U. S. Department of Commerce
National Oceanic and Atmospheric Administration
National Weather Service
National Centers for Environmental Prediction
5830 University Research Court
College Park, MD 20740

Technical Note

WAVEWATCH III ® development best practices [†].

Hendrik L. Tolman[‡] (Editor)
Environmental Modeling Center
Marine Modeling and Analysis Branch

Version 1.2, Feb. 2019

[†] MMAB Contribution No. 286.
[‡] Contact code manager, e-mail: NCEP.List.WAVEWATCH@NOAA.gov

This page is intentionally left blank.

Abstract

This guide describes best practices for code development of WAVEWATCH III®. This includes guidelines for packaging of codes delivered by general users to NCEP according to the WAVEWATCH III license, as well as instructions for co-developers on the use of the subversion depository at NCEP. The guide addresses codes, documentation and manuals.

Change log

version	svn rev.	date	comment
0.1	7869	May 14, 2010	Initial MMAB No. 286.
			Sec. 4 (regression testing)
			as placeholder only
1.0	12398	Feb. 18, 2011	Adding svn note p. ??.
			Adding svn warning p. ??.
			Section 4 updated.
1.1	38155	Mar. 18, 2014	Changed address on title page.
			Updating most sections.
			Adding Sec. ?? (code distr.).
			Finalize for release of v. 4.18.
1.2		Feb. 26, 2019	Prepare for release of v. 6.07.

(minor edits included in each new version)

Acknowledgments. Code management for WAVEWATCH III is provided by NCEP. Arun Chawla (NCEP), Henrique Alves (SRG at NCEP), Erick Rogers and Tim Campbell (NRL Stennis) contributed to this guide.

This guide is available as a pdf file from

http://polar.ncep.noaa.gov/waves

Contents

This page is intentionally left blank.

1 Introduction

The WAVEWATCH III ® wind wave model has a history dating back to the second half of the 1980s. It's history started with the development of the WAVEWATCH model at Delft university of technology (Tolman, 1989, 1990, 1991). The next step of development occurred at NASA, Goddard Space Flight Center in the early 1990s, with the development of WAVEWATCH II. This model was explicitly designed for (vector) super computing, and focused on improved numerics (Tolman, 1992a,b). Development of WAVEWATCH III at NCEP started in 1993. Compared to previous WAVEWATCH models, this model uses modified basic equations, and introduces the present model architecture. This model utilizes vector optimization, together with OpenMP or MPI parallel optimization, and hence can be run efficiently on most modern computer architectures. With model version 3.14, WAVEWATCH III has been trademarked and copyrighted, and has been distributed under an open-source style license (see section 1.2 of Tolman, 2009, or the web site[1]). Henceforth, WAVEWATCH III will be denoted as WW3.

Five public releases of WW3 have been made available (Tolman, 1999, 2002, 2009; Tolman and The WAVEWATCH III ® Development Group, 2014), including the current release version 5.16 (**(alias?)**, 2016). This best practices guide was first provided with model versions 3.14 for two reasons. First, coding standards are needed to foster and support community model development. This has become particularly important with the National Oceanographic Partnership Program (NOPP) project to improve all basic wave model source terms, which will run from 2010 through 2014, and which will use WW3 as a main development vehicle. This means that several teams will work simultaneously on the WW3 code. Second, there is a need for unifying coding approaches within NCEP. A first set of standards has been developed for the Community Radiative Transfer Model (CRTM) as presented in Van Delst (2008). Whereas it is unrealistic to retrofit all NCEP codes to a completely homogeneous coding standard due to the shear size of legacy codes, all basic precepts should be the same, and are consistent between the present guide and Van Delst (2008).

From the beginning, WW3 has been envisioned as a modeling framework, with various options for numerical and physical approaches, both for operational and research applications. With a focus on operations at NCEP, selections of numerical and physical approaches are done at the compile level of the code. This limits the complexity of the source code that is used in operations. For example, complex exact interaction codes are not compiled into the operational models at NCEP, and hence do not need to be maintained in the operational code versions. Compile level code selections are made using the native WW3 preprocessor and 'switches' in the source code, as described in full in the WW3 manuals as identified above.

Starting with the release model version 3.14, we were maintaining the code using subservion (Collins-Sussmann et al., 2004) and now are maintaining the code via git.

