# SWASH

## IMPLEMENTATION MANUAL

**SWASH version 10.05**

| | | |
|---|---|---|
| by | : | The SWASH team |
| mail address | : | Delft University of Technology<br>Faculty of Civil Engineering and Geosciences<br>Environmental Fluid Mechanics Section<br>P.O. Box 5048<br>2600 GA Delft<br>The Netherlands |
| e-mail | : | m.zijlema@tudelft.nl |
| website | : | http://www.tudelft.nl/swash |

# Contents

# Chapter 1

# Introduction

This Implementation Manual is a part of the total material to implement the SWASH model on your computer system. The total material consists of:

- the SWASH source code,
- the pre-built SWASH release for Windows,
- the User Manual,
- this Implementation Manual,
- some test cases.

All of the material can be found at the official SWASH homepage.

Since version 8.01, the SWASH source code is also hosted on GitLab and can be cloned from this repository. For details see Section 3.3.

On the SWASH website, general information is given about the model functionalities, physics and limitations of SWASH. Also, the modification history (or release notes) of SWASH and information on support are provided.

After downloading the material, you may choose between

- direct usage of the pre-built SWASH for Windows and
- implementation of SWASH on your computer system.

If you want to use the pre-built SWASH please read Chapters 5 and 6 for further information.

For the purpose of implementation, you have access to the source code of SWASH and additional files, e.g. for testing SWASH. Please read the copyright in this manual and in

the source code with respect to the terms of usage and distribution of SWASH. You are permitted to implement SWASH on your computer system. However, for any use of the SWASH source code in your environment, proper reference must be made to the origin of the software!

Implementation involves the following steps:

1. Copying the source code from the SWASH website to the computer system on which you want to run SWASH.
2. If necessary, applying patches for an upgrade of the source code due to e.g., bug fixes, resolved issues, new features, etc.
3. Making a few adaptions in installation-dependent parts of the code.
4. Compiling and linking the source code to produce an executable of SWASH.
5. Testing of the built SWASH.

After the last step you should have the built SWASH ready for usage. Note that steps 3 and 4 can be carried out fully automatically.

## 1.1 The material

The source tarball `swash-10.05.tar.gz` contains the SWASH source code and consists of the following files:

| | |
|---|---|
| general modules | : ocpmod.ftn |
| | SwashModule1.ftn90 |
| | SwashModule2.ftn90 |
| modules for flow | : SwashFlowdata.ftn90 |
| modules for rigid bodies | : SwashRigBoddata.ftn90 |
| modules for solvers | : SwashSolvedata.ftn90 |
| modules for support | |
| parallel MPI runs | : m_parall.ftn |
| | |
| main program | : Swash.ftn90 |
| | SwashMain.ftn90 |
| pre-processing routines | : SwashBackup.ftn90 |
| | SwashBCboundwave.ftn90 |
| | SwashBCshortwave.ftn90 |
| | SwashBCspecfile.ftn90 |
| | SwashBCspectrum.ftn90 |

SwashBCStokeswave.ftn90
SwashBCtransferfnc.ftn90
SwashBounCond.ftn90
SwashIntWavgen.ftn90
SwashCheckPrep.ftn90
SwashInit.ftn90
SwashInitBCtrans.ftn90
SwashInitCompGrid.ftn90
SwashInitCompUgrid.ftn90
SwashInitCond.ftn90
SwashInitSteady.ftn90
SwashInputField.ftn90
SwashInputGrid.ftn90
SwashReadBndval.ftn90
SwashReadInput.ftn90
SwashReadTestpnts.ftn90
SwashReqOutL.ftn90
SwashReqOutQ.ftn90

computational routines : SwashAmbCurrent.ftn90
SwashAntiCreep1DH.ftn90
SwashAntiCreep2DH.ftn90
SwashBotFrict.ftn90
SwashBreakPoint.ftn90
SwashComputFlow.ftn90
SwashComputStruc.ftn90
SwashComputTrans.ftn90
SwashComputTurb.ftn90
SwashDensity.ftn90
SwashDryWet.ftn90
SwashExpDep1DHflow.ftn90
SwashExpDep1DHtrans.ftn90
SwashExpDep2DHflow.ftn90
SwashExpDep2DHtrans.ftn90
SwashExpLay1DHflow.ftn90
SwashExpLayP1DHflow.ftn90
SwashExpLay1DHtrans.ftn90
SwashExpLay2DHflow.ftn90
SwashExpLayP2DHflow.ftn90
SwashExpLay2DHtrans.ftn90
SwashFlowDP.ftn90
SwashFloatObjects.ftn90
SwashForcesRigidBod.ftn90
SwashGeometrics.ftn90

