Climate Data Record (CDR) Program

Transitioning CDRs from Research to Operations

Guidelines Document

DRAFT

October, 2010

Transitioning CDRs from Research to Operations

TABLE of CONTENTS

Section 1. CDR Research-to-Operations Guidelines

1.1. Introduction. The intent of these guidelines is to define the internal NCDC organizational structure, processes, roles and responsibilities necessary to transition research quality CDRs to an operational state. This process is commonly called the CDR Research-to-Operations (R2O) Transition System. Figure 1 is a conceptual diagram illustrating the end-to-end CDR System. These guidelines only pertain to the R2O subsection of Figure 1.

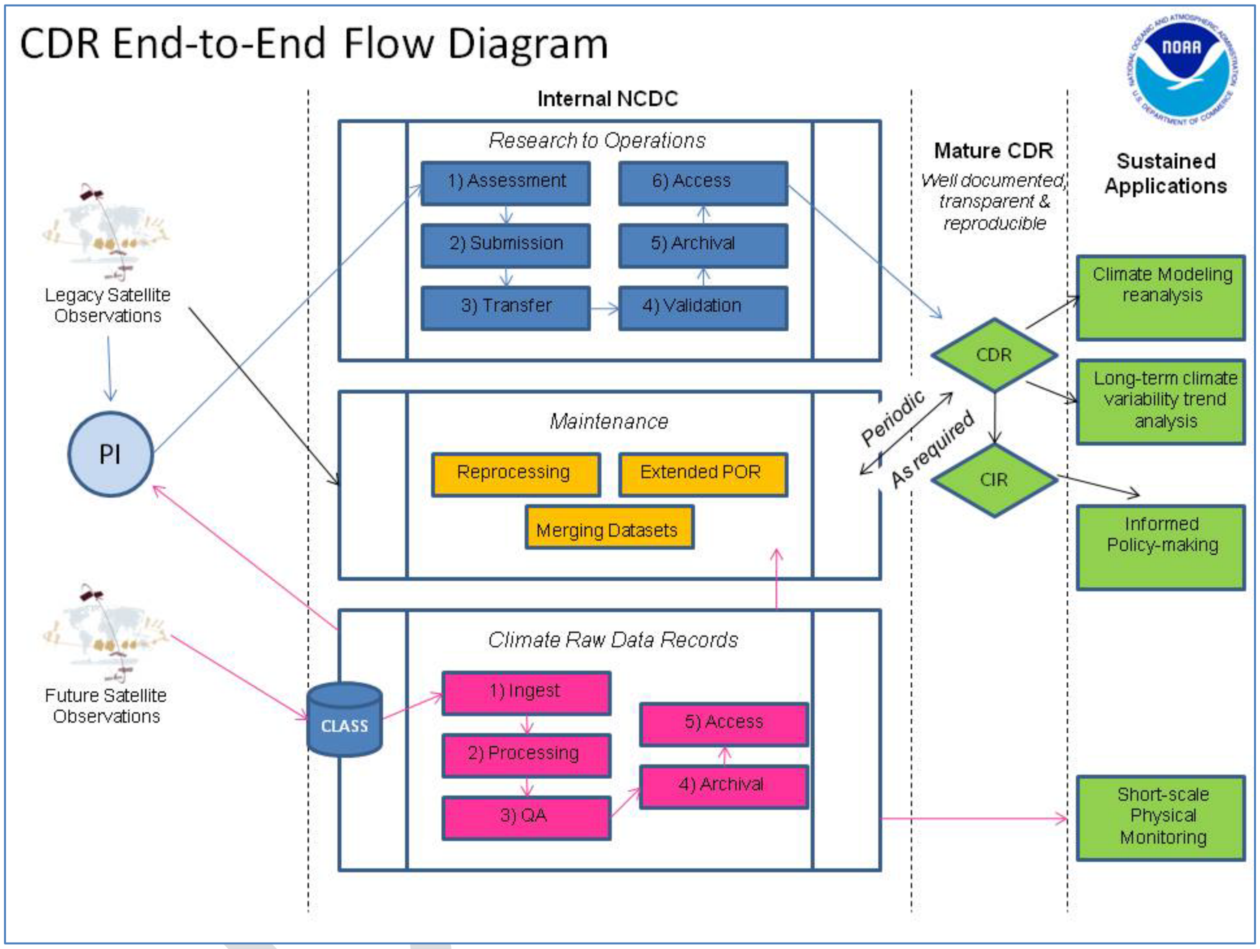


Figure 1. CDR End-to-End Flow Diagram

For the purpose of this document, 'external' is defined as CDR algorithm software, data and supporting documentation originating from Principle Investigators (PI) outside of NCDC as opposed to 'internal' which originates from within. Regardless of origin, these guidelines ensure all CDRs meet the GCOS standard of quality outlined in Appendix A-6 since NCDC is ultimately responsible for their validity. The maturity of an operational CDR can be described as either Initial Operating Capability (IOC) or Full Operating Capability (FOC). IOC processes describe the preliminary R2O requirements needed in order to make a CDR accessible by the public. Acknowledging there is no 'end state' to a climate record, FOC processes describe the stewardship that maintains continuous cyclical improvement to a CDR. These guidelines specifically deal only with IOC processes.

1.2. CDR R2O Overview. Figure 2 provides a flow diagram of the CDR R2O process internal to NCDC. There are essentially six phases toward preparing a CDR for IOC: assessment, submission, transfer, validation, archival and access. The entire process is a collaborative effort touching four separate branches spanning two divisions within NCDC as well as the PI. During the transition of four CDRs to operations in FY10, the average amount of time required was approximately 300 labor hours per CDR over a three month period. For coming years, successful transition of an increasing number of CDRs will require excellent coordination between all parties.

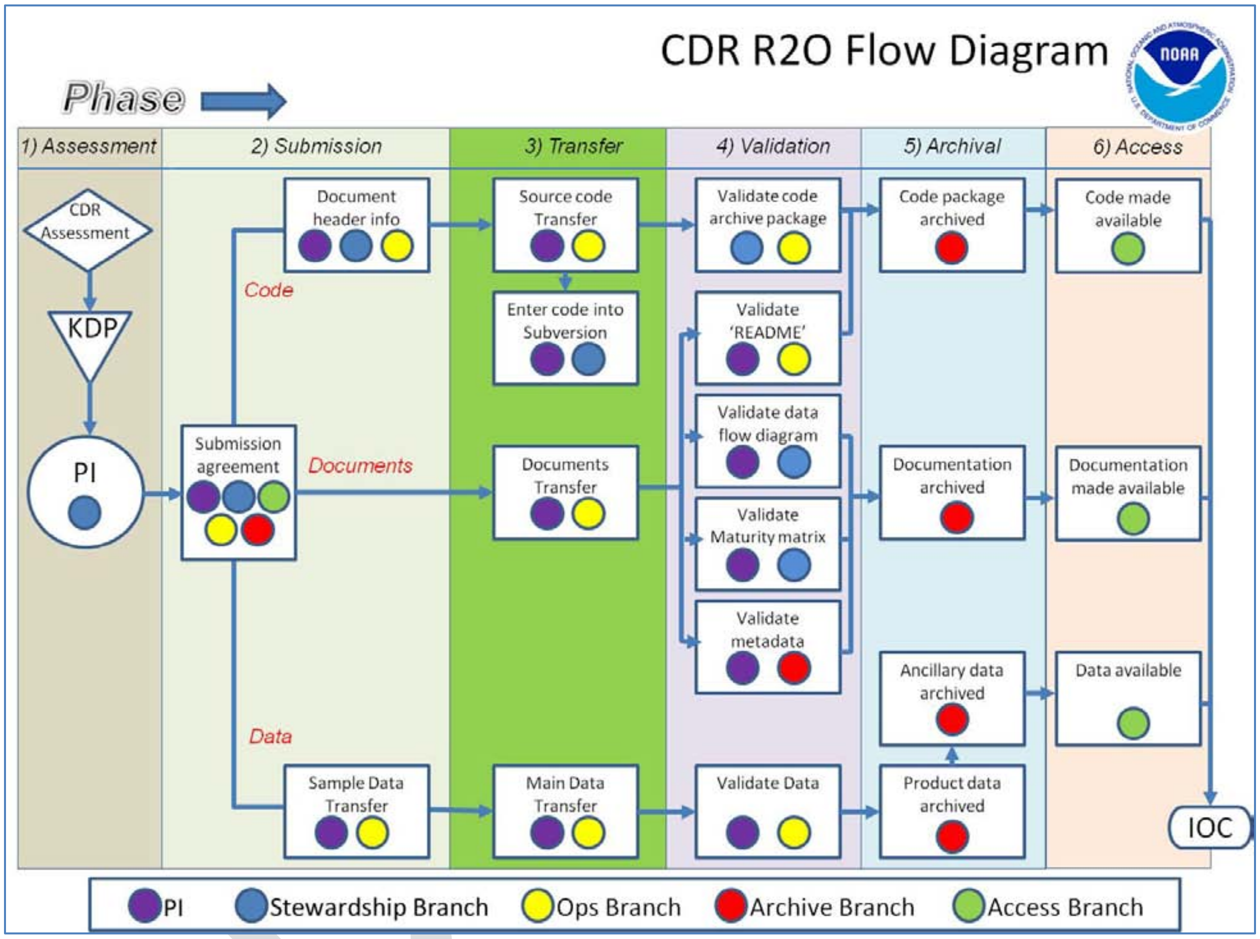


Figure 2. CDR R2O IOC Process

1.3. Definition of IOC. A CDR is considered IOC once it has met all the necessary requirements for public release, including being well-documented, transparent and reproducible. An IOC checklist is provided in Appendix A-5.