[1] http://polar.ncep.noaa.gov/waves/wavewatch/license.shtml

The master version of WW3 will be maintained and supported at NCEP. With the licensing of model version 3.14, users that develop code and/or modifications for WW3 are obligated to offer these back to NCEP, relative to the most recent version of WW3 available to them[2]. NCEP will then decide if such modifications and additions will be included in the master version of WW3, and will be responsible for including it in the authoratitive git repository.

In this context, coding standards, best practices for upgrading parts of WW3 and for adding new pieces to WW3, regression testing, and maintenance of documentation are essential. These issues are approached in Sections 2 (also including copyright statements), 3, 4, and 5, respectively. Finally, Section 6 discusses standards of code management using the subversion server at NCEP.

Some formatting practices for the WW3 manual are used in this guide. The `file` font will be used to identify files, scripts and command line entries. The `CODE` font is used to identify source code. Previous experience with WW3 is expected, and the use of, for instance, optional switches in the code, will not be explained here in detail.

[2] Public release or research version on svn server, depending on user access.

2 Programming style

WW3 is written in ANSI standard Fortran 90, fully modular, and with an internal dynamic data structure exclusively using use-associated data modules. All modules are internally documented with a style of documentation as illustrated in Fig. 2.1 and 2.2 for subroutines and modules, respectively. Examples of this can be found throughout the source code, and templates are also provided in 'user slot' routines for source terms and propagation schemes.

Is should be noted that WW3 consists of incomplete ('`.ftn`') FORTRAN files that require standard WW3 preprocessing. All changes and additions should be made in these files, not in extracted true FORTRAN files (for details see the manual).

The following is expected of codes provided to NCEP for inclusion in the official version of WW3:

i) Fully document the code following the outline described above.

ii) Follow the coding style of WW3, in particular :

   - For readability, code is written following the use of columns as in fixed format Fortran, even though codes are technically written in free format. Use typical indent strategies for loops and logical structures.
   - Code intended as permanent code is written in upper case, temporary (test) code is written in lower case. Note that we encourage the inclusion of permanent test output to be activated at compile time using the WW3 compile switches[3] like '`!/T`'. The latter test output should be coded in upper case as a permanent part of the code.

iii) Maintain an update log at the top of each module and for each individual routine or function, and update the last update date in the header of each module, function and routine, as has been done in the distribution version of WW3. If a module only contains one program element, only a single update log needs to be maintained. This is a legacy from code management before using subversion, but will be retained until further notice.

iv) Each subroutine, function or grouping should be embedded in a module to allow for full use association and internal automatic interface checks in Fortran compilers. File naming conventions include:

   - File names for elements of the basic wave model should start with `w3`.
   - Program elements related to the multi-grid capability should start with `wm`.

[3] See manual for details on use of compile level switches.

- Module file names should end in md (before the file extension).
- Files with main programs should be stored in file names starting with ww3_.
- The file extension.ftn identifies code elements that need to be preprocessed by the WW3 preprocessor w3adc to activate switches.
- Files with ready-to-use source code (no need for the WW3 preprocessor) are identified by the extension .f90. This includes external packages interfaced to WW3.

As examples w3snl2md.ftn is a module of the basic wave model (one of the S_{nl} source term options) that needs to be preprocessed by the WW3 preprocessor. ww3_grid.ftn contains the main program for the grid preprocessing. mod_constants.f90 is a part of a user-supplied package that does not require WW3 code preprocessing. Note that the only file not following the WW3 naming convention is constants.ftn, which contains a module with physical constants.

v) For now, we have been using the Fortran 90 standard. Required coding practices include:

- Use free format with style as described above.
- Use `IMPLICIT NONE` in each module.
- Do not use `COMMON` declarations. Eventually all major data structures should become part of the WW3 dynamical data structures (see manual), which are all contained in separate modules, and can be used by use association. See section 3 for suggestions on how to deal with these data structures during (initial) code development.
- Each module used in a given program element will need to be use associated with a `USE` statement. Where feasible, use

  `USE` *module_name*`, ONLY:` *used names*

  to avoid unintended use of variables in modules.
- For the same reason, use `PRIVATE` for general declarations in modules.
- Declare `INTENT` on all dummy argument list items.
- Do not use tab characters in the code (not in Fortran character set).
- Name `END`s fully both for readability and because several compilers will require this.
- Do not use numbered `DO` loops.
- Use `CYCLE` and `EXIT` instead of `GOTO`.