SwashHDiffZplane1DH.ftn90
SwashHDiffZplane2DH.ftn90
SwashHorzVisc.ftn90
SwashHydroLoads.ftn90
SwashImpDep1DHflow.ftn90
SwashImpDep2DHflow.ftn90
SwashImpLay1DHflow.ftn90
SwashImpLayP1DHflow.ftn90
SwashImpLay2DHflow.ftn90
SwashImpLayP2DHflow.ftn90
SwashKepsMod1DH.ftn90
SwashKepsMod2DH.ftn90
SwashReynoldsStress.ftn90
SwashLayerIntfaces.ftn90
SwashLogLaw.ftn90
SwashMotionRigidBod.ftn90
SwashPorousStruc.ftn90
SwashPorFricDep.ftn90
SwashPorFricLay.ftn90
SwashPresFlow.ftn90
SwashSolvers.ftn90
SwashSpongeLayer.ftn90
SwashUpdateData.ftn90
SwashUpdateDepths.ftn90
SwashUpdateFld.ftn90
SwashUpdDepu.ftn90
SwashUpdFlowFlds.ftn90
SwashUpdKBCrigb.ftn90
SwashUpdPress.ftn90
SwashVeget.ftn90
SwashVertVisc.ftn90
SwashWindStress.ftn90

routines for unstructured grids : SwashBndTopology.ftn90
SwashCheckGrid.ftn90
SwashCompUFlow.ftn90
SwashCompUnstruc.ftn90
SwashCompUTrans.ftn90
SwashCompUTurb.ftn90
SwashExpDepUflow.ftn90
SwashExpDepUtrans.ftn90
SwashExpLayUflow.ftn90
SwashExpLayUtrans.ftn90

| | |
|---|---|
| | SwashFlowUDP.ftn90 |
| | SwashImpDepUflow.ftn90 |
| | SwashImpLayUflow.ftn90 |
| | SwashInitBCUtrans.ftn90 |
| | SwashLayUIntfaces.ftn90 |
| | SwashPrintGridInfo.ftn90 |
| | SwashUBotFrict.ftn90 |
| | SwashUBreakPoint.ftn90 |
| | SwashUDryWet.ftn90 |
| | SwashUHorzVisc.ftn90 |
| | SwashUKepsMod.ftn90 |
| | SwashULogLaw.ftn90 |
| | SwashUpdateUData.ftn90 |
| | SwashUpdateUDepths.ftn90 |
| | SwashUpdUDepu.ftn90 |
| | SwashUpdUFlowFlds.ftn90 |
| | SwashUpdUPress.ftn90 |
| | SwashUServ.ftn90 |
| | SwashUSpongLayer.ftn90 |
| | SwashUWindStress.ftn90 |
| post-processing routines | : swanout2.ftn |
| | SwashAverOutp.ftn90 |
| | SwashCoorOutp.ftn90 |
| | SwashDecOutL.ftn90 |
| | SwashDecOutQ.ftn90 |
| | SwashElemOutp.ftn90 |
| | SwashFlobjOutp.ftn90 |
| | SwashOutput.ftn90 |
| | SwashQuanOutp.ftn90 |
| | SwashRunupHeight.ftn90 |
| | SwashVTKWriteHeader.ftn90 |
| | SwashVTKWriteData.ftn90 |
| | SwashVTKPDataSets.ftn90 |
| service routines | : SwashCleanMem.ftn90 |
| | SwashPrintSettings.ftn90 |
| | SwashServices.ftn90 |
| SWAN service routines | : swanser.ftn |
| routines for support parallel MPI runs | : swanparll.ftn |
| SWAN routines for unstructured grids | : SwanReadGrid.ftn90 |
| | SwanReadADCGrid.ftn90 |
| | SwanReadTriangleGrid.ftn90 |

| | |
|---|---|
| | SwanReadEasymeshGrid.ftn90 |
| | SwanCreateEdges.ftn90 |
| | SwanGridTopology.ftn90 |
| | SwanGridVert.ftn90 |
| | SwanGridCell.ftn90 |
| | SwanGridFace.ftn90 |
| | SwanFindPoint.ftn90 |
| | SwanPointinMesh.ftn90 |
| | SwanBpntlist.ftn90 |
| | SwanInterpolatePoint.ftn90 |
| | SwanInterpolateOutput.ftn90 |
| routines for installation | : ocpids.ftn |
| command reading routines | : ocpcre.ftn |
| miscellaneous routines | : ocpmix.ftn |
| SWAN modules for unstructured grids | : SwanGriddata.ftn90 |
| | SwanGridobjects.ftn90 |
| | SwanCompdata.ftn90 |

The SWASH source code is written in Fortran 90. Most of the routines are written in free form and are indicated by extension `f90`. Some routines are written in fixed form and depending on your system, the extension may be `for` or `f`. The conversion from `ftn` or `ftn90` to one of these extensions can be done automatically or manually; see Chapter 3.