1.4. CDR R2O Roles and Responsibilities

1.4.1. CDR Primary Lead. The Stewardship Branch Chief will designate a Primary Lead for each CDR research to operations initiative. This lead is responsible for 'cradle to grave' of the CDR to include initial assessment, liaison with the PI and coordinating the intra-NCDC collaborative effort to IOC. The lead will produce monthly updates for the PM and senior leadership that provide project status, future milestones, project risks in addition to a project schedule in the form of a Gantt chart.

1.4.2. Other RSAD CDR Team Leads. In addition to the CDR Primary Lead, additional team members are appointed from each RSAD branch (Stewardship, Operations and Archive) in order to assist the PI with the CDR submission agreement, documenting the code and validating the CDR documents and data.

1.4.3. Climate Services Division. The Climate Services Division will identify a POC from the Data Access Branch to facilitate CDR delivery to the public.

1.5. CDR R2O IOC Phases.

1.5.1 CDR Assessment. Prior to committing significant internal resources to any main effort, a comprehensive CDR assessment must be conducted by the NCDC Primary Lead to determine whether the CDR is mature enough for transition. This assessment includes feasibility and complexity studies, estimating the volume of data and determining the quality of documentation. The Primary Lead will then recommend to the CDR Program Manager whether the CDR is ready for IOC transition. This is the Key Decision Point (KDP) shown in Figure 2.

1.6.2 Submission Agreement. The submission agreement documents the agreed upon plan for how the data transfer and support will actually happen and how it involves the entire Data Center. Appendix A-1 is an example of a CDR submission agreement. This agreement is a living document as long as the data transfer is actively ongoing. The Primary Lead as well as the additional team members will stay actively engaged with the PI during the entire submission process. With the guidance provided by NCDC, the PI should have code, documents and data ready in no later than eight weeks.

1.6.3. Code Header Document. Appendix A-2 is an example of a code header document. With this document, each unit of code has a description of its file name, location, purpose, description, author, creation date, copyright and other pertinent information. Additional information on code headers as well as other programming standards is detailed in Appendix A-7.

1.6.4. Entering Code into Subversion. With the assistance of the PI, the Stewardship Branch team lead will enter the CDR code into the revision control system known as Subversion. Subversion is a tool designed to manage changes to documents, code and other information stored as computer files. Use of Subversion satisfies Global Climate Observing System (GCOS) recommendation 7, *'Version management of FCDRs and products, particularly in connection with improved algorithms and reprocessing'* (Ref. Appendix A-6).

1.6.5. Validation. CDR code, documentation and data need to be validated in order to ensure preparation according to provided guidelines and that the PI is following the submission agreement. It is optimal for NCDC to receive test files of CDR code, documents or data before shipment of the main set since validating a smaller sample may be both quicker and easier. Estimated time for validation is approximately one week (except for FGDC which is a day). Code is validated by assuring compliance with standards and guidance provided in Appendix A-7, *CDR General Programming Standards*. Depending on complexity, any failed validation may be returned to the PI for fix.

1.6.6. CDR Data Transfer and Archival. The estimated time to transfer a CDR set to NCDC is volume-dependent on the order of 1 TB a day. After the CDR is validated, transfers are sent to the HDSS tape drives. On average, 1-2 weeks are needed for the archival of the code, documents and data; however, this is dependent on volume in addition to the number of files.

1.6.7. Access Branch Actions. The role of the Access Branch is to package and make available to the public the CDR code, documentation and data components.

1.7. IOC Determination. IOC is considered the first iteration of a repeatable public-released CDR. Once all the IOC requirements are met, the CDR is releasable to the public. This means the CDR is well documented, transparent and reproducible.

1.8. FOC Processes. As stated in the GCOS guidance, CDRs require periodic improvements to an ever sliding climate record. R2O CDRs have to be updated as new observational datasets are acquired or reprocessing is needed. These updates may only be required to partly follow the processes outlined in Figure 1. An example is new datasets that need to run on coded algorithms that remain unchanged. Regardless of the complexity of the CDR update, the key to maintaining transparency is a well documented audit trial. These FOC processes will be detailed in a subsequent CDR roadmap.

A-1 Submission Agreement Example

Submission Agreement
Between the NCDC Stewardship Branch
And the NCDC Archive Branch
For Upper Tropospheric Water Vapor (UTWV) Products
From HIRS Channel 12 Brightness Temperatures
Including the Fundamental Climate Data Record (FCDR)

September 13, 2010

Introduction

This document represents the agreement that the National Climatic Data Center (NCDC) Stewardship Branch (the "Provider") and the NCDC Archive Branch (the "Archive") have reached for submitting the Provider's data, the Upper Tropospheric Water Vapor (UTWV) Products from High-Resolution Infrared Radiation Sounder (HIRS) Channel 12 Brightness Temperatures, including the Fundamental Climate Data Record (FCDR), to the Archive for long-term preservation. It represents a joint effort between the Provider and the Archive to accurately document the agreement and the expectations between the two groups.

Data Description

Intersatellite calibration is carried out for the UTWV data from clear-sky HIRS channel 12 measurement. As the intersatellite biases are scene brightness temperature dependent, an algorithm is developed to account for the varying biases with respect to brightness temperature. The bias correction data are derived from overlaps of monthly means of each 10-degree latitude belt. For the colder temperature range, data from the simultaneous nadir overpass observations are incorporated. The HIRS measurements from the NOAA series of polar orbiting satellites are calibrated to a baseline satellite. The time series of the intersatellite calibrated HIRS UTWV data from 1979 to present is constructed and anomaly data are computed.

- *Final edits to description for Archive Metadata are TBD*

Items to Be Submitted

1. HIRS UTWV for gridded Monthly Means, Daily Means, and Swath data in netCDF by version/release ("v01r01" will be the initial archived version)
2. HIRS UTWV input/ancillary data files by version/release (intermediate HIRS L1B channel 12 brightness temperatures and other data are TBD)
3. HIRS UTWV source code by version/release
4. HIRS UTWV documentation by version/release

Contacts

Provider Contacts:

Ed Kearns
RSAD Deputy Chief
NCDC / RSAD
+1 828-271-4328

ed.kearns@noaa.gov

Lei Shi
Physical Scientist
NCDC / RSAD / SB
+1 828-350-2005
lei.shi@noaa.gov

Ethan Shepherd
IT Specialist
NCDC / RSAD / SB
+1 828-257-3017
ethan.shepherd@noaa.gov

Archive Contacts:

Nancy Ritchey
Archive Branch Chief
NCDC / RSAD / AB
+1 828-271-4445
nancy.ritchey@noaa.gov

David Bowman
IT Specialist
NCDC / RSAD / AB
+1 828-271-4368
david.p.bowman@noaa.gov

Tammy Scott
IT Specialist
NCDC / RSAD / AB
+1 828-271-4175
tammy.scott@noaa.gov

Transfer Interface

The Provider will FTP push files to a directory on an NCDC server as specified by the Archive. The Provider will be responsible for verifying the success of an FTP session.

File Naming and Size

NetCDF Data:
Data files have the following naming convention:
HIRS-CH12_<*TYPE*>_v<*NN*>r<*NN*>_<*SATID*>_<*YYYY*><*DDD*>.nc

Where:
_ = file field delimiter
HIRS-CH12 = static field that identifies the HIRS channel 12 brightness temperature (UTWV) product series
<*TYPE*> = five to six characters identifying the data type, with the valid domain:
"MONGRD" for Monthly Means on a 2.5x2.5 degree grid
"DAYGRD" for Daily Means on a 2.5x2.5 degree grid
"SWATH" for aggregated granules at original HIRS L1B swath resolution (20km pixel)
v<*NN*>r<*NN*> = two-digit version number and two-digit release number corresponding with the data production

<SATID> = three character identifier for the POES or MetOp satellite that carried the HIRS instrument (only present for the Swath data type), with the valid domain:
  “T-N” for TIROS-N
  “N06” for POES-6
  “N07” for POES-7
  “N08” for POES-8
  “N09” for POES-9
  “N10” for POES-10
  “N11” for POES-11
  “N12” for POES-12
  “N14” for POES-14
  “N15” for POES-15
  “N16” for POES-16
  “N17” for POES-17
  “M02” for MetOp-A
<YYYY> = four-digit year, from “1978” to “2009”
<DDD> = three-digit day of year (“001” - “366”) (field is only present for Swath data type)
nc = file extension for netCDF

Example netCDF file names:
  HIRS-CH12_MONGRD_v01r01_2002.nc
  HIRS-CH12_DAYGRD_ v01r01_2002.nc
  HIRS-CH12_SWATH_ v01r01_N17_2002365.nc

Data are stored using netCDF-3.6.1. The gridded Monthly Means are yearly files with an average size of 1MB. The gridded Daily Means are also yearly files with an average size of TBD MB. There are approximately 25,000 daily Swath files for the 30+ year record with an average size of 2MB.

