In the steps below, we will use apt install to install a package on distributions based on Debian Linux. The command might be different on your distribution, e.g.:
Fedora, CentOS, RHEL or openSUSE: dnf install
Mageia: urpmi.

Prerequisites

Here is a list of tools and libraries that you will need to build and run the TELEMAC-MASCARET system.

Mandatory

To run TELEMAC-MASCARET the following software are mandatory:

- Python 3.7+
- Numpy 1.15+
- A Fortran compiler (GFortran 4.6.3 is the minimum)

Optional

Other dependencies are required to enable some features:

Parallelism

This allows the TELEMAC-MASCARET system to run in parallel which quite speeds up the execution of a case.

- MPI Distribution
- METIS library

Other partitioners

A partitioner is necessary when building TELEMAC-MASCARET with parallelism. During a parallel run, the mesh is split in multiple parts using the PARTEL module, which relies on METIS as the default partitioner.
However, another library can be used instead of METIS:

- SCOTCH library (check their website for information on how to compile)

A parallel version of PARTEL has been developed but is still experimental. It uses parallel partitioning instead of sequential and as such, requires a parallel implementation of METIS or SCOTCH:

- ParMETIS library (check their website for information on how to build it)
- PT-SCOTCH library (check their website for information on how to build it)

Additional Python libraries for validation

- SciPy
- Matplotlib

MED format support

This allows the TELEMAC-MACARET system to read and write files using the MED format.

- HDF5 library
- MED library

Parallel direct solver

This adds a parallel solver called MUMPS which is only used by the ARTEMIS module at the moment (option SOLVER = 9).

- MUMPS (check their website for information on how to compile)

AED2 support

This activates more processes in the WAQTEL module.

- AED2 library

GOTM support

This allows the use of the General Ocean Turbulence Model in the TELEMAC-3D module.

- GOTM library

Get the source code

The TELEMAC-MASCARET system is distributed in two ways that are described below.

Download the latest version

You can download a compressed archive of the latest stable release (v8p4r0), including the source code, documentation, and examples, from here.

Clone from the GitLab server

The TELEMAC-MASCARET source code repository is hosted on a GitLab server (see this page to get more information).

To check if you have Git installed on your computer run:

```
$ git --version
```

If this command does not work, follow the instructions on the link above to install Git.
If your connection is using a proxy, follow go these instructions first.

To clone the repository, enter:

```
$ git clone https://gitlab.pam-retd.fr/otm/telemac-mascaret.git
my_opentelemac
```

To install it somewhere else, replace `my_opentelemac` by the directory where you want to install it.

To install the latest version, enter the following commands after cloning:

```
 $ cd my_opentelemac
$ git checkout tags/v8p4r0
```

(optional) Proxy

If your connection is using a proxy, you can specify it using the following commands:

```
 $ git config --global http.proxy
https://<username>:<password>@<proxy_address>:<proxy_port>
$ git config --global https.proxy
https://<username>:<password>@<proxy_address>:<proxy_port>
```

Replace:

- **<proxy_address>** by the address to your proxy
- **<proxy_port>** by the port of your proxy
- **<username>** by the login to your proxy if any
- **<password>** by the password for your proxy if any

Installation of TELEMAC-MASCARET

In this part we describe how to install the TELEMAC-MASCARET system.
We will use the following terms (which will be explained in more details further on):

- **<root>** means the path to your TELEMAC-MASCARET sources.
- **<systel.cfg>** means your build configuration file.
- **<config>** points to the build configuration you are using.
- **<pysource>** refers to your environment file.

In the example below, we will install TELEMAC-MASCARET in `~/telemac-mascaret` (**<root>**) with two configurations (**gfortran** and **gfortranHPC**). All the files for these examples can be found at the end of each section.

Setting up your environment

This part explains how to create the **<pysource>** script.

