CODING guide for ORAC

Original guidelines: C. Poulsen: caroline.poulsen@stfc.ac.uk
Rutherford Appleton Laboratory

CHANGE RECORD

Issue	Date	Pages	Added by	Reason for change
1	11/08/2010	All	CP	New document
2	22/08/2010		CP	changes after consultation with ORAC group

Introduction

This document describes the coding guidelines that should be followed when writing or editing the ORAC community aerosol and cloud code. The documents are meant to be a living document. Comments and suggestions are welcome. The purpose of this document is to maintain and produce code that:

- maximize readability and comprehensibility
- encourage uniformity
- simplify maintenance
- assist portability
- promote reliability and fault tolerance

Licence agreement.

A copy of the ORAC licence agreement can be found here XXXXX
By downloading and using the ORAC code you are agreeing to

1. Acknowledging ORAC in any relevant publications
2. Feeding back into the code any improvements made/bugs found.

Specific guidelines for writing and editing ORAC community code

Configuration control

All code should be stored under configuration control.
The code repository can be found at http://proj.badc.rl.ac.uk/orac

Instructions on how to use the configuration control can be found in the ORAC SVN miniguide http://proj.badc.rl.ac.uk/orac/wiki/SVNGuide and the official subversion manual can be found at http://svnbook.red-bean.com/

Template

The following template should be used for all new routines

```
! Name:
!
!
! Purpose:
!
!
! Description and Algorithm details:
!
!
! Arguments:
!   Name      Type    In/Out/Both    Description
!
!
! Local variables:
!   Name      Type    Description
!
!
! History:
!
!
! $Id$
!
! Bugs:
!
!
```

Name: The name should fully describe the function of the routine
Purpose: A very short written description of the purpose of the routine
Description and algorithm details: A longer written description of the routine outlining inputs and out puts. to include where practical:

1. A short written description
2. Important equations and/or references to web pages or papers

Arguments:
Name Type In/Out/Both Description

Local variables:
Name Type Description

History: history of ALL changes made to code must include:

1. date of edit
2. name of editor

3. nature of the change

Bugs: A short description of anything known not to work with date and name of person responsible for this piece of work.

! Id. The first time this is checked into sub version with th e appropriate key word it will set the version number

Conventions for file naming

All text files will have the file extension ".txt" to distinguish them.

All binary files will have the file extension ".bin" to distinguish them

All fortran source code files will have the file extension ".f90" to distinguish them.

Conventions for error handling

Functions will indicate their success or failure by means of an int return value of STATUS_OK. Any function that can fail should return such a status code. Success will be indicated by a STATUS_OK value (which should be defined as 0), failure by anything else. If the function simply needs to indicate a failure, the value one should be returned. In more complex situations, a range of values may be appropriate.

Only functions that cannot fail (or more accurately, where the likelihood of failure is sufficiently small that it is not checked for) are allowed to return a value that is not a status value, e.g. basic mathematical functions can simply return the result value rather than a status code.

Conventions for Error Codes

Generic error codes are defined in table 1.

Code number	Name	Description

Conventions for test tools

ALL the code should have a DEBUG mode with extended output and NORMAL mode with minimal output. Any output (screen or file) is slow, computationally expensive and wasteful of resource (CPU and disk). Eg use orac_message.f90(to be written) program to print to screen if in debug mode. Specify via environment or compiler options?

Test inputs and outputs should be stored to allow for regression testing

Unit testing must be done. Clearly define test inputs/outputs

General Optimal Estimation coding guidelines

Developers should keep in mind the principals of the OE and community code and follow the guidelines below

1. Empirical algorithms and tests, e.g threshold tests should not be introduced unless absolutely necessary.
2. Code specific to a single instrument should not be introduced. Code should be introduced for generic application
3. Coding should be as modular as possible
4. Optional algorithms should be able to be switched on and off
5. blah blah

Procedure for demonstrating code improvements

The following steps are necessary to demonstrate before the community e.g at an ORAC meeting or by email/mailing list before the code can uploaded and distributed as a new version of ORAC. Users are encouraged to get a colleague to check code for readability and adherence to the rules before submission.

1. run code on one (or more??) test scenes listed below

2. Illustrate improvement with validation study.
3. Run 'test' idl code- (to be provided) which will produce histograms and images of test scenes which can be compared against previous versions.

Procedure for reporting bugs

It is inevitable that bugs will be identified and introduced in the ORAC community code. In this case the bug should be identified as soon as possible to

1. The Person who introduced the bug.

2. The code manager.

Barry suggests using a single defined method to repor t them too. Eg. the :BUG: :FIXME: and :TODO: notation within comment lines which can be picked up by many standard programming tools (including the Trac Wiki page I believe)
Trac can use these can then be assigned tickets to people to fix and then show when task completed. No doubt other similar tools though.

To be investigated….