Data Tar Files:
The Swath data files will be archived in compressed yearly tape archive (tar) files by satellite with the following tar file naming convention:
HIRS-CH12_ SWATH_v<*NN*>r<*NN*>_<*SATID*>_<*YYYY*>.tar.gz

Where:
_ = file field delimiter
HIRS-CH12_SWATH = static field that identifies the HIRS channel 12 brightness temperature (UTWV) Swath product
v<*NN*>r<*NN*> = two-digit version number and two-digit release number corresponding with the data production
<SATID> = three character identifier for the POES or MetOp satellite that carried the HIRS instrument (only present for the Swath data type)
<YYYY> = four-digit year, from “1978” to “2009”
tar = file extension for tar archive file
gz = GNU zip compression extension

Example tar file name:
  HIRS-CH12_SWATH_ v01r01_N17_2002.tar.gz

There will be no more than 366 Swath netCDF files in a yearly Swath tar file per satellite. Monthly Means and Daily Means will be archived as individual netCDF files and will not be archived as tar files.

Source Code:
Source code tar files will have the following naming convention:
hirs_ch12_source_code_v<*NN*>r<*NN*>.tar.gz

_ = file field delimiter
hirs_ch12 = static field that identifies the HIRS channel 12 brightness temperature (UTWV) product
source_code = static field that identifies source code for HIRS UTWV
v<*NN*>r<*NN*> = two-digit version number and two-digit release number corresponding with the data production of the same version/release
tar = file extension for tar archive file
gz = GNU zip compression extension

Example source code tar file name:
hirs_ch12_source_code_v01r01.tar.gz

Input/Ancillary Data:
Input/ancillary data files will have the following naming convention:
Archiving of intermediate HIRS products is TBD

Documentation:
Documents will have the following naming convention:
HIRS_UTWV_<*DOC_TYPE*>_ v<*NN*>r<*NN*>.<*ext*>

Where:
HIRS_UTWV = static field that identifies the HIRS UTWV product
<*DOC_TYPE*> = documentation type, with the domain:
"OAD" for Operational Algorithm Description
"Flowchart" for data flow processing diagram
"Code_Headers" for code headers (generated by ROBODoc software)
"Maturity_Matrix" for product's maturity ratings
v<*NN*>r<*NN*> = two-digit version number and two-digit release number corresponding with the data production of the same version/release
<*ext*> = appropriate file extension for the document, e.g., "doc" or "pdf"

All archived documentation will be stored in a single tar file with the following naming convention:
HIRS_UTWV_DOC_v<*NN*>r<*NN*>.tar

Where:
HIRS_UTWV_DOC = static field that identifies documentation for the HIRS UTWV product
v<*NN*>r<*NN*> = two-digit version number and two-digit release number corresponding with the data production of the same version/release
tar = file extension for tar archive file

Example documentation tar file name:
HIRS_UTWV_DOC_v01r01.tar

Manifests:
A manifest file with file information and MD5 checksum value for each archived file is required for ingest of the product, input/ancillary data, and source code files.

The format of the information in a manifest file is:
file_name,file_size,MD5_checksum (three values comma delimited per row with no spaces)

A manifest can contain any number of file checksum values (i.e., one manifest for each file or one manifest for the entire record).

Manifests will have the following naming convention:

manifest_<*yyyymmdd*>_<*HHMMSS*>.<*ext*>

Where:
_ = file field delimiter
manifest = static field that identifies manifest file
<*yyyymmdd*> = manifest file creation date stamp for unique manifest file name
<*HHMMSS*> = manifest file creation time stamp for unique manifest file name
<*ext*> = appropriate file extension for manifest file, e.g., "txt" or "xml"

Example manifest file name:
manifest_20100810_101500.txt

A manifest file will be provided for each Monthly Mean and Daily Mean netCDF file as well as each Swath tar file. A manifest is also expected for any input/ancillary data, source code and documentation tar files.

Submission Schedule
Data submission for the initial version (i.e., v01r01) of the product and supporting information will begin no later than September 16, 2010 and will finish no later than September 24, 2010. Source data and code for the corresponding initial version will also be submitted during September 2010.

Additional product versions are expected to be delivered as requested by the Provider and these will be archived in a similar fashion as the initial submission. A routine schedule may be established once the product is fully operational.

Validation
The Archive will use file name, size and MD5 checksum to verify the integrity of a delivered file.

Error Reconciliation
The Archive will report any unexpected file size or a duplicate file for a version to the Provider. The same procedure will be true for any file integrity or checksum error. A new corresponding manifest file will be required for re-submitted files from the Provider.

Confirmation
The Archive will provide confirmation of successful data ingest for a version or as requested by the Provider.

Quality Assurance
No Quality Assurance will be performed by the Archive.

Archive Storage
All files will be stored on the NCDC HDSS under DSI:
3629_<*XX*>
Where, <*XX*> is the two-digit DSI subset number.

The subset number will increment by one (e.g., "01", "02" … "99") with each new product release, however, a subset number will not necessarily correspond to the product version or release number stored under that subset.

Directory structure for the data will be by year and contain the Monthly Means and Daily Means netCDF files and the compressed yearly Swath tar files (all three data types):
/aab/36xx/3629_<*XX*>/CDR/<*YYYY*>/HIRS-CH12_MONGRD_v<*NN*>r<*NN*>_<*YYYY*>.nc
/aab/36xx/3629_<*XX*>/CDR/<*YYYY*>/HIRS-CH12_DAYGRD_v<*NN*>r<*NN*>_<*YYYY*>.nc
/aab/36xx/3629_<*XX*>/CDR/<*YYYY*>/HIRS-CH12_SWATH_ v<*NN*>r<*NN*>_T/N/M<??>_<*YYYY*>.tar.gz

Directory structure for ancillary or input data, source code, and documentation, respectively:
/aab/36xx/3629_<*XX*>/ANC/TBD
/aab/36xx/3629_<*XX*>/SRC/hirs_ch12_source_code_v<*NN*>r<*NN*>.tar.gz
/aab/36xx/3629_<*XX*>/DOC/HIRS_UTWV_DOC_v<*NN*>r<*NN*>.tar

Note that ancillary data archived under existing DSI directories will only be referenced.

Constraints

No constraints on data access, use or other.

Access and User Services

The Archive will write metadata that follows the FGDC CSDGM content standard with Extensions for Remote Sensing using information from the Provider. This metadata file, with the ID TBD, will be stored and published in the NOAA Metadata Manager Repository (NMMR) and from there harvested to metadata portals and clearinghouses for public users (URL: http://www.ngdc.noaa.gov/metadata/published/NCDC/TBD). User community access will be through THREDDS for data on 'eclipse', and access to data on tape will be through the HDSS Access System (HAS) - TBD. Provision of archived data on media will be provided through NCDC Customer Services.

Additional Terms

None - TBD

A-2 Code Header Document Example

TABLE OF CONTENTS

1. PROJECT/ERSST [PROJECTS]

[Top] [PROJECTS]

NAME

```
ersstv3b_oper_situ_only_col.sh
```

LOCATION

```
$ERSST/script/ersstv3b_oper_situ_only_col.sh
```

PURPOSE

```
To generate the analyzed Extended Reconstruction Sea Surface Temperature
(ERSST) on a 2-deg grid from in situ data (ship and buoy, NOT satellite: v3b
uses in situ only),and transfer to distribution directories.
```

DESCRIPTION

```
This is the main script that launches a series of fortran programs for
computing for a specific month and year (determined from the current machine
date). The operational runs will affect values in recent past years due the
long-term averaging

Also, the program uses output from a one-time climatological run (1880
to around 1985). Most programs write output for all the years sequentially in
one binary file. However, depending on the program, the month processed may
just be added to the the pre-exisitng file, or the entire file may be
rewritten from 1985 onward.

The processing is as follows:
First, in situ data (ship and buoy) is ftp'd from source locations.
Adjustments are made for distance of point obs from grid center, difference
in dependability of ship and buoy data, etc. and other quality checks are
made. The data is placed on a 2-deg grid, and anomalies are computed.
Adjustments are also made for sea ice presence. The sea ice data comes from
the daily OISST analysis. Statistical analysis is done in 2 steps:
1)The decadal or low frequency component is determined from the anomalies and
then the residuals are computed
2)The high frequency analysis is performed on the residuals.
```

The **ERSST** is then computed from the sum of the two components and error variance is estimated. After the **ERSST** computation, other programs are run to update related products(land and merged land-ocean SST) that use **ERSST.** These other products are continually updated on a different schedule, external to this script. Areal averages are computed for the Climate Monitoring group, and plots are made to check the **ERSST** output. Comparisons with v2 and the v3 (satellite) are also made, that are produced separately.