To ease the loading of the TELEMAC-MASCARET environment, we suggest using a file to set environment variables. You can find an example of such a file in `<root>/configs/pysource.template.sh`, that you can copy and edit.
It sets the following environment variables:

- HOMETEL: path to your installation (<root>)
- SYSTELCFG: path to yout configuration file (<systel.cfg>)
- USETELCFG: name of your configuration (<config>)
- SOURCEFILE: path to this file (<pysource>)

Set those four variables in your <pysource> to comply with your installation. Here is an example of the modified lines values:

```
export HOMETEL=$HOME/telemac-mascaret
export SYSTELCFG=$HOMETEL/configs/systel.cfg
export USETELCFG=gfortranHPC
export SOURCEFILE=$HOMETEL/configs/pysource.gfortranHPC.sh
```

We also recommend that you add information from prerequisites in there as well:
For example for metis add :

```
### METIS ------------------------------------------------------------
export METISHOME=~/opt/metis-5.1.0
```

Here is what the file looks like for gfortranHPC:

pysource.sh

```
# This file is a template for a Linux environment file
# running "source pysource.template.sh" will position all
# the necessary environment variables for telemac
# To adapt to your installation replace word <word> by their local
value
###
### TELEMAC settings
------------------------------------------------------------
###
# Path to telemac root dir
export HOMETEL=$HOME/telemac-mascaret/v8p4r0
# Adding python scripts to PATH
export PATH=$HOMETEL/scripts/python3:.:$PATH
# Configuration file
export SYSTELCFG=$HOMETEL/configs/systel.edf.cfg
# Name of the configuration to use
export USETELCFG=gfortranHPC
# Path to this file
export SOURCEFILE=$HOMETEL/configs/pysource.gfortranHPC.sh
### Python
# To force python to flush its output
export PYTHONUNBUFFERED='true'
### API
export PYTHONPATH=$HOMETEL/scripts/python3:$PYTHONPATH
export LD_LIBRARY_PATH=$HOMETEL/builds/$USETELCFG/lib:$HOMETEL/
builds/$USETELCFG/wrap_api/lib:$LD_LIBRARY_PATH
export PYTHONPATH=$HOMETEL/builds/$USETELCFG/wrap_api/lib:
$PYTHONPATH
###
### EXTERNAL LIBRARIES
------------------------------------------------------------
###
### METIS
------------------------------------------------------------
export METISHOME=~/opt/metis-5.1.0
```

If you intend to use only one configuration of the TELEMAC-MASCARET system, e.g. gfortranHPC, you can add this line to your `~/.bashrc`:

```
source $HOME/telemac-mascaret/configs/pysource.gfortranHPC.sh
```

(optional) Adding MED support

You need to add the following lines for both MED and HDF5 if they were not installed using your distribution package manager:

```
### HDF5 ------------------------------------------------------------
export HDF5HOME=~/opt/hdf5-1.10.7
export LD_LIBRARY_PATH=$HDF5HOME/lib:$LD_LIBRARY_PATH
### MED -------------------------------------------------------------
export MEDHOME=~/opt/med-4.1.0
export LD_LIBRARY_PATH=$MEDHOME/lib:$LD_LIBRARY_PATH
export PATH=$MEDHOME/bin:$PATH
```

(optional) Adding SCOTCH support

You need to add the following information for SCOTCH if it was not installed using your distribution package manager:

```
### SCOTCH -----------------------------------------------------------
export SCOTCHHOME=~/opt/scotch-6.0.0/build
export LD_LIBRARY_PATH=$SCOTCHHOME/lib:$LD_LIBRARY_PATH
```

(optional) Adding AED2 support

You need to add the following information for AED2:

```
### AED2 -------------------------------------------------------------
export AED2HOME=~/opt/aed2-1.2.0/build
export LD_LIBRARY_PATH=$AED2HOME/lib:$LD_LIBRARY_PATH
```

Loading the environment

Once that is done, you can load your environment:

```
$ source pysource.gfortranHPC.sh
```