- Use `CASE` statements with a default rather than `IF` statements for multiple selection tests.
- As a holdover of days long gone, short variable names have been used throughout the WW3 code. Although this makes it easy to keep documentation readable, it does not necessarily make it easy to understand the code at a glance. Feel free to use longer variable names to make the code more easily understandable.
- Up to now, there has been no need for explicit `KIND` declarations in WW3. If such declarations are needed, follow the standard set in Van Delst (2008).

vi) Provide documentation for the modules to be included in the WW3 manual. The manual is written in LaTeX. Required manual elements to be provided are

- Description or update of basic equations / physical parameterizations as needed.
- Description or update of numerical approaches as needed.
- Update of system documentation including description of parameters in the dynamical data structure of WW3.
- Document namelist options as applicable in ww3_grid in full in the example input file ww3_grid.inp. The example file is included in full in the manual.

The coding style does not imply that existing packages that are attached to WW3 need to be re-written in this style. However, it is strongly recommended that any such package should be fully documented inside the code. Typically, a user provided package will require an interface routine to WW3. Such an interface routine is expected to conform to the WW3 coding practices.

Note that the NWS claims copyright for all main elements of WW3, and generally will claim copyright for interface routines. Providers of packages to be included with the distribution of WW3 are encourage to provide copyright statements and disclaimers in these packages as appropriate (as NWS will not claim copyright of such packages).

```
!/ ------------------------------------------------------------------- /
      SUBROUTINE W3XXXX
!/
!/                  +-----------------------------------+
!/                  | WAVEWATCH III           NOAA/NCEP |
!/                  |            John Doe               |
!/                  |                        FORTRAN 90 |
!/                  | Last update :         01-Jan-2010 |
!/                  +-----------------------------------+
!/
!/    01-Jan-2010 : Origination.                        ( version 4.xx )
!/
!  1. Purpose :
!  2. Method :
!  3. Parameters :
!
!     Parameter list
!     ----------------------------------------------------------------
!     ----------------------------------------------------------------
!
!  4. Subroutines used :
!
!      Name      Type  Module   Description
!     ----------------------------------------------------------------
!      STRACE    Subr. W3SERVMD Subroutine tracing.
!     ----------------------------------------------------------------
!
!  5. Called by :
!
!      Name      Type  Module   Description
!     ----------------------------------------------------------------
!     ----------------------------------------------------------------
!
!  6. Error messages :
!  7. Remarks
!  8. Structure :
!  9. Switches :
!
!     !/S  Enable subroutine tracing.
!
! 10. Source code :
!
!/ ------------------------------------------------------------------- /
!/S      USE W3SERVMD, ONLY: STRACE
!/
      IMPLICIT NONE
!/
!/ ------------------------------------------------------------------- /
!/ Parameter list
!/
!/ ------------------------------------------------------------------- /
!/ Local parameters
!/
!/S      INTEGER, SAVE           :: IENT = 0
!/
!/ ------------------------------------------------------------------- /
!/
!/S      CALL STRACE (IENT, 'W3XXXX')
....
!/
!/ End of W3XXXX ----------------------------------------------------- /
!/
      END SUBROUTINE INSBTX
```

Fig. 2.1 : Documentation template for subroutines. Note that each subroutine is expected to include a call to the STRACE *subroutine to enable subroutine tracing inside WW3,*

```
!/ ------------------------------------------------------------------- /
      MODULE W3XXXXMD
!/                  +-----------------------------------+
!/                  | WAVEWATCH III           NOAA/NCEP |
!/                  |              John Doe             |
!/                  |                        FORTRAN 90 |
!/                  | Last update :         01-Jan-2010 |
!/                  +-----------------------------------+
!/
!/    01-Jan-2010 : Origination.                        ( version 4.xx )
!/
!/    Copyright 2010 National Weather Service (NWS),
!/       National Oceanic and Atmospheric Administration.  All rights
!/       reserved.  WAVEWATCH III is a trademark of the NWS.
!/       No unauthorized use without permission.
!/
!  1. Purpose :
!  2. Variables and types :
!
!      Name      Type  Scope    Description
!     ----------------------------------------------------------------
!     ----------------------------------------------------------------
!
!  3. Subroutines and functions :
!
!      Name      Type  Scope    Description
!     ----------------------------------------------------------------
!      W3XXXX    Subr. Public   ........
!     ----------------------------------------------------------------
!
!  4. Subroutines and functions used :
!
!      Name      Type  Module   Description
!     ----------------------------------------------------------------
!      STRACE    Subr. W3SERVMD Subroutine tracing.
!     ----------------------------------------------------------------
!
!  5. Remarks :
!  6. Switches :
!
!     !/S  Enable subroutine tracing.
!
!  7. Source code :
!/
!/ ------------------------------------------------------------------- /
!/
      PRIVATE
!/
      CONTAINS
!/ ------------------------------------------------------------------- /
      SUBROUTINE W3XXXX
.....
!/
!/ End of w3XXXX ----------------------------------------------------- /
!/
      END SUBROUTINE W3XXXX
!/
!/ End of module W3XXXXMD -------------------------------------------- /
!/
      END MODULE W3XXXXMD
```