You are allow to make changes in the source code of SWASH, but Delft University of Technology will not support modified versions of SWASH. If you want your modifications to be implemented in the authorized version of SWASH (the version on the SWASH homepage), you need to submit these changes to the SWASH team (e-mail: m.zijlema@tudelft.nl).

The SWASH source code is additionally accompanied by the following files:

| | |
|---|---|
| installation procedures | : INSTALL.README |
| | Makefile |
| | macros.inc |
| | which.cmd |
| | platform.pl |
| | switch.pl |
| run procedures | : SWASHRUN.README |
| | swashrun |
| | swashrun.bat |
| machinefile for parallel MPI runs | : machinefile |

edit file : swash.edt

On the SWASH homepage, you also find some test cases with some output files for making a configuration test of SWASH on your computer. You may compare your results with e.g. analytical or laboratory data. See Chapter 6 for further details.

# Chapter 2

# Use of patch files

Between releases of authorised SWASH versions, it is possible that bug fixes or new features are published on the SWASH homepage. These are provided by patch files that can be downloaded from the website. Typically, a patch can be installed on top of the existing source code. Patches are indicated by a link to **patchfile**. The names refer to the current version number supplemented with letter codes. The first will be coded 'A' (i.e. 10.05.A), the second will be coded 'B', the third will be coded 'C', etc. The version number in the resulting output files will be updated to 10.05ABC, indicating the implemented patches.

To use a patch file, follow the next instructions:

1. download the file (right-click the file and choose *save link as*)
2. place it in the directory where the source code of SWASH is located
3. execute `patch -p0 < patchfile`

After applying a patch or patches, you need to recompile the SWASH source code.

It is important to download the patch and not cut and paste it from the display of your web browser. The reason for this is that some patches may contain tabs, and most browsers will not preserve the tabs when they display the file. Copying and pasting that text will cause the patch to fail because the tabs would not be found. If you have trouble with patch, you can look at the patch file itself.

Note to Linux/UNIX users: the downloaded patch files are MS-DOS ASCII files and contain carriage return (CR) characters. To convert these files to UNIX format, use the command `dos2unix`. Alternatively, execute `cat 10.05.[A-C] | tr -d '\r' | patch` that apply the patch files 10.05.A to 10.05.C to the SWASH source code at once after which the conversion is carried out.

Note to Windows users: `patch` is a UNIX command. Download the patch program from the SWASH website, which is appropriate for Windows operating system.

# Chapter 3

# Installation

## 3.1 Introduction

SWASH can be installed on various architectures, including laptops and supercomputers. The portability of the SWASH source code is guaranteed by the use of standard ANSI Fortran 90. (See also the manual Programming rules.) Hence, virtually all Fortran compilers can be used for installing SWASH. It should be noted that there are two Fortran commands used in the source code of SWASH (v8.01+) which were introduced in later versions of Fortran: stream I/O (Fortran 2003 standard) and execution OS command line (Fortran 2008 standard). They are, however, supported by currently maintained Fortran compilers, including gfortran and Intel® Fortran.

The SWASH source code also supports parallelization, which enables a considerable reduction in the wall-clock time for relatively large CPU-demanding calculations. A message passing modelling is employed based on the Message Passing Interface (MPI) standard that enables communication between independent processors. Hence, users can optionally run SWASH on a Linux cluster.

The SWASH software can be build in the usual way via GNU make or from scratch. This building process is explained in Section 3.2. However, since version 8.01, the option to install SWASH using CMake is supported and is elaborated in Section 3.3.

## 3.2 Classic build instructions

### 3.2.1 Configuring the build

The material on the SWASH website provides a `Makefile` and two Perl scripts (`platform.pl` and `switch.pl`) that enables the user or developer to install SWASH on the computer in a proper manner. For this, the following platforms, operating systems and compilers are supported (and tested):

| **platform** | **OS** | **F90 compiler** |
|---|---|---|
| Intel/AMD desktop/laptop | Linux | gfortran |
| Intel Core desktop/laptop | Linux | Intel® |
| Intel Xeon desktop/laptop | Linux | Intel® |
| x86-64 processor-based system | Linux | Portland Group |
| Intel/AMD desktop/laptop | Linux | Lahey |
| Intel Core desktop/laptop | MS Windows | Intel® |
| Intel Xeon desktop/laptop | MS Windows | Intel® |
| Intel/AMD desktop/laptop | MS Windows | Lahey |
| MacBook | macOS | gfortran |
| MacBook | macOS | Intel® |
| SGI Origin 3000 (Silicon Graphics) | IRIX | SGI |
| IBM SP | AIX | IBM |
| Compaq True 64 Alpha (DEC ALFA) | OSF1 | Compaq |
| Sun SPARC | Solaris | Sun |
| PA-RISC (HP 9000 series 700/800) | HP-UX v11 | HP |
| IBM Power6 (pSeries 575) | Linux | IBM |
| Power Mac G4 | Mac OS X | IBM |

If your computer and available compiler is mentioned in the table, you may consult Section 3.2.2 for a complete build of the software without making any modifications. If desired, you may install SWASH manually; see Section 3.2.3.