Configuring TELEMAC-MASCARET

This part explains how to create the **<systel.cfg>** file.
In the folder `<root>/configs` you can find a few configuration files, some may fit your configuration.
Here is a quick description of what some of the existing configurations handle:

- **systel.cis-debian.cfg** Configuration for sequential and parallel configuration on Debian.
- **systel.cis-fedora.cfg** Configuration for sequential and parallel configuration on Fedora.
- **systel.cis-opensuse.cfg** Configuration for sequential and parallel configuration on openSUSE.
- **systel.cis-ubuntu.cfg** Configuration for sequential and parallel configuration on Ubuntu.
- **systel.cis-windows.cfg** Configuration for sequential and parallel configuration on Windows.
- **systel.hrw.cfg** Configurations used by H.R.Wallingford.
- **systel.edf.cfg** Configurations used by EDF, contains parallel and debug configuration for GNU, Intel and NAG on Debian and a couple of configurations for clusters.
- **systel.cis-hydra.cfg** Configuration for a Hydra cluster.

Now we are going to describe how to create one from scratch that contains two configurations:

- **gfortran** A basic serial configuration.
- **gfortranHPC** A basic parallel configuration.

First we list all the configurations available in the file:

```
# _____                                  _____________________________________\\
# ____/ TELEMAC Project Definitions /_____________________________________/\\
#
[Configurations]
configs: gfortran gfortranHPC
```

Then we fill the `general` section that will set the default values for every configuration so we'll not have to repeat parameters that are common with every configuration.

```
#
# General
# Global declarations that are set by default for all the configurations
[general]
language: 2
modules:  system
version:  v8p4
#
options: api
#
f2py_name: f2py3
pyd_fcompiler: gnu95
#
sfx_zip:    .zip
sfx_lib:    .a
sfx_obj:    .o
sfx_mod:    .mod
sfx_exe:
```

```
#
#
val_root:   <root>/examples
#
val_rank:   all
#
mods_all:   -I <config>
#
incs_all:
#
libs_all:
#
cmd_obj_c: gcc -fPIC -c <srcName> -o <objName>
```

Here are a few explanations about what the parameters stand for:

- language: defines the language the TELEMAC-MASCARET system should use (1: French, 2: English).
- modules: the list of modules to compile (system means all. You can remove some by adding `-modulename`).
- version: the version.
- sfx_*: extension of the different files (zipped files, libraries, compiled object files, modules, executables).
- val_root: path of examples for validation.
- val_rank: defaut rank for validation.
- mods_all: include command for TELEMAC-MASCARET modules.
- incs_all: include command for compilation.
- cmd_obj_c: command to compile C files (for MASCARET).
- options: compile additional things (here the TELEMAC-MASCARET API)
- f2py_name: name of the f2py command to use (here f2py3 is the Python 3 version)
- pyd_compiler: name of the compiler used by f2py (you can get that name by running `f2py -c –help-fcompiler`)

Now we add the informations for the gfortran configuration:

```
#
# Gfortran
#
[gfortran]
brief: Gfortran compiler
#
cmd_obj:    gfortran -c -cpp -fPIC -O2 -fconvert=big-endian -frecord-marker=
4 -DHAVE_VTK <mods> <incs> <f95name>
cmd_lib:    ar cru <libname> <objs>
cmd_exe:    gfortran -fPIC -fconvert=big-endian -frecord-marker=4 -lpthread
-lm -o <exename> <objs> <libs>
```

Where:

- brief: is a small description of your configuration
- cmd_obj: is the command to compile an object file from a Fortran file
- cmd_lib: is the command to generate a library

- cmd_exe: is the command to generate an executable

THen we add the gfortranHPC configuration:

```
[gfortranHPC]
brief: GFortran compiler using Open MPI
#
mpi_cmdexec: mpirun -machinefile MPI_HOSTFILE -np <ncsize> <exename>
#
cmd_obj:    mpif90 -c -cpp -fPIC -O2 -fconvert=big-endian -frecord-marker=4
-DHAVE_MPI -DHAVE_VTK <mods> <incs> <f95name>
cmd_lib:    ar cru <libname> <objs>
cmd_exe:    mpif90 -fPIC -fconvert=big-endian -frecord-marker=4 -lpthread
-lm -o <exename> <objs> <libs>
#
libs_all: -L$METISHOME/lib -lmetis
```

The new parameters are:

- mpi_cmdexe: define the command to run an MPI job.
- libs_all: options that will be added to the cmd_exe command mainly links to external libraries.

