ESA - ComMoDE (Building instructions)

WINDOWS Systems - MSVC Native building and installation

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

- Compliant **Windows** 10/11 Desktop or higher. Never tested on Windows Server distributions.
- Visual Studio Community suite with C/C++ build essential tools enabled, for x64 architectures. (tested with Community Visual Studio 17 2022). (https://visualstudio.microsoft.com/). Beware: it is essential to install the workload to develop C++ desktop application. Without it, you will not be able to use the cl.exe compiler(see later). In order to do so visit this link: https://learn.microsoft.com/en-us/visualstudio/install/modify-visual-studio?view=vs-2022: run the Visual Studio Installer, click Modify, then on tab Workloads select the "Desktop Development with C++". Click on modify on the bottom of the page and wait until the update is finalized.
- cmake (version >= 3.22) (https://cmake.org/download/)
- doxygen + graphviz (version >= 1.8.5) (https://www.doxygen.nl/download.html)
- python3 distribution (version >= 3.8.x, <= 3.10.x https://www.python.org/. Beware: download a native Windows python suite, don't use python coming from emulators like msys2, cygwin, nor interpreters from Microsoft Store etc...). Please also avoid any python installer package with "rc" (meaning "release candidate") suffix.

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 Developer PowerShell for Visual Studio (Main app menu/Visual Studio folder or searching in the search bar "Developer PowerShell"). **It is important to get the PowerShell related to Visual Studio instead of a normal PowerShell**, because in the first case the compiler environment is already loaded in the shell.You can check if it is the right powershell typing:

   ```
   >>cl.exe
   ```

   you should get back info on a valid MSCV compiler found for x64 (64 bit) architecture. Please note: you can load also the right Visual studio developer environment from command line of a regular powershell typing :

   ```
   >> & 'C:\Program Files\Microsoft Visual Studio\2022\Community\Common7\Tools\Launch-VsDevShell.ps1' -Arch amd64 -HostArch amd64
   ```

   Change 2022 in the path, according to the type of Visual Studio you are using (2019, 2017). Further info can be found here: https://learn.microsoft.com/en-us/visualstudio/ide/reference/command-prompt-powershell?view=vs-2022 .

   Please note, if you are not able to get any cl.exe the reasons are only 2: you're taking the wrong powershell (pick the developer powershell!), or you missed to install the "Desktop development with C++" workload in Visual Studio (see before).

3. Move inside the project folder location and identify the build script file **build.ps1** and be sure to have execution permission for it. To check permissions you can run:

   ```
   >>Get-ExecutionPolicy -Scope CurrentUser
   ```

   If the command returns one of AllSigned, Restricted or Undefined, enable the executability of the script typing:

   ```
   >>Set-ExecutionPolicy -ExecutionPolicy ByPass -Scope CurrentUser
   ```

   For futher information on execution policies please consult : https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_execution_policies?view=powershell-7.3

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

   - **-G** <Cmake Visual Studio Generator>: mandatory, it specifies a string needed by cmake to generate a compliant project for Visual Studio. At the moment the recommended option is to use "Visual Studio 17 2022", which corresponds to latest version of VS Community. It is possible to generate project for early versions of Visual Studio like 16 or 15, passing respectively "Visual Studio 16 2019" and "Visual Studio 15 2017". It is strongly recommended to avoid older distros. Pick up the string corresponding to the right version of Visual Studio actually in use onto your machine.

   - **-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 ~\ESA-ComMoDE_v1.0 will be used (in windows sounds like C:\Users\johndoe\ESA-ComMoDE_v1.0).

   - **-python** <full path to python3 interpreter>: mandatory, it will allow the user to specify the path to the python.exe interpreter to be used in ComMoDE building. Usually Windows collect all python interpreters in paths like C:\Users\johndoe\AppData\Local\Programs\Python\PythonXYY\python.exe, where XYY is the number formed by the python Major and Minor version (for example 3.10.x version will be found in ...Python\Python310\python.exe). If not expressed, the script will exit requiring a valid python command path. **BEWARE**: having the Python command exported to User **Path** variable will inhibit the correct building and installation of the boost libraries onto your system. So if the python interpreter that you intend to use was exported to Path during python installation stage or explicitly manipulating the Environment variable, this will subtly cause your boost installation to fail. Unfortunately there are no hack nor workround known at the moment, except for momentarily erasing the python paths C:\Users\johndoe\AppData\Local\Programs\Python\PythonXYY and C:\Users\johndoe\AppData\Local\Programs\Python\PythonXYY\Scripts from the Path environment variable. Before running the build.ps1 script, we strongly recommend to :

     - Go to toolbar "Search" and digit "System Environment Variables"
     - Click on "Edit the System Environment Variables" -> Environment Variables button
     - Select **Path** on "User variables for johndoe" and click **Edit**
     - Remove the entry related to Python interpreter you intend to use: C:\Users\johndoe\AppData\Local\Programs\Python\PythonXYY, C:\Users\johndoe\AppData\Local\Programs\Python\PythonXYY\Scripts
     - Those can be restored manually into the Path Enviromnent Variable **after** the ComMoDE building is finished.

   - **-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 and a custom python 3.10 (prefix C:\Users\johndoe\ESA-ComMoDe_v1.0 and documentation disabled) :

     ```
     >>.\build.ps1 -python C:\Users\johndoe\AppData\Local\Programs\Python\Python310\python.exe
     ```

     Otherwise to customize prefix, and enabling doc:

     ```
     >>.\build.ps1 -prefix C:\Users\johndoe\Desktop\myApps\myComMoDeInstallation -python
     C:\Users\johndoe\AppData\Local\Programs\Python\Python310\python.exe -enableDoc
     ```

     Or customizing just the prefix, with no no doc:

     ```
     >>.\build.ps1 -prefix C:\Users\johndoe\Desktop\myApps\myComMoDeInstallation -python
     C:\Users\johndoe\AppData\Local\Programs\Python\Python310\python.exe
     ```

   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 Invoke-WebRequest
   - 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 dll/pyd 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 integration test (core_integration_test) will be found in subfolder **bin** of installation path. To run it, access the bin folder from a normal powershell and type:

   ```
   >>.\core_integration_test.exe
   ```

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, from a regular powershell access the bin folder and activate the python environment dedicated to ComMode:

   ```
   >>.\ComMoDE-env\Scripts\activate
   ```

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

   ```
   >>.\bin\ComMoDE-env\Scripts\activate
   ```

   If the enviroment is ready, a decoration (ComMoDE-env) will appear on the powershell 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
   ```

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.ps1 -G ..." 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.