AUTHOR
Chunying Liu

CREATION DATE
04/01/2008

COPYRIGHT
THIS SOFTWARE AND ITS DOCUMENTATION ARE CONSIDERED TO BE IN THE PUBLIC DOMAIN AND THUS ARE AVAILABLE FOR UNRESTRICTED PUBLIC USE. THEY ARE FURNISHED "AS IS." THE AUTHORS, THE UNITED STATES GOVERNMENT, ITS INSTRUMENTALITIES, OFFICERS, EMPLOYEES, AND AGENTS MAKE NO WARRANTY, EXPRESS OR IMPLIED, AS TO THE USEFULNESS OF THE SOFTWARE AND DOCUMENTATION FOR ANY PURPOSE. THEY ASSUME NO RESPONSIBILITY (1) FOR THE USE OF THE SOFTWARE AND DOCUMENTATION; OR (2) TO PROVIDE TECHNICAL SUPPORT TO USERS.

MODIFICATION HISTORY
04/30/2008 - Chunying modified from /raid2/**ERSST**/ftn/ersstv3b_oper_situ_only_col.sh to remove satellite data
08/28/2009 - Viva Banzon added comments
08/17/2010 - Viva Banzon removed Land and merged comments after Liu removed related codes The conditions for writing ascii for previous decade were put in but left commented since that was not in the original script

INPUTS
buoyship_quarter/nqyyyymm.Z - ftp ship and buoy data
Outputs from the Fortran programs are passed to other Fortran programs with each program building/modifying the data for the next program

OUTPUTS
Updated **ERSST** integer and data files **ERSST**/datat/**ERSST**-v3b/situ/**ERSST**.v3b.yr1.yr2.asc **ERSST**/datat/**ERSST**-v3b/situ/**ERSST**.esd.v3b.yr1.yr2.asc
NetCDF formated **ERSST** file **ERSST**/data/netcdf-v3b/situ/**ERSST**.yyyymm.nc

PARAMETERS
smult = 4 - standard deviation multiplier (range 2-6) for QC of in situ data

VARIABLES
$chyr = two digit year
$chmon = two digit month

SUBPROGRAMS

```
monice1d-med-oper.f
gtsqc.situ.v3b.f
ice1t2.f
ssta.merg.situ.v3b.f
lfsst.situ.v3b.f
hfsst.situ.v3b.f
sst2d.situ.v3b.f
err.norm.map2.upd.situ.v3b.f
dati2.upd.situ.v3b.f
ersst_netcdf.situ.v3b.f
```

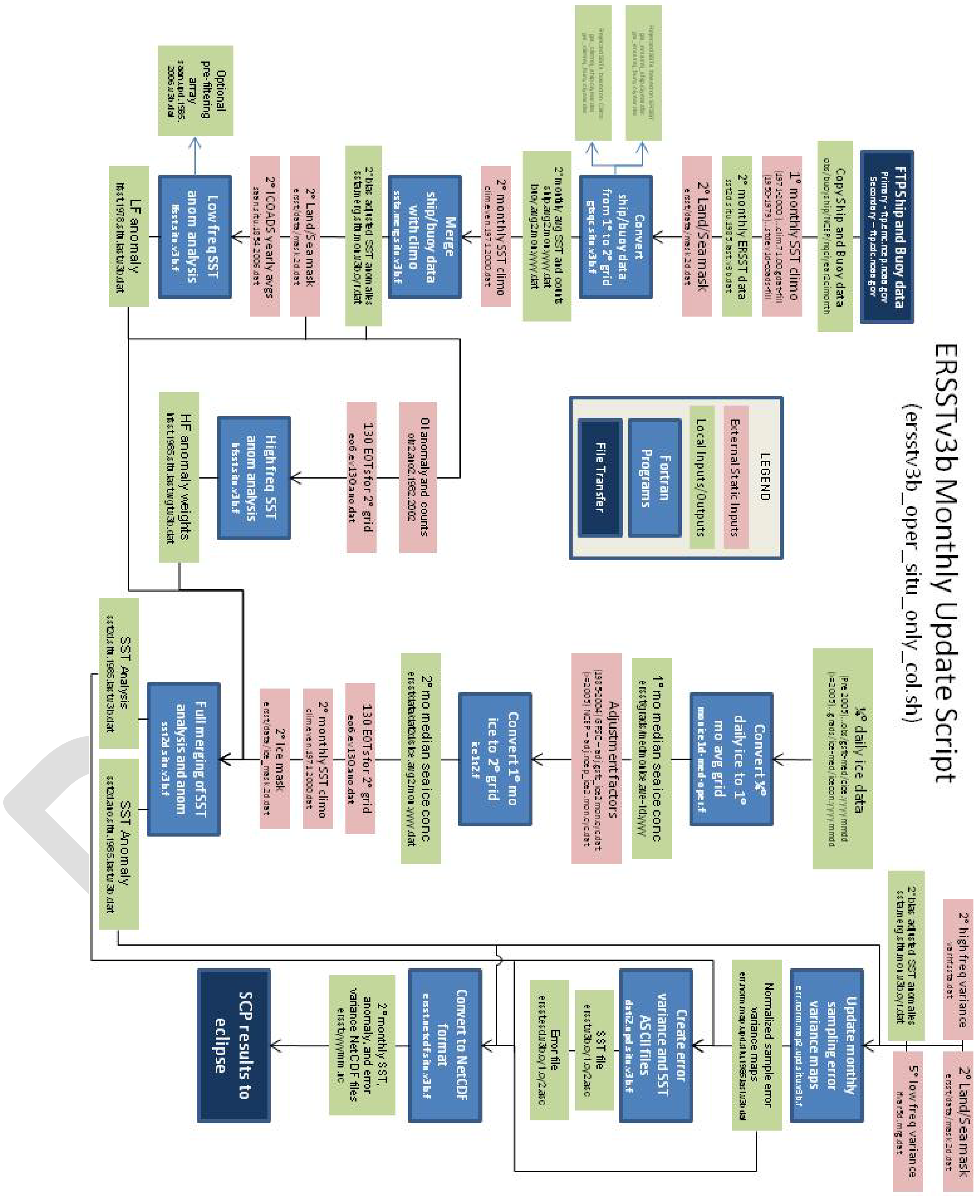
LIBRARY

```
netcdf bin library at: /usr/local/netcdf-3.6.1/bin
netcdf modules at:/usr/local/netcdf-3.6.1/include
date function at: /lib/w3lib
```

