The ADCIRC Developers Guide

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Introduction **1**

2 Development Strategy **1**

3 Coding Standards **1**

3.1 The Basics 1

3.2 Maintainability 1

4 Release Process **1**

5 Subversion **2**

5.1 Policies 2

5.2 Instructions 3

5.2.1 Compare Revisions 3

5.2.2 Package Up Code From Repository 3

5.2.3 Pull Old Version from Repository 3

5.2.4 Make a Branch 3

1 Introduction

This document describes the guidelines that ADCIRC developers should use when developing new code for ADCIRC and its related utilities. It also presents a short tutorial for Subversion (svn), the version control software that we use to coordinate development and ease the task of merging in new features.

2 Development Strategy

The ADCIRC development strategy consists of the maintenance of two separate versions at any given time: the stable version and the development version. The stable version only receives bug fixes, while the development version receives bug fixes as well as new features.

At some point, the development version becomes stable. At that time, a stable branch is created from it, and then new features are added to the previosly stable code, creating a new development version. At that time, the major version number will be incremented, e.g., stable v48.xx will become the next development version as v49.00.

The minor version number is changed each time there is a change in the code and the modified code is committed back to the repository. Details of the implementation of this strategy with Subversion are provided in the section entitled "Subversion".

3 Coding Standards

ADCIRC has had many individual contributors and has received code accretions over many years. A set of uniform coding standards has not been defined, and as a result, ADCIRC contains many different styles of Fortran. This section provides a set of style guidelines for contributing code to ADCIRC.

3.1 The Basics

- IMPLICIT NONE at the beginning of each subroutine.
- Fixed form, never exceeding the 72 column limit, even for comment lines.
- When adding code to an existing subroutine, make the new code match the style of the surrounding code, even if you'd prefer another style.

3.2 Maintainability

When adding code that will be used in slightly different ways in different contexts, make it a subroutine, rather than cutting and pasting several times and making small changes to each cut-and-pasted section. Although it is faster to write code with cut-and-paste, the resulting code is harder to maintain, since each cut-and-pasted section will have to be modified individually later. Also, it is easier to make mistakes when working with cut-and-pasted code.

Rick has expressed a desire for greater modularity … particularly with the number of variables in the global module. When adding a major new feature, please consider the modularity of the data it requires. In other words, if new variables can be made private to a particular module rather than global, please do so.

4 Release Process

The release process informal, but generally includes the following steps:

- Set and stick to a list of new features as the goal for the new release.
- Have a consensus among ADCIRC developers that the goal has been achieved.

- Run tests that cover the new features as well as making sure old features still work.
- Fix issues revealed by failed tests.
- Distribute the release candidate code to collaborative users to make sure their runs perform as expected on the new code.
- Fix issues revealed by user dismay.
- Update the docs to reflect the changes. If input files or file formats have changed, produce release notes to highlight the changes.
- Publicly announce, release, and brag about a shiny new version of ADCIRC.

These tests are often referred to as "regression tests," i.e., a means to detect a regression in correctness or functionality due to changes in the code. A regression test suite can be built up pretty easily. There are a number of small test cases on adcirc.org, so the initial set of tests should consist of those. We may also want to differentiate between tests for correctness and tests for functionality. The former is making sure you're getting the expected model results and the latter is making sure that nothing is broken (i.e., adcprep, i/o, message passing, etc).

The test suite gets built up based on different purposes. One may want to exercise more options and more cases, but it is also important to test for known bugs that might have been reintroduced. In otherwords, once a bug is discovered and fixed, a test should be created specifically for that bug to make sure it doesn't appear again.

And lastly, a small test suite should be included with each distribution and a more comprehensive should be created that runs during development and before a release. A step in this direction has been made using ant in the autotest directory.

5 Subversion

Subversion is a version control system. That is, it provides a means for many software developers to collaborate on a single software project.

A subversion repository is a storehouse of code for a particular project. Subversion repositories consist of a trunk, which is the mainline code that everyone shares, and any number of branches that are created by and for individual users.

A branch may be created by a developer that has a lot of changes to make over a longer period of time. This lone developer can work on a branch without affecting the mainline code. Once the changes in the branch are satisfactory, the branch can be merged back to the trunk, making the changes available to all.

A further advantage of Subversion is that it automates the process of merging new features into ADCIRC. For example, in the past, without Subversion, developers modified their local copies of ADCIRC and wanted to merge the modifications into the mainline code. This required hours or days of painstakingly examining each change they had made and cutting and pasting the changes one by one into the up-to-date version of ADCIRC. With Subversion, the software examines the changes that were made locally and makes corresponding changes to the mainline version in an automated way.

5.1 Policies

Working with subversion requires greater coordination among ADCIRC developers. For example, if two developers change the same line of code in two different branches in two different ways, this will create a conflict that will have to be resolved manually.

Furthermore, if one developer makes changes to trunk that prevent the code from compiling, it will be difficult for other developers to continue working if they update to the latest code. As a result, ADCIRC development with Subversion will adhere to the following policies:

- Communicate. Jason Fleming is responsible for making sure ADCIRC development is smooth, pain-free and productive---when in doubt, email him (jason.fleming@seahorsecoastal.com).

*Also use the adcirc-dev mailing list to keep everyone informed of what you are doing. *If you are not on the adcirc-dev mailing list and would like to be, email Jason Fleming. * Make a branch to develop new features, rather than making changes to trunk. * Don't commit changes to trunk that prevent the code from compiling, at least on your platform. You should also confirm that your code compiles with and without NetCDF.

5.2 Instructions

Comprehensive documentation for Subversion is available: http://svnbook.red-bean.com/.

The ADCIRC code is hosted at the following Subversion repository location:

```
https://adcirc.renci.org/svn/adcirc
```

To check out code from the trunk, please type

```
svn checkout https://adcirc.renci.org/svn/adcirc/trunk
```

Here are a few examples of things that we commonly do with svn.

5.2.1 Compare Revisions

In order to see the changes between revisions of the repository, type (for example)

```
svn diff  -r 15:16 owiwind.F
```

5.2.2 Package Up Code From Repository

In order to create a tar file that contains just the ADCIRC code and excludes the .svn directories, the first step is to export the code to a new subdirectory. For example:

```
svn export ./trunk ./adc99_99
```

Would copy the local files under version control from the trunk subdir to the adc99_99 subdir, excluding the .svn files. Particular revisions of the svn repository can also be extracted, see the svn documentation for details on how to do this.

5.2.3 Pull Old Version from Repository

One advantage of using svn is that it is easy to go back to an older version of the code, if necessary. The first thing to do in this case is usually to look at the svn log to see what changes were made, so that you can select the right version of the repository to extract. Go to the subdirectory of the code tree that you want an older version of and issue the command

```
svn log
```

Look at the log entries, and note the revision that you'd like to extract. Let's say you want to retrieve adcirc48/trunk at revision 65 of the repository (for example). Issue the following command:

```
svn checkout -r 65 https://adcirc.renci.org/svn/adcirc/trunk
```

5.2.4 Make a Branch

To make a branch off trunk to add a new feature and call it the myfeature branch:

```
svn copy https://adcirc.renci.org/svn/adcirc/trunk https://adcirc.renci.org/svn/adcirc/bra
```

Then to start using the newly created branch:

```
svn checkout https://adcirc.renci.org/svn/adcirc/branches/myfeature
```