure to build and install the code:

1. Obtain the code zip package and unzip it (or git clone it, if available from the github remote).
2. Open a shell and access the project folder location.
3. Identify the build script file **build.sh** and be sure to have execution permission on it. If not, you can activate it with:

   ```
   >>chmod a+x build.sh
   ```

4. The build script will expose the following arguments to customize installation:

   - **-prefix** <full custom installation path>: optional, it will allow the User to specify the folder where the software will be installed. If not expressed, the default folder $HOME/ESA-ComMoDE_v1.0 will be used. Since no uninstall mechanism is provided, we officialy advise against the use of default system folders /usr, /usr/local, /opt and similar as installation prefix, to avoid as much as possible polluting your current OS.
   - **-python** <full path to python3 interpreter>: optional, it will allow the user to specify the path to the python3 interpreter to be used in ComMoDE building. The option is useful to specify the correct python in case of OS supporting multiple python distributions. If not expressed, the script will use the python3 command available in the shell.
   - **-enableDoc** : optional, if expressed it will build and install the ComMoDE doxygen documentation. If not, no documentation will be provided.

     For example, to automatically build the code with all default option (prefix $HOME/ESA-ComMoDe_v1.0, shell python3 and documentation disabled) :

     ```
     >>./build.sh
     ```

     Otherwise to customize prefix, python command and enabling doc:

     ```
     >>./build.sh -prefix /home/johndoe/Desktop/myComMoDeInstallation -python /usr/bin/python3.9 -enableDoc
     ```

     Or customizing just the prefix, but using default shell python3 and no doc:

     ```
     >>./build.sh -prefix /home/johndoe/Desktop/myComMoDeInstallation
     ```

   The current script will perform a lot of actions underground. The most important are:

   - create a temporary build folder *build* inside the project folder to compile source code
   - automatical build boost-python compliant with your current python interpreter, retrieving sources from boost.org remote with wget
   - build the ComMoDE c++ native coreAPI and create its python bindings as a python wheel PyComMoDE
   - build documentation with doxygen if enableDoc is activated.
   - install the libraries into the chosen installation prefix
   - set up a custom python environment dedicated to ComMoDE (ComMoDE-env) with all needed python packages (setuptools, wheel, PyComMoDE, numpy, matplotlib and PySide2)
   - run the functional tests.

   The execution time can be long (5-10 minutes), expecially the first time, because of the boost building. You can follow the status of boost installation checking the boost_install.log inside the build dir.

5. A first example of c++ integration test (core_integration_test) will be found in subfolder **bin** of installation path. To run it, access the bin folder and type:

   ```
   >>./core_integration_test
   ```

6. A second example of python binded integration test (core_integration_test_binded.py) will be found in subfolder **bin** of installation path. To run it, access the bin folder and activate the python environment dedicated to ComMode:

   ```
   >>source ./ComMoDE-env/bin/activate
   ```

   If the enviroment is ready, a decoration (ComMoDE-env) will appear on the shell row like:

   ```
   (ComMoDe-env)>>
   ```

   Then to run the binded test:

   ```
   (ComMoDe-env)>> python core_integration_test_binded.py
   ```

   To deactivate the ComMoDE environment:

   ```
   (ComMoDe-env)>> deactivate
   ```

7. A full functional ComMoDE GUI interface is available in the subfolder **gui** inside installation directory, but first the python environment dedicated to ComMode in the **bin** subfolder must be activated. Go in the installation directory:

   ```
   >>source ./bin/ComMoDE-env/bin/activate
   ```

   If the enviroment is ready, a decoration (ComMoDE-env) will appear on the shell row like:

   ```
   (ComMoDe-env)>>
   ```

   Then to run the gui:

   ```
   (ComMoDe-env)>> cd gui
   (ComMoDe-env)>> python ComMoDE-gui.py
   ```

   To deactivate the ComMoDE environment:

   ```
   (ComMoDe-env)>> deactivate
   ```

   For problems visualizing the ComMoDe GUI please consult the appendix **PySide2 Troubleshoots Rendering on Linux OS** for some practical hints.

8. For developers only, you can recompile and install any successive modification of the core-API code re-running as much time you want the "build.sh -prefix ..." script: this will rebuild the code with the new modifications and install it to the prescribed prefix. Please note the boost building and installation, if successfull the first time, it will be safely skipped.