Note that for a successful installation, a Perl package must be available on your computer. Usually, it is available for macOS, Linux and a UNIX-like operating system. Check it by typing `perl -v`. You can download Perl for MS Windows from Strawberry Perl. The Perl version should be at least 5.0.0 or higher!

Before starting the build process, the user may first decide how to run the SWASH program. There are two run modes:

- serial runs or
- parallel runs on a multi-core PC or a Linux cluster.

For a typical depth-averaged flume computation, it may be sufficient to choose the serial mode, i.e. one SWASH program running on one processor. The parallel mode is more convenient for a relatively large CPU-demanding multi-layer flume (2DV) or basin-like (2DH/3D mode) calculation.

For a proper installation of MPI-based application on Windows, please consult Section 3.2.4.

### 3.2.2 Building SWASH using GNU make

Carry out the following steps for building SWASH on your computer.

1. An include file containing some machine-dependent macros must be created first. This file is called `macros.inc` and can be created by typing

```
make config
```

2. Now, SWASH can be built for serial or parallel mode, as follows:

| **mode** | **instruction** |
|---|---|
| serial | `make ser` |
| parallel | `make mpi` |

IMPORTANT NOTES:

- To Windows users:
  - To execute the above instructions, just open a command prompt.
  - To build SWASH on Windows platforms by means of a Makefile you need the command-line utility `Nmake`, which is provided by the Microsoft® Visual Studio.
  - This installation currently supports Intel® MPI library for Windows. See Section 3.2.4 for further information.
- One of the commands `make ser` and `make mpi` must be preceded once by `make config`.
- If desirable, you may clean up the generated object files and modules by typing `make clean`. If you want to delete any stray files from a previous compilation, just type `make clobber`.
- If you are unable to install SWASH using the Makefile and Perl scripts for whatever reason, see Section 3.2.3 that includes instructions for a custom installation.

### 3.2.3 Building SWASH from scratch

It is recommended to consult Section 3.2.2 for a complete build of SWASH on your computer. However, if you want to build SWASH on your system from scratch, then please follow the instructions below.

### Modifying the source code

To compile SWASH on your computer system properly, some subroutines should be adapted first depending on the operating system, use of compilers and the wish to use MPI for parallel runs. This can be done by removing the switches started with '!' followed by an indentifiable prefix in the first 3 or 4 columns of the subroutine. A Perl script called `switch.pl` is provided in the material that enables the user to quickly select the switches to be removed. This script can be used as follows:

```
perl switch.pl [-dos] [-unix] [-f95] [-mpi] [-cray] [-sgi]
               [-cvis] [-timg] [-matl4] [-impi] *.ftn[90]
```

where the options are all optionally. The meaning of these options are as follows.

`-dos`, `-unix` Depending on the operating system, both the TAB and directory separator character must have a proper value (see also Chapter 4). This can be done by removing the switch !DOS or !UNIX, for Windows and Linux/UNIX platforms, respectively, in the subroutines `OCPINI` (in `ocpids.ftn`) and `TXPBLA` (in `swanser.ftn`). For other operating system (e.g., Macintosh), you should change the values of the following variables manually: DIRCH1, DIRCH2 (in `OCPINI`), TABC (in `OCPINI`) and ITABVL (in `TXPBLA`).

`-f95` If you have a Fortran 95 compiler or a Fortran 90 compiler that supports Fortran 95 features, it might be useful to activate the `CPU_TIME` statement in the subroutines `SWTSTA` and `SWTSTO` (in `swanser.ftn`) by removing the switch !F95 meant for the detailed timings of several parts of the SWASH calculation. Note that this can be obtained with the command TEST by setting **itest=1** in your command file.

`-mpi` For the proper use of MPI, you must remove the switch !MPI at several places in the files `swanparll.ftn`, `Swash*2DHflow.ftn90` and `m_parall.ftn`.

`-cray`, `-sgi` If you use a Cray or SGI Fortran 90 compiler, the subroutines `OCPINI` (in `ocpids.ftn`) and `FOR` (in `ocpmix.ftn`) should be adapted by removing the switch !/Cray or !/SGI since, these compilers cannot read/write lines longer than 256 characters by default. By means of the option RECL in the OPEN statement sufficiently long lines can be read/write by these compilers.

