GSI Gridpoint Statistical Interpolation

Advanced User's Guide

Version 3.5

August, 2016

Ming Hu
National Oceanic and Atmospheric Administration (NOAA)/Earth System Research Laboratory Cooperative Institute for Research in Environmental Sciences (CIRES)

Chunhua Zhou, Hui Shao, Don Stark, and Kathryn Newman
National Center for Atmospheric Research (NCAR)

DTC Developmental Testbed Center

Acknowledgement

This user's guide is constructed with contributions from distributed GSI developers. We give our special acknowledgement to these contributors and reviewers, including, but not limit to:

National Centers for Environmental Prediction (NCEP) Environmental Modeling Center (EMC):
John Derber, Russ Treadon, Mike Lueken, Wan-Shu Wu, Andrew Collard, and Ed Safford

National Center for Atmospheric Research (NCAR):
Xiang-Yu Huang, Syed Rizvi, Zhiquan Liu, and Arthur Mizzi

National Oceanic and Atmospheric Administration (NOAA) Earth System Research Laboratory (ESRL):
Steve Weygandt, Dezso Devenyi, Joseph Olson, and Jeff Beck

Shanghai Meteorological Service:
Min Sun

The GSI community support and code management effort is sponsored by NOAA's Office of Oceanic and Atmospheric Research (OAR). This work is also facilitated by NCAR. NCAR is supported by the National Science Foundation (NSF).

Foreword

This document, designed for experienced users, includes advanced knowledge, features, and skills of GSI as well as details of assimilation of specific data types. Users may use as a reference for their special research topics. To read this guide, users should already read and understand the content in the *GSI User's Guide*.

This version of Advanced GSI User's Guide was released with the community GSI version 3.5 in August 2016. Please note, not like the basic GSI user's guide which is being updated every year and closely follows the GSI release code, this advanced user's guide, as a reference, is only being updated as needed and therefore doesn't pertain to one specific code release.

There are 10 Chapters in this document:

DTC may update the content of this advanced User's Guide, if needed, between releasees. For the latest version of this document, please visit the GSI User's Website at

http://www.dtcenter.org/com-GSI/users.v3.5/docs/index.php

Please send questions and comments to:

gsi-help@ucar.edu

For referencing this document, please use:

Developmental Testbed Center, 2016: Gridpoint Statistical Interpolation Advanced User's Guide Version 3.5. Available at http://www.dtcenter.org/com-GSI/users.v3.5/docs/index.php, 119 pp.

For referencing the general aspect of the GSI community effort, please use:

Shao, H., J. Derber, X.-Y. Huang, M. Hu, K. Newman, D. Stark, M. Lueken, C. Zhou, L. Nance, Y.-H. Kuo, B. Brown, 2016: Bridging Research to Operations Transitions: Status and Plans of Community GSI. Bulletin of the American Meteorological Society, doi:10.1175/BAMS-D-13-00245.1, in press

Table of Contents

Chapter 1: Overview

Purpose of the Advanced GSI User's Guide

This document is the second part of the GSI User's Guide. For the history of GSI and its community efforts, please refer to the Overview of the *GSI User's Guide*, released with each code version.

While the *GSI User's Guide* focuses on basic information for compiling, running, and diagnosing GSI, this *Advanced GSI User's Guide* is intended to help users who have mastered the fundamental portion of the GSI and would like to apply GSI for specific research topics that need more advanced knowledge and skills.

Unlike *the GSI User's Guide*, which is released annually with the official release, the Advanced User's Guide will initially release with the official release but may be updated after the release based on needs and contributions from users and developers. The latest release time and subversion will be indicated on the title page of this document.

Some of the contents of this *Advanced User's Guide* are not updated to match the official release of the GSI code like the fundamental portion. Therefore, users are advised to refer to the relative content with caution, as there may be differences between the content and the code. Please contact the GSI help desk with any issues with using this guide.

Some of the sections and chapters have only titles in this release (no content). These are place hold for important topics of the GSI. The content will be added in the future as knowledge and resources are available to update the topic. Users and developers are very welcome to make any contributions to the guide, either with updated content or with new additions.

This document is intended to provide useful assistance to experienced GSI users and developers for advancing GSI development and research.

Subversion release log:

Version	Release time	Modifications
3.3.0.0	06/20/2014	Initial release with official release 3.3
3.3.0.1	07/07/2014	Fix typos in Equation 1-5 in Chapter 4
3.3.0.2	08/12/2014	Fix typos in Table in Section 5.5. Add step 5 in radar reflectivity analysis in section 9.3
3.4.0.0	08/09/2015	Update on the use of anavinfo file in section 3.2 Add section 9.2.1 Data Preprocessing of Radar Radial Velocity Assimilation within GSI Add section 9.2.2 The Processes of the read_radar.f90 code

		Update in section 10.1 GSI global analysis Update in the namelist (Appendix A) based on version 3.4
3.5.0.0	08/08/2016	Remove section 10.1 GSI global analysis to GSI User's Guide Chapter 6. Added the introduction to the GSI 4D hybrid EnVar as section 10.1. Delete Appendix A because it is in GSI User's Guide Appendix C

Structure of this User's Guide:

The User's Guide is organized as follows:

Chapter 2 provides detailed information on software installation, including description of examples for tailoring the building system on non-standard computing platforms.

Chapter 3 contains advanced topics related to running and diagnosing GSI

Chapter 4 illustrates the GSI data assimilation technique and minimization procedure

Chapter 5 introduces major processes and subroutines associated with GSI I/O, observation ingestion, and innovation calculation.

Chapter 6 illustrates concept of background error covariance, estimation of static background error covariance as well as how GSI processes background error information.

Chapter 7 provides information regarding observation processing for GSI. It contains basic skills for BUFR/PrepBUFR files, including how to encode, decode, and append new data into these types of files. It also provides information on GSI BUFR interface, NCEP processes for BUFR/PrepBUFR files, and the observation error adjustment procedure inside GSI.

Chapter 8 discusses radiance data assimilation in GSI, including data ingestion, quality control, bias correction, and other associated procedures.

Chapter 9 discusses radar data assimilation in GSI.

Chapter 10 describes various GSI operational applications.

Chapter 2: Software Installation

2.1 Modifying the GSI Build Environment

The GSI build system is designed to compile on most standard Unix/Linux systems. Typically, if the WRF model builds on a system, GSI will build there as well. The lack of standardization of Linux HPC environments, specifically from big vendors such as SGI and IBM, may necessitate minor customization of the GSI build settings for those computing environments.

Typical build problems seen can be traced back to issues with the location of libraries, MPI wrappers for the compiler, or the support utilities such as cpp. These sorts of issues can usually be solved by customizing the default configuration file settings. Unfortunately this may involve an iterative process where the build parameters are modified, the compile script is run, build errors diagnosed, and the process repeated.

2.2 Understanding the Build System

The GSI build system uses a collection of data files and scripts to create a configuration resource file that defines the local build environment.

At the top most level there are four scripts. The `clean` script removes everything created by the build. The `configure` script takes local system information and queries the user to select from a collection of build options. The results of this are saved into a resource file called `configure.gsi`. Once the `configure.gsi` file is created, the actual build is initiated by running the `compile` script. The `compile` script then calls the top-level `makefile`, substitutes in settings from the configure file, and builds the source code.

Name	Content
`makefile`	Top-level makefile
`arch/`	Build options and machine architecture specifics
`clean`	Script to clean up the directory structure
`configure`	Script to configure the build environment for compilation. Creates a resource file called `configure.gsi`
`compile`	Script for building the GSI system. Requires the existence of the `configure.gsi` prior to running

The `compile` script uses the resource file `configure.gsi` to set paths and environment variables required by the compile. The `configure` script generates the resource file `configure.gsi` by calling the Perl script `Config.pl`, located in the `arch/` directory. The script `Config.pl` combines the build information from the files in the `arch/` directory with machine specific and user provided build information to construct the `configure.gsi` resource file.

A "clean" script is provided to remove the build objects from the directory structure. Running ./`clean` scrubs the directory structure of the object and module files. Running a clean-all ./`clean -a` removes everything generated by the build, including the library files, executables, and the configure resource file. Should the build fail, it is strongly recommended that the user run a ./`clean -a` prior to rerunning the `compile` script.

The `arch/` directory contains a number of files used to construct the configuration resource file `configure.gsi.`

File name	Description
`preamble`	Uniform requirements for the code. Currently only contains shell information and comments.
`configure.defaults`	Selection of compilers and options. Users can edit this file if a change to the compilation options or library locations is needed. It can also be edited to add a new compilation option if needed.
`postamble`	Standard compilation ("make") rules and dependencies

Most users will not need to modify **any** of these files unless experiencing significant build issues. Should a user require a significant customization of the build for their local computing environment, those changes would be saved to the `configure.defaults` file only after first testing these new changes in the temporary `configure.gsi` file.

2.2.1 Configuration Resource File

The configuration resource file `configure.gsi` contains build information, such as compiler flags and paths to system libraries, specific to a particular machine architecture and compiler.

To illustrate its contents, lets look at the resource for the Linux Intel/gcc build.

```
# Settings for Linux x86_64, Intel/gnu compiler (ifort & gcc)    (dmpar,optimize)#
```

The header describes the overall build environment

- Linux x86 with 64 bit word size
- Uses Intel Fortran and GNU C compilers

The link path points to an Intel version of NetCDF and the OpenMP libraries.

```
LDFLAGS        =  -Wl,-rpath,/usr/local/netcdf3-ifort/lib -openmp
```

The code directory location and include directory:

```
COREDIR        =  $HOME/comGSIv3.4_EnKFv1.0
```

```
        INC_DIR        =  $(COREDIR)/include
```

Compiler definitions

- Intel ifort Fortran compiler
- GNU gcc C compiler

```
    SFC            =  ifort
    SF90           =  ifort -free
    SCC            =  gcc
```

The include paths for GSI source code and NetCDF:

```
  INC_FLAGS      =  -module $(INC_DIR) -I $(INC_DIR) -I /usr/local/netcdf3-ifort/include
```

The default Fortran compiler flags for the main source code:

```
    FFLAGS_DEFAULT =  -fp-model precise -assume byterecl -convert big_endian
    FFLAGS_FULLOPT =  -O3
    FFLAGS         =  $(FFLAGS_OPT) $(FFLAGS_DEFAULT) $(INC_FLAGS) $(LDFLAGS) -DLINUX
```

Note that the flag ‘convert big_endian” switches the byte order from the native “little endian” to “big endian.” This allows GSI to ingest “big endian” binary files and there by maintaining compatibility with legacy NOAA output.

The default Fortran compiler flags for the external libraries:

```
        FFLAGS_BACIO   =  -O3 $(FFLAGS_DEFAULT)
        FFLAGS_BUFR    =  -O3 $(FFLAGS_DEFAULT) $(FFLAGS_i4r8)
        CFLAGS_BUFR    =  -O3 -DUNDERSCORE
        FFLAGS_CLOUD   =  -O3 $(FFLAGS_DEFAULT)
        FFLAGS_CRTM    =  -O2 $(FFLAGS_DEFAULT)
        FFLAGS_GFSIO   =  -O3 $(FFLAGS_DEFAULT) $(FFLAGS_i4r4)
        FFLAGS_SFCIO   =  -O3 $(FFLAGS_DEFAULT) $(FFLAGS_i4r4)
        FFLAGS_SIGIO   =  -O3 $(FFLAGS_DEFAULT) $(FFLAGS_i4r4)
        FFLAGS_SP      =  -O3 $(FFLAGS_DEFAULT) $(FFLAGS_i4r8)
        FFLAGS_W3      =  -O3 $(FFLAGS_DEFAULT)
        #
```

The default CPP path and flags. If your system has multiple versions of cpp and you do not wish to use the version in your path, it may be necessary to specify the specific version here

```
        CPP            =  cpp
        CPP_FLAGS      =  -C -P -D$(BYTE_ORDER) -D_REAL8_ -DWRF -DLINUX
        CPP_F90FLAGS   =  -traditional-cpp -lang-fortran
```

The MPI compiler definitions:

```
        DM_FC          =  mpif90 -f90=$(SFC)
        DM_F90         =  mpif90 -free -f90=$(SFC)
        DM_CC          =  gcc
```

A few comments should be made here about the use of the `mpif90` wrapper to invoke the “parallel” compiler build. The default version of the build shown here has the additional flag `-f90=$(SFC)` following the call to `mpif90`. This flag specifies what compiler is to be used for the parallel build. In this example `SFC = ifort` there by telling the script to use the Intel compiler. This is the standard with the open source versions of MPI such as MPICH2 and OPENMPI. Supercomputer venders such as SGI, CRAY, and IBM no longer follow this convention. Depending on the vendor, including the `-f90=` flag results in, at the least, compiler warnings, and at most, compiler errors. Because of this situation, the release code has an extra build option for each of the compilers ,“Vendor supplied MPI,” which removes the `-f90=` flag from the build rules.

Unfortunately this is not the end of this story. The two vendors SGI MPT and IBM PE have done away with the `mpif90` wrapper completely and instead prefer to call the Intel compiler directory with an additional MPI flag:

```
DM_FC          =  ifort
DM_F90         =  ifort -free
```

This will be addressed in next section illustrating how to modify the build rules.

The default C compiler flags:

```
CFLAGS  =  -O0 -DLINUX -DUNDERSCORE
CFLAGS2 =  -DLINUX -Dfunder -DFortranByte=char -DFortranInt=int -DFortranLlong='long
long'
```

The default library paths and names

- Variable `LAPACK_PATH` needs to point to the MKL library location
- The library names may be different on other systems

```
MYLIBsys       = -L$(LAPACK_PATH) -mkl=sequential
```

NetCDF path information

- Older versions of NetCDF only have the single library `-lnetcdf`. If you are using an older version you may need to remove the first library name.

```
NETCDFPATH      =  /usr/local/netcdf3-ifort
NETCDFLIBS      =  -lnetcdff -lnetcdf $(NETCDF_PATH)
```

It should not be necessary to modify anything below the NetCDF environment variables.

2.2.2 Modification Example

To demonstrate how one would go about modifying the configuration resource file, the generic Linux/Intel configuration will be ported to build on an SGI MPT Linux cluster called Zeus. Zeus comes with a vender-supplied version of MPI, which necessitates modification of the MPI paths.

The first change is that Zeus does not use the traditional MPI wrappers such as `mpif90` to invoke the compiler. Instead the Intel compiler is called directly with an additional `-lmpi` flag to specify an MPI build. Therefore the `DM` compiler definitions become:

```
DM_FC          =  ifort
DM_F90         =  ifort -free
DM_CC          =  gcc
```

Next, additional link flags for MPI are needed. These are in **bold**.

```
LDFLAGS        =  -Wl,-rpath,/usr/local/netcdf3-ifort/lib -L$MPI_ROOT/lib -lmpi -openmp
```

Then add the path to the MPI include directory, along with the additional Fortran flag.

```
FFLAGS_DEFAULT =  -msse2 -fp-model precise -assume byterecl  -I$MPI_ROOT/include
```

An equivalent include path for the C flags are also needed.

```
CFLAGS         =  -O0 -DLINUX -DUNDERSCORE -I$MPI_ROOT/include
```

These changes should be saved to the users `configure.gsi` resource file and tested. Once they are confirmed to work, they may be moved into the `configure.defaults` file located in the `arch/` directory as a new build target.

To save your new build configuration, open the file `configure.defaults`, located in the *`arch/`* directory. You will notice that it contains a collection of platform/compiler specific entries. The first entry is for the IBM platform, using the xlf compiler with 64-bit word size. This entry is indicated by a label at the top of the block starting with the tag `#ARCH`. For the 64-bit IBM build, the tag is:

```
#ARCH AIX 64-bit   #dmpar
```

The block for the 64-bit IBM build is immediately followed by the 32-bit IBM build entry, which is indicated by the tag:

```
#ARCH AIX 32-bit   #dmpar
```

with each subsequent build specification is delineated by a similar tag.

For our port of the generic Intel build to Zeus, locate the tag for the Linux/Intel build with 64 bit words. Its header looks like this:

```
#ARCH Linux x86_64, Intel compiler (ifort & gcc) # (dmpar,optimize)
```

Duplicate this entry and give it a unique name by modifying the ARCH entry.

```
#ARCH Linux x86_64, Intel compiler SGI MPT (ifort & gcc) #
(dmpar,optimize)
```

Then update the variables to match the settings in the `configure.gsi` resource file tested previously, and save your changes. Now when you run the ./`configure` script, there will be a new build option for an SGI MPT build.

Chapter 3: Advanced Topics on Run and Diagnosis

The basic skills of running GSI and diagnosing GSI results are introduced in the Chapter 3 and Chapter 4 of the GSI User's Guide. This chapter discusses some complex issues for advanced users to further tune and diagnosis GSI runs.

