In the step below we will use “apt-get install” to install package.
This command is mainly for Debian distribution.
The command might be different on your distribution for example.
It is “urpmi” for Mandriva/Mageia.
It is “yum install” for Fedora.

Prerequisites

Here is a list of the software you will need to compile and run the TELEMAC-MASCARET system.

In the text below <version> is to be replace by the version of telemac you are installing (v8p0r0, v8p1r0…)

Mandatory

To run TELEMAC-MASCARET the following software are mandatory:

- Python 3.x.x (installation_linux_python)
- Numpy 1.8.3 (installation_linux_numpy)
- A Fortran compiler (Gfortran 4.6.3 is the minimum) installation_linux_gfortran

Optional

Then there are a couple of others dependencies which necessary to activate certain functionalities:

Parallelism

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

- MPI Distribution (installation_linux_mpi)
- Metis 5.1.x (installation_linux_metis)

Addtional python library for validation

- scipy (installation_scipy)
- matplotlib (installation_matplotlib)

MED Format

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

- Hdf5 (installation_linux_hdf5)
- MEDFichier (installation_linux_med)

Other partitioners

This is only available if you have the prerequisites for parallelism. During a parallel run the mesh is split in multiple parts via the software partel. Partel is using by default metis for the partitioning step. Here is another software that can be used instead of metis:

- Scotch (You can have information on how to compile on their website)

A parallel version of partel was developped but is not fully working. It is using a parallel partitioning instead of a sequential one. Here is a list of two partitioners that are compatible: Parallel partitioner for parallel version of partel (Feature not available in v7p2r0)

- ParMetis (You can have information on how to compile on their website)
- PtScoctch (You can have information on how to compile on their website)

Parallel direct solver

This adds a new parallel solver called MUMPS (option SOLVER = 9).

- MUMPS (You can have information on how to compile on theirwebsite)

AED Library

This will activate more process in the waqtel module installation_linux_aed

---

Get the source code

The TELEMAC-MASCARET system is distributed via two means that are described below:

Download the zipped version

Link not available at the moment for v8p1
You can download here the two zipped files of the latest stable release (v7p2r0 here):

- The source code and the documentation here
- The examples here

Upload from svn server

The TELEMAC-MASCARET system is stored on a subversion server (For more information see here).

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

```
svn –version
```

If this command does not work, install subversion with the following command:

```
sudo apt-get install subversion
```

Here are the information we will use to download the sources:

- **Address** http://svn.opentelemac.org/svn/opentelemac/ (default http protocole using port 80)
- **Source code repository Path** `/tags/<version>/` (for the latest stable version of the source code)
- **Username** `ot-svn-public`
- **Password** `telemac1*`

If your connection is using a proxy go here first
The commands are then:

```
mkdir ~/telemac-mascaret/<version>
svn co
http://svn.opentelemac.org/svn/opent
elemac/tags/<version>
~/telemac-mascaret/<version>
–username=ot-svn-public
–password=telemac1*
```

To install it somewhere else replace `~/telemac-mascare/<version>` by where you want to install it.
To install the trunk replace `tags/<version>` by `trunk`.

(optional) Proxy

If your connection is using a proxy edit the file ~/.subversion/servers and under the line `[global]`:

```
http-proxy-host = <proxy_address>
http-proxy-port = <proxy_port>
http-proxy-username = <username>
http-proxy-password = <password>
```

Replace:

- **<proxy_address>** By the adress to your proxy
- **<proxy_port>** By the port of your proxy
- **<username>** By the login to your proxy
- **<proxy_adress>** By the password for your proxy

Installation of TELEMAC-MASCARET

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

- **<root>** will mean the path to your TELEMAC-MASCARET sources.
- **<systel.cfg>** This will mean you configuration file.
- **<config>** This will point to your configuration.
- **<pysource>** This is your environment file.

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

Setting your environment

This part will describe how to create **<pysource>**.

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`.
So copy this file.
It sets the following environment variables:

- HOMETEL: Path to your installation (<root>)
- SYSTELCFG: Path to yout configuration file (<systel.cfg>)
- USETELCFG: The 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/yugi/telemac-mascaret/<version>
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/build_debian
```

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/yugi/telemac-mascaret/v8p0r0
# 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/wrap_api/lib:
$LD_LIBRARY_PATH
export PYTHONPATH=$HOMETEL/builds/$USETELCFG/wrap_api/lib:
$PYTHONPATH
###
### EXTERNAL LIBRARIES
------------------------------------------------------------
```

```
###
### METIS
#-------------------------------------------------------------------
export METISHOME=/home/yugi/opt/metis-5.1.0/arch/debian
```

If you will be using only one version of TELEMAC-MASCARET you can add the content of this file in your `~/.bashrc`.