`-cvis` The same subroutines `OCPINI` and `FOR` need also to be adapted when the Compaq Visual Fortran compiler is used in case of a parallel MPI run. Windows systems have a well-known problem of the inability of opening a file by multiple SWASH executables. This can be remedied by using the option SHARED in the OPEN statement for shared access. For this, just remove the switch !CVIS.

`-timg` If the user want to print the timings (both wall-clock and CPU times in seconds) of different processes within SWASH then remove the switch !TIMG. Otherwise, no timings will be keeped up and subsequently printed in the `PRINT` file.

`-matl4` By default, the created binary Matlab files are of Level 5 MAT-File format and are thus compatible with MATLAB version 5 and up. In this case the switch !MatL5 must be removed. However, some machines do not support a 1-byte unit for the record length (e.g. IBM Power6). At those computers, the binary Matlab files must be formatted of Level 4. In this case the switch !MatL4 must be removed while the switch !MatL5 should not be removed. Level 4 MAT-files are compatible with MATLAB versions 4 and earlier. However, they can be read with the later versions of MATLAB.

`-impi` Some Fortran compilers do not support `USE MPI` statement and therefore, the module `MPI` in `m_parall.ftn` must be included by removing the switch !/impi.

For example, you work on a Linux cluster where MPI has been installed and use the Intel® Fortran compiler (that can handle Fortran 95 statements), then type the following:

```
perl switch.pl -unix -f95 -mpi *.ftn *.ftn90
```

Note that due to the option `-unix` the extension `ftn` is automatically changed into `f` and `ftn90` into `f90`.

### Compiling and linking SWASH source code

After the necessary modifications are made as described in the previous section, the source code is ready for compilation. All source code is written in Fortran 90 so you must have a Fortran 90 compiler in order to compile SWASH. The source code cannot be compiled with a Fortran 77 compiler. If you intended to use MPI for parallel runs, you must use the command `mpif90` (or `mpiifort` in case of the Intel® compiler) instead of the original compiler command.

The SWASH source code complies with the ANSI Fortran 90 standard, except for a few cases, where the limit of 19 continuation lines is violated. We are currently not aware of any compiler that cannot deal with this violation of the ANSI standard.

When compiling SWASH you should check that the compiler allocates the same amount of memory for all INTEGERS, REAL and LOGICALS. Usually, for these variables 4 bytes are allocated, on supercomputers (vector or parallel), however, this sometimes is 8 bytes. When a compiler allocates 8 bytes for a REAL and 4 bytes for an INTEGER, for example, SWASH will not run correctly.

Furthermore, SWASH can generate binary MATLAB files on request, which are unformatted. Some compilers, e.g. Intel® Fortran, measured record length in 4-byte or longword units and as a consequence, these unformatted files cannot be loaded in MATLAB. Hence, in such as case a compiler option is needed to request 1-byte units, e.g. for Intel® Fortran this is `/assume:byterecl` (Windows) or `-assume byterecl` (Linux/UNIX).

The modules must be compiled first. Several subroutines use these modules. These subroutines need the compiled versions of the modules before they can be compiled. You can find here below the complete list of modules in the proper order.

- `ocpmod.f`
- `SwashModule1.f90`,`SwashModule2.f90`
- `m_parall.f`
- `SwanGriddata.f90`, `SwanGridobjects.f90`, `SwanCompdata.f90`
- `SwashFlowdata.f90`, `SwashSolvedata.f90`

Linking should be done without any options nor using shared libraries (e.g. math or NAG). It is recommended to rename the executable to `swash.exe` after linking.

Referring to the previous example, compilation and linking may be done as follows:

```
mpif90 <list of modules> ocpmix.f swanser.f Swash*.f90 Swan*.f90 -o swash.exe
```

### 3.2.4 Building with MPI support

SWASH can be built with support for MPI. It is assumed that MPI has been installed already in the Linux environment. However, this is probably not the case for Windows. At any rate, the Intel® MPI library may be employed to build an MPI application. This library is included in the Intel® oneAPI HPC Toolkit. In this respect, the following steps need to be made first

- make sure that the variables `INCS_MPI` and `LIBS_MPI` in the file `macros.inc` are emptied, and
- change the value of the variable `F90_MPI` by replacing `ifort` by `mpiifort`.

Build SWASH by executing the command `make mpi`.

## 3.3 Building SWASH with CMake

CMake is a cross-platform build system that creates native build files (for use with a generator GNU Make, Nmake or Ninja) for command line builds or project files for an IDE (e.g. Visual Studio). CMake makes use of configuration files that control the build process. We recommend to use CMake 3.12+ for building SWASH. There are installers available for Windows, Linux and macOS. See the download page for installation instructions.