Fig. 2.2 : Documentation template for modules. Copyright statement to be adapted as appropriate.

This page is intentionally left blank.

3 Adding to the model

WW3 is designed as a highly plug-compatible code. Source term and propagation approaches can be included as self-contained modules, with limited changes needed to the interface of routine calls in W3SRCE, W3WAVE, and in the point post-processing programs only. General users can experiment with new approaches in user slots that are provided as dummy model slots like W3SNLX in the file w3snlxmd.ftn for the nonlinear interactions. General users are expected to provide these 'user slot' routines to NCEP for inclusion in subsequent versions of WW3, following the instruction in this guide and in the documentation of routines like W3SNLX. Such codes should be self-contained in the way described below.

When providing a module for a source term like W3SNLX or for a propagation scheme the following programming guidelines should be followed:

i) Follow coding guidelines as outlined in the previous section.

ii) Provide a file with necessary modifications to W3SRCE and all other routines that require modification.

iii) Provide a test case with expected results.

Furthermore, the module needs to be self-contained in the following way.

i) All saved variables connected with this source term need to be declared in the module header. Upon acceptance as permanent code, they will be converted to the WW3 dynamic data structure.

ii) Provide a separate computation and initialization routine. In the submission, the initialization should be called from the computation routine upon the first call to the routine. Upon acceptance as permanent code, the initialization routine will be moved to a more appropriate location in the code (i.e., being absorbed in ww3_grid or being moved to W3IOGR).

When such packages are provided to NCEP, NCEP may choose to not include the package, or to provide the package as a 'user slot routine' like W3SNLX, with some minor work of users required to install these routines, or may choose to fully integrate the routines as a standard option in WW3.

Co-developers of WW3 with access to the subversion server are expected to fully integrate the new modules in the experimental versions of WW3, using software selection switches as provided by the NCEP code managers. It is, nevertheless, strongly recommended that initially data structures are kept internal to the modules that are being developed, and that data for the modules are only included in the dynamic data structure of WW3 when the module is mature. This will make code development and unification much easier when multiple developers are working on the code simultaneously.

The integration of new modules may involve adding new software selection switches to the source code. When this happens it is necessary to also add the new software selection switches to the build scripts. Specifically, the new switches must be properly encoded in model/bin/make_file.sh and model/bin/w3_new. This is required so that switch incompatibilities can be trapped at compile time and source files affected by changes to model/bin/switch can be correctly identified and recompiled. Documentation for new software selection switches must be added to Chapter 5.4 of the manual.

The above approach are applicable to inherently modular elements of WW3 such as source terms or propagation schemes. For more intricate changes to the code, please consult the WW3 code managers[4] on how to proceed with developing and providing code upgrades.

[4] Mail to NCEP.EMC.wavewatch@NOAA.gov

4 Regression testing

Up to model version 3.14, the wave model was distributed with a suite of test cases in the tests directory. This set of tests was inconvenient to use for regression testing, as it needed manual intervention and compilation of the code for each individual test. In model version 4.07, an automated script was introduced by Erick Rogers and Tim Campbell of NRL Stennis to convert the traditional tests cases to regression tests that could be run in an automated manner. The automated regression tests were first maintained in the nrltest directory, and are now kept in the regtests directory. The previous tests directory has been removed, moving the remaining real-world examples in the new cases directory. A more complete documentation of the test cases and of the regression test tools can be found in Section 5.6 in The WAVEWATCH III ® Development Group (WW3DG) (2019). The regression tests in regtests cover basic tests to assure that features such as propagation and source terms work. The following conventions are used in naming the test cases:

- ww3_tp1.*n*: Tests for one-dimensional propagation (regular grids).
- ww3_tp2.*n*: Tests for two-dimensional propagation (all grids).
- ww3_ts*n*: Tests for source term integration (some including propagation).
- ww3_bt*d*.*n*: Tests for wave-bottom interaction. ($d = 1$ or 2, indicating one- or two-dimensional test cases.)
- ww3_ic*d*.*n*: Tests for wave-ice interaction. ($d = 1$ or 2, indicating one- or two-dimensional test case.)
- mww3_test_*nn*: Tests cases for the multi-grid wave model.

these idealized test cases should be used for regression testing when new code is added to the repository; i.e., these tests should result in different answers compared to the previous codes only when expected.

