Regression testing

W. Erick Rogers
Oceanography Division
Naval Research Laboratory
Stennis Space Center, MS

The WAVEWATCH III Team + friends
Marine Modeling and Analysis Branch
NOAA / NWS / NCEP / EMC

NCEP.list.WAVEWATCH@NOAA.gov
NCEP.list.waves@NOAA.gov
Outline

Covered in this lecture:

- additional documentation
- naming convention for regression tests
- run_test script
- compilers and MPI
- bugs and broken things
Additional documentation

- WAVEWATCH III® development best practices
  - NCEP website (v3.14):
    http://polar.ncep.noaa.gov/waves/wavewatch/wavewatch.shtml#documentation
  - In repository (v4, restricted access): {branch or trunk}/guide/report.pdf

- run_test wikis (restricted access):
  - https://svnemc.ncep.noaa.gov/trac/ww3/wiki/NrlTest
  - https://www7320.nrlssc.navy.mil/Alvin/index.php/WW3_Test_Cases
Naming conventions for test cases:

- `ww3_tp1N`: Tests for one-dimensional propagation
- `ww3_tp2N`: Tests for two-dimensional propagation
- `ww3_tp3N`: Tests for two-dimensional propagation in unstructured grids
- `ww3_tsN`: Tests of source terms
- `ww3_tpsN`: “realistic” tests with both propagation and source terms
- `mww3_testN`: Simple tests for multi-grid wave model
- `mww3_caseN`: “realistic” tests for multi-grid wave model
Note added by Erick Feb 12 2013 (after class ended) : Some content on this slide is not legible via projector. Prior to use in a subsequent class, the “screen grabs” should be converted to larger font powerpoint text boxes.

NCEP svn server
Note added by Erick Feb 12 2013 (after class ended) : Some content on this slide is not legible via projector. Prior to use in a subsequent class, the “screen grabs” should be converted to larger font powerpoint text boxes.

/regtests/

regtests on NCEP svn server
run_test

run_test script introduction:

- shell script: `/regtests/bin/run_test`
  - `diff_ww3, run_suite, cleanup, run_cmp` provided in same directory: not covered here, but may also be useful
- each major test case occupies a directory, e.g. `/regtests/ww3_tp1.1/
  - sub-types are available via multiple “run-time selectable” variants of `switch, ww3_multi.inp, and ww3_grid.inp` in `/regtests/{test name}/input/`
  - additional sub-types can be created by individual users
  - “run-time selectable” variants of `ww3_shel.inp, ww3 strt.inp, ww3_trck.inp, ww3_bound.inp` are not supported (yet), but users may customize them and/or use `cp` or `ln` to select variants
  - multiple variants of `ww3 outf.inp, ww3 ounf.inp, ww3 outp.inp, ww3 ounp.inp, ww3 prep.inp` exist. These are not “run-time selectable”: all existing variants are executed with every run.

If you have added a feature, you must also add a regression test (or sub-test) which utilizes your new feature.
**run_test script execution:**

- execute without arguments (or with –h argument) for screen dump re: usage
- -c : change compiler (required on first use)
- -g : select a non-default grid (single grid cases)
- -i : select a non-default directory for input files
- -s : select a non-default switch file
- -w : use a non-default work directory
- -m : select grid set (multi-grid cases)
- -r : run only one program (e.g. `ww3_prep`)
- -q : quit after running program (e.g. `ww3_grid`)
- -n and –p : instructions for parallel runs
- -a and –e : select or modify `wwatch3.env` file
- -o select standard or NetCDF output
run_test script execution:

- running run_test for the first time in a “fresh” export/checkout of a branch:
  - `wwatch3.env` file will be created interactively*
  - “-c” must be used to set `comp` and `link` files*
- most run_test arguments are optional (i.e. defaults exist)
  - exception: multi-grid case require “-m”
  - “-s” is needed if file with default name `{test name}/input/switch` does not exist
- example commands:
  - `./bin/run_test -s ST1 ../model/ ww3_ts1`
  - `./bin/run_test -m grdset_a -n 3 -p mpirun -s PR3_MPI_SCRIP -w work_a_PR3_MPI_SCRIP ../model mww3_test_03`