Ninja is one of the many build generators to create executable files and libraries from source code. The way it works is very similar to GNU make; for example, it does not rebuild things that are already up to date. We recommend Ninja because it is faster than GNU make. Ninja can be downloaded from its git repository.

### 3.3.1 Build instructions

For a proper build, the release code including the required CMake files can be downloaded or cloned from the SWASH git repository hosted on TU Delft GitLab.

Next, carry out the following steps.

**1. clone the repository and navigate to the top level source directory**

```
git clone https://gitlab.tudelft.nl/citg/wavemodels/swash.git && cd swash
```

**2. create the build directory**

At the top of SWASH source directory execute the following commands

```
mkdir build
cd build
```

This step is required to perform an out-of-source build with CMake, that is, build files will not be created in the `/swash/src` directory.

**3. build the software**

Two CMake configuration files are provided as required for the build. They are placed in the following source directories: `./swash/CMakeLists.txt` and `./swash/src/CMakeLists.txt`. The following two CMake commands should suffice to build SWASH

```
cmake .. -G Ninja
cmake --build .
```

The first command refers to the source directory where the main configuration file is invoked. The second command carries out the building in the build directory. The package is actually built by invoking Ninja.

**4. install the package**

```
cmake --install .
```

The default install directory is `/usr/local/swash` (Unix-like operating systems, including macOS) or `C:\PROGRAM FILES\swash` (Windows). Instead, you may install SWASH in any other user-defined directory, as follows

```
cmake --install . --prefix /your/defined/directory
```

After installation a number of subdirectories are created. The executables end up in the `/bin` directory, the library files in `/lib`, and the module files in `/mod`. Additionally, the `/doc` folder contains the pdf documents, the folder `/tools` consists of some useful scripts and the `/misc` directory contains all of the files that do not fit in other folders (e.g., a machinefile and the edit file `swash.edt`).

Please note that the installation can be skipped (though not recommended). Executables and libraries are then located in subdirectories of the build directory.

### 3.3.2 Configuring the build

The build can be (re)configured by passing one or more options to the CMake command with prefix `-D`. A typical command line looks like

```
cmake .. -D<option>=<value>
```

where `<value>` is a string or a boolean, depending on the specified option. The table below provides an overview of the non-required options that can be used.

| option | description | default value |
|---|---|---|
| CMAKE_INSTALL_PREFIX | user-defined installation path | /usr/local/swash |
| CMAKE_Fortran_COMPILER | full path to the Fortran compiler | determined by CMake |
| MPI | enable build with MPI | OFF |
| CMAKE_VERBOSE_MAKEFILE | provide verbose output of the build | OFF |

For example, the following commands

```
cmake .. -GNinja -DMPI=ON
cmake --build .
```

will configure SWASH to be built created by Ninja that supports parallel computing using the MPI paradigm. Note that CMake will check the availability of MPI libraries within your environment.

The system default Fortran compiler (e.g., f77, g95) can be overwritten as follows

```
cmake .. -DCMAKE_Fortran_COMPILER=/path/to/desired/compiler
```

Finally, if CMake fails to configure your project, then execute

```
cmake .. -DCMAKE_VERBOSE_MAKEFILE=ON
```

which will generate detailed information that may provide some indications to debug the build process.

### 3.3.3 Clean up the build files

To remove the build directory and all files that have been created after running `cmake --build .`, run at the top level of your project the following command:

```
cmake -P clobber.cmake
```

(The `-P` argument passed to CMake will execute a script `<filename>.cmake`.)

# Chapter 4

# User dependent changes and the file `swashinit`

SWASH allows you to customize the input and the output to the wishes of your department, company or institute or yourself. This can be done by changing the settings in the initialisation file `swashinit`, which is created during the first time SWASH is executed on your computer system. The changes in `swashinit` only affect the runs executed in the directory that contains that file.

A typical initialisation file `swashinit` may look like:

```
    6                               version of initialisation file
Delft University of Technology      name of institute
    3                               command file ref. number
INPUT                               command file name
    4                               print file ref. number
PRINT                               print file name
    4                               test file ref. number
                                    test file name
    6                               screen ref. number
99999                               highest file ref. number
$                                   comment identifier
[TAB]                               TAB character
\                                   dir sep char in input file
/                                   dir sep char replacing previous one
    7                               default time coding option
   -1                               grid partitioning option
    1                               option to merge output files
  100                               speed of processor   1
  100                               speed of processor   2
  100                               speed of processor   3
```

Explanation:

- The version number of the initialisation file is included in the file so that SWASH can verify whether the file it reads is a valid initialisation file. The current version is **6**.