More generally, there are two reasons for providing regression tests. The first is the responsibility of a code developer not to break previously existing code. The regression testing gives a developer the tools to check this systematically and rigorously. The second is to avoid that a contributor of code has to provide large efforts in keeping the contributed code functional. By providing a regression testing for your new code, the responsibility of keeping code function will first and foremost fall on developers of new code, not on the contributor of already accepted code. As already outlined above, this implies that developers of new model options should both run relevant regression tests during development, and provide new (options for existing) regression tests for their code additions. Some more details of how and when to regression test will follow below.

To facilitate regression testing, a shell script, regtests/bin/run_test is provided along with input files for all test cases. The test cases are organized such that each major test case has its own directory. Within each test case directory, it is possible to have more test cases, for instance by compiling the code with different switches, using different grid configurations, using different model inputs, and running the code in either a shared or a distributed compute environment. A regression test is run using command-line commands only. All options can be displayed by running the script without arguments or with the -h option

```
run_test

run_test -h
```

the output of which is displayed in Fig. 4.1 at the end of this section.

When editing input files, it is recommended that developers avoid "checking in" trivial changes as updates to the git repository. In the case of ww3_grid.inp and switch, this can easily be avoided by keeping non-version-controlled variants of these files for local editing and accessing them via the -g and -s switches, respectively.

Most WW3 developers will not have reason to edit the run_test script itself, i.e. it is treated as a black box. However, developers who would like to extend the capabilities of the script (e.g. by adding new command line switches) and commit these changes to the repository for sharing are encouraged to do so.

With the set of simple test cases, an a large number of switch and grid options, several hundred unique regression tests have been created. A matrix of all possible regression tests can be generated with the tools provided in the regtests/bin directory, as described in Section 5.6 in The WAVEWATCH III $^{®}$ Development Group (WW3DG) (2019). The tools also allow for quickly sub-setting the full matrix by test case or other filter options. The NCEP code managers will run the full matrix of regression tests for trunk updates, running the matrix on both the old trunk, and the proposed new trunk, and using the tools in regtests/bin to do bit-wise comparisons of the model results. As it may take up to 12 hours to run the full matrix on our supercomputers, it makes no sense to run the full set of regression tests during development of new options.

During development of code, common sense should prevail. For instance, when developing a new source term package, it is worth while to regularly run some of the source term tests to assure that the other source term options are not broken, and only occasionally run the entire matrix. Note that the NCEP developers do expect that a branch presented for integration with the trunk has been tested at least once with the entire matrix of regression tests by the developer.

There are two basic ways in which the regression testing can be done. One is to keep copy of the trunk from which the development branch is created, and have a set of regression tests run for both the trunk and the development branch, and compare the results with the tools provide. The other is to do the regression testing "in place". In this case, when creating a branch, run the selected regression test(s) to create a baseline identifying it with a version identifier. For instance, checking the PR3 and UQ propagation options while working on another propagation scheme, a baseline could be generated with (and possibly other command line options)

```
./bin/run_test -s PR3 -w work_PR3_OQ_v4.00 ../.  ww3_tp1.1
```

and after modifications are made to the code, the regression test is re-run in a in a different work directory, e.g.,

```
./bin/run_test -s PR3 -w work_PR3_UQ_v4.01 ../.  ww3_tp1.1
```

after which results can be compared using tools provided or with direct Linux file comparison tools. Note that for a test case to run to completion is a necessary but not sufficient indication of success.