3.1 Convergence Information from File fort.220

In file *fort.220*, users can find more detailed minimization information about each iteration. The following example uses the first two iterations to explain the meaning of each value:

```
Minimization iteration          0
```

1)

```
 J= 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.118329009942697698E+05
    0.190285043373867524E+05 0.401338098573457983E+05 0.468178247339593265E+04
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.185513606236281089E+05 0.100380070802053093E+06 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00
```

2)

```
 b=-0.310927744401462716E+04 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.410038969466198277E+06
    0.723694789994664703E+06 0.287063341578062015E+06 0.294343224843158402E+05
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.357063077297121385E+06 0.259969713154007923E+08 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00
```

3)

```
 c= 0.310927744401462659E+08 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.595529299391540965E+08
    0.968629709661563464E+08 0.320963665012150593E+08 0.207128030095876056E+07
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.141290320912310097E+09 0.325532123706306098E+11 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00
```

4)

```
EJ= 0.277433945264109695E+02 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.111001385595905754E+05
    0.177294227112127074E+05 0.396140623480684926E+05 0.462763172936301229E+04
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.179761712946645360E+05 0.741628809765858670E+05 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00
```

5)

```
 stepsize estimates =  0.100000000000000005E-03  0.944604610006952448E-03
                      0.944604610006930960E-03
 stepsize stprat    =  0.894135600291783517E+00  0.227490031410343044E-13
 stepsize guesses   =  0.100000E-03  0.900000E-04  0.110000E-03  0.000000E+00  0.944605E-03
                      0.935159E-03  0.954051E-03
 penalties          =  0.000000E+00  0.559315E+03 -0.552732E+03  0.588939E+04 -0.234810E+05
                      -0.234780E+05 -0.234780E+05
pcgsoi: gnorm(1:2)  3.10927744401462748 6E+07  3.109277444014627486E+07
costterms Jb,Jo,Jc,Jl  =   1   0  0.000000000000000000E+00  1.946084290880794579E+05
                      0.000000000000000000E+00  0.000000000000000000E+00
cost,grad,step,b,step? =   1   0  1.946084290880794579E+05  5.576089529423489694E+03
                      9.446046100069309653E-04  0.000000000000000000E+00  good
estimated penalty reduction this iteration   1   0  2.937037807406784850E+04
                      1.509203800251368577E-01%
penalty and grad reduction WRT outer and initial iter=   1   0  1.000000000000000000E+00
                      1.000000000000000000E+00  1.000000000000000000E+00
                      1.000000000000000000E+00

Minimization iteration          1
 J= 0.277433945264109712E+02 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.111001385595905754E+05
    0.177294227112127073E+05 0.396140623480684925E+05 0.462763172936301231E+04
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.179761712946645356E+05 0.741628809765858657E+05 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00
 b=-0.103454706240741566E+06 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.471859414082640217E+06
    0.836887408435857958E+06 0.444144650683562874E+06 0.493857599036264164E+05
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.143436922590024884E+06-0.241501731777713210E+08 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00
 c= 0.744926644276673220E+08 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.187038212807235956E+09
    0.245074032578180438E+09 0.116810874950575594E+09 0.631952836573825778E+07
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.327987495483898158E+09 0.597416983300266383E+11 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00
EJ= 0.907421173071728001E+02 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.104139003066849756E+05
    0.165779309617118123E+05 0.390129888433980952E+05 0.456584643970885767E+04
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.175622634769410845E+05 0.567994645981838240E+05 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00
    0.000000000000000000E+00 0.000000000000000000E+00
 stepsize estimates =  0.944604610006930965E-03  0.577090170662210155E-03
0.577090170662210132E-03
 stepsize stprat    =  0.636840580602851869E+00  0.386854261310281249E-16
 stepsize guesses   =  0.944605E-03  0.850144E-03  0.103907E-02  0.000000E+00  0.577090E-03
0.571319E-03  0.582861E-03
 penalties          =  0.000000E+00 -0.367282E+04  0.475604E+04  0.120164E+05 -0.819848E+04
-0.819646E+04 -0.819646E+04
pcgsoi: gnorm(1:2)  3.502903930399505794E+07  3.502903930399504304E+07
costterms Jb,Jo,Jc,Jl  =   1   1  2.774339452641097026E+01  1.652103076194851892E+05
0.000000000000000000E+00  0.000000000000000000E+00
cost,grad,step,b,step? =   1   1  1.652380510140116094E+05  5.918533543369932886E+03
5.770901706622101049E-04  1.126597414824659582E+00  good
estimated penalty reduction this iteration   1   1  2.021491427007579478E+04
1.038748134641514359E-01%
penalty and grad reduction WRT outer and initial iter=   1   1  8.490796199748631423E-01
1.061412933228467859E+00  8.490796199748631423E-01  1.126597414824660026E+00
```

For each inner iteration, there are 5 sections of outputs. The 1st iteration is labeled with numbers 1 to 5, with a detailed explanation below:

1 – 4) detailed information on the cost function (`J=`), b term for estimate stepsize (`b=`), c term for estimate stepsize (`c=`), estimate terms in penalty (`EJ`). There are 32 (8 + number of observation types) items listed in each and the meanings of these items are:

```
1 contribution from background, satellite radiance bias, and
   precipitation bias
2  place holder for future linear linear term
3  contribution from dry pressure constraint term (Jc)

4  contribution from negative moisture constraint term (Jl/Jq)
5  contribution from excess moisture term (Jl/Jq)
6  contribution from negative gust constraint term
7  contribution from negative vis constraint term
8  contribution from negative pblh constraint term
```

9-32: contributions to Jo from different observation types:

```
9  contribution from ps observation term
10 contribution from t observation term
11 contribution from w observation term
12 contribution from q observation term
13 contribution from spd observation term
14 contribution from srw observation term
15 contribution from rw observation term
16 contribution from dw observation term
17 contribution from sst observation term
18 contribution from pw observation term
19 contribution from pcp observation term
20 contribution from oz observation term
21 contribution from o3l observation term (not used)
22 contribution from gps observation term
23 contribution from rad observation term
24 contribution from tcp observation term
25 contribution from lagrangian tracer
26 contribution from carbon monoxide
27 contribution from modis aerosol aod
28 contribution from level modis aero aod
29 contribution from in-situ pm2_5 obs
```

```
30 contribution from gust monoxide
31 contribution from vis aerosol aod
32 contribution from pb1h modis aero aod
```

For further understand of these terms, it is suggested that the users check *stpcalc.f90* for the code including the above information.

Some terms in section 5 are explained below:

- `stepsize estimates:` final step size estimates
- `stepsize stprat:` convergence in stepsize estimation
- `gnorm(1:2):` 1=(norm of the gradient)^2, 2= (norm of the gradient)^2
- `Jb,Jo,Jc,Jl`**:** the values of cost function, background term (`Jb`), observations term (`Jo`), dry pressure constraint term (`Jc`), and negative and excess moisture term (`Jl`).
- `cost,grad,step,b :` see explanations in the 1st part of this section.
- `estimated penalty reduction this iteration:`
  (penalty current solution- estimate of penalty for new solution),
  (penalty current solution- estimate of penalty for new solution)/(original penalty)
- `penalty and grad reduction WRT outer and initial iter=`
  Penalty reduction to the 1st inner loop value, Grad reduction to the 1st inner loop value , Penalty reduction to the original (1st outer) value, Grad reduction to the original (1st outer) value

3.2 Use Bundle To Configure Control, State Variables And Background Fields

Since the GSI release version 3.0, the control variables, state variables, and background fields can be configured through a new info file named "*anavinfo*". Different GSI applications need a different *anavinfo* file to setup the control variables, state variables, and background fields. In the *./fix* directory of the release package, there are many example *anavinfo* files for different GSI applications. Because this is a work in progress, users should use one of the sample *anavinfo* files instead of making a new one. The released GSI run script has added the link for this new info file.

Below is an example of an *avaninfo* file for an ARW (*anavinfo_arw_netcdf*) case:

```
met_guess::
!var     level    crtm_use    desc              orig_name
  cw       30       10        cloud_condensate  cw
#  ql        30       10        cloud_liquid      ql
#  qi        30       10        cloud_ice         qi
#  qr        30       10        rain              qr
#  qs        30       10        snow              qs
#  qg        30       10        graupel           qg
```

```
::

state_vector::
!var     level  itracer amedge  source     funcof
 u        30      0      no     met_guess    u
 v        30      0      no     met_guess    v
 tv       30      0      no     met_guess    tv
 tsen     30      0      no     met_guess    tv,q
 q        30      1      no     met_guess    q
 oz       30      1      no     met_guess    oz
 cw       30      1      no     met_guess    cw
 p3d      31      0      yes    met_guess    p3d
 ps        1      0      no     met_guess    p3d
 sst       1      0      no     met_guess    sst
::
control_vector::
!var     level  itracer as/tsfc_sdv  an_amp0   source  funcof
 sf       30      0       1.00        -1.0     state    u,v
 vp       30      0       1.00        -1.0     state    u,v
 ps        1      0       0.50        -1.0     state    p3d
 t        30      0       0.70        -1.0     state    tv
 q        30      1       0.70        -1.0     state    q
 oz       30      1       0.50        -1.0     state    oz
 sst       1      0       1.00        -1.0     state    sst
 cw       30      1       1.00        -1.0     state    cw
 stl       1      0       1.00        -1.0     motley   sst
 sti       1      0       1.00        -1.0     motley   sst
::
```

There are three sections in this file:

`met_guess::`	section to configure background fields
`state_vector::`	section to configure state variables
`control_vector::`	section to configure control variables

In each section, the 1st column sets up the variable name and 2nd column sets up the vertical levels. The 4th column in the section `control_vector` is the normalized scale factor for the background error variance. Please be aware that starting from GSI version 3.4, the vertical levels (2nd column) in the anavinfo file should exactly match the vertical levels of the GSI background field. And the variables might also be different between different versions of GSI code and different GSI applications. Users are advised to modify the anavinfo file that comes with the release code to suit their own application.

3.3 Using Observations Station Uselist And Rejection List In GSI

The GSI tries to use all available observations but has also to make significant efforts to avoid bad observations getting into the analysis. The data quality control before GSI and the gross check inside GSI are two major ways to find and toss the bad observations. In addition, GSI can also use station rejection list and uselist to further control which data should be used in the GSI. The rejection list assumes all observations should be used in the GSI analysis except ones in the rejection list, while the uselist assumes all observations should NOT be used except ones in the uselist.

3.3.1 Surface Observation Rejection And Use List

GSI has many kinds of surface rejection list and uselist files. Those files are listed and explained in the following table. If those files are not existing in a GSI run, then the function of using rejection list and uselist will be turned off automatically.

File name used in GSI	Rejection list and uselist array in GSI	Content	Sample files in fix directory
mesonetuselist	cprovider	mesonet provider names from the uselist	nam_mesonet_uselist.txt
w_rejectlist	w_rjlist	station names from the reject list for wind	new_rtma_w_rejectlist
t_rejectlist	t_rjlist	station names from the reject lists for temperature	new_rtma_t_rejectlist
t_day_rejectlist	t_day_rjlist		new_rtma_t_day_rejectlist
t_night_rejectlist	t_night_rjlist		new_rtma_t_night_rejectlist
p_rejectlist	p_rjlist	station names from the reject list for surface pressure	new_rtma_p_rejectlistmore
q_rejectlist	q_rjlist	station names from the reject lists for specific humidity	new_rtma_q_rejectlist
q_day_rejectlist	q_day_rjlist		new_rtma_q_day_rejectlist
q_night_rejectlist	q_night_rjlist		new_rtma_q_night_rejectlist
mesonet_stnuselist	csta_winduse	'good' mesonet station names from the station uselist	nam_mesonet_stnuselist.txt
wbinuselist	csta_windbin	wind direction stratified wind accept lists	new_rtma_wbinuselist

Note, this table is based on the subroutine *init_rjlists* in file *sfcobsqc.f90.*

At the beginning of subroutine *read_prepbufr*, the subroutine *init_rjlists* is called to read station names from the rejection list and uselist files. When a surface observation is read in, subroutine *get_usagerj* is called to compare the station name with the rejection list and uselist to reset the usage flag of the observation.

For rejection list of temperature, moisture, surface pressure, and wind observation other than mesonet wind:

- if incoming *usage* value is >=6. then do nothing since *read_prepbufr* has already flagged this observation and assigned a specific usage value to it;
- if *usage* value is < 6 and those observations are found in the rejection list, set *usage*=5000.
- if *usage* value is < 6 and those observations are not found in the rejection list, keep the original usage value.

Now, only mesonet wind observation has both uselist and rejection list, the details of apply those lists are if *usage* value is < 6, then:

- set *usage* = 6000 and check if this wind observation is found in one of the three uselist:
    - mesonet provider names uselist
    - 'good' mesonet station names uselist
    - wind direction stratified wind accept lists

  if found this station in uselist, then set original *usage* value, otherwise, the *usage* flag of this station is 6000.
- After uselist check, all mesonet observations then go through the rejection list just as other surface wind observations to check if toss this station. So, the stations flagged to use in uselist check may be flagged to large value again in the rejection list.

As a background knowledge, the observation with *usage* flag larger than outer loop number will not be used in the GSI analysis. The above check of the rejection list and uselist are summarized in the following table:

Observation type	List type	Rejection list and uselist array in GSI	If station name match, *Usage* flag change to
Temperature:	Reject list	t_rjlist t_day_rjlist t_night_rjlist	r5000 r5100 r5100
Moisture	Reject list	q_rjlist q_day_rjlist q_night_rjlist	 r5100 r5100
Ps	Reject list	p_rjlist	r5000
surface wind other than mesonet	Reject list	w_rjlist	r5000
Mesonet wind	Uselist	*Set all mesonet obs to usage_rj=r6000, then* • cprovider • csta_winduse • csta_windbin	 usage_rj0 usage_rj0 usage_rj0
	Reject list	w_rjlist	r6100 r6200

3.3.2 Aircraft Observation Rejection

GSI also has rejection list for aircraft observations (PrepBUFR type 129 to 140 and 229 to 240), which are listed and explained in the following table. Again, if those files are not existing in a GSI run, then the function of using rejection list and uselist will be turned off automatically.

File name used in GSI	Array in GSI	Content	Sample files in fix directory
current_bad_aircraft	t_aircraft_rjlist	Aircraft tag number from the reject list for temperature	rap_current_bad_aircraft.txt
	w_aircraft_rjlist	Aircraft tag number from the reject list for wind	
	q_aircraft_rjlist	Aircraft tag number from the reject list for moisture	

The rejection lists for aircraft are used in the same way just like the rejection list for surface data. But the rejection list for temperature, wind, and moisture are save in the same file.

Chapter 4: GSI Theory

The GSI was developed originally as a three-dimensional variational (3DVAR) data assimilation system. It has been evolving to an Ensemble-Var hybrid system in recent years.

As a reference for users to understand the basic GSI analysis procedure, a brief summary of the 3DVAR mathematical theory and the minimization steps used in the GSI is given in this Chapter.

4.1 3DVAR Equations:

The basic 3DVAR equation is:

$$J = \tfrac{1}{2}(x_a - x_b)^T B^{-1}(x_a - x_b) + \tfrac{1}{2}(Hx_a - o_o)^T O^{-1}(Hx_a - o_o) + J_c \qquad (1)$$

where:

x_a: Analysis fields
x_b: Background fields
B : Background error covariance matrix
H : Observation operator
o_o: Observations
O: Observation error covariance
J_c: Constraint terms (e.g., dynamical constraint, moisture constraint)

Define an analysis increment (Δx=) $x = x_a - x_b$, then equation (1) becomes:

$$J = \tfrac{1}{2}x^T B^{-1} x + \tfrac{1}{2}(H(x_b + x) - o_o)^T O^{-1}(H(x_b + x) - o_o) + J_c \qquad (2)$$

By assuming the linearity of the observation operator H, equation (2) can be written as:

$$J = \tfrac{1}{2}x^T B^{-1} x + \tfrac{1}{2}\big(Hx - (o_o - Hx_b)\big)^T O^{-1}\big(Hx - (o_o - Hx_b)\big) + J_c \qquad (3)$$

Next, define the observation innovation as $o = o_o - Hx_b$, equation (3) becomes:

$$J = \tfrac{1}{2}x^T B^{-1} x + \tfrac{1}{2}(Hx - o)^T O^{-1}(Hx - o) + J_c \qquad (4)$$

4.2 Iterations To Find The Optimal Results

To improve convergence ,GSI preconditions its cost function by defining a new variable $y = B^{-1}x$. Equation (4), in terms of the new variable y, becomes:

$$J = \tfrac{1}{2}y^T By + \tfrac{1}{2}(HBy - o)^T O^{-1}(HBy - o) + J_c \qquad (5)$$

Using the chain rule, the gradients of background and observation parts of the cost function (4) with respect to x and cost function (5) with respect to y have the form:

$$\nabla_x J = B^{-1}x + H^T O^{-1}(Hx - o) \qquad (6)$$
$$\nabla_y J = B^T y + B^T H^T O^{-1}(HBy - o) = B\nabla_x J \qquad (7)$$

Equations (6) and (7) are simultaneously minimized by employing an iterative Conjugate Gradient process.

Start by assuming:

$$x^0 = y^0 = 0$$

Then iterate over *n*:

$$\nabla_x J^n = B^{-1}x^{n-1} + H^T O^{-1}(Hx^{n-1} - o) = y^{n-1} + H^T O^{-1}(Hx^{n-1} - o)$$
$$\nabla_y J^n = B\nabla_x J^n$$

$$Dir\cdot x^n = \nabla_y J^n + \beta Dir\cdot x^{n-1}$$
$$Dir\cdot y^n = \nabla_x J^n + \beta Dir\cdot y^{n-1}$$

$$x^n = x^{n-1} + \alpha Dir\cdot x^n$$
$$y^n = y^{n-1} + \alpha Dir\cdot y^n$$

Until either the maximum number of iterations has been reached or the gradient is sufficiently minimized.

During the above iteration, the β is calculated in subroutine *pcgsoi* and the stepsize (α) is calculated in subroutine *stpcalc*.

Please note that the current version GSI has more minimization options in addition to the one described above. Such as:

- Minimize cost function using sqrt(B) preconditioner when namelist variable *lsqrtb* is set to true.
- Minimization using Bi-conjugate gradient for minimization when namelist variable *lbicg* is set to true

4.3 Analysis Variables

Typically, there are seven analysis variables used in GSI analysis:

Stream function (ψ)
Unbalanced velocity potential (χ)
Unbalanced virtual temperature (T)
Unbalanced surface pressure (P)
Pseudo relative humidity [qoption =1] or normalized relative humidity [qoption=2]
Ozone mixing ratio (only for global GSI)
Cloud condensate mixing ratio (only for global GSI)

With broader application of GSI for chemical data assimilation, some new variables, such as trace gases, aerosols, and chemistry are added as analysis variables. Also, gust and visibility were added as analysis variables for RTMA application.

Chapter 5: GSI Code Structure

This Chapter introduces the basic code structure of the GSI. Section 5.1 describes the main processes of the GSI consisting of the three main routines. Sections 5.2 to 5.5 introduce the code related to four important parts of GSI: background IO, observation ingestion, observation innovation calculation, and minimization iteration.

5.1 Main Process

At the top most level of abstraction, the GSI code is divided into three phases; the initialization, the run, and the finalize phase. The philosophy behind this division is to create a modular program structure with tasks that are independent of one another.

The main top-level driver routine is called *gsimain* and is located in the file `gsimain.f90`. Ninety percent of `gsimain.f90` is a variety of useful Meta data.