(optional) Adding MED

You need to add the following information for both MED and hdf5 if there are not installed in /usr.

```
### HDF5 ------------------------------------------------------------------
export MEDHOME=~/opt/hdf5-1.8.14/build_debian
export LD_LIBRARY_PATH=$HDF5HOME/lib:$LD_LIBRARY_PATH
### MED -------------------------------------------------------------------
export MEDHOME=~/opt/med-5.2.0/build_debian
export LD_LIBRARY_PATH=$MEDHOME/lib:$LD_LIBRARY_PATH
export PATH=$MEDHOME/bin:$PATH
```

(optional) Adding SCOTCH

You need to add the following information for Scotch if it is not installed in /usr.

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

(optional) Adding AED

You need to add the following information for AED if it is not installed in /usr.

```
### AED --------------------------------------------------------------------
export AEDHOME=~/opt/aed-2.1.0/build_debian
export LD_LIBRARY_PATH=$AEDHOME/lib:$LD_LIBRARY_PATH
```

Loading the environment

Once that is done, to load your environment type in a terminal:

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

Configuring TELEMAC-MASCARET

This part will describe how to create **<systel.cfg>**.
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 handles:

- **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 nag, intel, gfortran on Debian and a couple of configurations for clusters.
- **systel.salome.cfg** Configurations for salome_hydro.
- **systel.cis-hydra.cfg** Configuration for an 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:  v8p1
#
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>
```

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

- language: defines in which language the TELEMAC-MASCARET will be in (1: French, 2: English).
- modules: the list of modules to compile (system means all. You can remove some by adding `-modulename`).
- version: Name of 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 in static
- f2py_name: Name of the f2py to use (here f2py3 is the python3 version)
- pyd_compiler: Name of the compiler in f2py (you can get that name by running ”“f2py -c –help-fcompiler)

Now we add the informations for gfortran:

```
#
# Gfortran
#
[gfortran]
brief: Gfortran compiler 4.9.2
#
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

Here is the configuration for gfortranHPC

```
[gfortranHPC]
brief: Gfortran compiler 4.9.2 with open_mpi for a debian 8
#
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 replace `gfortran` by `mpif90` in the `cmd_exe` and `cmd_lib` also we added -DHAVE_MPI in cmd_obj to active 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:  v8p1
#
```

```
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>
#
# _____           ____________________________________
# ____/ Calibre9 ____________________________________/
#
# Gfortran
#
[gfortran]
brief: Gfortran compiler 4.9.2
#
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 4.9.2 with open_mpi for a debian 8
#
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
```

(optinal) Adding MED

Here are the parameters that you have to edit to active the MED feature:

- `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

Here are the parameters that you have to edit to active the SCOTCH feature:

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

Note that if you use Scotch you still need metis.

(optional) Adding AED

Here are the parameters that you have to edit to active the AED feature:

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

Compiling TELEMAC-MASCARET

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

So first source your environment, by typing the following command in a terminal:

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

Then type in a terminal:

```
config.py
```

Which show what your configuration is. It should something like that:

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


... parsing configuration file: /home/yugi/telemac-mascaret/v8p1r0/configs/
systel.cfg


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

gfortranHPC:

    +> Gfortran compiler 4.9.2 with open_mpi for a debian 8
    +> 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 is alright just type in the terminal:

```
compile_telemac.py
```

If at the end you have `My Work is done` it worked properly.

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
```

If at the end you have `My Work is done` it works properly.

Additional configurations

Here we will give additional parameters to add to install TELEMAC-MASCARET on a cluster and how to compile the Telemac2d 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:

- When you run a job everything is run in the cluster queue system.
- When you run a job, only the execution of the TELEMAC-MASCARET executable is run through the cluster queue system. Partitioning is run on the frontal. 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 one, add the following options:

- `hpc_stdin` it 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 compilation before 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 form.

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.
4. replace the options argument by “options: api” (remove the static keyword)

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

```
[C9.gfortran.dyn]
brief: Gfortran compiler 4.9.2 with open_mpi for a debian 8 all libraries
are compiled in dynamic
```

```
#
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