Note that run_test has utility beyond regression testing. Perhaps the most useful feature is that it makes it possible to fully document (or communicate) a model simulation using a single line of text, "./bin/run_test ..." without ambiguity. Furthermore, the utility allows for direct comparison of options in the model. For instance, the GSE test can be used to easily generate GrADS output for all different propagation schemes on a regular grid with otherwise identical model settings.

```
Usage: run_test [options] source_dir test_name
Required:
  source_dir : path to top-level of WW3 source
  test_name  : name of test case (directory)
Options:
  -a ww3_env       : use WW3 environment setup file <ww3_env>
                   :   *default is <source_dir>/bin/wwatch3.env
                   :   *file will be created if it does not already exist
  -c cmplr         : setup comp & link files for specified cmplr
  -C coupl         : invoke test using <coupl> coupled application
                   :   OASIS : OASIS3-mct ww3_shel coupled application
                   :   ESMF  : ESMF ww3_multi coupled application
  -d               : invoke main program using gdb (non-parallel)
  -e               : prompt for changes to existing WW3 environment
  -f               : force pre- and post-processing programs to be compiled
                   : non-MPI (i.e., with SHRD switch); default is all programs
                   : compiled with unmodified switch settings
  -g grid_string   : use ww3_grid_<grid_string>.inp
  -G               : create GrADS data files using gx_outX.inp
  -h               : print usage and exit
  -i inpdir        : use inputs in test_name/<inpdir> (default  test_name/input)
  -m grid_set      : execute multi-model test
                   :   *grid names are obtained from input/<grid_set>
                   :   *ww3_multi_<grid_set> will execute instead of ww3_shel
                   :   *to execute a single model test case with ww3_multi use
                   :    grid_set = none
  -n nproc         : specify <nproc> processors for parallel run
                   :   *some <runcmd> programs do not require <nproc>
                   :   *ignored if -p <runcmd> or -O is not specified
  -N               : use namelist (.nml) input instead of .inp (if available)
  -o outopt        : limit output post-processing based on <outopt>
                   :   native : post-process only native output
                   :   netcdf : post-process only NetCDF output
                   :   both   : post-process both native and NetCDF output
                   :   * default is native
                   :   * note that required input files must be present for
                   :     selected output post-processing to occur
  -O               : parallel run using OpenMP paradigm and OMP_NUM_THREADS
                     environment variable and number of processors defined with
                     the -n np option
  -p runcmd        : run in parallel using <runcmd> to start program
                   :   *MPICH or OpenMPI: mpirun or mpiexec (default <nproc> = 1)
                   :   *IBM with Loadleveler: poe (no <nproc> required)
                   :   *LSF: mpirun.lsf (no <nproc> required)
  -q program       : exit script after program <program> executes
  -r program       : only execute program <program>
  -s switch_string : use switch_<switch_string>
  -S               : create stub file <finished>. with end data and time.
                     tests not executed if file is found.
  -t nthrd         : Threading option. (this is system dependant and can be used
                   : only for the hybrid option)
  -w work_dir      : run test case in test_name/work_dir (default test_name/work)
```

Fig. 4.1 : Description of arguments for run_test *script.*

5 Manual and documentation

The WW3 manual and other WW3 documents like this guide are written in LaTeX. Since these are dynamic documents, the corresponding files are maintained in git, together with the WW3 source code, script and auxiliary files. Because the manual is rather large, it has been stored in several .tex files. During the development of model version 4.18, most sub-sections were placed in their own file, to minimize conflicts in editing when many contributors work on the manual simutaneously. The main files (and directories) making up the manual are

manual.tex	Main .tex file, mainly combining the .tex files below into the complete manual.
defs.tex	User defined LaTeX constructs used in the manual.
start.tex	Title page and table of contents set up.
intro.tex	Chapter: Introduction, using directory intro.
eqs.tex	Chapter: Governing equations, using directory eqs.
num.tex	Chapter: Numerics, using directory num.
run.tex	Chapter: Running the model, using directory run.
impl.tex	Chapter: Installing the model, using directory impl.
sys.tex	Chapter: System documentation, using directory sys.
app.tex	Appendices, using directory app.
manual.bib	BibTex database with references used in the manual.
jas.bst	Bibliography style file used for the manual.

All files for the main chapters and appendices manage individual LaTeX files for individual sections (appendices) in the chapter. This way, new options are documented in their own sub-file, and can therefore be easily managed. See the main chapter files for the contents of the chapter directories. Note that the chapter directories also include files for many individual figures.

A makefile is provided to compile the manual. The default make target will compile a PDF version of the manual (manual.pdf). Other make targets available are: "inp" (create LaTeXversions in current directory of the ww3_*.inp and gx_*.inp input files), as well as outputs of the ww3_gspl.sh and run_test scripts, "clean" (remove all files created during compile of manual). The following external programs are required and must be found in the user PATH: latex, bibtex and dvipdf.