- Major change history
- List of input and output files
- List of subroutines and modules
- List of external libraries
- Complete list of exit states
- A discussion of important namelist options

Possibly the most important of these is the list of exit codes. Should the GSI run fail from an internal error, the exit code may provide sufficient insight to resolve the issue. The final lines of `gsimain.f90` consist of the three main calls to initialize, run and finalize. The table below summarizes each of these phases.

gsimain.f90	main steps in each call
call gsimain_initialize (gsimod.F90)	• gsi_4dcoupler_parallel_init • MPI initialize • Initialize defaults of variables in modules • Read in user input from namelist • 4DVAR setup if it is true (*not supported)* • Check user input for consistency among parameters for given setups • Optional read in namelist for single observation run • Write namelist to standard out • If this is a wrf regional run, the run interface with wrf: **call convert_regional_guess** (details in section 6.2.2) • Initialize variables, create/initialize arrays • Initialize values in radinfo and aeroinfo
call gsimain_run (gsimod.F90)	• Call the main GSI driver routine **call gsisub(mype)** (check next page for steps in gsisub)
If 4DVAR, then: *call gsi_4dcoupler_final_traj*	
call gsimain_finalize (gsimod.F90)	• Deallocate arrays • MPI finalize

GSI main process (continue)

subroutine gsisub (gsisub.F90)
high level driver for GSI

If not ESMF

- Allocate grid arrays
- Get date, grid, and other information from background files

End if not ESMF

- If *single observation test*:
  Create prep.bufr file with single obs in it
- If *regional analysis*:
  Read in Level 2 land radar winds and create radar wind superob file
  call radar_bufr_read_all
- If *initial pass*:
  Read info files for assimilation of various observations
- If *initial pass*:
  Computer random number for precipitation forward model
- Complete setup and execute external and internal minimization loops

```
if (lobserver) then
   if initial pass: call observer_init
                    call observer_run
   if last pass: call observer_finalize
else
   call glbsoi(mype)
endif
```

- If *last pass*:
  Deallocate arrays

Note: lobserver = if true, calculate observation departure vectors only.

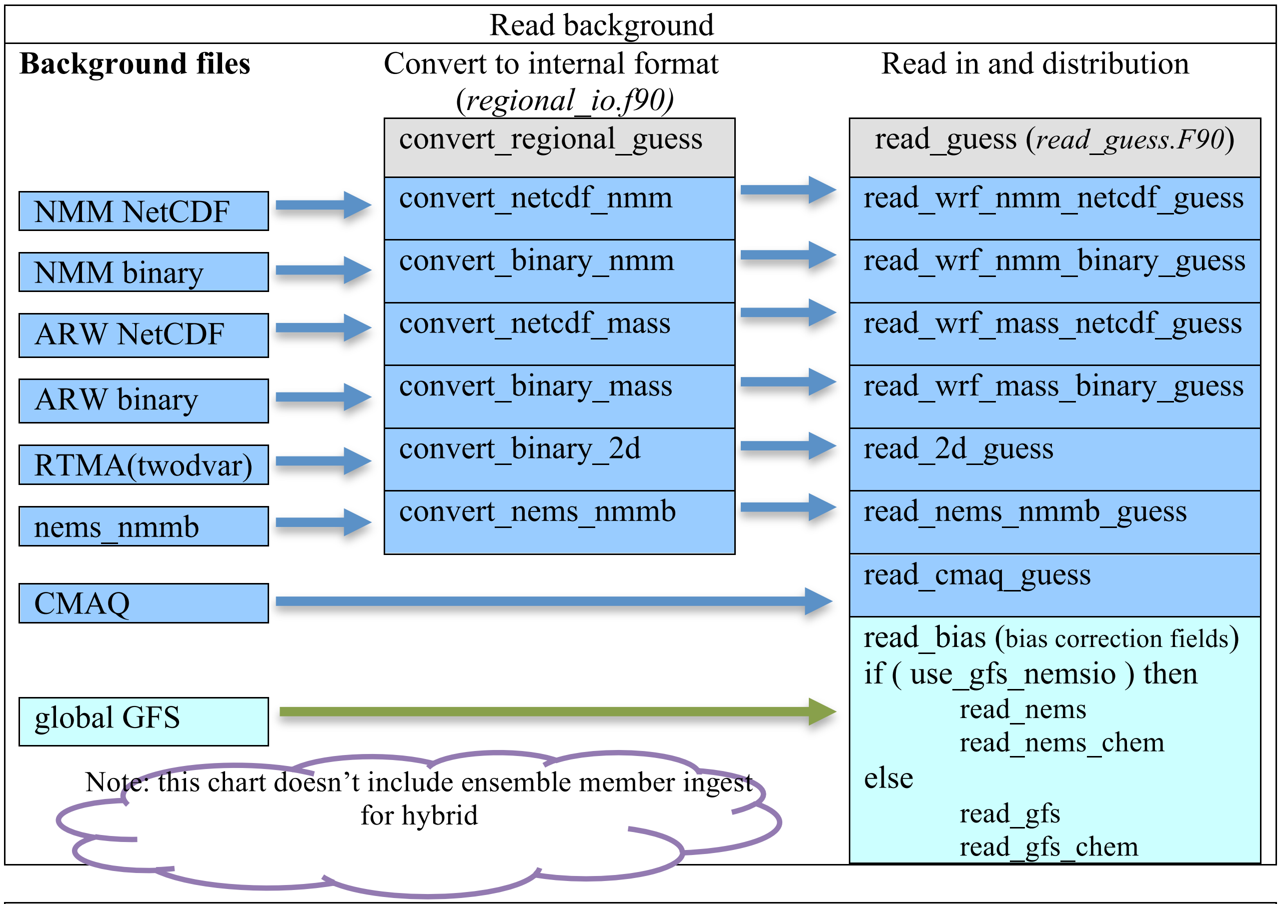
subroutine glbsoi (glbsoi.f90)
driver for GSI

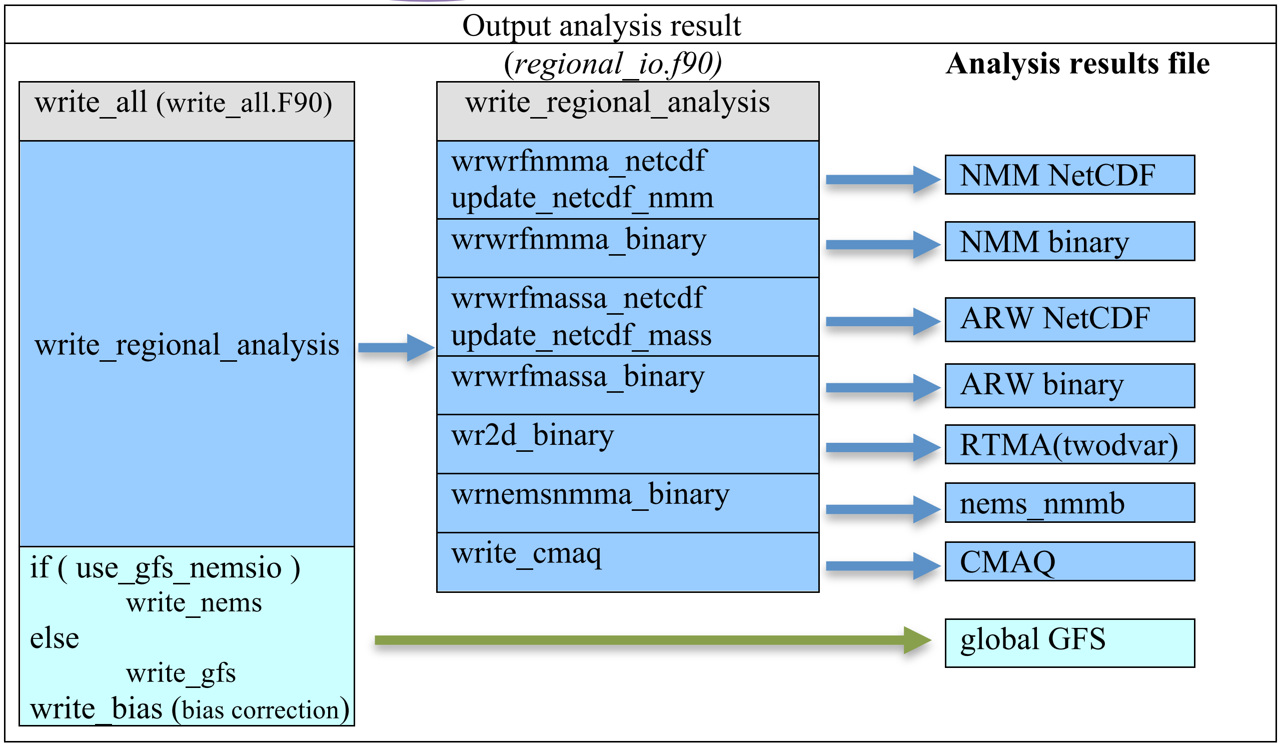
- Initialize timer for this procedure
- If *l_hyb_ens is true*, then initialize machinery for hybrid ensemble 3dvar
- Check for alternative minimizations
- Initialize observer
- Check GSI options against available number of guess time levels
- Read observations and scatter
- Create/setup background error and background error balance
- If *l_hyb_ens is true*, then read in ensemble perturbations
- If *4d-var and not 1st outer loop*, then read output from previous minimization.
- Set error (variance) for predictors (only use guess)
- Set errors and create variables for dynamical constraint
- Main outer analysis loop

```
do jiter=jiterstart,jiterlast
➢ Set up right hand side of analysis equation
  call setuprhsall (details in section 6.2.4)
➢ Set up right hand side of adjoint of analysis
  equation if forecast sensitivity to observations
➢ Inner minimization loop
  if (laltmin) then
     if (lsqrtb) call sqrtmin
     if (lbicg)  call bicg
  else
     call pcinfo
     call pcgsoi (details in section 6.2.5)
  endif
➢ Save information for next minimization
➢ Save output of adjoint of analysis equation
end do ! jiter
```

- Calculate and write O-A information
- Deallocate arrays
- Write updated bias correction coefficients
- Finalize observer
- Finalize timer for this procedure

5.2 GSI Background IO (for 3DVAR)





5.3 Observation Ingestion

Data type *(ditype)*	Observation type *(obstype)*		Subroutine that reads data
conv	t, q, ps, pw, spd, mta_cld, gos_ctp, gust, vis		read_prepbufr
	uv	from satwnd	read_satwnd
		Not from satwnd	read_prepbufr
	sst	from mods	read_modsbufr
		not from mods	read_prepbufr
	srw		read_superwinds
	tcp		read_tcps
	lag		read_lag
	rw (radar winds Level-2)		read_radar
	dw (lidar winds)		read_lidar
	rad_ref		read_RadarRef_mosaic
	lghtn		read_lightning
	larccld		read_NASA_LaRC
	pm2_5		read_anowbufr
	pblh		read_pblb
rad (satellite radiances)	(platform) not AQUA	amsua	read_bufrtovs (TOVS 1b data)
		amsub	
		msu	
		mhs	
		hirs4,3,2	
		ssu	
	(platform) AQUA	airs	read_airs (airs data)
		amsua	
		hsb	
	atms		read_atms
	iasi		read_iasi
	cris		read_cris
	sndr, sndrd1/2/3/4		read_goesndr (GOES sounder data)
	ssmi		read_ssmi
	amsre_low/mid/hig		read_amsre
	ssmis, ssmis_las/uas/img/env		read_ssmis
	goes_img		read_goesimg
	seviri		read_seviri
	avhrr_navy		read_avhrr_navy
	avhrr		read_avhrr
ozone	subuv2, omi, gome, o3lev, mls		read_ozone
co	mopitt		read_co
pcp	pcp_ssmi, pcp_tmi, pcp_amsu,pcp_stage3		read_pcp
gps	gps_ref, gps_bnd		read_gps
aero	aod		read_aerosol

Note: This table is based on *subroutine read_obs* in *read_obs.F90*:

- Data type is saved in array *ditype*
- Observation type is save in array *obstype.* In namelist, the observation type is *dtype*

Each observation type uses one or more processors to read in the data and then write the data into a intermediate file called *obs_input.**, where * is a processor ID that is used to read in certain observation type.

Then in *subroutine* ***obs_para*** (*obs_para.f90*), each processor reads through all *obs_input.** files, pick observations within its sub-domain, and save them into a file called: *pe*.obs-type_outer-loop*, where * is 4 digital processor ID.

5.4 Observation Innovation Calculation

Data type *(ditype)*	Observation type *(obstype)*		Subroutine calculate innovation
conv	t		setupt
	uv		setupw
	q		setupq
	ps		setupps
	pw		setuppw
	spd		setupspd
	sst		setupsst
	srw		setupsrw
	tcp		setuptcp
	lag		setuplag
	rw (radar winds Level-2)		setuprw
	dw (lidar winds)		setupdw
	pm2_5		setuppm2_5
	gust		setupgust
	vis		setupvis
	pblh		setuppb1h
rad (satellite radiances)	(platform) not AQUA	amsua	setuprad
		amsub	
		msu	
		mhs	
		hirs4,3,2	
		ssu	
	(platform) AQUA	airs	
		amsua	
		hsb	
	atms		
	iasi		
	cris		
	sndr, sndrd1, sndrd2 sndrd3, sndrd4		
	ssmi		
	amsre_low/mid/hig		
	ssmis ssmis_*		
	goes_img		
	seviri		
	avhrr_navy		
	avhrr		
ozone	subuv2, omi, gome,		setupozlay
	o3lev, mls		setupozlev
pcp	pcp_ssmi, pcp_tmi, pcp_amsu,pcp_stage3		setuppcp
co	mopitt, subuv2		setupco
gps	gps_ref		setupref
	gps_bnd		setupbend

Note: this table is based on *subroutine setuprhsall* in *setuprhsall.f90*:
- Data type is saved in array *ditype*
- Observation type is in array *obstype*

- The observation departure from the background of each outer loop is calculated in *subroutine setuprhsall.*
- A array (*rdiagbuf*) that holds observation innovation for diagnosis is generated in each setup routine. (Also see A.2)
- The index of the data array for temperature in *setupt* is list below:

index		**content**
1	ier	obs error
2	ilon	grid relative obs location (x)
3	ilat	grid relative obs location (y)
4	ipres	pressure
5	itob	t observation
6	id	station id
7	itime	observation time in data array
8	ikxx	observation type
9	iqt	flag indicating if moisture obs available
10	iqc	quality mark
11	ier2	original-original obs error ratio
12	iuse	use parameter
13	idomsfc	dominant surface type
14	iskint	surface skin temperature
15	iff10	10 meter wind factor
16	isfcr	surface roughness
17	ilone	longitude (degrees)
18	ilate	latitude (degrees)
19	istnelv	station elevation (m)
20	iobshgt	observation height (m)
21	izz	surface height
22	iprvd	observation provider
23	isprvd	observation subprovider
24	icat	data level category
25	iptrb	t perturbation

5.5 Inner Iteration

The inner iteration loop of GSI is where the cost function minimization is computed. GSI provides several minimization options, but here we will focus on the preconditioned conjugate gradient method. The inner iteration of the GSI variational analysis is performed in subroutine pcgsoi (*pcgsoi.f90*). inside the following loop:

```
inner_iteration: do iter=0,niter(jiter)

     …

end do inner_iteration
```

The main steps inside the loop are listed as a table below with the corresponding code and the terms of equation in Section 6.1.

Steps in inner iteration	Code in pcgsoi.f90	Corresponding equations in Chapter 4 (variables are defined in Chapter 4)
Gradient of observation term	*call intall*	$H^T O^{-1}(Hx^{n-1} - o)$
Add gradient of background term	*gradx(i)=gradx(i)+yhatsave(i)*	$\nabla_x J^n = H^T O^{-1}(Hx^{n-1} - o) + y^{n-1}$
Apply background error covariance	*call bkerror(gradx,grady)*	$\nabla_y J^n = B\nabla_x J^n$
Calculate norm of gradients		b=β $\beta = \frac{(\nabla_x J^n - \nabla_x J^{n-1})^T \nabla_y J^n}{(\nabla_x J^n - \nabla_x J^{n-1})^T Dir \cdot x^n}$

Calculate new search direction	*dirx(i)=-grady(i)+b*dirx(i)*
	*diry(i)=-gradx(i)+b*diry(i)*
		$Dir\cdot x^n = \nabla_y J^n + \beta Dir\cdot x^{n-1}$
		$Dir\cdot y^n = \nabla_x J^n + \beta Dir\cdot y^{n-1}$

Calculate stepsize	*call stpcalc*	stp=$\alpha = \frac{\sum a}{\sum b}$

Update solution inside *stpcalc*	*xhatsave(i)=xhatsave(i)+stp*dirx(i)*
	*yhatsave(i)=yhatsave(i)+stp*diry(i)*
		$x^n = x^{n-1} + \alpha Dir\cdot x^n$
		$y^n = y^{n-1} + \alpha Dir\cdot y^n$

For detailed steps, advanced developers are suggested to read through the code and send questions to gsi_help@ucar.edu.

Chapter 6: Static Background Error Covariance

The background error covariance is most important part of variational analysis method to determine the impact ratio, distribution, and relations of the analysis increments. In this Chapter, we will discuss the issues related to static background error covariance used in the GSI analysis.

6.1 What Is Background Error Covariance

Background error covariance plays a very important role in determining the quality of variational analysis for NWP models. It controls what percentage of the innovation becomes the analysis increment, how each observation impacts a broad area, and the balance among different analysis variables.

Since most of the data assimilation background is model forecasts from a prior time step, the background error covariance matrix (**B**) can be defined as the error covariance of model forecasts:

$$[Forecast\ (x) - Truth\ (x_{truth})]$$

Since the actual state of atmosphere (truth) is not known, the forecast errors need to be estimated. When estimating forecast errors, the most common methods are the “NMC method” and “ensemble method”. In the “NMC method”, forecast errors are estimated with the difference of two (typically 12 and 24 hours) forecasts valid for the same time. In the “ensemble method”, the forecast errors are estimated with ensemble perturbations (ensemble - ensemble mean).

Because of the size of the model variables, the full size of a **B** matrix is extremely large. It is typically on the order of 10^6x10^6, which in its present form cannot be stored in any computer. This problem is simplified by using an ideal set of analysis variables for which the analysis is performed. These are generally referred to as “analysis control variables”. The analysis control variables are selected such that the cross-correlations between these variables are minimum, which means less off-diagonal terms in **B**. The cross dependency among these analysis control variables is removed. The balance between analysis variables (such as mass and wind fields) are achieved with pre-computed “regression coefficients”. Further, the forecast errors are modeled as a Gaussian distribution with pre-computed variances and “lengthscale” parameters for each of the analysis control variables. We will use the following sub-sections to briefly introduce how GSI processes these pre-computed background error statistics and applies them in a GSI analysis.

To achieve desired regression coefficients, variance, and lengthscale parameters, offline computation should be conducted with a sufficiently large data set for a period of time, typically, more than one month. For this purpose, a separate utility called “*gen_be*” can be used. It is released as a stand-alone tool for the generation of the background error covariance matrix based on the forecasts from a user defined forecast system. Details about this utility can be found in the 2012 GSI residential tutorial lecture by Rizvi *et al.* (the lecture slides are available on-line at the GSI User's Page).

6.2 Processing Of Background Error Matrix

The GSI package has several files in *~/comGSI_v3.2/fix/* to hold the pre-computed background error statistics for different GSI applications with different grid configurations. Since the GSI code has a build-in mechanism to interpolate the input background error matrix to any desired analysis grid, the following two background error files can be used to specify the **B** matrix for any GSI regional application.