LANGUAGE

```
Linux Bourne shell script
```

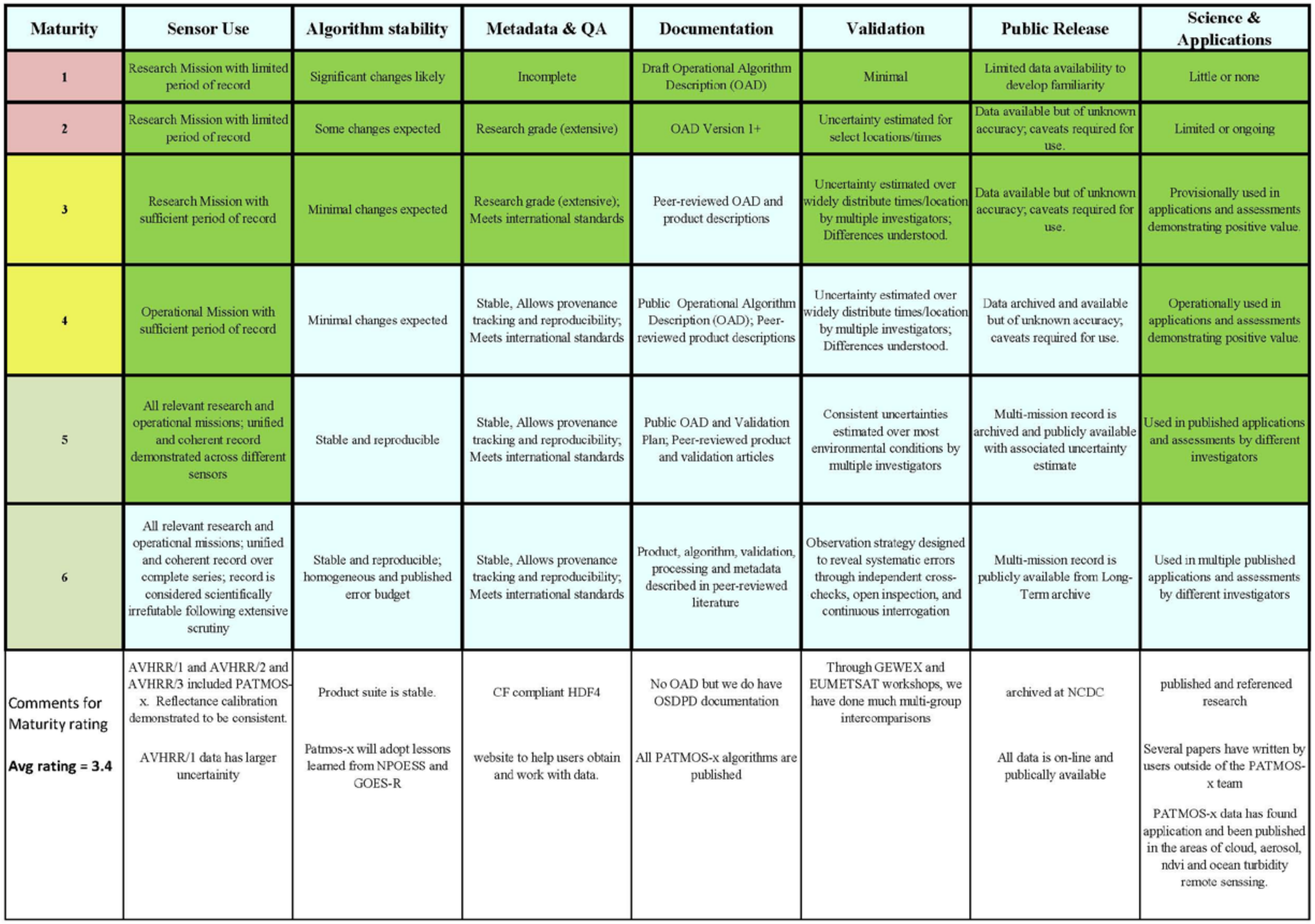
A-3 CDR Data Flow Diagram


ERSSTv3b Monthly Update Script
(ersstv3b_oper_situ_only_col.sh)
LEGEND
External Static Inputs
Local Inputs/Outputs
Fortran Programs
File Transfer
FTP Ship and Buoy data
Primary - ftp.emc.ncep.noaa.gov
Secondary – ftp.cdc.noaa.gov
Copy Ship and Buoy data
obs/buoyship/NCEP/nqcyear2cimonth
1° monthly SST climo
(1971-2000)...clim.71.00.gdat-fill
(1950-1979)...stdev1d-coads-fill
2° monthly ERSST data
sst2d.situ.1985.last.v3b.dat
2° Land/Seamask
ersst/data/mask2d.dat
Convert ship/buoy data from 1° to 2° grid
gtsqc.situ.v3b.f
2° monthly avg SST and counts
ship.avg2mon.yyyy.dat
buoy.avg2mon.yyyy.dat
2° monthly SST climo
clim.even.1971.2000.dat
Merge ship/buoy data with climo
ssta.merg.situ.v3b.f
2° bias adjusted SST anomalies
sstamerg.situ.mon.v3b.cyr.dat
2° Land/Seamask
ersst/data/mask2d.dat
2° ICOADS yearly avgs
sseansitu.1854.2006.dat
Low freq SST anom analysis
lfsst.situ.v3b.f
Optional pre-filtering array
saan.upd.1985.2006.v3b.dat
LF anomaly
lfsst.1978.situ.last.v3b.dat
OI anomaly and counts
oiv2.ano2.1982.2002
130 EOTs for 2° grid
eo6.ev130.ano.dat
High freq SST anom analysis
hfsst.situ.v3b.f
HF anomaly weights
hfsst.1985.situ.last.wgt.v3b.dat
¼° daily ice data
(Pre 2005)...obs/gsfc-med/cice.yyyymmdd
(>=2005)...grads/ice-med/icecon.yyyymmdd
Convert ¼° daily ice to 1° mo avg grid
mon.ice1d-med-oper.f
1° mo median sea ice conc
ersstgrads/medmon.ice.ave-1d.yyyy
Adjustment factors
(1985-2004) GFSC – adj.gsfc_ice2mon.cyc.dat
(>=2005) NCEP – adj.ncep_ice2.mon.cyc.dat
Convert 1° mo ice to 2° grid
ice1t2.f
2° mo median sea ice conc
ersstdata/dat2d.ice.avg2mon.yyyy.dat
130 EOTs for 2° grid
eo6.ev130.ano.dat
2° monthly SST climo
clim.even.1971.2000.dat
2° Ice mask
ersst/data/ice_mask2d.dat
Full merging of SST analysis and anom
sst2d.situ.v3b.f
SST Analysis
sst2d.situ.1985.last.v3b.dat
SST Anomaly
sst2d.ano.situ.1985.last.v3b.dat
2° high freq variance
varhfssta.dat
2° Land/Seamask
ersst/data/mask2d.dat
2° bias adjusted SST anomalies
sstamerg.situ.mon.v3b.cyr.dat
5° low freq variance
lfvar5d.mrg.dat
Update monthly sampling error variance maps
err.norm.map2.upd.situ.v3b.f
Normalized sample error variance maps
err.norm.map.upd.situ.1985.last.v3b.dat
Create error variance and SST ASCII files
dat2.upd.situ.v3b.f
SST file
ersst.v3b.cyr1.cyr2.asc
Error file
ersst.esd.v3b.cyr1.cyr2.asc
Convert to NetCDF format
ersst.netcdf.situ.v3b.f
2° monthly SST, anomaly, and error variance NetCDF files
ersst.yyyymm.nc
SCP results to eclipse

A-4.1. CDR Maturity Matrix (as of Oct 2010)

Maturity	Sensor Use	Algorithm Stability	Metadata &QA	Documentation	Validation	Public Release	Science & Applications
1	Research Mission with limited period of record	Significant changes likely	Incomplete	Draft operational Algorithm Description (OAD)	Minimal	Limited data availability to develop familiarity	Little or none
2	Research Mission with limited period of record	Some changes expected	Research grade (extensive)	OAD Version 1+	Uncertainty estimated for select locations and times	Data available but of unknown accuracy, caveats required for use	Limited or ongoing
3	Research Mission with sufficient period of record	Minimal changes expected	Research grade (extensive); Meets international standards	Peer-reviewed OAD and product descriptions	Uncertainty estimated over widely distributed times and locations by multiple investigators; differences understood	Data available but of unknown accuracy, caveats required for use	Provisionally used in applications and assessments demonstrating positive value
4	Operational Mission with sufficient period of record	Minimal changes expected	Stable; Allows provenance tracking and reproducibility and meets international standards	Public Operational Algorithm Description (OAD); peer-reviewed product descriptions	Uncertainty estimated over widely distributed times and locations by multiple investigators; differences understood	Data archived and available but of unknown accuracy; caveats required for use	Operationally used in applications and assessments demonstrating positive value
5	All relevant research and operational missions; unified and coherent record demonstrated across different sensors	Stable and reproducible	Stable; Allows provenance tracking and reproducibility and meets international standards	Public OAD and Validation Plan; Peer-reviewed product and validation articles	Consistent uncertainties estimated over most environmental conditions by multiple investigators	Multi-mission record is archived and publically available with associated uncertainty estimate	Used in published applications and assessments by different investigators
6	All relevant research and operational missions; unified and coherent record over complete series; record is considered scientifically irrefutable following extensive scrutiny	Stable and reproducible; homogeneous and published error budget	Stable; Allows provenance tracking and reproducibility and meets international standards	Product, algorithm, validation, processing and metadata described in peer-reviewed literature	Observation strategy designed to reveal systematic errors through independent cross-checks, open inspection and continuous interrogation	Multi-mission record is publically available from long-term archive	Used in multiple published applications and assessments by different investigators

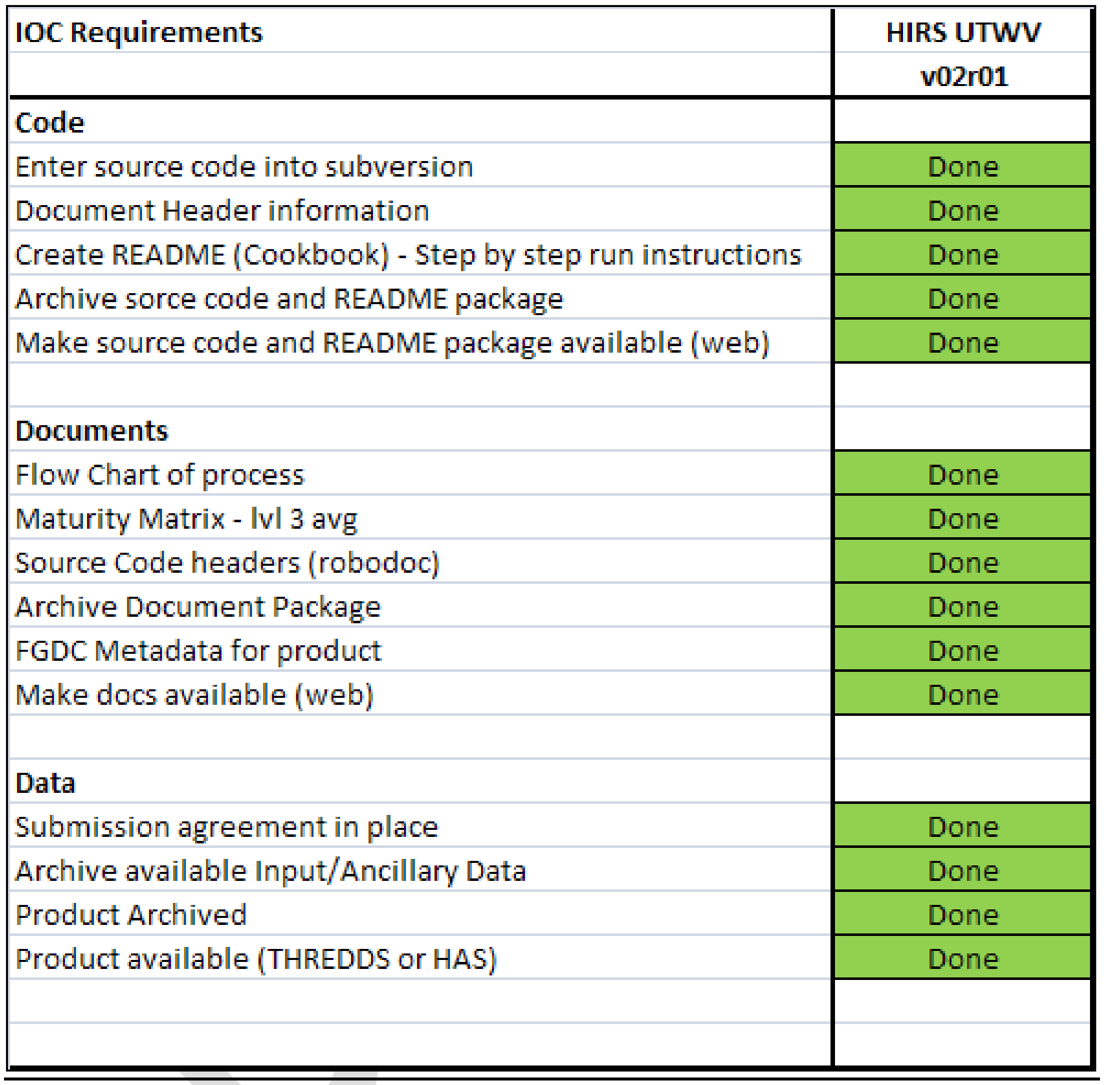
A-4.2. CDR Maturity Matrix Example for PATMOS-X, FY10