We can also see that we hace replaced `gfortran` by `mpif90` in the `cmd_exe` and `cmd_lib` and that we have added -DHAVE_MPI in cmd_obj to enable MPI in the sources.

You can find the full file below.

systel.cfg

```
# _____
_____________________________________
# ____/ TELEMAC Project Definitions
/_____________________________________/
#
[Configurations]
configs: gfortran gfortranHPC
# _____
_____________________________________________
# ____/ General /_____________________________________________/
# Global declarations that are set by default for all the
configurations
[general]
language: 2
modules:  system
version:  v8p4
#
options: static api
#
f2py_name: f2py3
pyd_fcompiler: gnu95
#
```

```
sfx_zip:    .zip
sfx_lib:    .a
sfx_obj:    .o
sfx_mod:    .mod
sfx_exe:
#
#
val_root:   <root>/examples
#
val_rank:   all
#
mods_all:   -I <config>
#
incs_all:
#
libs_all:
#
cmd_obj_c: gcc -fPIC -c <srcName> -o <objName>
#
[gfortran]
brief: GFortran compiler
#
cmd_obj:    gfortran -c -cpp -fPIC -O2 -fconvert=big-endian
-frecord-marker=4 -DHAVE_VTK <mods> <incs> <f95name>
cmd_lib:    ar cru <libname> <objs>
cmd_exe:    gfortran -fPIC -fconvert=big-endian -frecord-marker=4
-lpthread -lm -o <exename> <objs> <libs>
#
# Gfortran HPC
#
[gfortranHPC]
brief: GFortran compiler using Open MPI
#
mpi_cmdexec: mpirun -machinefile MPI_HOSTFILE -np <ncsize> <
exename>
#
cmd_obj:    mpif90 -c -cpp -fPIC -O2 -fconvert=big-endian
-frecord-marker=4 -DHAVE_MPI -DHAVE_VTK <mods> <incs> <f95name>
cmd_lib:    ar cru <libname> <objs>
cmd_exe:    mpif90 -fPIC -fconvert=big-endian -frecord-marker=4
-lpthread -lm -o <exename> <objs> <libs>
#
libs_all: -L$METISHOME/lib -lmetis
```

(optional) Adding MED support

Here are the parameters you have to edit to activate MED support :

- incs_all: add -I$MEDHOME/include
- cmd_obj: add -DHAVE_MED
- libs_all: add -L$HDF5HOME/lib -lhdf5 -L$MEDHOME/lib -lmed -lstdc++ -lz

(optional) Adding SCOTCH suport

Here are the parameters you have to edit to activate SCOTCH support:

- incs_all: add -I$SCOTCHHOME/include
- cmd_obj: add -DHAVE_SCOTCH
- libs_all: add -L$SCOTCHHOME/lib -lsctoch -lsctocherr

Note that even if you use SCOTCH, you still need METIS.

(optional) Adding AED2 support

Here are the parameters that you have to edit to active AED2 support:

- incs_all add -I$AED2HOME/include -I$AED2HOME/mod
- cmd_obj add -DHAVE_AED2
- libs_all add -L$AED2HOME/lib -laed2

Compiling TELEMAC-MASCARET

First of all we need to check that the environment is set properly.