- *nam_nmmstat_na.gcv* : contains the regional background error statistics, computed using forecasts from the NCEP's NAM model covering North America. The values of this B matrix cover the northern hemisphere with 93 latitude lines from -2.5 degree to 89.5 degree with 60 vertical sigma levels from 0.9975289 to 0.01364.
- *nam_glb_berror.f77.gcv* : contains the global background errors based on the NCEP's GFS model, a global forecast model. The values of this B matrix covers global with 192 latitude lines from -90 degree to 90 degree and with 42 vertical sigma levels from 0.99597 to 0.013831.

Also included in this release package is the background error matrix for RTMA GSI:

- new_rtma_regional_nmm_berror.f77.gcv

These background error matrix files listed above are Big Endian binary files. In the same directory, *nam_nmmstat_na.gcv_Little_Endian* and *nam_glb_berror.f77.gcv_Little_Endian* are their Little Endian versions for certain computer platforms that cannot compile GSI with the Big Endian option. In this release version, GSI can be compiled with the Big Endian option with PGI and Intel, but not with gfortran compiler.

All the parameters for the global background error statistics are latitude dependent. In the case of the regional background error statistics, regression coefficients of velocity potential as well as variances and horizontal lengthscales for all the control variables are latitude dependent. The remaining parameters such as regression coefficients for unbalanced "surface pressure", "temperature" and vertical lengthscales for all the fields do not vary with latitude.

In the GSI code, the background error statistics are initially read in at their original sigma levels and interpolated vertically in log (sigma) coordinates on the analysis vertical sigma levels. In subroutines "*prewgt*" and "*prewgt_reg*", lengthscales (both horizontal and vertical) and variance information are read in and then vertically interpolated to analysis grids by calling "*berror_read_wgt"* and "*berror_read_wgt_reg"*, while the balance information is read in and vertically interpolated to analysis grids by calling "*berror_read_bal*" and "*berror_read_bal_reg*", respectively for global and regional applications.

Table 6.1 shows the list of arrays in which the original background error statistics are read by the various subroutines discussed above.

Table 6.1 The information on arrays used by GSI background error matrix

Category	Array name	Dimension	Content
Balance (Horizontal regression coefficients)	*agvi*	0:mlat+1,1:nsig,1:nsig	Regression coefficients for stream function and temperature
	wgvi	0:mlat+1,1:nsig	Regression coefficients for stream function and surface pressure
	bvi	0:mlat+1,1:nsig	Regression coefficients for stream function and velocity potential
Horizontal and vertical influence scale	*hwll*	0:mlat+1,1:nsig,1:nc3d	horizontal lengthscales for stream function, unbalanced velocity potential, unbalanced temperature, and relative humidity
	hwllp	0:mlat+1, nc2d	horizontal lengthscale for unbalanced surface pressure
	vz	1:nsig, 0:mlat+1, 1:nc3d	Vertical lengthscale for stream function, unbalanced velocity potential, unbalanced temperature, and relative humidity
variance	*corz*	1:mlat,1:nsig,1:nc3d	Square root of variance for stream function, unbalanced velocity potential, unbalanced temperature, and relative humidity
	corp	1:mlat,nc2d	Square root of variance for unbalanced surface pressure

Note: mlat = number of latitude in original background error coefficient domain,
nsig = number of vertical levels in analysis grid
nc3d = number of 3 dimensional analysis variables
nc2d = number of 2 dimensional analysis variables

Horizontal interpolation of regression coefficients to the desired grid is done for global and regional applications respectively in subroutines "*prebal*" and "*prebal_reg*", residing in the "*balmod.f90*" module. Horizontally interpolated regression coefficients on the desired grid are stored in "*bvz*", "*agvz*",*"wgvz"* and "*bvk*", "*agvk*", "*wgvk*" arrays for global and regional applications, respectively. These regression coefficients are used in subroutine *balance* to build the respective balance part of velocity potential, temperature, and surface pressure fields.

In subroutines "*prewgt_reg*" and "*prewgt"*, horizontal and vertical lengthscales (*hwll, hwllp, vz*) and variance (*corz, corp*) information are horizontally interpolated and adjusted with the corresponding input tuning parameters ("*vs*","*hzscl*", "*as3d"* and *"as2d"*) supplied through *gsiparm.anl* and *anavinfo.txt*. Desired information is finally processed and transformed to new arrays such as "*slw*", "*sli*", "*dssv*" and "*dssvs*", which are subsequently used for recursive filter applications both in the horizontal and vertical directions. The variance array: *dssv* is an allocated array for 3D variables with dimensions *"lat", "lon", "nsig", "variables"*. The *dssvs* is an allocated array for 2D variables with dimensions "*lat", "lon", "variables"*. For both of these arrays, allocation of variables is decided by the input parameters supplied via "*anavinfo*" and from the background grid configuration.

6.3 Apply Background Error Covariance

According to the variational equations used in the GSI, the background error covariance is used to calculate the gradient of the cost function with respect to *y* based on the gradient of the cost function with respect to *x*, which can be represented below following Section 6.1.2:

$\nabla_y J = B\nabla_x J$ (subroutine *bkerror(gradx,grady)*)

Because **B** is very complex and has a very large dimension in most data analysis domains, in reality, it must be decomposed into several sub-matrices to fulfill its function step by step. In GSI, the **B** matrix is decomposed into the following form:

$$B = B_{balance} V B_Z (B_x B_y B_y B_x) B_Z V B^T_{balance}$$

The function of each sub-matrix is explained in table 6.2:

Table 6.2 the function of sub-B matrix

Sub-matrix of B	Function	Subroutine	GSI files
$B_{balance}$	balance among different variables	*balance*	*balmod.f90*
$B^T_{balance}$	adjoint of balance equation	*tbalance*	*balmod.f90*
V	Square root of variance	*bkgvar*	*bkgvar.f90*
B_Z	vertical smoother	*frfhvo*	*smoothzrf.f90*
$B_x B_y B_y B_x$	Self-adjoint smoothers in West-East (B_x) and South-North (B_y) direction	*smoothrf*	*smoothzrf.f90*

The composition of B is achieved by calling *bkerror* in following three steps:

Step 1. Adjoint of balance equation ($B^T_{balance}$) is done by calling *tbalance*

Step 2. Apply square root of variances, vertical and horizontal parts of background error correlation by calling subroutine *bkgcov*

- Multiply by square root of background error variances (V) by calling *bkgvar;*
- Apply vertical smoother (B_Z) by calling *frfhvo;*
- Convert from subdomain to the full horizontal field distributed among processors by calling *general_sub2grid;*

- Apply self-adjoint smoothers in West-East (B_x) and South-North (B_y) direction by calling *smoothrf*. Smoothing in the horizontal is achieved by calling *ryxyyx* at each vertical sigma level in a loop over number of vertical sigma levels (*nlevs*). Smoothing for three horizontal scales is done with the corresponding weighting factors (*hswgt*) and horizontal lengthscale tuning factors (*hzscl*);
- The horizontal field is transformed back to respective subdomains by calling *general_grid2sub;*
- Apply vertical smoother (B_Z) by calling *frfhvo;*
- Multiply by the square root of background error variances (V) by calling *bkgvar.*

Step 3. Application of balance equation ($B_{balance}$) is done by calling *balance*

In this step the balance part of velocity potential, temperature and surface pressure is computed from the stream function filled by using the corresponding regression coefficients as follows:

```
velocity potential = unbalanced velocity potential + $B_{Balance(st \to vp)}$ stream function
temperature        = unbalanced temperature        + $B_{Balance(st \to t)}$ stream function
surface pressure   = unbalanced surface pressure   + $B_{Balance(st \to p)}$ stream function
```

Chapter 7 Observations

The observation types that can be used by GSI and how to add or remove certain observation have been discussed in detail in the GSI User's Guide. But there are more issues related to observations that users should know when they apply their own data with GSI or want o improve the use of data. As an operation system, GSI development team has invested significant effort to improve the data process inside and outside GSI.

In this chapter, we will discuss several important observation issues for better application of the GSI, including:

- Process BUFR/PrepBUFR files
- Understand GSI interface to the observations
- The basic knowledge on NCEP observation files
- Observation error inflation inside the GSI

The first three topics are tailored from the "BUFR/PrepBUFR User's Guide" to help users process observations for GSI more quickly. If users have problem to understand the BUFR/PrepBUFR process or want to learn more details of the DC BUFR table and more examples on PrepBUFR process, please check BUFR User's Page and the BUFR User's Guide:

http://www.dtcenter.org/com-GSI/BUFR/index.php

7.1 Process BUFR/PrepBUFR Files

7.1.1 introduction

BUFR (**B**inary **U**niversal **F**orm for the **R**epresentation of meteorological data) is Table Driven Data Representation Forms approved by the World Meteorological Organization (WMO) for operational use since 1988. Since then, it has been used for the representation and exchange of observational data, as well as for archiving of all types of observational data in operation centers, including National Center for Environmental Prediction (NCEP).

BUFR is a self-descriptive table driven code form that offers great advantages of flexibility and expandability compared with the traditional alphanumeric code form as well as packing to reduce message sizes.

As one of the operation centers, NCEP converts and archives all observational data received into a BUFR tank and provides several kinds of BUFR files for its global and regional numerical weather forecast systems. These BUFR files are used by the NCEP operational data analysis system, Gridpoint Statistical Interpolation (GSI), as the standard data sources. Therefore, it is one of DTC's GSI user support tasks to provide suitable documentation for community GSI users to acquire basic knowledge and skills to use BUFR form.

In this Section, a set of simple example programs is employed to explain how to process BUFR/PrepBUFR files. The PrepBUFR is the NCEP term for "prepared" or QC'd data in BUFR format (NCEP convention/standard). These examples are Fortran codes and are available in the community GSI release version 3 and later package under directory *./util/bufr_tools/.* Through these examples, users can easily understand the usage of several commonly used BUFRLIB subroutines, and how these subroutines, together with DX BUFR table, are worked together to encode, decode, append BUFR/PrepBUFR files. These examples can also serve as a starting point for users to solve their specific BUFR file processing problems.

The examples studied in this section include:

bufr_encode_sample.f90:	Write one temperature observation with location and time into a BUFR file.
bufr_decode_sample.f90:	Read one temperature observation with location and time out from the BUFR file.
bufr_append_sample.f90:	Append one temperature observation with location and time into an existing BUFR file.

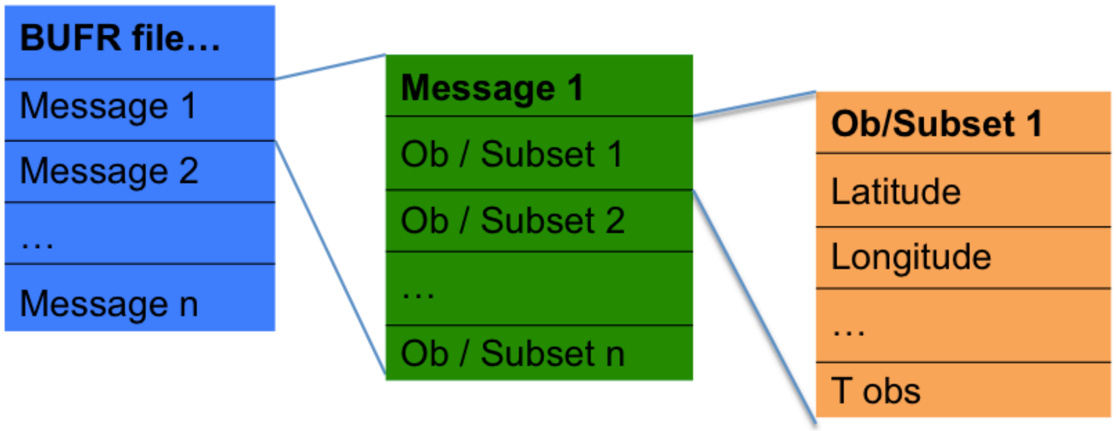
Please note that these examples are based on the NCEP BUFRLIB. We will use examples to introduce commonly used BUFRLIB subroutines and functions and the code structure of BUFR processing.

BUFR/PrepBUFR file structure

BUFR file structure should be described as: "A BUFR message contains one or more BUFR data subsets. Each data subset contains the data for a single report from a particular observing site at a particular time and location, in addition to time and location information. Typically each data subset contains data values such as pressure, temperature, wind direction and speed, humidity, etc. for that particular observation. Finally, BUFR messages themselves are typically stored in files containing many other BUFR messages of similar content." Therefore, if we summarize in a top-down fashion, we would say:

"A BUFR file contains one or more BUFR messages,
each message containing one or more BUFR data subsets,
each subset containing one or more BUFR data values. "

We can also represent the BUFR/PrepBUFR file structure using the following figure.



7.1.2 Encode, Decode, Append A Simple BUFR File

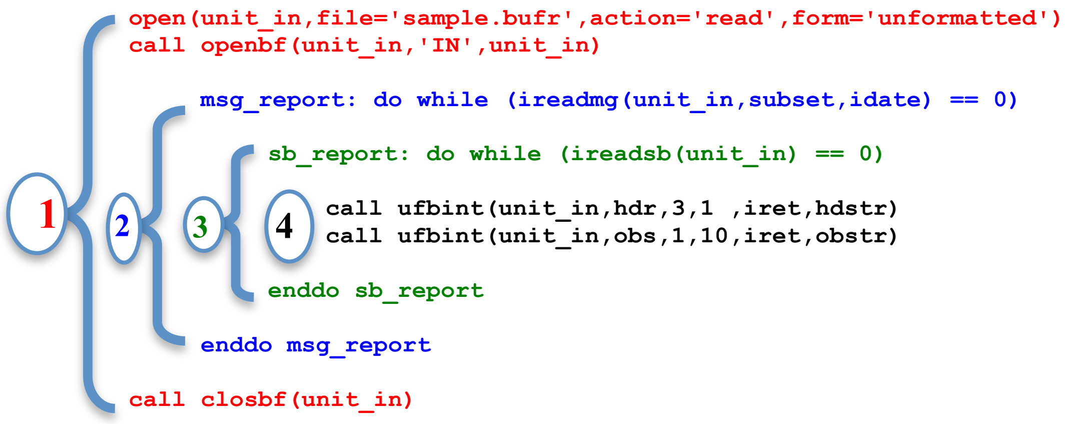
7.1.2.1 Decoding/Reading Data From A Simple BUFR File

The following is from the code *bufr_decode_sample.f90*, which shows how to read specific observation values (among a large variety) out from a BUFR file.

```
program bufr_decode_sample
!
! example of reading observations from bufr
!
 implicit none

 character(80):: hdstr='XOB YOB DHR'
 character(80):: obstr='TOB'
 real(8) :: hdr(3),obs(1,10)

 integer :: ireadmg,ireadsb
 character(8) subset
 integer :: unit_in=10
 integer :: idate,iret,num_message,num_subset

! decode
 open(unit_in,file='sample.bufr',action='read',form='unformatted')
 call openbf(unit_in,'IN',unit_in)
 call datelen(10)
   num_message=0
   msg_report: do while (ireadmg(unit_in,subset,idate) == 0)
      num_message=num_message+1
      num_subset = 0
      write(*,'(I10,I4,a10)') idate,num_message,subset
      sb_report: do while (ireadsb(unit_in) == 0)
      num_subset = num_subset+1
      call ufbint(unit_in,hdr,3,1 ,iret,hdstr)
      call ufbint(unit_in,obs,1,10,iret,obstr)
      write(*,'(2I5,4f8.1)') num_subset,iret,hdr,obs(1,1)
      enddo sb_report
   enddo msg_report
 call closbf(unit_in)

end program
```

Specifically, this example will read all temperature observation values with observation location and time from a BUFR file named *sample.bufr*.

The structure of the above FORTRAN BUFR decoding code matches the top-down hierarchy of a BUFR file. To better illustrate this structure, the code is divided into four different levels:



```
open(unit_in,file='sample.bufr',action='read',form='unformatted')
call openbf(unit_in,'IN',unit_in)

    msg_report: do while (ireadmg(unit_in,subset,idate) == 0)

        sb_report: do while (ireadsb(unit_in) == 0)

           call ufbint(unit_in,hdr,3,1 ,iret,hdstr)
           call ufbint(unit_in,obs,1,10,iret,obstr)

        enddo sb_report

    enddo msg_report

call closbf(unit_in)
```

- The 1st Level: the three RED lines are the first level (file level) statements, which open/close a **BUFR file** for decoding.
- The 2nd Level: the two BLUE lines are the second level (message level) statements, which read in **BUFR messages** from the BUFR file. Each loop reads in one message until the last message in the file is reached.
- The 3rd Level: the two GREEN lines are the third level (subset level) statements, which read in **BUFR data subsets** from a BUFR message. Each loop reads in one subset until the last subset in the message is reached.
- The 4th Level: The BLACK lines are the fourth level (data level) statements, which read in user picked **data values** into user defined arrays from each BUFR subset.

All BUFR encode, decode, and append programs have the same structure as listed here. The message loop (`msg_report`) and subset loop (`sb_report`) are needed only if there are multiple messages in a file and multiple subsets in a message, which is the case for most types of observations.

There are several commonly used BUFRLIB subroutines/functions in the code. We will explain the usage of each of them in detail based on the NCO BUFRLIB document. Users are encouraged to read the explanations carefully in parallel to the example code to understand the usage of each function. Understanding the usage of these functions and BUFR file structure are key to successfully processing all NCEP BUFR files.

1st level (file level): open a BUFR file

```
open(unit_in,file='sample.bufr',action='read',form='unformatted')
call openbf(unit_in,'IN',unit_in)
     …
call closbf(unit_in)
```

- The ***open*** command: Fortran command to link a BUFR file with a logical unit. Here the action is '*read*' because we want to decode (read) only. The form is always "unformatted" because the BUFR file is a binary stream.

- ***openbf***:

```
CALL OPENBF  ( LUBFR, CIO, LUNDX )

Input arguments:
    LUBFR        INTEGER       Logical unit for BUFR file
    CIO          CHAR*(*)     'IN' or 'OUT' or 'APX'  (or NUL', 'NODX',
                                                 'SEC3' or 'QUIET')
    LUNDX        INTEGER       Logical unit for BUFR tables
```

This subroutine identifies to the BUFRLIB software a BUFR file that is connected to logical unit *LUBFR*. The argument *CIO* is a character string describing how the file will be used, e.g. 'IN' is used to access an existing file of BUFR messages for reading/decoding BUFR, and 'OUT' is used to access a new file for writing/encoding BUFR. An option 'APX' behaves like 'OUT', except that output is then appended to an existing BUFR file rather than creating a new one from scratch, and there are also some additional options 'NUL', 'NODX', 'SEC3', 'QUIET'. It will be sufficient to further consider only the 'IN', 'OUT', 'APX' cases for the purposes of this discussion. The third argument *LUNDX* identifies the logical unit of DX BUFR table. Except when *CIO*='SEC3', every BUFR file that is presented to the BUFRLIB software must have a DX BUFR tables file associated with it, and these tables may be defined within a separate ASCII text file or, in the case of an existing BUFR file, may be embedded within the first few BUFR messages of the file itself, and in which case the user needs to set *LUNDX* to the same value as *LUBFR*. In any case, note that *LUBFR* and *LUNDX* are logical unit numbers; therefore, the user must have already associated these logical unit numbers with actual filenames on the local system, typically via a FORTRAN "OPEN" statement.