* if not done manually
When to perform regression testing:

- after major changes to your branch
  - thorough set of tests needed
- prior to merge from your branch to trunk
  - thorough set of tests needed
- after minor/incremental change to your branch
  - usually a single regression test is enough

Which regression tests to use:

Common sense applies. Examples:

- if you are adding a new dissipation term, then you probably didn’t break the propagation (lower priority to test)
- if you are modifying NL1 code, you should check impact on all source term packages (ST1, ST2, etc.) (higher priority to test)
How to use results:

- verify that test runs to completion
  - this is sufficient to catch most problems
- however, if you have reason to worry (e.g. if you have made major changes)
  - run earlier version of code, and verify that differences are expected (non-graphical method: can use “diff” on the work directories), or
  - visualize results, or
  - both of the above
Compilers:

- Developers should occasionally test with an alternate compiler
- This is much more critical than you might expect.
- Everyone should include gfortran in their testing (since it’s free, you don’t have a good excuse not to...)

MPI:

- Developers should test with MPI prior to any merge to trunk
- This can be done on a workstation
- Rule applies even if your changes have no obvious connection with MPI

More machines, more compilers, more eyeballs ➔ more likely to find problems
Bugs and otherwise broken code

- Find bugs early via `run_test`, ideally, before you commit to repository
  - bugs found late create version-control nightmares
- If you find out too late that your code is broken:
  - halt all unrelated development in this branch until problem is fixed
  - if bug remains unresolved, you may want to back out the problem revision, or split off a new branch from a prior bug-free revision
- Think ahead: If you have the (good) habit of checking in revisions in small increments, this will make it easier to find out when/where/why code became broken.
  - brute force method: combine “svn export” with “run_test” to identify revision
What to do if you think that the code is broken

- If the problem is in your branch only
  - make sure that it doesn’t get merged to trunk
  - work with your branch’s dev team to fix it
- If the problem is more general
  - Notify the “trunk authorities”. At time of writing, this is H. Alves. Do not complain to a 3rd party and expect him/her to notify the authorities.
  - Create/identify a test case that can be executed with run_test which exhibits the problem. If you created a new test case, check it into the repository. (In some cases, it may be best to create a new branch for this. There is also a “regtests” branch that can be used.)
  - Open a ticket (or help trunk authorities to do so). Include the run_test command in the ticket documentation.
The end

End of lecture
NRL wiki re: run_test cases
NRL wiki re: run_test cases

Example run_test commands:

```
./bin run_test -g precurs -s PR1_precurs -w work_PR1_precurs $WW3_precurs ww3_tpi.3 # baseline
./bin run_test -g precurs -s PR2_precurs -w work_PR2_precurs $WW3_precurs ww3_tpi.3 # baseline
./bin run_test -g precurs -s PR3_precurs -w work_PR3_precurs $WW3_precurs ww3_tpi.3 # baseline
./bin run_test -s PR1 -w work_PR1_v4.01 $WW3_curve ww3_tpi.3 # verify
./bin run_test -s PR2 -w work_PR2_v4.01 $WW3_curve ww3_tpi.3 # verify
./bin run_test -s PR3 -w work_PR3_v4.01 $WW3_curve ww3_tpi.3 # verify
```

PR2, GSE control is OFF by default (perhaps because it affects stability) PR3, GSE control is ON by default

Verification: diff of ascii results (curvilinear=capable code v4.01) vs (baseline=pre-curvilinear code, v4.00)

**ww3_tpi.4**

Test script for WW-III, one-dimensional propagation. spectral refraction (X).
Summary: tpi.4 = N R 1d refraction (X) (left-right)
Origination: Hendrik Tolman, Jun 2002