Maturity	Sensor Use	Algorithm stability	Metadata & QA	Documentation	Validation	Public Release	Science & Applications
1	Research Mission with limited period of record	Significant changes likely	Incomplete	Draft Operational Algorithm Description (OAD)	Minimal	Limited data availability to develop familiarity	Little or none
2	Research Mission with limited period of record	Some changes expected	Research grade (extensive)	OAD Version 1+	Uncertainty estimated for select locations/times	Data available but of unknown accuracy; caveats required for use.	Limited or ongoing
3	Research Mission with sufficient period of record	Minimal changes expected	Research grade (extensive); Meets international standards	Peer-reviewed OAD and product descriptions	Uncertainty estimated over widely distribute times/location by multiple investigators; Differences understood.	Data available but of unknown accuracy; caveats required for use.	Provisionally used in applications and assessments demonstrating positive value.
4	Operational Mission with sufficient period of record	Minimal changes expected	Stable, Allows provenance tracking and reproducibility; Meets international standards	Public Operational Algorithm Description (OAD); Peer-reviewed product descriptions	Uncertainty estimated over widely distribute times/location by multiple investigators; Differences understood.	Data archived and available but of unknown accuracy; caveats required for use.	Operationally used in applications and assessments demonstrating positive value.
5	All relevant research and operational missions; unified and coherent record demonstrated across different sensors	Stable and reproducible	Stable, Allows provenance tracking and reproducibility; Meets international standards	Public OAD and Validation Plan; Peer-reviewed product and validation articles	Consistent uncertainties estimated over most environmental conditions by multiple investigators	Multi-mission record is archived and publicly available with associated uncertainty estimate	Used in published applications and assessments by different investigators
6	All relevant research and operational missions; unified and coherent record over complete series; record is considered scientifically irrefutable following extensive scrutiny	Stable and reproducible; homogeneous and published error budget	Stable, Allows provenance tracking and reproducibility; Meets international standards	Product, algorithm, validation, processing and metadata described in peer-reviewed literature	Observation strategy designed to reveal systematic errors through independent cross-checks, open inspection, and continuous interrogation	Multi-mission record is publicly available from Long-Term archive	Used in multiple published applications and assessments by different investigators
Comments for Maturity rating **Avg rating = 3.4**	AVHRR/1 and AVHRR/2 and AVHRR/3 included PATMOS-x. Reflectance calibration demonstrated to be consistent. AVHRR/1 data has larger uncertainity	Product suite is stable. Patmos-x will adopt lessons learned from NPOESS and GOES-R	CF compliant HDF4 website to help users obtain and work with data.	No OAD but we do have OSDPD documentation All PATMOS-x algorithms are published	Through GEWEX and EUMETSAT workshops, we have done much multi-group intercomparisons	archived at NCDC All data is on-line and publically available	published and referenced research Several papers have written by users outside of the PATMOS-x team PATMOS-x data has found application and been published in the areas of cloud, aerosol, ndvi and ocean turbidity remote sensing.

A-5 Internal NCDC Checklist for CDR

(as of Oct 2010)



IOC Requirements	HIRS UTWV
	v02r01
Code	
Enter source code into subversion	Done
Document Header information	Done
Create README (Cookbook) - Step by step run instructions	Done
Archive sorce code and README package	Done
Make source code and README package available (web)	Done
Documents	
Flow Chart of process	Done
Maturity Matrix - lvl 3 avg	Done
Source Code headers (robodoc)	Done
Archive Document Package	Done
FGDC Metadata for product	Done
Make docs available (web)	Done
Data	
Submission agreement in place	Done
Archive available Input/Ancillary Data	Done
Product Archived	Done
Product available (THREDDS or HAS)	Done

A-6 GCOS 12 Requirements for CDR

Table 1. GCOS 12 Requirements for CDR[1]

1. Full description of all steps taken in generation of FCDRs and ECV products, including algorithms used, specific FCDRs used and characteristics and outcomes of validation activities.
2. Application of appropriate calibration/validation activities.
3. Statement of expected accuracy, stability and resolution (time, space) of the product including a comparison with the GCOS requirements.
4. Assessment of long-term stability and homogeneity of the product.
5. Information on the scientific review process related to FCDR/product construction (including algorithm selection), FCDR/product quality and applications.
6. Global coverage of FCDRs and products where possible.
7. Version management of FCDRs and products where possible.
8. Arrangements for access to the FCDRs, products and all documentation.
9. Timeliness of data release to the user community to enable monitoring activities.
10. Facility for user feedback.
11. Application of a quantitative maturity matrix.
12. Publication of a summary (webpage or peer-reviewed article) documenting point-by-point the extent to which this guideline has been followed.

[1] Guidelines for the Generation of Datasets and Products Meeting GCOS Requirements, GCOS-143 (WMO/TD No. 1530), May 2010

A-7 CDR General Programming Standards

1. INTRODUCTION

The Climate Data Record (CDR) Program receives a wide variety of source codes that need to be deployed in a full-time operational setting. Source codes are written in various programming languages and idiosyncratic styles and lack coordinating documentation in most cases. The resulting software is often costly to maintain, as the source code may have been mislaid, the code may be difficult to read and understand, documentation may be inadequate, or the original developers may no longer be available to help maintain their code.

The purpose of developing common software programming standards is to streamline the research to operations transition process. Implementation of these coding standards will shift costs away from operations and maintenance as the problems are resolved upstream. Promoting the accountability of the developers and scientists to create standardized software programs will benefit the CDR program as a whole.

Having common programming standards used by all will aid in cross-organization communication and implementation of codes. It will also produce a software catalog that:

- Is robust
- Is readily portable (platform independent)
- Is modular and reusable
- Is inexpensive to implement and maintain operationally
- Is written in a widely used and supported language
- Has a common look and structure
- Adheres to best programming practices
- Is well documented
- Is easily readable and understandable

1.1 Programming Standards and Guideline Definitions

It is recognized that certain stylistic suggestions which make code easier to read (e.g. lining up attributes, or using all lower case or mixed case) are subjective and therefore should not have the same weight as techniques and practices that are known to improve code quality. For this reason, these standards are divided into three components; Standards, Guidelines and Recommendations (SGRs):

Standard: Aimed at ensuring portability, readability and robustness. Compliance with this category is mandatory.

Guideline: Good practices. Compliance with this category is strongly encouraged. The case for deviations will need to be argued by the programmer.

Recommendation: Compliance with this category is optional, but is encouraged for consistency.

These three standards will thus be found in the above format throughout this document, indicating the weight of a particular standard. If possible, all standards, guidelines and recommendations should be followed when programming. Else, programmers should include these components whenever possible, keeping in mind their respective weight. Please refer to these definitions as needed.

1.2 Reference Documents

This document was derived from SPSRB's ***General Programming Principles and Guidelines***, Version 2.0; September 2010. It is found at http://projects.osd.noaa.gov/spsrb/standards_software_coding.htm

2. Formatting Basics

Eric S. Raymond, in his book *The Art of Unix Programming*, summarizes the Unix philosophy as the widely-used engineering philosophy, ***"Keep it Simple, Stupid"*** (KISS Principle). He then describes how he believes this overall philosophy is applied as a cultural Unix norm:

a. *Rule of Simplicity*: Design for simplicity; add complexity only where you must.
b. *Rule of Modularity*: Write simple parts connected by clean interfaces.
c. *Rule of Clarity:* Clarity is better than cleverness.
d. *Rule of Composition*: Design programs to be connected to other programs.
e. *Rule of Transparency*: Design for visibility to make inspection and debugging easier
f. *Rule of Parsimony*: Write a big program only when it is clear by demonstration that nothing else will do.
g. *Rule of Repair*: When you must fail, fail noisily and as soon as possible.
h. *Rule of Economy*: Programmer time is expensive; conserve it in preference to machine time.
i. *Rule of Generation*: Avoid hand-hacking; write programs to write programs when you can.
j. *Rule of Optimization*: Prototype before polishing; get it working before you optimize it.
k. *Rule of Extensibility*: Design for the future, because it will be here sooner than you think.