Currently, as many as 32 BUFR files can be simultaneously connected to the BUFRLIB software for processing. Of course, each one must have a unique *LUBFR* number and be defined to the software via a separate call to subroutine *OPENBF*.

In this example, LUBFR=LUNDX= *unit_in* since BUFR table is already embedded within the BUFR messages of the file itself. CIO uses 'IN' for reading BUFR file.

- ***closbf***:
  Since *OPENBF* is used to initiate access to a BUFR file, *CLOSBF* would be used to terminate this access:

```
CALL CLOSBF  ( LUBFR )

 Input argument:
    LUBFR     INTEGER      Logical unit for BUFR file
```

This subroutine severs the connection between logical unit *LUBFR* and the BUFRLIB software. It is always good to call *CLOSBF* for every *LUBFR* that was identified via *OPENBF*; *CLOSBF* will actually execute a FORTRAN "CLOSE" on logical unit *LUBFR* before returning, whereas *OPENBF* did not itself handle the FORTRAN "OPEN" of the same *LUBFR*.

Now that we have covered the library subroutines that operate on the BUFR file level, and recalling the BUFR file structure that was previously discussed, it is now time to continue on to the BUFR message level:

2nd level (message level): read in messages

```
msg_report: do while (ireadmg(unit_in,subset,idate) == 0)
          …
enddo msg_report
```

- Function ***ireadmg***:

```
IRET = IREADMG  (LUBFR, CSUBSET, IDATE)

Input argument:
   LUBFR      INTEGER      Logical unit for BUFR file

Output arguments:
   CSUBSET    CHAR*(*)     Table A mnemonic (name/type) for BUFR message
   IDATE      INTEGER      Section 1 date-time for BUFR message
   IRET       INTEGER      Return code:
                             0 = normal return
                            -1 = no more BUFR messages in LUBFR
```

Subroutine *IREADMG* reads the next BUFR message from the given BUFR file pointed to by *LUBFR,* returns *IRET* as its function value. It reads the next BUFR message into internal arrays within the BUFRLIB software (from where it can be easily manipulated or further parsed) rather than passed back to the application program directly. If the return code *IRET* contains the value -1, then there are no more BUFR messages within the given BUFR file, and the file will be automatically disconnected from the BUFRLIB software via an internal call to subroutine *CLOSBF*. Otherwise, if *IRET* returns with the value 0, then the character argument *CSUBSET* will contain the Table A mnemonic, which describes a type of data subset, and the integer argument *IDATE* will contain the date-time in format of YYMMDDHH or YYYYMMDDHH determined by subroutine *DATELEN*.

In this example, the loop *meg_report* will use *ireadmg* function to read all message in from the BUFR file until getting a none-zero return value (IRET=-1).

After *IREADMG* reads a BUFR message into the internal arrays, we can get into the 3rd level of the code to read a data subset from that internal message:

3rd level (subset level): read in data subsets

```
sb_report: do while (ireadsb(unit_in) == 0)
     …
enddo sb_report
```

- Function ***ireadsb***:

```
IRET = IREADSB  ( LUBFR )

Input argument:
   LUBFR      INTEGER       Logical unit for BUFR file
Output arguments:
   IRET       INTEGER       Return code:
                              0 = normal return
                             -1 = no more BUFR data subsets in
                                  current BUFR message
```

Function *IREADSB* reads a data subset from the internal arrays. A return code value of -1 within *IRET* indicates that there are no more data subsets within the given BUFR message.

Again, in this example, the loop *sb_report* will use *ireadsb* function to read all subset in from the internal array until getting a none-zero return value (IRET=-1).

Once a subset has been successfully read with IRET=0, then we are ready to call the data-level subroutines in order to retrieve actual data values from this subset:

4th level (data level): read in picked data values

This is the level where observation values are read into user-defined arrays. To understand how to read in observations from a BUFR subset, the following two questions need to be addressed:

1) How do I know what kind of data are included in the subset (or a BUFR file)?

This question can be answered by checking the content of a BUFR table and mnemonics. Chapter 3 of the BUFR User's Guide discusses the BUFR table and mnemonics in detail.

Here we illustrate how to use the BUFR table to solve the problem directly. As an example, an excerpt from the BUFR table in `sample.bufr` for the message type `ADPUPA` is shown below. We will use this table information to illustrate how to track observation variables in `ADPUPA` (the upper level data type):

```
|------------------------------------------------------------------------|
| MNEMONIC | NUMBER | DESCRIPTION                                         |
|----------|--------|-----------------------------------------------------|
| ADPUPA   | A48102 | UPPER-AIR (RAOB, PIBAL, RECCO, DROPS) REPORTS      |
| AIRCAR   | A48103 | MDCRS ACARS AIRCRAFT REPORTS                       |
```

```
| MNEMONIC | SEQUENCE                                                    |
|----------|-------------------------------------------------------------|
| ADPUPA   | HEADR  SIRC  {PRSLEVEL}  <SST_INFO>  <PREWXSEQ>  {CLOUDSEQ}  |
| ADPUPA   | <CLOU2SEQ>  <SWINDSEQ>  <AFIC_SEQ>  <TURB3SEQ>               |

| HEADR    | SID  XOB  YOB  DHR  ELV  TYP  T29  TSB  ITP  SQN  PROCN  RPT  |
| HEADR    | TCOR  <RSRD_SEQ>                                             |
```

```
|------------------------------------------------------------------------|
| MNEMONIC | NUMBER | DESCRIPTION                                         |
|----------|--------|-----------------------------------------------------|
| SID      | 001194 | STATION IDENTIFICATI
| XOB      | 006240 | LONGITUDE                                          |
| YOB      | 005002 | LATITUDE                                           |
| DHR      | 004215 | OBSERVATION TIME MINUS CYCLE TI                    |
| ELV      | 010199 | STATION ELEVATION                                  |
| TYP      | 055007 | PREPBUFR REPORT TYP                                |
```

```
|------------------------------------------------------------------------|
| MNEMONIC | SCAL | REFERENCE   | BIT | UNITS                |-------------|
|----------|------|-------------|-----|----------------------|-------------|
|          |      |             |     |                      |-------------|
| SID      |    0 |           0 |  64 | CCITT IA5            |-------------|
| XOB      |    2 |      -18000 |  16 | DEG E                |-------------|
| YOB      |    2 |       -9000 |  15 | DEG N                |-------------|
| DHR      |    3 |      -24000 |  16 | HOURS                |-------------|
| ELV      |    0 |       -1000 |  17 | METER                |-------------|
| TYP      |    0 |           0 |   9 | CODE TABL            |-------------|
```

The four color boxes here are used to separate the different parts of the BUFR table, which can also be marked as Part 1 (red), Part 2 (blue), Part 3 (yellow), and Part 4 (green) in the order they are listed above.

As discussed before, *IREADMG* reads in a message with three output arguments. The first output argument is:

```
CSUBSET      Table A mnemonic for BUFR message
```

It returns the message type (also called data type). This message type is the starting point to learn what types of observations are included in this message. The description of message types can be found in the first section of a BUFR table, that is the Part 1 (red) in the sample BUFR table.

Here, if `CSUBSET` has the value of `ADPUPA`, the contents of this message or all subsets (third level) are upper air reports (like rawinsonde). A search of `ADPUPA` in the BUFR table returns the first two lines of Part 2 (blue), in which `ADPUPA` is followed by a sequence of items like: `HEADR SIRC {PRSLEVEL}….` If we then search for `HEADR` in the same file, we can find the last two lines in Part 2 (blue), in which `HEADR` leads the sequence containing `SID XOB YOB DHR ELV TYP ….`

If we then search for `SID XOB YOB DHR ELV TYP` in the same file, we can find the definition of these items in Part 3 (yellow). Clearly, the message type `ADPUPA` includes variables like station ID, observation location (longitude, latitude), observation time, etc. These are important variables to describe an observation. If we keep searching for other items under `ADPUPA`, we can also find lots of observation variables are included in `ADPUPA`. Please note that a complete list of all variables in a message type could be very long and complex, but we don't need to learn about all of them - we only need to know what we need for our specific application.

The last part of the BUFR table (Part 4, green) includes useful unit information for a variable; for example, the unit of `XOB` is `DEG` (degree) and the unit of `DHR` is `HOURS` (hours). Users will not likely need to make use of the scale, reference, and bit information.

There are lots of other details on BUFR tables, but the above information should be sufficient for now to learn about BUFR file processing applications using the NCEP BUFRLIB software with the examples in this Chapter.

2). How Do I Tell BUFRLIB To Only Read In Specific Data Information?

From the BUFR table discussion above, we can see a message or a subset could include lots of information. In this example, we only wants to read in temperature observation, along with its longitude, latitude, and observation time. Here we will use this example to illustrate how to solve this question. From the BUFR table, for the message type `ADPUPA`, the name of longitude, latitude, and time in the BUFR table are `'XOB YOB DHR'` within the sequence `HEADER`. Similarly, the name of the temperature observation can be found as `'TOB'` in the sequence `{PRSLEVEL}` (not shown in the example BUFR table). Actually, most conventional message types contain such observation information.

In the example code, the first several lines define the information we want to read:

```
        character(80):: hdstr='XOB YOB DHR'
        character(80):: obstr='TOB'
        real(8) :: hdr(3),obs(1,10)
```

`hdstr` is a string of blank-separated names (mnemonics) associated with array `hdr`, while `obstr` is another string associated with array `obs`. Please note that arrays (`hdr and obs`) have to be defined as REAL*8 arrays. Now let's first learn the usage of subroutine *ufbint* which is called in the following two lines.

```
call ufbint(unit_in,hdr,3,1 ,iret,hdstr)
call ufbint(unit_in,obs,1,10,iret,obstr)
```

- ufbint

```
CALL UFBINT  ( LUBFR, R8ARR, MXMN, MXLV, NLV, CMNSTR )

Input arguments:
    LUBFR     INTEGER       Logical unit for BUFR file
    CMNSTR    CHAR*(*)      String of blank-separated mnemonics
                            associated with R8ARR
    MXMN      INTEGER       Size of first dimension of R8ARR
    MXLV      INTEGER       Size of second dimension of R8ARR
                            OR number of levels of data values
                            to be written to data subset

Input or output argument (depending on context of LUBFR):
    R8ARR(*,*) REAL*8       Data values written/read to/from
                            data subset

Output argument:
    NLV       INTEGER       Number of levels of data values
                            written/read to/from data subset
```

Subroutine UFBINT writes or reads specified values to or from the current BUFR data subset within the internal arrays, with the direction of the data transfer being determined by the context of *LUBFR*, if *LUBFR* points to a BUFR file that is open for input (i.e. reading/decoding BUFR), then data values are read from the internal data subset; otherwise, data values are written to the internal data subset. The actual data transfer occurs through the use of the two-dimensional REAL*8 array *R8ARR* whose actual first dimension *MXMN* must always be passed in. The call argument *MXLV*, on the other hand, contains the actual second dimension of *R8ARR* only when *LUBFR* points to a BUFR file that is open for input (i.e. reading/decoding BUFR); otherwise, whenever *LUBFR* points to a BUFR file that is open for output (i.e. writing/encoding BUFR), *MXLV* instead contains the actual number of levels of data values that are to be written to the data subset (and where this number must be less than or equal to the actual second dimension of *R8ARR*). In either case, the input character string *CMNSTR* always contains a blank-separated list of "mnemonics" which correspond to the REAL*8 values contained within the first dimension of *R8ARR*, and the output argument *NLV* always denotes the actual number of levels of those values that were written/read to/from the second dimension of *R8ARR*, where each such level represents a repetition of the mnemonics within *CMNSTR*. Note that, when *LUBFR* points to a BUFR file that is open for output (i.e. writing/encoding BUFR), we would certainly expect that the output value *NLV* is equal to the value of *MXLV* that was input, and indeed this is the case unless some type of error occurred in storing one or more of the data levels.

In this case, after we run the two BUFRLIB subroutines, longitude (`XOB`), latitude (`YOB`), and observation time (`DHR`) will be read into array `hdr` and temperature observations (`TOB`) is read into array `obs`. The array contents should be:

- hdr(1) - longitude
- hdr(2) - latitude
- hdr(3) - time
- obs(1,1) - temperature observation in 1st level (single level)
- obs(1,2) - temperature observation in 2nd level for multi-level observation
- obs(1,3) - temperature observation in 3rd level for multi-level observation
- ...

Because these two lines are inside the message and subset loops, we can get temperature observation with location and time from all observations in the BUFR file. If data subsets contain some missing data, the data values in the array are assigned as 10.0E10.

Now, only one BUFRLIB subroutine *datelen* left in the code needs to be explained:

- *datelen*:

```
CALL DATELEN  ( LEN )
Input argument:
    LEN      INTEGER      Length of Section 1 date-time values to
                          be output by message-reading subroutines
                          such as READMG, READERME, etc.
                            8 =   YYMMDDHH (i.e. 2-digit year)
                           10 = YYYYMMDDHH (i.e. 4-digit year)
```

This subroutine allows the user to specify the format for the `IDATE` output argument that is returned by `READMG`.

7.1.2.2 Encoding/Writing Data Into A Simple BUFR File

The following is from the program *bufr_encode_sample.f90*, which shows how to write a few observation variables into a new BUFR file.

```
program bufr_encode_sample
!
!  example of writing one value into a bufr file
!
 implicit none

 character(80):: hdstr='XOB YOB DHR'
 character(80):: obstr='TOB'
 real(8) :: hdr(3),obs(1,1)
 character(8) subset
 integer :: unit_out=10,unit_table=20
 integer :: idate,iret

! set data values
 hdr(1)=75.;hdr(2)=30.;hdr(3)=-0.1
 obs(1,1)=287.15
 idate=2008120100  ! YYYYMMDDHH
 subset='ADPUPA'   ! upper-air reports

! encode
 open(unit_table,file='table_prepbufr.txt')
 open(unit_out,file='sample.bufr',action='write' &
            ,form='unformatted')
 call datelen(10)
 call openbf(unit_out,'OUT',unit_table)
   call openmb(unit_out,subset,idate)
      call ufbint(unit_out,hdr,3,1,iret,hdstr)
      call ufbint(unit_out,obs,1,1,iret,obstr)
      call writsb(unit_out)
   call closmg(unit_out)
 call closbf(unit_out)

end program
```

Specifically, this example will write one temperature observation value with observation location and time to a BUFR file named as *sample.bufr*.

Here, we can see the BUFR encode procedure has the same structure as the decode procedure: file level, message level, subset level, which are marked in the same color as the decode example in Section 7.1.2.1. The major difference between encode and decode are highlighted in bold in the code and explained below:

- **`open(unit_table,file='table_prepbufr.txt')`**
  To encode some observation values into a new BUFR file, a pre-existing BUFR table file is necessary and needs to be opened.

- `open(unit_out,file='sample.bufr',action=`**`'write'`**`,form='unformatted')`
  The `action` in Fortran open command has to be "**write**".

- `call openbf(unit_out,`**`'OUT',unit_table`**`)`
  The second input parameter is set to "OUT" to access a new file for writing. The third parameter is the logical unit of BUFR table file so that BUFR table will be written into BUFR file. Please check the detailed explanation for *openbf* in section 7.1.2.1.

- **`call openmb(unit_out,subset,idate)`**

```
        CALL OPENMB  ( LUBFR, CSUBSET, IDATE )
```

```
Input arguments:
    LUBFR     INTEGER       Logical unit for BUFR file
    CSUBSET   CHAR*(*)      Table A mnemonic for type of BUFR
                            message to be opened
    IDATE     INTEGER       Date-time to be stored within
                            Section 1 of BUFR message
```

This function opens and initializes a new BUFR message for eventual output to *LUBFR*, using the arguments *CSUBSET* and *IDATE* to indicate the type and time of message to be encoded. It only opens a new message if either *CSUBSET* or *IDATE* has changed, and otherwise will simply return while leaving the existing internal message unchanged, so that subsequent data subsets can be stored within the same internal message. For this reason, *OPENMB* allows for the storage of an increased number of data subsets within each BUFR message and therefore improves overall encoding efficiency. Regardless, whenever a new BUFR message is opened and initialized, the existing internal BUFR message (if any) will be automatically closed and written to output via an internal call to the following subroutine:

- `call closmg(unit_out)`

```
CALL CLOSEMG ( LUBFR )
Input arguments:
    LUBFR     INTEGER       Logical unit for BUFR file
```

Furthermore, since, in the case of a BUFR file that was opened for input, each subsequent call to subroutine I*READMG* will likewise automatically clear an existing message from the internal arrays before reading in the new one, for this reason, it is rare to ever see subroutine *CLOSMG* called directly from within an application program!

- **`call writsb(unit_out)`**

```
CALL WRITSB  ( LUBFR )

Input argument:
    LUBFR     INTEGER       Logical unit for BUFR file
```

This subroutine is called to indicate to the BUFRLIB software that all necessary data values for this subset have been stored and thus that the subset is ready to be encoded and packed into the current message for the BUFR file associated with logical unit *LUBFR*. However, we should note that the BUFRLIB software will not allow any single BUFR message to grow larger than a certain size (usually 10000 bytes, although this can be increased via a call to subroutine *MAXOUT*);

Before this subroutine, we can see two consecutive calls to the subroutine `ufbint`, which is the same as in the decode example. However, this time, the strings `hdstr` tells the BUFR subroutine `ufbint` that the array `hdr` holds longitude, latitude and observation time, the string `obstr` tells `ufbint` that the array `obs` holds temperature observations. The data subset is ready and written into the BUFR file via call **writsb**.