First, source your environment using the following command:
```
$ source pysource.gfortranHPC.sh
```

Then, to display your configuration, enter:
```
$ config.py
```

It should display something like that:

```
Loading Options and Configurations
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


... parsing configuration file: /home/yugi/telemac-mascaret/v8p4r0/configs/
systel.cfg


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
gfortranHPC:

    +> GFortran compiler using Open MPI
    +> root:     /home/yugi/telemac-mascaret/v8p0r0
    +> module:   splitsel / mascaret / ad / tomawac / damocles / partel /
postel3d
               / artemis / parallel / diffsel / gretel / api / waqtel /
stbtel
               / bief / sisyphe / nestor / hermes / telemac3d / telemac2d /
special

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


My work is done
```

If everything went fine, you can now build the whole system:
`$ compile_telemac.py`

A successfull build will always end with the following message: `My Work is done`.

Running a job

Now that the compilation is over let us check that it works.

Go into <root>examples/telemac2d/gouttedo And type in the terminal:
`$ telemac2d.py t2d_gouttedo.cas --ncsize=4`

Again, a successfull run will display the message `My Work is done`.

Additional configurations

Here, we give additional parameters to install the TELEMAC-MASCARET system on a cluster and how to compile the Python API.

Compile on a cluster

You will need to follow the installation procedure for a parallel installation first. Then, there are two ways to run on a cluster:

- Running everything on the cluster queueing system.
- Running only the execution of the TELEMAC-MASCARET executable the cluster queueing system. Partitioning is run on a frontal node. The gathering will have to be done by hand.

To use `runcode.py` in a standard mode and not take into account your modification for the cluster, just add –`mpi` to the command.

For the first solution, add the following options:

- `hpc_stdin`: defines your batch file (i.e. the parameter file you use to run a job in queue).
- `hpc_runcode`: the command to run instead of the classic execution of TELEMAC-MASCARET.
- `par_cmdexec`: defines the command for PARTEL, you might need to change it to comply with your cluster.

Here is an example of a configuration of a cluster:

```
brief: Intel 16.0.4 compiler with open_mpi 1.6.5_tuned on the EDF athos
cluster
#
language: 2
modules:  system
version:  trunk
#
sfx_zip:    .zip
sfx_lib:    .a
sfx_obj:    .o
sfx_mod:    .mod
sfx_exe:
#
#
val_root:   <root>/examples
#
val_rank:   all
#
mods_all:   -I <config>

#
options:    mpi hpc
par_cmdexec: srun -n 1 -N 1 <config>/partel < PARTEL.PAR >> <partel.log>
mpi_cmdexec: mpirun -np <ncsize> <exename>
#
hpc_stdin: #!/bin/bash
  #SBATCH --job-name=<jobname>
  #SBATCH --output=<jobname>-<time>.out
```

```
  #SBATCH --error=<jobname>-<time>.err
  #SBATCH --time=<walltime>
  #SBATCH --ntasks=<ncsize>
  #SBATCH --partition=<queue>
  ##SBATCH --exclude=cn[0000-0000,0000]
  #SBATCH --exclusive
  #SBATCH --nodes=<ncnode>
  #SBATCH --ntasks-per-node=<nctile>
  source <root>/configs/pysource.<configName>.sh
  <py_runcode>
#
hpc_runcode: cp HPC_STDIN ../;cd ../;sbatch < <hpc_stdin>
#
cmd_obj:    mpif90  -c -cpp -convert big_endian -O2 -DHAVE_MPI -DHAVE_MED
-DHAVE_VTK <mods> <incs> <f95name>
cmd_lib:    ar cru <libname> <objs>
cmd_exe:    mpif90  -o <exename> <objs> <libs>
#
incs_all: -I $MEDHOME/include
libs_all: -lm -L$MEDHOME/lib -lmed -L$HDF5HOME/lib -lhdf5 -ldl -lstdc++ -lz
          -L$METISHOME/lib -lmetis
#
cmd_obj_c: gcc -c <srcName> -o <objName>
```

For the second one:

- `hpc_stdin`: Batch script that runs the executable.
- `hoc_cmdexec`: The command to run the Batch script.