The programming principles described here and in the language-specific coding documents adhere to the "KISS" principles above.

2.1 Program Unit Organization

Standard: *Program units shall start with a series of descriptive header comments (2.2), followed by well organized source code (2.3-2.6).*

2.2 Headers

These headers must contain the items listed as a standard and should contain the other items for good measure. Please provide the fields in the order which they are presented to maintain consistency.

Standard: *Every new program unit shall contain a header comment section.*

Designate information required in the header with the following keywords:

a. *NAME*: The name of the program unit.

b. *LOCATION*: The relative path where the program is located.

c. *PURPOSE*: A brief description of the program unit function (e.g., 1-2 sentences).

d. *DESCRIPTION:* A description of the program unit processing (e.g., diagrams, PDL).

e. *AUTHOR:* Who created the program unit.

f. *CREATION DATE:* When the program unit was created.

g. *COPYRIGHT:* Standard statement allowing unrestricted public use (see Example 1).

h. *EXTERNALS:* Modules and subroutines needed from an external source.

i. *SUBROUTINES:* Any subroutines contained internally in the program text.

j. *INPUTS*: A description of the program unit inputs (e.g., parameters, files).

k. *OUTPUTS*: A description of the program unit outputs (e.g., parameters, files).

l. *HISTORY*: The revision history of the program unit

Guideline: *Additional header items listed below should be included as necessary.*
Designate information with the following keywords:

a. *REFERENCE*: The reference(s) to program unit design materials (e.g., requirements document, design document, standards, algorithm decisions).

b. *USAGE*: What the program is using (e.g., a calling sequence).

c. *RETURN VALUES*: Parameter values to be returned from the program unit.

d. *DEPENDENCIES*: A description of the program unit dependencies (e.g., HW/SW dependencies, INCLUDE files, operating systems, initialization, using a specific compiler).

e. *ERROR CODES/EXCEPTIONS:* Description of, or link to, the error codes used in the program unit.

f. *NOTES*: Any other information needed to increase understanding of the program unit.

2.2.1 Example 1: Sample Header for the Module Noise

```
!----------------------------------------------------------------------
! NAME
!   noise.f
!
! LOCATION
!   $SRC_DIR/noise.f
!
! PURPOSE
!   The F90 module is designed to contain subroutines which are
!   available for calls from external program units, unless declared
!   with the private attribute
!
! DESCRIPTION
!   This module contains the various subroutines related to the
!   handling of the instrument noise, such as generation of artificial
!   noise or the computation of the noise
!
!
! AUTHOR
!   S.A. Boukabara
!
! CREATION DATE
!   2007-03-15
!
! COPYRIGHT
!   THIS SOFTWARE AND ITS DOCUMENTATION ARE CONSIDERED TO BE IN THE PUBLIC
!   DOMAIN AND THUS ARE AVAILABLE FOR UNRESTRICTED PUBLIC USE.  THEY ARE
!   FURNISHED "AS IS." THE AUTHORS, THE UNITED STATES GOVERNMENT, ITS
!   INSTRUMENTALITIES, OFFICERS, EMPLOYEES, AND AGENTS MAKE NO WARRANTY,
!   EXPRESS OR IMPLIED, AS TO THE USEFULNESS OF THE SOFTWARE AND
!   DOCUMENTATION FOR ANY PURPOSE. THEY ASSUME NO RESPONSIBILITY (1) FOR
!   THE USE OF THE SOFTWARE AND DOCUMENTATION; OR (2) TO PROVIDE TECHNICAL
!   SUPPORT TO USERS.
!
! EXTERNALS
!   misc
!   IO_Noise
!
! SUBROUTINES
!   LoadNoise
!   ComputeNoise
!   NoiseOnTopOfRad
!   GenerateNoiseErr
!   BuildMatrxFromDiagVec
!   GRNF
!
! INPUTS
!   N/A
!
! OUTPUTS
!   N/A
!
! HISTORY
!   2007-03-15   Original Code - S.A. Boukabara IMSG Inc. @ NOAA/NESDIS/ORA
!
!----------------------------------------------------------------------
```

2.3 Program Unit Size

***Standard**: The maximum number of characters per line is 90.*

Some old FORTRAN compilers have a 72 character limit (hopefully no one is still using those compilers). The 90 characters standard is good for readability and maintenance.

***Recommendation**: Each program unit is kept as small and simple as possible to perform a specific task.*

Use multiple, smaller routines with well-defined functions rather than a larger routine that does a lot of things. Unwieldy program units spanning hundreds of lines should be examined to see if they can be segmented.

2.4 Naming Convention

Standard: *Names cannot be identical to reserved words or implementation supplied function names.*

This creates confusion, so it must be completely avoided.

Recommendation*: Naming conventions within a programming community are under continual development as programmers communicate with each other and agree to adopt particular conventions.*

Recommendation*: When writing code, the names of files, subroutines, functions, modules and variables created by a programmer are often up to the programmer.*

A possible exception could be if a project to integrate several algorithms from disparate sources into a single program unit developed standardized abbreviations for common variables. As this document deals with code that will be transitioned to operations and could potentially remain in operations for many years it is important that the code be readily maintainable and easily understood.

Recommendation*: The names of files, subroutines, functions and variables can be extremely useful in making code more readable.*

Choosing names may seem not very important, but insisting on meaningful names helps a programmer to organize thoughts and produce code that is readable and reviewable.

Recommendation*: Avoid names that look alike (e.g do not use characters that resemble each other).*

For example, names like 2 and Z, 0 and O, 5 and S, or I and 1.

Recommendation*: Name program units to indicate their purpose.*

This serves two main purposes. First, find a library routine that you can use to implement your desired function and second, avoid using names that are similar to library routines.

Recommendation*: Name symbolic variables to indicate what they are, not what values they may contain.*

Recommendation*: Names should be as mnemonically descriptive as possible, subject to constraints imposed by language standards.*

2.5 Indentations

Indentation shall be used consistently to enhance readability throughout a program.

Standard: *Each indentation should use at least two spaces.*

Standard: *A comment line should be indented in the same way as the following executable line of code.*

Standard: *Statements in nested loops should be indented so that all statements in the same nesting are indented by the same amount.*

Guideline: *Statements in inner nested loops should be indented by a greater amount than statements in* *outer nested loops.*

2.5.1 Example 2: Use of Indentations in Nested Loops

The following example shows how the use of indentations clarifies the intent of the code.

Good example of indentation usage:

```
*******************************************************
   !  Loop over values of x and y
   !  Compute the sides of a right triangle
   !  Then compute the square of the hypotenuse

     DO i=1,5
        x = x_value(i)

        DO j=1,4
           y = y_value(j)

           a = x + 6
           b = y / 4.5

           c_squared(i,j) = a * a + b * b

        END DO
     END DO
*******************************************************
```

bad example of indentation usage:

```
*******************************************************
   !  Loop over values of x and y

         DO i=1,5
         x = x_value(i)
         DO j=1,4
         y = y_value(j)
   !  Compute the sides of a right triangle
              a = x + 6
         b = y / 4.5

   !  Compute the square of the hypotenuse
              c_squared(i,j) = a * a + b * b

     !        Close the loops
         END DO
              END DO
*******************************************************
```

2.6 Nesting

Standard: *The nesting of parentheses in logical and arithmetic expressions shall be limited to four (4) levels.*

If an expression requires a greater level of nesting, it shall be separated into more than one expression.

2.6.1 Example 3: Use of Parentheses and Nesting

The following example shows how the use of parentheses and nesting is used to clarify the order of mathematical operations.

Good example of parentheses and nesting usage:

```
***************************************************************************
         tan_z = TAN ( ( pi * za ) / 180 )
         tan_v = TAN ( ( pi * va ) / 180 )

         fac = ( tan_z * tan_v ) * SIN ( pi * (tan_z / 180) )

***************************************************************************
```

Bad example of parentheses and nesting usage:

```
***************************************************************************
         fac = ( TAN ( pi * za / 180 ) * TAN ( pi * va / 180 )
         &     * SIN ( pi *  TAN ( pi * za  / 180 ) / 180) )
***************************************************************************
```

3. Declarations and Return

3.1 Variable Declarations

***Standard**: Align each declaration type name in the same column to improve readability.*

***Standard**: Avoid extremely long or continuation lines in a declaration statement by using multiple statements.*

***Standard**: List several variables of a single type on a line alphabetically.*

***Standard**: Explicitly dimension all arrays using parameters as much as possible to specify array dimensions/sizes. We recognize that it is not always possible to do so.*

***Standard**: Use of dynamic memory allocation is encouraged.*

3.2 Input and Output (I/O)

***Standard**: Identify the input and output variables in the header.*

***Recommendation**: Separate Input, Output, and Processing functions in a program so that all Input functions precede all Processing functions and are followed by all Output functions.*

Exceptions to this rule occur when memory constraints require dynamic allocation of memory within the processing function. When dynamic allocation is used, input and output functions within processing functions should be clearly identified by comments that identify the input/output variables with references to the Preamble and/or design documents.