7.1.2.3 Appending Data To A Simple BUFR File

The following is from the program *bufr_append_sample.f90*, which shows how to append a new observation variable into an existing BUFR file.

```
program
! sample of appending one observation into bufr file
 implicit none

 character(80):: hdstr='XOB YOB DHR'
 character(80):: obstr='TOB'
 real(8) :: hdr(3),obs(1,1)

 character(8) subset
 integer :: unit_out=10,unit_table=20
 integer :: idate,iret

! set data values
 hdr(1)=85.0;hdr(2)=50.0;hdr(3)=0.2
 obs(1,1)=300.0
 idate=2008120101  ! YYYYMMDDHH
 subset='ADPSFC'   ! surface land reports

! get bufr table from existing bufr file
 open(unit_table,file='table_prepbufr_app.txt')
 open(unit_out,file='sample.bufr',status='old',form='unformatted')
 call openbf(unit_out,'IN',unit_out)
 call dxdump(unit_out,unit_table)
 call closbf(unit_out)

! append
 open(unit_out,file='sample.bufr',status='old',form='unformatted')
 call datelen(10)
 call openbf(unit_out,'APN',unit_table)
   call openmb(unit_out,subset,idate)
      call ufbint(unit_out,hdr,3,1,iret,hdstr)
      call ufbint(unit_out,obs,1,1,iret,obstr)
      call writsb(unit_out)
   call closmg(unit_out)
 call closbf(unit_out)

end program
```

Specifically, this example will append one temperature observation value with observation location and time to an existing BUFR file named as *sample.bufr*.

If we compare this code with the example code for encoding, we can find the code structure and BUFRLIB functions used are very similar in two codes. But there is a key point that needs special attention for appending:

- <u>Appending has to use the exact same BUFR table as the existing BUFR file.</u>
  To ensure this, we add the following three lines to the code in order to extract the BUFR table from the existing BUFR file:

```
call openbf(unit_out,'IN',unit_out)
call dxdump(unit_out,unit_table)
call closbf(unit_out)
```

  Let's learn subroutine dxdump.

```
CALL DXDUMP  ( LUBFR, LDXOT )

Input arguments:
    LUBFR INTEGER        Logical unit for BUFR file
    LDXOT INTEGER        Logical unit for output BUFR tables file
```

  This subroutine provides a handy way to view the BUFR table information that is embedded in the first few messages of a BUFR file. The user needs only to have identified the file to the BUFRLIB software via a prior call to subroutine *OPENBF*, and then a subsequent call to subroutine *DXDUMP* will unpack the embedded tables information and write it out to the file pointed to by logical unit *LDXOT*. The output file is written with ASCII-text table format. Subroutine *DXDUMP* can be most useful for learning the contents of archive BUFR files.

In this example, the BUFR table embedded in the BUFR file *sample.bufr* will be read in and written into a text file called *table_prepbufr_app.txt.*

Comparing with the encode example again, there are two more slight differences in setups, which are highlighted in the code as Bold and explained below:

- In the Fortran open command, the status has to be set as **'old'** because appending requires an existing BUFR file.
- In the subroutine *openbf*, the existing BUFR file and dumped BUFR table are connected to BUFRLIB, the second input parameter has to be set as '**APN**'.

7.1.3 Encode, Decode, Append The PrepBUFR File

In last section, we use three simplified examples to illustrate the code structure of the BUFR file process (read, write and append) and explained commonly used BUFRLIB functions in the example code. In this section, we will learn how to use the skills we learned in previous sections to process a PrepBUFR file, which is one of major BUFR files used in GSI for all conventional observations and retrieved standard observations.

7.1.3.1 Decoding/Reading Data From A PrepBUFR File

The following is from the code *prepbufr_decode_all.f90*, which reads all major conventional observations and BUFR table out from a PrepBUFR file.

```
program prepbufr_decode_all
!
! read all observations out from prepbufr.
! read bufr table from prepbufr file
!
 implicit none

 integer, parameter :: mxmn=35, mxlv=250
 character(80):: hdstr='SID XOB YOB DHR TYP ELV SAID T29'
 character(80):: obstr='POB QOB TOB ZOB UOB VOB PWO CAT PRSS'
 character(80):: qcstr='PQM QQM TQM ZQM WQM NUL PWQ     '
 character(80):: oestr='POE QOE TOE NUL WOE NUL PWE     '
```

Compared to the mnemonic list used in the examples in 7.1.2.1, a clear difference here is that more BUFR table mnemonics are involved because we want to read all major observations, such as temperature (TOB), moisture (QOB), Pressure(POB), Height (ZOB), wind (UOB and VOB). Also, we want to read the quality flags and observation errors with these observations at the same time. Here is a list of content in these mnemonics strings:

- hdstr: defines report header information including the station ID, longitude, latitude, time, report type, elevation, satellite ID, data dump report type.
- `obstr`: defines observation for pressure, specific humidity, temperature, height, u and v component of wind, total precipitable water, data level category, surface pressure.
- `qcstr`: defines the quality markers for each of observation variables listed in the string `obstr`.
- `oestr`: defines the observation error for each of observation variables listed in the string `obstr`.

More detailed information on these mnemonics can be found from the BUFR table named with "*prepobs_prep.bufrtable*", which is a text file dumped out during the decoding process.

```
      real(8) :: hdr(mxmn),obs(mxmn,mxlv),qcf(mxmn,mxlv),oer(mxmn,mxlv)
```

The associated arrays are defined to hold the data values of mnemonics specified in `hdstr`, `obstr, qcstr, oestr`. Note, `mxmn`=35, `mxlv`=250, which make the array can hold up to 250 levels of observations with up to 35 mnemonics in each level.

```
       INTEGER        :: ireadmg,ireadsb
       character(8)   :: subset
       integer        :: unit_in=10,unit_table=24,idate,nmsg,ntb


       character(8)   :: c_sid
       real(8)        :: rstation_id
       equivalence(rstation_id,c_sid)
```

From our earlier discussions, it was noted that data values are normally read from or written to BUFR subsets using REAL*8 arrays via subroutine. The character values are read and written in the same way using a REAL*8 variable. Here, `rstation_id` is real(8); `c_sid` is character(8); then FORTRAN EQUIVALENCE is used to covert the station ID from REAL*8 to string that can be easily read by humans.

```
      integer        :: i,k,iret
```

```
      open(unit_table,file='prepobs_prep.bufrtable')
```
Fortran open command to link BUFR table with a logical unit, `unit_table`.

```
      open(unit_in,file='prepbufr',form='unformatted',status='old')
```
Fortran open command to link a PrepBUFR file with a logical unit, `unit_in`.

```
      call openbf(unit_in,'IN',unit_in)
```
Connect the PrepBUFR file to BUFRLIB. Since BUFR table is embedded in the PrepBUFR file, the third argument is the same as first argument in this call.

```
      call dxdump(unit_in,unit_table)
```
Dump BUFR table out from the existing PrepBUFR file and write to a ASCII file named "*prepobs_prep.bufrtable*" through unit `unit_table`.

```
      call datelen(10)
```
Specifies the date format as YYYYMMDDHH.

```
      nmsg=0
         msg_report: do while (ireadmg(unit_in,subset,idate) == 0)
           nmsg=nmsg+1
            ntb = 0
           write(*,*)
           write(*,'(3a,i10)') 'subset=',subset,' cycle time =',idate
           sb_report: do while (ireadsb(unit_in) == 0)
```

The `msg_report` loop reads each of messages until reaching the end of file. The `sb_report` loop reads each of data subsets within the current message until the end of the message.

```
             ntb = ntb+1
             call ufbint(unit_in,hdr,mxmn,1   ,iret,hdstr)
             call ufbint(unit_in,obs,mxmn,mxlv,iret,obstr)
             call ufbint(unit_in,oer,mxmn,mxlv,iret,oestr)
             call ufbint(unit_in,qcf,mxmn,mxlv,iret,qcstr)
```

Calling subroutine *ufbint* to read data based on mnemonics defined in `hdstr, obstr, oestr, qcstr` from a subset and write to corresponding arrays `hdr,obs, oer, qcf`. The `iret` is the actual returned number of pressure levels which have be read in even though `mxlv`=250.

```
             rstation_id=hdr(1)
             write(*,*)
             write(*,'(2I10,a14,8f14.1)') ntb,iret,c_sid,(hdr(i),i=2,8)

            DO k=1,iret
               write(*,'(i3,a10,9f14.1)') k,'obs=',(obs(i,k),i=1,9)
               write(*,'(i3,a10,9f14.1)') k,'oer=',(oer(i,k),i=1,7)
               write(*,'(i3,a10,9f14.1)') k,'qcf=',(qcf(i,k),i=1,7)
             ENDDO

           enddo sb_report
         enddo msg_report

      call closbf(unit_in)
      end program
```

From this PrepBUFR decoding example, we can see that the code structure and functions used are the same as the simple decoding example in section 7.1.2.1. But this example defines more mnemonics and larger dimensions of the REAL*8 arrays to read all major observation elements from the PrepBUFR file, including observation values, quality markers, and observation errors.

7.1.3.2 More Examples On Processing Prepbufr Files

In BUFR/PrepBUFR User's Guider, there are more examples on how to processing the PrepBUFR files used by GSI. Please read that document if needed:

prepbufr_encode_surface.f90:	Write a surface observation into a PrepBUFR file.
prepbufr_encode_upperair.f90:	Write an upper air observation into the PrepBUFR file.
prepbufr_append_surface.f90:	Append a surface observation into an existing

	PrepBUFR file.
prepbufr_append_upperair.f90:	Append an upper air observation into an existing PrepBUFR file.
prepbufr_append_retrieve.f90:	Append a retrieved data into an existing PrepBUFR file.
bufr_decode_radiance.f90:	Read TOVS 1B radiance observations and BUFR table out from the radiance BUFR file.

7.3 GSI BUFR Interface

GSI has a set of code to ingest and process observation data from BUFR/PrepBUFR files for the analysis. This section will first explain the procedure of observation ingest and process within the GSI system. Then, we provide 4 examples from GSI observation ingesting subroutines to illustrate how GSI interfaces with the BUFR files.

7.3. 1 GSI Observation Data Ingest And Process Procedure

As an important component of any data analysis system, observation data ingesting and processing is a key part of the GSI system. The data types that can be used in the GSI analysis and the corresponding subroutines that read in these data types are listed in the section 5.3 of the Advanced GSI User's Guide. But there are more details that users should know to be able to handle the observation data in GSI with confidence and flexibility. This section introduces the complete structure of GSI observation data ingesting and processing step-by step, including run scripts and namelist setup, data ingesting driver routine, read subroutines, observation data partition, and innovation calculation.

- *Step 1: Link BUFR/PrepBUFR file to GSI recognized names in GSI run scripts*

In the GSI run script, there is a section to link the BUFR/PrepBUFR files to GSI recognized file names in the GSI run directory. The script looks like:

```
# Link to the prepbufr data
ln -s ${PREPBUFR} ./prepbufr

# Link to the radiance data
# ln -s ${OBS_ROOT}/gdas1.t12z.1bamua.tm00.bufr_d amsuabufr
# ln -s ${OBS_ROOT}/gdas1.t12z.1bhrs4.tm00.bufr_d hirs4bufr
# ln -s ${OBS_ROOT}/gdas1.t12z.1bmhs.tm00.bufr_d  mhsbufr
```

Clearly, the PrepBUFR file: *gdas1.t12z.prepbufr.nr*, which is the file pointed by *${PREPBUFR},* and the BUFR files: *gdas1.t12z.1bamua.tm00.bufr_d* and *gdas1.t12z.1bhrs4.tm00.bufr_d* are the files we downloaded from NCEP data hub. The names of these files are determined by NCEP based on the operation systems that use the files. The BUFR files used in GSI can also be the observation files generated by users and named by users. But GSI itself doesn't recognize the names of these files. So, in the GSI run scripts, these files must be linked to the GSI run directory with a name that GSI knows.

In the section 3.1 of the GSI User's Guider has a table that lists all the GSI recognized data file names at the left column, the contents of the data files at the middle column, and the sample GDAS BUFR/PrepBUFR file names at the left column. The following is a sample of the table.

GSI Name	Content	Example file names
prepbufr	Conventional observations, including ps, t, q, pw, uv, spd, dw, sst, from observation platforms such as METAR, sounding, et al.	gdas1.t12z.prepbufr
amsuabufr	AMSU-A 1b radiance (brightness temperatures) from satellites NOAA-15, 16, 17,18, 19 and METOP-A	gdas1.t12z.1bamua.tm00.bufr_d
amsubbufr	AMSU-B 1b radiance (brightness temperatures) from satellites NOAA15, 16,17	gdas1.t12z.1bamub.tm00.bufr_d
radarbufr	Radar radial velocity Level 2.5 data	ndas.t12z. radwnd. tm12.bufr_d
gpsrobufr	GPS radio occultation observation	gdas1.t12z.gpsro.tm00.bufr_d
ssmirrbufr	Precipitation rate observations fromSSM/I	gdas1.t12z.spssmi.tm00.bufr_d
hirs4bufr	HIRS4 1b radiance observation from satellite NOAA 18, 19 and METOP-A	gdas1.t12z.1bhrs4.tm00.bufr_d
msubufr	MSU observation from satgellite NOAA 14	gdas1.t12z.1bmsu.tm00.bufr_d

So, in the GSI run script, the files in the right column are linked to the run directory with a new name at the left column. As a matter of fact, the file names in the left column can be changed if users prefer to do so and know how to change them in the GSI namelist data file setup section. But we recommend to leave the file names as is because the current names in the left column are a good indication of the contents of the corresponding BUFR observation files and are used by many the GSI applications.

- *Step 2: GSI Namelist data configuration section:* ***&OBS_INPUT***

In the GSI namelist, section *&OBS_INOUT* is used to setup data usage such as the links between data types and data files, data time window, and satellite data thinning. The following is a sample of the namelist section *&OBS_INOUT*:

```
&OBS_INPUT
 dmesh(1)=120.0,dmesh(2)=60.0,dmesh(3)=60.0,dmesh(4)=60.0,dmesh(5)=120,time_window_max=1.5,
 dfile(01)='prepbufr',  dtype(01)='ps',    dplat(01)=' ',   dsis(01)='ps',        dval(01)=1.0,  dthin(01)=0,
 dfile(02)='prepbufr'   dtype(02)='t',     dplat(02)=' ',   dsis(02)='t',         dval(02)=1.0,  dthin(02)=0,
 dfile(03)='prepbufr',  dtype(03)='q',     dplat(03)=' ',   dsis(03)='q',         dval(03)=1.0,  dthin(03)=0,
 dfile(04)='prepbufr',  dtype(04)='uv',    dplat(04)=' ',   dsis(04)='uv',        dval(04)=1.0,  dthin(04)=0,
……
 dfile(27)='msubufr',   dtype(27)='msu',   dplat(27)='n14', dsis(27)='msu_n14',   dval(27)=2.0,  dthin(27)=2,
 dfile(28)='amsuabufr', dtype(28)='amsua', dplat(28)='n15', dsis(28)='amsua_n15', dval(28)=10.0, dthin(28)=2,
 dfile(29)='amsuabufr', dtype(29)='amsua', dplat(29)='n16', dsis(29)='amsua_n16', dval(29)=0.0,  dthin(29)=2,
```

Users may notice that the first column, *dfile*, is the GSI recognized file names listed in the section 3.1 of the GSI User's Guider . The 2nd column, *dtype*, is the observation type. The 3rd column, *dplat*, is satellite platform ID. And the 4th column, *dsis*, is the data type from convinfo file or Sensor/instrument/satellite flag from satinfo file.

In the GSI data ingesting driver, it is the data type, *dtype*, that is used to decide which routine to call for reading the data from the corresponding input file defined by *dfile*. For example, when the GSI reaches the code to read "`t`", it will open file `'prepbufr'` (`dfile(02)`) to read temperature in. Or when the GSI reaches the point to read in AMSU-A from NOAA 16, it will open file `'amsuabufr'` (`dfile(29)`) to read in the data. From the namelist setup, it is possible that GSI reads in "*t*" from one PrepBUFR file (`dfile(02)`) but reads in '*q*' from another PrepBUFR file (`dfile(03)`), which gives more flexibility to control the data used in the GSI analysis.

- *Step 3: GSI data ingest driver*

In GSI, subroutine *read_obs* (inside file *read_obs.F90*) is used to read, select, and reformat observation data. It is the driver for routines that read different types of the observational data. This routine loops through all data types listed in *dtype* and checks the data usage and file availability. If the data file exists and the info files indicate the use of the data type, one or several processors will be assigned to read the data from the corresponding file setup in `dfile`. Please refer to the section 4.3 of the GSI User's Guide for more information on using the info file to control data usage. Here we give two chunks of the code from subroutine read_obs as examples to illustrate how to find routines that read different observation data types.

<u>Example 1: Process conventional (prepbufr) data</u>

```
!
  if(ditype(i) == 'conv')then
     if (obstype == 't'  .or. obstype == 'uv' .or. &
         obstype == 'q'  .or. obstype == 'ps' .or. &
         obstype == 'pw' .or. obstype == 'spd'.or. &
         obstype == 'mta_cld' .or. obstype == 'gos_ctp'  ) then
         call read_prepbufr(nread,npuse,nouse,infile,obstype,lunout,twind,sis,&
             prsl_full)
         string='READ_PREPBUFR'
```

From this chunk of the code, we can see the subroutine *read_prepbufr* will be used to read the data type *'t', 'uv', 'q', 'ps', 'pw', 'spd', 'mta_cld', 'gos_ctp'* from PrepBUFR file saved in "*infile*".

<u>Example 2: Process TOVS 1b data</u>

```
!
      if (platid /= 'aqua' .and. (obstype == 'amsua' .or. &
           obstype == 'amsub' .or. obstype == 'msu'   .or.  &
           obstype == 'mhs'   .or. obstype == 'hirs4' .or.  &
           obstype == 'hirs3' .or. obstype == 'hirs2' .or.  &
           obstype == 'ssu')) then
         llb=1
         lll=1
         if((obstype == 'amsua' .or. obstype == 'amsub' .or. obstype == 'mhs') .and. &
            (platid /= 'metop-a' .or. platid /='metop-b' .or. platid /= 'metop-c'))lll=2
         call read_bufrtovs(mype,val_dat,ithin,isfcalc,rmesh,platid,gstime,&
              infile,lunout,obstype,nread,npuse,nouse,twind,sis, &
              mype_root,mype_sub(mm1,i),npe_sub(i),mpi_comm_sub(i),llb,lll)
         string='READ_BUFRTOVS'
```

From this chunk of the code, we can see the subroutine *read_bufrtovs* will be used to read many kinds of radiance data such as *'amsua', 'amsub', 'msu', 'mhs', 'hirs', 'ssu'* from radiance BUFR file saved in "*infile*". But these radiance data are not observed by AQUA.