Install ComMoDE compliant python3 on Centos7

Default python3 available in Centos7 repository is too old (3.6.x) for the ComMoDe app. The most straightforward way is to download a more recent version of python and compile it from scratch onto the Centos7 system. Below is reported in detail the procedure to download a python 3.8.12 from python.org ftp repository and install onto your system step by step. Please note: the same procedure can be reused to install a different python version (simply changing the version of the target package 3.8.12) and on other linux systems (finding the corrispective required system packages, here specified for centos7/yum, for ubuntu can be named differently.)

1. Generally Update the system :

   ```
   >>sudo yum -y install epel-release
   >>sudo yum -y update
   ```

2. Install Development Tools, that is the gcc/g++ compiler and related utilities to compile and build c/c++ code.

   ```
   >>sudo yum -y groupinstall "Development Tools"
   ```

3. Install other packages, essential to build python

   ```
   >>sudo yum -y install openssl-devel bzip2-devel libffi-devel xz-devel
   ```

4. Install wget and download the Python-3.8.12.tgz archive

   ```
   >>sudo yum -y install wget
   >>wget https://www.python.org/ftp/python/3.8.12/Python-3.8.12.tgz
   ```

5. Expand the Python-3.8.12.tgz archive and access the expanded folder

   ```
   >>tar xvf Python-3.8.12.tgz
   >>cd Python-3.8.12
   ```

6. Run the configure script and install python. Please do not forget the --enable-shared option, otherwise a static Python .a lib will be produced. Static libraries create conflicts building the ComMoDe wheel.

   ```
   >>./configure --enable-optimizations --enable-shared
   >>sudo make altinstall
   ```

   If installation runs fine, the interpreter will be installed in /usr/local/bin/ with name python3.8. The final python libraries will be installed in /usr/local/lib. Please check there's no libpython3.8.a static libraries in it. If it is there, remove it and repeat the procedure using --enable-shared.

7. Python3.8 libraries must be visible on the shell. To do so you can export the /usr/local/lib in the LD_LIBRARY_PATH as:

   ```
   >>export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
   ```

   or to make it permanent edit your $HOME/.bashrc file and copy the above string into it. Then to enable modifications on bashrc, close/open again the shell or type:

   ```
   >>source $HOME/bashrc
   ```

8. Finally if typing the command /usr/local/bin/python3.8 you can access the python shell, you're good to go. As side note, building python from scratch will automatically enable pip and virtualenv installation alongside it.

PySide2 Troubleshoots Rendering on Linux OS

The current section should be ignored if the your OS is currently mounting a valid Graphical User Interface like GNOME, KDE or similar, that means your OS is already equipped to renderer and visualize Graphical apps (in simply words you have a desktop with icons and you can interact with them). Usually the ComMoDE-GUI (base on PySide2 bindings to QT to create the graphical part) fails to start for missing dependencies on:

- GL libraries (base libraries for rendering and interacting with graphical objects), usually coming updating drivers connected to your graphical borad.
- pulseaudio suite (is the standard linux audio server)
- server X (that makes rendering of graphical object possible) and its xcb client (bindings that makes possible calls to X server from C code, that is the application you're trying to run)

A system with an already installed and functioning GUI environment should have all of these at disposal, but sometimes things don't go smoothly. Usually there are some very specific libraries missing onto your system, and there is no specific recipe to fix all the problems. We found out that the most robust method to track down any missing library is to have a lot of patience and do the following:

1. Export the local variable QT_DEBUG_PLUGINS as 1, typing in the shell

   ```
   >>export QT_DEBUG_PLUGINS=1
   ```

2. Run the ComMoDe GUI as at point 7) of **LINUX Systems - building and installation** section
3. If something is missing the Python interpreter will raise an error of missing library and/or the xcb section enabled by QT_DEBUG_PLUGINS will complain of a missing library. You need to detect it, and install the correspondent package that provides that library into your system.

Once all the missing libraries are recovered, everything should runs fine eventually.

WSL USERS: If you are using WSL/WSL2 linux sub-systems run from a Windows host, an effective fix (that covers the majority of troubles) is to install on Windows Host a VCXsrv application (https://sourceforge.net/projects/vcxsrv/) and follow istructions reported in this hack here: https://github.com/microsoft/WSL/issues/4793#issuecomment-577232999