- The initialisation file provides a character string containing the name of the institute that may carry out the computations or modifying the source code. You may assign it to the name of your institute instead of `Delft University of Technology`, which is the present value.

- The standard input file and standard print file are usually named `INPUT` and `PRINT`, respectively. You may rename these files, if appropriate.

- The unit reference numbers for the input and print files are set to 3 and 4, respectively. If necessary, you can change these numbers into the standard input and output unit numbers for your installation. Another unit reference number is foreseen for output to screen and it set to 6. This is useful if print output is lost due to abnormal end of the program, while information about the reason is expected to be in the print file. There is also a unit number for a separate test print file. In the version that you downloaded from SWASH homepage, this is equal to that of the print file so that test output will appear on the same file as the standard print output.

- The comment identifier to be used in the command file is usually '$', but on some computer system this may be inappropriate because a line beginning with '$' is interpreted as a command for the corresponding operating system (e.g., VAX systems). If necessary, change to '!'.

- To insert [TAB] in the initialisation file, just use the TAB key on your keyboard.

- Depending on the operating system, the first directory separation character in `swashinit`, as used in the input file, may be replaced by the second one, if appropriate.

- Date and time can be read and written according to various options. The following options are available:

  1. 19870530.153000 (ISO-notation)
  2. 30-May-87 15:30:00
  3. 05/30/87 15:30:00
  4. 15:30:00
  5. 87/05/30 15:30:00
  6. 8705301530 (WAM-equivalence)
  7. 153000.000

Option 7 only represents the time (hours, minutes, seconds and milliseconds) and hence, does not taken into account the date. Note that this option can not be used for a simulation longer than a day. In most cases, this option is the appropriate one for SWASH. Note that the ISO-notation has no millenium problem, therefore the ISO-notation is recommended. In case of other options, except the last one, the range of valid dates is in between January 1, 1911 and December 31, 2010 (both inclusive).

- For a parallel MPI run the computational grid needs to be decomposed into a number of subdomains. This decomposition is based on the grid partition. Two methods are available:

  1. stripwise manner
  2. orthogonal recursive bisection (ORB)

  The ORB method starts with a single part (the entire domain) after which each part is recursively partitioned by bisecting it, until all parts have been created. The bisection direction is swapped in each direction. This partitioning method can only be applied in the 2D multi-layered mode. In the other modes, 1D or depth-averaged, a stripwise partitioning will be applied. This will be done automatically by choosing option $-1$. Otherwise, you may choose option 1 for stripwise partitioning along the $x-$ or $y-$axis, option 2 for ORB partitioning, option 3 for stripwise partitioning along the $x-$axis or option 4 for stripwise partitioning along the $y-$axis.

- In case of a parallel MPI run, SWASH either merge the processor-dependent output files into a single file and subsequently delete the individual files, or keep the processor-dependent output files on disk. The option values are:

  0. do not merge the processor-dependent output files
  1. merge the processor-dependent output files

  Option 1 is the default.

- In case of a parallel MPI run at the machine having a number of independent processors, it is important to assign subdomains representing appropriate amounts of work to each processor. Usually, this refers to an equal number of grid points per subdomain. However, if the computer has processors which are not all equally fast (a so-called heterogeneous machine), then the sizes of the subdomains depend on the speed of the processors. Faster processors should deal with more grid points than slower ones. Therefore, if necessary, a list of non-default processor speeds is provided. The given speeds are in % of default = 100%. As an illustrating example, we have two PC's connected via an Ethernet switch of which the first one is 1.5 times faster than the second one. The list would be

```
150    speed of processor 1
100    speed of processor 2
```

Based on this list, SWASH will automatically distribute the total number of active grid points over two subdomains in an appropriate manner. Referring to the above example, with 1000 active points, the first and second subdomains will contain 600 and 400 grid points, respectively.

# Chapter 5

# Run instructions

In this chapter it is assumed that you have a built SWASH available on your computer, either after installation as described in Chapter 3, or downloaded from the SWASH website (for Windows).

Before running SWASH you must first complete a command file. Consult the SWASH User Manual how to specify the various settings and instructions to SWASH concerning the (input) grids, boundary conditions, physics, numerics and output. To help you in editing a command file for SWASH input, the file `swash.edt` is provided which contains the complete set of commands.

After completing the command file, you may run SWASH. Two command-line utilities are provided among the source code, one for running SWASH on the Windows platform, called `swashrun.bat`, and one for running SWASH on the Linux/UNIX platform, called `swashrun`. Basically, the run procedure carries out the following actions:

- Copy the command file with extension `sws` to `INPUT` (assuming `INPUT` is the standard file name for command input, see Chapter 4).
- Run SWASH.
- Copy the file `PRINT` (assuming `PRINT` is the standard file name for print output, see Chapter 4) to a file which name equals the command file with extension `prt`.