The example input files (ww3_*.tex and gx_*.tex) required for the programs described in run.tex are automatically generated during compilation of the manual. The source input files are copied from $(INPDIR). The default setting for INPDIR, "../inp", can be overridden by setting INPDIR either on the make command-line or in the environment.

This method assures that the example input files provided with the code are the files displayed in the manual.

Note that the manual consist of both a conventional manual and a basic system documentation. The following standards should be used in writing LaTeX contributions to the manual:

- Use American spelling and grammar.
- Use dynamic references to equation, chapter and section numbers, etc. Do not use any hardwired reference numbers when referring to equations, sections etc.
- Use BibTex exclusively for references to other work. Do not write any references directly into the text.
- Do not use excessive line lengths in the `.tex` files. We typically use a maximum line length of 78 characters and 'auto-fill-mode' when writing or updating `.tex` files using emacs.
- When adding contributions to the manual, add a note of the update to the introduction, so that users of the public releases have a concise log of upgrades since the previous model release.
- If you have no LaTeX capability or experience, contact the WW3 code managers to determine an acceptable method of delivering contributions to the manual.

For general users we will provide a recent manual package when they are ready to provide their manual contributions. For co-developers, the most recent version of the manual will be available on github.

6 Git repository

Starting with model version 3.14, the WW3 model is maintained using subversion (Collins-Sussmann et al., 2004). Now the code is maintained via git on github. Information on how to contibute code and use git will now be maintained on the github wiki. Please see the github wiki for more information.

This page is intentionally left blank.

References

Collins-Sussmann, B., B. W. Fitzpatrick and C. M. Pilato, 2004: *Version control with subversion.* O'Reilly, 320 pp.[1]

The WAVEWATCH III ® Development Group (WW3DG), 2016: User manual and system documentation of WAVEWATCH III ® version 5.16. Tech. Note 329, NOAA/NWS/NCEP/MMAB, College Park, MD, USA, 326 pp. + Appendices.

The WAVEWATCH III ® Development Group (WW3DG), 2019: User manual and system documentation of WAVEWATCH III ® version 6.07. Tech. Note 333, NOAA/NWS/NCEP/MMAB, College Park, MD, USA, 465 pp. + Appendices.

Tolman, H. L., 1989: The numerical model WAVEWATCH: a third generation model for the hindcasting of wind waves on tides in shelf seas. Communications on Hydraulic and Geotechnical Engineering 89-2, Delft Universtity of Technology, ISSN 0169–6548, 72 pp.

Tolman, H. L., 1990: Wind wave propagation in tidal seas. Communications on Hydraulic and Geotechnical Engineering 90-1, Delft Universtity of Technology, ISSN 0169–6548, 135 pp. (Doctoral Thesis).

Tolman, H. L., 1991: A third-generation model for wind waves on slowly varying, unsteady and inhomogeneous depths and currents. *J. Phys. Oceanogr.*, **21**, 782–797.

Tolman, H. L., 1992a: Effects of numerics on the physics in a third-generation wind-wave model. *J. Phys. Oceanogr.*, **22**, 1,095–1,111.

Tolman, H. L., 1992b: Effects of the Gulf Stream on wind waves in SWADE. in *Proc. 23rd Int. Conf. Coastal Eng., Venice, Italy*, pp. 712–725. ASCE.

Tolman, H. L., 1999: User manual and system documentation of WAVEWATCH III version 1.18. Tech. Note 166, NOAA/NWS/NCEP/OMB, 110 pp.

Tolman, H. L., 2002: User manual and system documentation of WAVEWATCH III version 2.22. Tech. Note 222, NOAA/NWS/NCEP/MMAB, 133 pp.

Tolman, H. L., 2009: User manual and system documentation of WAVEWATCH III ™ version 3.14. Tech. Note 276, NOAA/NWS/NCEP/MMAB, 194 pp. + Appendices.

Tolman, H. L. and The WAVEWATCH III® Development Group, 2014: User manual and system documentation of WAVEWATCH III ® version 4.18. Tech. Note 316, NOAA/NWS/NCEP/MMAB, 282 pp. + Appendices.

Van Delst, P., 2008: CRTM: Fortran95 coding guidelines. Technical report, Joint Center for Satellite Data Assimilation.

[1] Updated versions available online at http://subversion.tigris.org/.