Here is an example from `systel.cis-hydra.cfg`

```
brief: parallel mode on the HPC queue, using mpiexec within the queue.
   In that case, the file partitioning and assembly are done by the
   python on the main node.
   (the merge would have to be done manually)
   The only difference with hydru is the presence of the key
   hpc_cmdexec. Of course, you also need the key hpc_stdin.
#
mpi_hosts:    mgmt01
mpi_cmdexec: /apps/openmpi/1.6.5/gcc/4.7.2/bin/mpiexec -wdir <wdir> -n <
ncsize> <exename>
#
par_cmdexec:   <config>/partel  < PARTEL.PAR >> <partel.log>
#
hpc_stdin: #!/bin/bash
   #PBS -S /bin/sh
   #PBS -o <sortiefile>
   #PBS -e <exename>.err
   #PBS -N <jobname>
   #PBS -l nodes=<nctile>:ppn=<ncnode>
   #PBS -q highp
```

```
   source /etc/profile.d/modules.sh
   module load gcc/4.7.2 openmpi/1.6.5/gcc/4.7.2
   <mpi_cmdexec>
   exit
#
hpc_cmdexec:  chmod 755 <hpc_stdin>; qsub <hpc_stdin>
#
cmd_obj:    gfortran -c -cpp -O3 -fconvert=big-endian -DHAVE_MPI
-frecord-marker=4 <mods> <incs> <f95name>
cmd_exe:    /apps/openmpi/1.6.5/gcc/4.7.2/bin/mpif90 -fconvert=big-endian
-frecord-marker=4 -lpthread -v -lm -o <exename> <objs> <libs>
```

Dynamic compilation for TelApy (Python API)

With the above compilation, you already have the API but you cannot have a User Fortran in your study. For that, the TELEMAC-MASCARET system and all its external libraries must be compiled in dynamic mode.

For each prerequisite you can find a section **Dynamic installation** if there is a modification to do for the installation.

For the rest, follow the standard installation.

Then in <systel.cfg> you need to adapt your configuration so that TELEMAC-MASCARET is compiled in dynamic. For that you need to change three things in <systel.cfg>:

1. Change the extension of libraries (sfx_lib) `.so` on most of the Linux distributions.
2. Add the option -fPIC (or equivalent for your compiler).
3. Change the command `cmd_lib` to generate a dynamic library.

Here is an example of a configuration adapted from `gfortranHPC`. The `[general]` section is the same as above.

```
[gfortran.dyn]
brief: GFortran compiler with Open MPI in dynamic mode
#
options: api
#
sfx_lib:    .so
#
mpi_cmdexec: mpirun -machinefile MPI_HOSTFILE -np <ncsize> <exename>
#
cmd_obj:    mpif90 -c -cpp -fPIC -O2 -fconvert=big-endian -frecord-marker=4
-DHAVE_MPI -DHAVE_MUMPS -DHAVE_MED -DHAVE_VTK <mods> <incs> <f95name>
cmd_lib:    mpif90 -fPIC -shared -fconvert=big-endian -frecord-marker=4
-lpthread -lm -o <libname> <objs>
cmd_exe:    mpif90 -fPIC -fconvert=big-endian -frecord-marker=4 -lpthread
-lm -o <exename> <objs> <libs>
#
```

```
libs_all: -L$MUMPSHOME/lib -ldmumps -lmumps_common -lpord
          -L$SCALAPACKHOME/lib -lscalapack
          -lblas
          -lm -L$MEDHOME/lib -lmed -L$HDF5HOME/lib -lhdf5 -ldl -lstdc++ -lz
          -L$METISHOME/lib -lmetis

cmd_obj_c: gcc -c -fPIC <srcName> -o <objName>
```

Then you need to run `compile_telemac.py`. You can find an example in python3/TelApy_api/gouttedo.py

From:
http://wiki.opentelemac.org/ - **open TELEMAC-MASCARET**

Permanent link:
http://wiki.opentelemac.org/doku.php?id=installation_on_linux

Last update: **2022/12/08 10:53**