On other operating system a similar procedure can be followed. For parallel MPI runs, the program `mpirun` or `mpiexec` may be needed instead (usually provided in an MPI distribution).

Before calling the run procedure, the environment variable `PATH` need to be adapted by including the pathname of the directory where `swash.exe` can be found. In case of Windows, this pathname can be specified through the setting *Environment Variables*. (Hit the Window key plus R to get command prompt. Then type *sysdm.cpl*, go to *Advanced* and

select *Environment Variables*. Note that you must be an administrator in order to modify an environment variable.) In case of Linux or UNIX running the bash shell (sh or ksh), the environment variable `PATH` may be changed as follows:

```
export PATH=${PATH}:/usr/local/swash
```

if `/usr/local/swash` is the directory where the executable `swash.exe` is resided. In case of the C shell (csh), use the following command:

```
setenv PATH ${PATH}:/usr/local/swash
```

If appropriate, you also need to add the directory path where the `bin` directory of MPI is resided to `PATH` to have access to the command `mpirun` or `mpiexec`.

The provided run utilities enable the user to properly and easily run SWASH both serial as well as parallel. Note that for parallel MPI runs, the executable `swash.exe` should be accessible by copying it to all the multiple machines or by placing it in a shared directory. When running the SWASH program, the user must specify the name of the command file. However, it is assumed that the extension of this file is `sws`. Note that contrary to Linux/UNIX, Windows does not distinguish between lowercase and uppercase characters in filenames. Next, the user may also indicate whether the run is serial or parallel. In case of Windows, use the run procedure `swashrun.bat` from a command prompt:

```
swashrun filename [nprocs]
```

where `filename` is the name of your command file without extension (assuming it is `sws`) and `nprocs` indicates how many processors need to be launched for a parallel MPI run (do not type the brackets; they just indicate that the parameter `nprocs` is optional). By default, $nprocs = 1$.

The command line for the UNIX script `swashrun` is as follows:

```
./swashrun -input filename -mpi n
```

where `filename` is the name of your command file without extension. Note that the script `swashrun` need to be made executable first, as follows:

```
chmod +rx ./swashrun
```

The parameter `-mpi n` specifies a parallel run on $n$ cores using MPI. The parameter `-input` is obliged, whereas the parameter `-mpi n` can be omitted (default: $n = 1$). To redirect screen output to a file, use the sign >. Use an ampersand to run SWASH in the background. An example:

```
./swashrun -input l51con01 -mpi 4 > swashout &
```

For a parallel MPI run, you may also need a `machinefile` that contains the names of the nodes in your parallel environment. Put one node per line in the file. Lines starting with the # character are comment lines. You can specify a number after the node name to indicate how many cores to launch on the node. This is useful e.g., for multi-core processors. The run procedure will cycle through this list until all the requested processes are launched. Example of such a file may look like:

```
# here, eight processes will be launched
node1
node2:2
node4
node7:4
```

Note that for Windows platforms, a space should be used instead of a colon as the separation character in the `machinefile`.

SWASH will generate a number of output files:

- A print file with the name `PRINT` that can be renamed by the user with a batch (DOS) or script (UNIX) file, e.g. with the provided run procedures. For parallel MPI runs, however, a sequence of `PRINT` files will be generated (`PRINT-001`, `PRINT-002`, etc.) depending on the number of processors. The print file(s) contain(s) the echo of the input, information concerning the iteration process, possible errors, timings, etc.

- Numerical output (such as table and block output) appearing in files with user provided names.

- A file called `Errfile` (or renamed by the run procedures as well as more than one file in case of parallel MPI runs) containing the error messages is created only when SWASH produces error messages. Existence of this file is an indication to study the results with more care.

- A file called `ERRPTS` (or renamed by the run procedures as well as more than one file in case of parallel MPI runs) containing the grid points, where specific errors occured during the calculation, such as non-convergence of an iterative matrix solver. Existence of this file is an indication to study the flow in that grid point with more care.

# Chapter 6

# Testing SWASH

The SWASH package consists of one executable file (`swash.exe`), a command file (`swash.edt`) and a run procedure (`swashrun.bat` or `swashrun`). The executable for Windows can be obtained from the SWASH web site, but see Chapter 5 for further details. The input and output to a number of test problems is provided on the SWASH homepage. The files with extension `sws` are the command files for these tests; the files with extension `bot` are the bottom files for these tests, etc. This input can be used to make a configuration test of SWASH on your computer. Compare the results with those in the provided plot files.

To run the SWASH program for the test cases, at least 500 MBytes of free internal memory is recommended. For more realistic cases 1 to 2.5 GBytes may be needed, whereas for more simple 1D cases significant less memory is needed (less than 100 MBytes).