3.3 Check Return Values

Standard: *Check for error return values, even from functions that "can't" fail. It is recommended that the following convention be used for error return values:*

- A value of zero indicates the function completed successfully.
- A negative value indicates the function failed.
- A positive value indicates the function completed successfully but encountered something unexpected.
- Include the system error text for every system error message.

Standard: *Take special care with I/O statements since these are usually affected by events beyond the control of the programmer.*

Include an item which causes control to be transferred to the statement attached to that label in the event of an error. This must, of course, be an executable statement and in the same program unit.

3.3.1 Example 4: I/O Statements

The following example shows how to use an executable statement properly.

```
*************************************************************************
         READ(UNIT=IN, FMT=*, ERR=999) VOLTS, AMPS
         WATTS = VOLTS * AMPS
         rest of program in here . . . . . and finally
         STOP
    999  WRITE(UNIT=*,FMT=*)'Error reading VOLTS or AMPS'

         END
*************************************************************************
```

Standard: *Handle the end-of-file condition when reading past the end of a sequential/ internal file.*

Recommendation: *Programmers should check the success of any dynamic memory allocation or deallocation.*

3.3.2 Example 5: Use of Dynamic Memory Allocation

Check for the success of this allocation. Note there is an error if STAT is not equal to zero.

```
**********************************************************
         ALLOCATE(x(N,N),STAT=alloc_stat)
         If(STAT.eq.0)then
         . . .
**********************************************************
```

4. Readability

4.1 Readability of General Programs

Consistency is the key to making programs easily readable.

Standard: *Begin each program unit at the top a new page.*

Standard: *Use a maximum of 90 characters per line of code.*

Standard: *Alphabetic case shall be used consistently to enhance readability throughout a program.*

Standard: *Place spaces before and after compound expressions (relational operators, reserved words, identifiers, and arithmetic operators) to enhance readability of compound expressions.*

4.1.1 Example 6: Use of Consistent Variable Spaces and Size

The following example shows how the consistent use of variables clarifies the intent of the code.

Good examples of parentheses and spacing usage:

```
************************************************************
         tempC = ( tempF – 32.0 ) * 5 / 9


         C = A + ( B * X )


         d = e + f + y
************************************************************
```

Bad example of parentheses and spacing usage:

```
************************************************************
         C=a+B*x
************************************************************
```

Guideline: Blocking with blank lines shall be used consistently to enhance readability throughout a program.

Guideline: All comment lines that are followed by an executable line of code should be separated from the executable line of code either by a single blank line or by no blank line. This is an optional matter of style that should be used consistently throughout a program.

4.1.2 Example 7: Use of Blocking and Comment Lines

The following example shows how the use of proper comment line formatting clarifies the intent of the code.

Good example of blocking and comment line usage:

```
***********************************************************
     ! Compute the sides of a right triangle

       a = x + 6
       b = y / 4.5

     ! Compute the square of the hypotenuse

       c_squared = a * a + b * b

***********************************************************
```

Good example of blocking and comment line usage:

```
***********************************************************
     ! Compute the sides of a right triangle
       a = x + 6
       b = y / 4.5

     ! Compute the square of the hypotenuse
       c_squared = a * a + b * b
***********************************************************
```

Bad example of blocking and comment line usage:

```
***********************************************************
    ! Compute the sides of a right triangle
      a = x + 6
      b = y / 4.5
     ! Compute the square of the hypotenuse
      c_squared = a * a + b * b
***********************************************************
```

Guideline: *The evaluation of logical and arithmetic expressions shall be clarified through the use of parentheses and spaces.*

4.1.3 Example 8: Use of Parentheses in Logical Expressions

The following example shows how the use of parentheses clarifies the intent of the code.

Good example of parentheses usage:

```
***********************************************************
         pk = pk - 1.0 + ( 0.5 * REAL(ning) )
***********************************************************
```

Bad example of parentheses usage:

```
***********************************************************
         pk = pk - 1.0 + 0.5 * REAL(ning)
***********************************************************
```

5. Best Practices

5.1 No Hardcoding

Standard: *Paths should not be hardcoded into software.*

Paths should not be spelled out in absolute paths but rather relative paths, and then defined in config files. (ex. `$SRC_DIR/noise.f` where `$SRC_DIR` is defined in a config file)

Guideline: *Variables should not be hardcoded into software.*

It is good to keep the code as flexible as possible. Global constants such as mathematical or geophysical constants, such as Pi or the Earth's radius, should be contained in a single constants file. Examples of software elements that should not be hardcoded include file paths, temporal data (Year, Month or day), spatial extents (latitudes or longitudes), etc.

5.2 Development

Standard: *Always verify inputs.*

When you code a read statement from any data source, know what you expect and check for that, whether it is a number within a certain range, a date string, a filename, an array, etc. Check string lengths, integer size, and array bounds, for example. Be especially careful if you are reading input to allocate memory, such as from a file header. If that content is corrupt or larger than anticipated, one program could consume all of the machine's memory resource. If you do not read what you expect, generate an error and stop processing.

Standard: *Clean up after the application.*

Include a clean-up script in the application package. Things that can potentially fill a disk, if not removed include: input data, output data, log files, internal mail messages, intermediate files. (Software security sources recommend the institution of disk quotas, so that one out of control application does not impact other applications on the shared system.)

Standard: *Be careful with usage of* `rm -rf`.

If you supply this command with a variable that holds a path name you want removed, make sure that path exists. If the variable is undefined, or simply defined as "/", this command will recursively erase everything in the entire file system owned by that user and everything to which that user has write permission.

Standard: *Monitor all changes.*

When developing on a shared system, or after making a change on an operational system, always verify that something running via a scheduler is working as expected. Monitor at least one run of the software to ensure proper function. Changes made and not monitored are a frequent source of problems in operations.

Standard: *Avoid system calls if there is a system function interface call available in the language you are using.*

For example, do not use a system call to ftp when you can use the Perl ftp module instead. This improves portability of the software.

***Guideline:** Watch for hung programs and standard output.*

It is convenient to direct the standard output and standard error from a program to a log file for diagnostic purposes. However, a program hung in an infinite loop can spew output to a log file and potentially fill the disk. Therefore, a software package should always be able to limit file sizes and check for hung processes. The run time for a process can be obtained using the ps system command and the process can be killed using the `kill` system command. Any resources the program had should be freed.

***Guideline:** Tailor error messages to the appropriate audience.*

Too much information confuses the recipient of the message and tends to be ignored. Messages sent to the operators need less technical information than messages intended for the application programmer.

***Guideline:** Use compiler options; turn on all checking during development.*

Eliminate warnings by modifying the code – do not just look at errors that prevent the compile. This can catch many flaws. To improve operational performance, the final recompile usually removes most of this checking for operational processing. But compiler options should be turned on again when verifying maintenance changes.

***Guideline:** Watch the use of multiple thread situations as this can be dangerous.*

An application can behave differently than expected if more than one instance is kicked off at once. Satellite data can arrive in spurts. Think about whether shared intermediate files and ancillary files, or simultaneous writing to logs or output could be a problem. Scripts or control software (such as OPUS) can be used to ensure an application remains single threaded.

6. Security Concerns

This document lists general issues that apply to all software. Language specific considerations are addressed in those specific documents (for example, C is specifically prone to buffer overflows). These practices will result in code that is more reliable, easier to maintain, and less likely to impact other applications on the same system.

6.1 Security Issues

The issue of software security has become a high priority within NOAA. How this will affect the code we develop for our scientific applications is still evolving. Most of the information available on the internet refers to applications that sit on web servers. Very few of our applications are exposed to the public in this fashion. Most of our applications reside on servers shared with other applications. When one considers availability and reliability under the umbrella of software security, it becomes apparent that there are issues programmers need to consider. This section outlines some potential security issues to avoid when developing code for operational use. We call these "security issues" not because they specifically constitute a vulnerability to outside attack, but because they are software defects that may pose a risk to all operational programs and hardware running on a given machine or within a network. These design flaws can take down software, hardware, and networks, and because they occur internally can have devastating impacts.

Standard: *Never hardcode passwords*.

This is a security violation. Set up keys when there is a need to transfer data between systems.

Standard: *Limit frequency and number of connections via ftp/sftp or scp.*

Failure to do so can consume bandwidth on a remote server creating an unintended denial of service attack. Scripts are often used to automate these connections and transfers. A script can be constructed so it sleeps (using the *sleep* system command) between cycles in which is initiates connections. This will slow the frequency of access. Also, the software can be written so that it limits the total number of connections allowed at any given time.

Standard: *Own the source code.*

Do not use executable files obtained from outside sources. If there is a requirement to use executable files without obtaining the source code, security must verify that they were obtained from a trusted source.

Standard: *Check ownership and permissions.*

Make sure consideration is made to allow only the appropriate user ID (the production ID for operational systems) to modify and run the application.

Recommendation: *Root configure the system so that each user is also allowed a certain number of processes he is allowed to run (defined as maxprocs) at any instant.*