In the subroutine *read_obs*, users can find similar portion of the code deciding which subroutine is used to read in the data for certain data type. For each subroutine, the input variables always includes parameters like:

infile = *dfile* of the namelist section *&OBS_INOUT*
obstype = *dtype* of the namelist section *&OBS_INOUT*
sis = *dsis* of the namelist section *&OBS_INOUT*

- *Step 4: Read in observations and initial check of the observations*

The data types and the corresponding GSI subroutines that read in these data types are listed in the table of section 5.3. From the table, we can see there are 28 subroutines employed by GSI to read in different kinds of BUFR/PrepBUFR files. Also from the table, we can easily find the GSI subroutine that actually reads in the certain observations from the BUFR/PrepBUFR files. The same subroutines also do the quality control to the observation data, data thinning, and checks to insure that the data are in the analysis domain and time window.

These read_* subroutines listed in the table of section 5.3 are the GSI interface to the BUFR/PrepBUFR that users should check when trying to analyze their own data using the GSI system. We will discuss how to check the structure of these read_* subroutines in section 7.3.2 of this Chapter.

After we read in the observations for each element, such as "*t*", "*q*", "*wind*", GSI will write out observations for certain element in the analysis domain and time to one binary file, which will be read in again by the next step for data partitioning into sub-domains (if run with multiple processors).

- *Step 5: sub-domain partition*

When GSI runs in parallel mode, both the background and the observation data need to be partitioned into sub-domains. This step is done after the observation data have been read in and saved in the internal format. The code to assign and distribute observations to sub-domains is call "*obs_para*", which is a subroutine inside the file "*obs_para.f90*". Please note that after this step, the observations from all observation elements are saved in the same binary file for each processor.

- *Step 6: innovation calculation*

As an important step of the data analysis system, observation innovation calculation also involves lots of code. The section 5.4 of the Advanced GSI User's Guide provides a table to list innovation calculations for the different kinds of observation elements. We will not introduce these calculations in this document but would like to remind users that innovation calculation is also a key component in the use of observation data in the analysis.

7.3.2 The BUFR Decoding In GSI Read Files

From the previous section, we can see that there are many steps involved in the GSI system to ingest and process the observation data from BUFR/PrepBUFR files for the final analysis. To encode new data for the GSI, the best way to start is reading the related GSI code for BUFR/PrepBUFR data ingesting and checking the mnemonics used in the code to figure out the data needed in the GSI. In the table of section 5.3, we have provided a complete list of GSI subroutines for the observation data ingesting. Here we will give 2 examples to illustrate how to extract the GSI BUFR interface from the GSI read_* subroutines and delete other functions that are not related to the BUFR decoding from the subroutine, such as observation location and time checking, data thinning, and quality control checking, etc.

Example 1: read_prebufr.f90

The file read_prepbufr.f90 is in GSI source code directory (./src/main) and it reads conventional data from the PrepBUFR file. Specific observation types read by this routine include surface pressure, temperature, winds (components and speeds), moisture, total precipitable water, and cloud and weather. This file has over one thousand lines and most of the code are not related to the PrepBUFR decoding. Here, as an example, we deleted all the code that are not for PrepBUFR decoding and shortened the file down to 197 lines. The full code is listed in the Appendix B and can be downloaded from the Examples Page of the BUFR website. Here we will only show the mnemonics used by the GSI PrepBUFR decoding to get an idea what are the GSI expected variables from the PrepBUFR file.

```
data hdstr  /'SID XOB YOB DHR TYP ELV SAID T29'/
data hdstr2 /'TYP SAID T29 SID'/
data obstr  /'POB QOB TOB ZOB UOB VOB PWO CAT PRSS' /
data drift  /'XDR YDR HRDR                    '/
data sststr /'MSST DBSS SST1 SSTQM SSTOE          '/
data qcstr  /'PQM QQM TQM ZQM WQM NUL PWQ     '/
data oestr  /'POE QOE TOE NUL WOE NUL PWE     '/
data satqcstr  /'QIFN'/
data prvstr /'PRVSTG'/
data sprvstr /'SPRVSTG'/
data levstr  /'POB'/
data metarcldstr /'CLAM HOCB'/      ! cloud amount and cloud base height
data metarwthstr /'PRWE'/           ! present weather
data metarvisstr /'HOVI'/           ! visibility
data geoscldstr /'CDTP TOCC GCDTT CDTP_QM'/
```

Compared to the PrepBUFR processing examples we provided, we can see that there is more information expected by the GSI PrepBUFR interface. Please note that not all the variables listed in the above mnemonics are needed for a GSI run. Some are for certain special GSI applications only, such as the cloud observations, which are used in the Rapid Refresh system only. So, if users only want to generate a PrepBUFR file that contains a part of the observations expected by these mnemonics, the GSI still can run successfully and use the observation data to get a final analysis. But from the previous introduction to the GSI observation data processing procedure, users can see that there are many steps involved in the data usage in the GSI analysis. A complete picture of the data flow in GSI system will be very helpful for users who work on data impact studies with GSI, especially when they need to generate the new PrepBUFR file for their new data.

Example 2: read_airs.f90:

The file read_airs.f90 is in GSI source code directory (./src/main) and it reads BUFR format AQUA radiance (brightness temperature) observations. This file has 768 lines. To simplify this example, we deleted all the code that is not related to the BUFR decoding and shortened the file down to 82 lines. The full code is listed in the Appendix and can be download from the Examples Page of the BUFR website. Again, we will only show the lines that include mnemonics used by decoding to get an idea what variables are expected by GSI from the AIRS BUFR file.

```
allspotlist='SIID YEAR MNTH DAYS HOUR MINU SECO CLATH CLONH SAZA BEARAZ FOVN'

 call ufbrep(lnbufr,allchan,1,n_totchan,iret,'TMBR')

 call ufbint(lnbufr,aquaspot,2,1,iret,'SOZA SOLAZI')
```

Here, we highlight the mnemonics and we will leave then for users to find out the exactly meaning of these mnemonics by checking the BUFR table.

Summary:

In the course of preparing this document and extending the BUFR/PrepBUFR support for GSI, we outline portions of 4 GSI BUFR ingest interface files for users to reference:

read_prepbufr.f90
read_airs.f90
read_bufrtovs.f90
read_gps.f90

Users can find these files in the Examples Page of the BUFR user's website. There is makefile provided with these files to help users properly compile the code. These files can also be used to decode the corresponding NCEP operation PrepBUFR/BUFR files.

7.4 NCEP Generated BUFR Files

7.4.1 Knowledge on NCEP BUFR/PrepBUFR Files

NCEP saves most of the observation data in WMO BUFR format. PrepBUFR is the final step in preparing most of the observations for data assimilation, the NCEP term for "prepared" or QC'd data in BUFR format (NCEP convention/standard). Please note that a PrepBUFR file is still a BUFR file, but has more QC information. NCEP uses PrepBUFR files to organize conventional observations and satellite retrievals as well as other related information (such as quality marks) into single files. The BUFRLIB software and BUFR table are needed for processing BUFR/PrepBUFR files.

NCEP generates different BUFR/PrepBUFR files for each of its operation systems. The "PrepBUFR" includes the major conventional observations for assimilation into the various NCEP analyses, including the North American Model (NAM) and NAM Data Assimilation System (NDAS), unified grid-point statistical interpolation analysis (GSI) (the "NAM" and "NDAS" networks), the Global Forecast System and Global Data Assimilation System unified GSI (the "GFS" and "GDAS" networks), the Climate Data Assimilation System SSI (the "CDAS" network), the Rapid Update Cycle (the "RUC" network) and the Real Time Mesoscale Analysis (the "RTMA" network).

In this section, we will briefly introduce several types of BUFR/ PrepBUFR files mostly accessed by the research community to help users decide which one is the best for their GSI applications. Each type of BUFR/PrepBUFR file has its own coverage, data cut-off time, and quality control procedures, which result in different quality marker values for the same observation in different files.

- File name convention:

  The following is a list of example file names we collected from NCEP FTP site:

  gdas1.t00z.prepbufr.nr
  gfs.t00z.gpsro.tm00.bufr_d
  ndas.t18z.1bamub.tm03.bufr_d
  nam.t00z.aircar.tm00.bufr_d.nr
  ndas.t18z.prepbufr.tm03.nr

  These file names reflect information of the observations within the file. Let us explain the meaning of the filenames, segment by segment, separated by dots:

  - The 1st section is the operation system name, indicating which operation system this file is created/used for. For example: gdas1 is for the Global Data Assimilation System (GDAS), gfs for the Global Forecast System (GFS), ndas for the North American Data Assimilation System (NDAS), nam for the North American Mesoscale (NAM) forecast system.
  - The second section is analysis hour, indicating which analysis hour this file is used for. For example: t00z is for 00Z analysis, t18z for 18Z analysis.
  - The third section is data type, indicating what kinds of data are included in the file. For example: prepbufr is for conventional observations, gpsro for GSPRO, 1bamub for AMSU-B, and aircar for aircraft observations.

- From fourth section, there is different information for different operational files:
    - bufr_d tells us it is a BUFR format file. We may think prepbufr as a special data “format” here.
    - nr tells us that the file only includes non-restricted data (we can only access non-restricted data).
    - tm00 and tm03, where the two digital number is hours. They also indicate the time of the file used in the analysis. When the number is 00, the file analysis time is the same as showed in the second segment. When it is a number larger than 0, it indicates the analysis time of the file is the time in the second segment minus this number. For example: the analysis time for ndas.t18z.1bamub.tm03.bufr_d is 15Z (18Z - 03h = 15Z). This file is used in the catch up cycles during NDAS that have a delayed analysis start time to wait for more observations.

- Data coverage and cut off time:

  Each operational system requires different data types, data coverage, cut off time, and quality control procedures. The details of these setups need a long technical note to describe but here we can briefly introduce some major features of each file:

    - GDAS (gdas1) covers the global and has the latest cut off time (6 hours), which means it includes most of the available real-time observation data.
    - GFS (gfs) covers the global but has a shorter cut off time (2:45 hours) compared to GDAS.
    - NDAS(ndas) covers the North America and has a longer cut off time than NAM, which means it includes more real-time data than NAM.
    - NAM(nam) covers the North America but has a shorter cut off time comparing to others.
    - Data quality control processes for PrepBUFR files in each observation system are different but their results are reflected as quality markers, which can be easily checked by decoding the specific PrepBUFR file.
    - For data types in each PrepBUFR file, please check the following section.

- Code table for PrepBUFR report types

  The complete list of the conventional observation types (and their BUFR codes) used by each NCEP operation system are documented at the following links:

  Global GFS and GDAS GSI analyses:

http://www.emc.ncep.noaa.gov/mmb/data_processing/prepbufr.doc/table_2.htm

Global CDAS/reanalysis systems:

http://www.emc.ncep.noaa.gov/mmb/data_processing/prepbufr.doc/table_3.htm

Regional NAM and NDAS GSI analyses:

http://www.emc.ncep.noaa.gov/mmb/data_processing/prepbufr.doc/table_4.htm

Rapid Update Cycle (RUC) 3DVAR analysis:

http://www.emc.ncep.noaa.gov/mmb/data_processing/prepbufr.doc/table_5.htm

Here we give a simplified table for the most commonly used data types:

7.4.2 BUFR/PrepBUFR Data Resources for Community Users

There are several sources to get real-time and archived atmospheric observations and model forecasts. Some of them provide NCEP operation BUFR/PrepBUFR files for community. Below is a list we are aware of. Users are welcome to send us new data source links to share with the community.

Data in BUFR format

- NCEP NOMADS Site:
  - PrepBufr for GDAS (Global) - 1 month buffer:
    http://nomads.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/
  - PrepBufr for NDAS (North America) - 1 month buffer:
    http://nomads.ncep.noaa.gov/pub/data/nccf/com/nam/prod/

- NCEP FTP Site:
  - PrepBufr for GDAS (Global) - 3 day buffer:
    ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/
  - PrepBufr for NDAS (North America) - 3 day buffer:
    ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/nam/prod/

- NCDC NOMADS Site:
  - PrepBufr for GDAS (Global) - archive starting May 2007:
    http://nomads.ncdc.noaa.gov/data/gdas/

- NCAR Computational and Information Systems Laboratory (CISL) Research Data Archive (RDA) Site:
    - **DS337.0**: NCEP ADP Global Upper Air and Surface Observations (PREPBUFR and NetCDF PB2NC Output) - archive starting May 1997:
      http://dss.ucar.edu/datasets/ds337.0/
    - **DS337.0 Subset**: Interactive tool for running PB2NC over a specified time period and geographic region:
      http://dss.ucar.edu/datasets/ds337.0/forms/337_subset.php

7.5 Observation Error Adjustment

The actual observation errors used in GSI analysis start with the “external” (either from PrepBUFR files or an error table file) observation errors in the *obserr* array and go through multiple adjustments based on observation quality, vertical sigma location, observation density, time of the observations, etc. The major adjustments occur in *read_prepbufr.f90* and some are listed as follows:

1. Observation errors for each variable are bonded by their corresponding lower limits. Currently, these lower limits are hard coded and prescribed in *read_prepbufr*. The observation error limits for temperature, moisture, wind, surface pressure and total precipitable water are: *terrmin=0.5, qerrmin*=0.1*, werrmin*=1.0*, perrmin*=0.5*, pwerrmin*=1.0, respectively.
2. Observation errors are adjusted based on the quality markers from the prepbufr data files. If the quality markers from prepbufr are larger than a threshold value (*lim_qm* in *read_prepbuf.f90r*), the corresponding observation errors are adjusted to a very large number ($1.0x10^6$, which indicates a bad observation and will not make any impact on the analysis results). If the quality markers are smaller than *lim_qm*, the observation errors are adjusted based on the vertical location and vertical distribution of the observations. Please refer to the BUFR/PrepBUFR User’s Guide for more details on the quality markers and the values of *lim_qm*.
3. If an observation quality marker is either 3 or 7, the observation error can be inflated by setting *inflate_error* as true. The value of the inflation factor may be set based on observation types. However, currently it is fixed as 1.2.
4. For certain observation types (e.g., T), their observation errors are amplified by a factor of 1.2 if the observation locations are above 100 hPa.

Besides the above-mentioned adjustments, observation errors are further inflated during the observation innovation calculation (e.g., in the subroutines listed in section 3.2.4 of the Advanced User’s Guide) when the observation is located either lower than the lowest analysis level or higher than the highest analysis level. In the same routine, GSI performs gross error checks and, if *oberror_tune* is set to true, observation error tuning (this function is not discussed in this document).

Chapter 8: Satellite Radiance Data Assimilation

Satellite radiance data analysis is one of the most advanced and important features in the GSI system. GSI has developed complex functions and code components to ingest, analyze, bias correct, and monitor radiance observations from various satellite instruments. In this chapter, we will discuss these satellite radiance analysis related aspects from the users point of view, including how to correctly setup and run GSI with radiance observations, how to check and understand the radiance analysis results, bias correction, and monitoring radiance observations. Related code structure will also be described to help advanced users to further investigate and apply radiance data analysis with GSI.

8.1. Satellite Radiance Data Ingest And Distribution

8.1.1 Link Radiance BUFR Files To GSI Recognized Names

All radiance observations used by the GSI are saved in the BUFR format. For detailed information on the BUFR format and its processing techniques, please see the BUFR/PrepBUFR User's Guide, which is available on line:

http://www.dtcenter.org/com-GSI/BUFR/docs/index.php

In the Section 3.1 of this user's guide, we introduced all GSI BUFR/PrepBUFR observation files and the GSI recognized observation file names in table 3.1. From this table, we can see most of the BUFR files are used for satellite radiance data. Here, we will use a small part of the table to explain the link between the GSI name and the file name:

GSI Name	Content	Example file names
amsuabufr	AMSU-A 1b radiance (brightness temperatures) from satellites NOAA-15, 16, 17,18, 19 and METOP-A	gdas1.t12z.1bamua.tm00.bufr_d
amsubbufr	AMSU-B 1b radiance (brightness temperatures) from satellites NOAA15, 16,17	gdas1.t12z.1bamub.tm00.bufr_d

The right column of the table gives example radiance BUFR files that can be downloaded from the NCEP data servers (please see BUFR/PrepBUFR user's guide for the naming convention for these files), while the left column is the data file name that GSI expects during observation data ingestion. The middle column is a brief explanation of the data content in each file.

As explained in section 5.2.1, running radiance data analysis with GSI could be as simple as linking the radiance BUFR files to the GSI run directory with the GSI recognized name in the run script. For example, if we add the following two lines to the GSI run script:

```
# Link to the radiance data
ln -s ${OBS_ROOT}/gdas1.t12z.1bamua.tm00.bufr_d amsuabufr
ln -s ${OBS_ROOT}/gdas1.t12z.1bamub.tm00.bufr_d amsubbufr
```

we should see that AMSU-A and AMSU-B observations are analyzed in the GSI analysis, as illustrated in the rest of Section 5.2. Here, we will give more detail on the setup and usage of the GSI recognized observation file name (GSI name) in the left column of the table 3.1.

The GSI names, `amsuabufr` and `amsubbufr`, are actually decided by the parameters in the GSI namelist section OBS_INPUT. As an example, the relevant part of OBS_INPUT is:

```
dfile(28)='amsuabufr', dtype(28)='amsua', dplat(28)='n15',     dsis(28)='amsua_n15',
dfile(29)='amsuabufr', dtype(29)='amsua', dplat(29)='n16',     dsis(29)='amsua_n16',
dfile(30)='amsuabufr', dtype(30)='amsua', dplat(30)='n17',     dsis(30)='amsua_n17',
dfile(31)='amsuabufr', dtype(31)='amsua', dplat(31)='n18',     dsis(31)='amsua_n18',
dfile(32)='amsuabufr', dtype(32)='amsua', dplat(32)='metop-a', dsis(32)='amsua_metop-a',
dfile(33)='airsbufr',  dtype(33)='amsua', dplat(33)='aqua',    dsis(33)='amsua_aqua',
dfile(34)='amsubbufr', dtype(34)='amsub', dplat(34)='n15',     dsis(34)='amsub_n15',
dfile(35)='amsubbufr', dtype(35)='amsub', dplat(35)='n16'      dsis(35)='amsub_n16',
dfile(36)='amsubbufr', dtype(36)='amsub', dplat(36)='n17',     dsis(36)='amsub_n17',
```

Please note that the last two columns of the OBS_INPUT have been excluded for conciseness. From this list, we can see the content of *dfile* is the GSI name, which is the observation file name recognized by GSI, while *dtype* and *dplat* indicate the radiance instruments and the satellite name associated with the GSI name in *dfile*. The *dsis* is the radiance observation type that is the combination of the instruments and satellite names. This list tells us that the GSI expects NOAA-15 AMSU-A radiance observations from a BUFR file with name *amsuabufr*. It also reads in the NOAA-18 AMSU-A observations from the same file. For NOAA-17 AMSU-B observations, GSI will read them in from a file named *amsubbufr*.

It is possible to change the GSI name in *dfile* to a user specified name (for example, `'amsuagsi'` rather than `'amsuabufr'`) as long as the GSI name (`amsuabufr`) in the link from the BUFR file (`gdas1.t12z.1bamua.tm00.bufr_d`) to the GSI name has also been changed. The following demonstrates the process required to change the name in *dfile.*

Set new name in namelist section OBS_NPUT:

```
dfile(28)='amsuagsi',  dtype(28)='amsua', dplat(28)='n15',     dsis(28)='amsua_n15',
dfile(29)='amsuagsi',  dtype(29)='amsua', dplat(29)='n16',     dsis(29)='amsua_n16',
```

Then change the GSI name in the run script:

```
ln -s ${OBS_ROOT}/gdas1.t12z.1bamua.tm00.bufr_d amsuagsi
```

It is advised to use the GSI names provided in the released run script because they describe the contents of the file well and are used by many users. However, the flexibility to setup a different GSI name does give GSI more ability to analyze radiance observations from different resources. For example, if we want GSI to assimilate NOAA-15 AMSU-A observations from a BUFR file named *gdas1.t12z.1bamua.tm00.bufr_d* and NOAA-16 AMSU-A observations from another BUFR file named *gdas2.t12z.1bamua.tm00.bufr_d*, we can setup the run script and namelist as follows:

Set the GSI names in the namelist section OBS_INPUT:

```
dfile(28)='amsuabufr',  dtype(28)='amsua', dplat(28)='n15',     dsis(28)='amsua_n15',
dfile(29)='amsuagsi',   dtype(29)='amsua', dplat(29)='n16',     dsis(29)='amsua_n16',
```

And then, link them in the run script:

```
      ln -s ${OBS_ROOT}/gdas1.t12z.1bamua.tm00.bufr_d amsuabufr
      ln -s ${OBS_ROOT}/gdas2.t12z.1bamua.tm00.bufr_d amsuagsi
```

Now, GSI will read in NOAA-15 AMSU-A observations from the GSI file *amsuabufr*, which is the BUFR file *gdas1.t12z.1bamua.tm00.bufr,* and read in NOAA-16 AMSU-A observations from another GSI file *amsuagsi*, which is the BUFR file *gdas2.t12z.1bamua.tm00.bufr*.

A common user mistake in the setup of the radiance data analysis is forgetting to add the radiance observation type the user wants to use into the OBS_INPUT. Some users may notice that NOAA-19 AMSU-A is not on the list of the OBS_INPUT setups in release version 3.0. To use GSI to analyze NOAA-19 AMSU-A observations with the run script and name list from release version 3.0, users need to add one more line in OBS_INPUT, for example:

```
dfile(79)='amsuabufr', dtype(79)='amsua', dplat(79)='n19',     dsis(79)='amsua_n19',
```

Where index *79* for this new line is from the existing number of parameter "*ndat*" in namelist section *SETUP* plus 1. The "*ndat*" should also be set to 79.

In this case, NOAA-19 AMSU-A observations should be included in the BUFR file that `amsuabufr` is linked to. The released run script will be continually updated to include new satellite platforms, however users are suggested to double-check the content of the BUFR file and the setup of the namelist if desired data types are missing from the analysis.

The radiance data normally need to be thinned in the analysis, the last column (`dthin(26)=1,`) in the namelist section OBS_INPUT is used to setup radiance data thinning. The details of radiance data thinning are described in section 3.3 under item 7.

More detailed control on how to use each channel of certain radiance observation types in the GSI analysis can be achieved by setting up the satinfo file. The use of the satinfo file was previously introduced in section 4.3. Please note the satinfo file may be structured differently in different released versions.

8.1.2 GSI Code To Ingest Radiance Data

GSI has a set of files (subroutines) named *read_*.f90* to read in different types of observations, including satellite radiance. The table in Section 6.2.3 gives a complete list of such subroutines. Below is an excerpt of the table that applies to radiance data:

Data type *(ditype)*	Observation type *(obstype)*		Subroutine that reads data
rad (satellite radiances)	(platform) not AQUA	amsub	read_bufrtovs (TOVS 1b data)
		amsua	
		msu	
		mhs	
		hirs4,3,2	
		ssu	
	sndr, sndrd1, sndrd2 sndrd3, sndrd4		read_goesndr (GOES sounder data)
	ssmi		read_ssmi

From this table, we can see TOVS 1b observations from the NOAA and METOP satellites are read in by subroutine *read_bufrtovs* and the GEOS sounder and SSMI observations are read in by subroutine *read_goesndr* and *read_ssmi*.

In general, GSI reads in radiance observations from external BUFR files, picks the observations within the analysis domain and time window, performs thinning based on the coarse grid setup in OBS_INPUT, and saves them into an intermediate binary file using a general data format across all observation types.

In this user's guide, we will use subroutine *read_bufrtovs* (*read_bufrtovs.f90*) as an example to introduce some important aspects of GSI radiance observation ingesting. All these aspects can be extended to other radiance ingesting subroutines because they share the same code structure and BUFR techniques. We hope these points can help advanced users learn the detailed content inside the GSI radiance observation process and add new observations for their GSI application.

- BUFR file ingesting

The basic structure of BUFR file ingesting has two loops to read in every message (`read_subset)` from the BUFR file and then read in all observations (`read_loop`) from each message. In the subroutine *read_bufrtovs*, the two loops are marked by the following code:

```
!    Loop to read bufr file
     next=0
     read_subset: do while(ireadmg(lnbufr,subset,idate)>=0)
             ...
        read_loop: do while (ireadsb(lnbufr)==0)

             ...

!       End of bufr read loops
     enddo read_loop
  enddo read_subset
  call closbf(lnbufr)
```

The content of each observation needed by GSI can be found by searching the BUFR mnemonics (bold in following code sample), for example, the following lines of the code give a list of mnemonics included in the subroutine:

```
hdr1b ='SAID FOVN YEAR MNTH DAYS HOUR MINU SECO CLAT CLON CLATH CLONH HOLS'
if (atms) hdr1b ='SAID FOVN YEAR MNTH DAYS HOUR MINU SECO CLAT CLON CLATH CLONH HMSL'

hdr2b ='SAZA SOZA BEARAZ SOLAZI'

 call ufbrep(lnbufr,data1b8,1,nchanl,iret,'TMBR')
 call ufbrep(lnbufr,data1b8,1,nchanl,iret,'TMBRST')
```

An explanation of each mnemonic can be easily found from the BUFR table used to generate this BUFR file. Users can get this BUFR table on-line, from decoding the BUFR file, or checking the BUFR file *bufrtab.012* in the fix directory of the release package. For example, a search for **SAZA SOZA** in *bufrtab.012*, we found the following two lines:

```
| SAZA     | 007024 | SATELLITE ZENITH ANGLE                                   |
| SOZA     | 007025 | SOLAR ZENITH ANGLE                                       |
```

These lines tell us that GSI needs to read in satellite zenith angle and solar zenith angle for each observation profile.

- Data selection in reading process

In the data ingesting subroutine, only observations within the analysis domain (for regional applications) and time window are processed for the thinning. After establishing a coarse grid based on the setups in the parameters from OBS_INPUT, GSI starts a smart selection of radiance fields of view for the coarse grid. This processing of radiance data thinning not only selects the nearest radiance observation in a coarse grid, but also considers the quality of the radiance observations. The observation for each grid box is chosen based on its quality through a combined penalty value that considers the following criteria:

1. Remove observations where the key channels are bad
2. Prefer observations that have a larger number of good channels
3. Skip observations that the Field of View (FOV) are out of range

4. Prefer profiles that are over better surface fields. For many observation types, the order is (best to worst): sea, sea ice, snow/ land, mixed but this may vary by instrument.
5. Prefer observations based on available data quality predictors

- Internal observation data format

After data thinning, the best quality radiance observation for each coarse grid is then saved with surface status (calculated from the background) in a two-dimensional array called "*data_all*". The 1st dimension of the array saves all information about one observation and the 2nd one loops through the observations. The code that assigns the content of the array starts like:

```
data_all(1 ,itx)= rsat                    ! satellite ID
data_all(2 ,itx)= t4dv                    ! time
data_all(3 ,itx)= dlon                    ! grid relative longitude
data_all(4 ,itx)= dlat                    ! grid relative latitude
```

and ends like:

```
do i=1,nchanl
   data_all(i+nreal,itx)=data1b8(i)
end do
```

The code itself gives clear notation on the content of the 1st dimension of the array except for the last three lines. For example, it clearly tells us the first 4 items in the array are satellite ID (`rsat`), observation time (`t4dv`), and grid relative longitude (`dlon`) and latitude (`dlat`). However, there is no clear notation for `data_all(i+nreal,itx)`, a little search for the array `data1b8` indicates it contains the brightness temperature from all channels in an observation profile.

After reading and processing all observations in the BUFR file and saving them in the data array "*data_all*", this array is written to an intermediate binary file at the end of the subroutine *read_bufrtovs*.

- Observation count in *stdout* file

From the *stdout* file, we can see the following information counting the data during the data ingesting stage, an example from the case in Chapter 5:

```
READ_BUFRTOVS: file=amsuabufr  type=amsua      sis=amsua_n15            nread=    128055
ithin= 2 rmesh= 60.000000 isfcalc= 0 ndata=      53932 ntask=  1
```

This tells us that the subroutine read_bufrtovs is reading NOAA-15 AMSU-A observations from file *amsuabufr*. There are 128055 observations (profile number * channels number) read in from the BUFR file and 53932 observations kept for further processing after data selection and thinning.

8.1.3 Information On Ingesting And Distribution

The analysis in GSI is done in each subdomain for MPI runs. The observation number in each sub-domain can be found in the *stdout* file. All data types are listed in the stdout file as shown in the following example, using the same example as section 5.2.2:

```
OBS_PARA: ps                      2352      2572      8367      2673
OBS_PARA: t                       4617      4331     12418      4852
OBS_PARA: q                       3828      3908     11096      3632
OBS_PARA: pw                        89        31       141        23
OBS_PARA: uv                      5704      4835     15025      4900
OBS_PARA: sst                        0         0         2         0
OBS_PARA: hirs4     metop-a          0         0       416       731
OBS_PARA: amsua     n15           2563      1323      1048      1669
OBS_PARA: amsua     n18           1002      2119         0       390
OBS_PARA: amsua     metop-a          0         0      1268      2279
OBS_PARA: amsub     n17              0         0      1717      2891
OBS_PARA: hirs4     n19            244      1093         0       235
OBS_PARA: amsua     n19            651      3486         0       469
```

Please note the number in each subdomain is the number of the radiance profiles, not the number of observed channels. Each profile includes many channels. For example, each HIRS observation has 19 channels, each MSU has 4 channels, each AMSU-A has 15 and AMSU-B has 5, each MHS has 5 and SSU has 3.

8.2. Radiance Observation Operator

The observation operator for radiance observations is very complex and out of the scope of this user's guide. Here, we only briefly introduce some features of the radiance observation operator. The Community Radiative Transfer Model (CRTM) developed by JCSDA is employed by the GSI system to transform control variables into simulated radiance or brightness temperatures. This operator can be illustrated by the following equation:

$$y=K(x,z)$$

where:

y are simulated radiance observations;
x are profiles of temperature, moisture, and ozone;
K is the radiative transfer equation (CRTM);
z are unknown parameters such as the surface emissivity, CO_2 profile, methane profile, etc.

In GSI, x (including surface conditions) are calculated based on the background fields and then are put into the CRTM functions (K) to calculate the simulated radiance observations y. When unknowns in $K(x, z)$ are too large, which may be from the formulation of K or unknown variables (z), observed radiance data cannot be reliably used and must be removed during quality control. Examples of this include when clouds, trace gases, or aerosols exist in the observed column. The description of radiance data quality control can be found in the next section. For advanced users needing to learn the details of the radiance observation operator in GSI, please check the corresponding subroutine listed in the right column of the section 6.2.4 table.

Because GSI uses the CRTM functions as part of the radiance observation operator, the CRTM coefficients have to be available during the radiance data analysis. In the GSI release package, these CRTM coefficients are linked to the running directory by the run script before the GSI starts to run. The details of linking CRTM coefficients can be found in Chapter 3 in the introduction of the GSI run scripts. Please note that the GSI run script does not know which kind of radiance observations will be used in the analysis. The script links all the CRTM coefficients for the radiance observation types listed in the *satinfo* file. After reading in radiance observations from BUFR files, GSI recognizes which kind of radiance observations to be used and only reads in the corresponding coefficients needed. Therefore, users only need to check whether the CRTM coefficients of the user interested radiance data types are linked correctly. At the same time, users can ignore the warning information on the missing CRTM coefficients if those coefficients are for the radiance data types that are not used in the application.

8.3. Radiance Observation Quality Control

The quality control (QC) may be the most important aspect of satellite data assimilation. Unlike conventional observations from a prepbufr file, which includes the quality markers from the NCEP quality control process, the satellite radiance BUFR file does not include observation quality information. Instead, the quality control for radiance observations is inside the GSI.

The GSI radiance data quality control starts right after the radiance observations are read in (such as in read_bufrtovs.f90). We can think of the processing of radiance data thinning as a part of the quality control because the thinning process selects the best quality observations. The major radiance data quality control step is after the calculation of the radiance observation departure in file *setuprad.f90*. Many QC steps are employed to capture problematic satellite data, which mainly come from the following 4 sources:

- Instrument problems
- Clouds and precipitation simulation errors
- Surface emissivity simulation errors.
- Processing errors (e.g., wrong height assignment, incorrect tracking, etc.)

In GSI, each instrument has its own quality control subroutine. All these subroutines are in the file *qcmod.f90* and are listed as follows for reference:

subroutine name	Quality Control for
qc_ssmi	ssmi, amsre, and ssmis
qc_seviri	seviri data
qc_ssu	ssu data
qc_goesimg	GOES image
qc_msu	msu data
qc_irsnd	ir sounder data(hirs, goessndr, airs, iasi, cris)
qc_avhrr	avhrr and avhrr_navy
qc_amsua	amsua data
qc_mhs	amsub, mhs and hsb data
qc_atms	atms data

After calculating the radiance observation departure from the background and bias correction, these QC functions are called for each instrument to either toss the bad (questionable) observations or inflate the low confidence observations. The number of filtered observations by these QC functions is summarized in the radiance fit file (*fort.207*) as 7 QC categories (steps). To help users understand the meanings of these numbers in the radiance fit file, we will briefly introduce these QC steps in subroutine *qc_amusa* in the following table. Please note these QC categories may have different meaning for different instruments:

Category	Quality Control steps	Action to observations
QC1	Cloud affected profile, (factch4 > 0.5)	Toss channel 1-6, 16
QC2	Inaccurate emissivity /surface temperature estimate over sea	Toss channel 1-6, 16
QC3	Cloud affected profile (Scattering index factch6 > 1.0)	Toss channel 1-7, 16
QC4	Inflate observation error over high terrain (>2000m)	Inflate channel 7 observation error
QC5	Inflate observation error over high terrain (>4000m)	Inflate channel 8 observation error
QC6	Retrieved could liquid water path > 1.0	Part of QC1
QC7	Part of Scattering index > 1.0	Part of QC3

Using the same example as section 4.5.2:

```
sat    type       penalty         nobs   iland  isnoice  icoast ireduce   ivarl nlgross
n15    amsua      19769.16042371 4149     673     1475     268     1311   30453       0
                  qcpenalty        qc1     qc2      qc3     qc4      qc5     qc6     qc7
                  19769.16042371  883      63     2127     183        0      20      46
```

Using the above table, we can understand numbers listed under qc1 to qc7. Listed below also includes the explanation of the numbers not in the above table, for a complete understanding of this part of the radiance fit file. For other portions of the fit file, please see the introduction in section 4.5.2.

From the above example, we see there are 4149 NOAA-15 AMSU-A profiles after thinning, among which there are:

673 profiles over land (iland)
1475 profiles over snow or ice (isnoice)
268 profiles over coast (icoast)
1311 profiles within tropics that has reduced qc bounds (ireduce)
30453 channels that failed in the general gross check (ivarl)
0 channels that passed the general gross check but failed the nonlinear gross check (nlgross)

883 profiles were tossed because of cloud affect based on factch4
63 profiles were tossed because of inaccurate emissivity /surface temperature estimate over sea
2127 profiles were tossed because of cloud affect based on factch6
183 profiles have inflated observation error because of high terrain (>2000m)
0 profiles have inflated observation error because of high terrain (>4000m)
20 profiles meet criterion QC6 (part of qc 1)
46 profiles meet criterion QC7 (part of qc 3)

So, nearly ¾ of the observations were tossed because of cloud effects.

8.4. Bias Correction For Radiance Observations

Using bias correction to correct the system bias in the satellite radiance observations is one of the key steps to get a successful satellite radiance data assimilation. This section will introduce the basic theory of the GSI bias correction, the procedures and configurations of the bias correction in the GSI system, an explanation of the namelist, satinfo, and coefficients for bias correction, the use of the angle bias correction utility, and discussions of some common issues users encounter in the application of the GSI bias correction.

8.4.1. Bias Correction For Satellite Observations

Observation bias can systematically damage the data assimilation results and, consequently, the quality of the forecasting system. Biases in satellite observations are of particular concern because they may larger than the signal and damage the numerical weather prediction system in a very short period of time.

Biases between the satellite observations and the model may come from the following sources:

- satellite instrument itself (e.g. poor calibration or characterization, or adverse environmental effects);
- radiative transfer model (RTM) linking the atmospheric state to the radiation measured by the satellite (e.g. errors in the physics or spectroscopy, or from non-modeled atmospheric processes);
- systematic errors in the background atmospheric state provided by the NWP model used for monitoring.

In GSI, satellite observation bias is represented as a linear regression based on N state-dependent predictors $P_i(\mathbf{x})$, with associated coefficients β_i :

$$\boldsymbol{BC} = \sum_{i=1}^{N} \beta_i \cdot \boldsymbol{P}_i(\boldsymbol{x})$$

Since the bias correction is applied to the radiance departures, this is equivalent to using the modified definition of the observation operator:

$$\tilde{\mathrm{H}}(\mathbf{x}, \boldsymbol{\beta}) = \mathrm{H}(\mathbf{x}) + \mathrm{BC}(\boldsymbol{\beta}, \mathbf{x})$$

The training of the bias correction consists in finding the vector $\boldsymbol{\beta}$ that allows the best fit between the NWP fields **x** and the observations. This is obtained by minimizing the following cost function:

$$J(\boldsymbol{\beta}) = \frac{1}{2}[\mathbf{y} - \tilde{H}(\mathbf{x}, \boldsymbol{\beta})]^{\mathrm{T}}[\mathbf{y} - \tilde{H}(\mathbf{x}, \boldsymbol{\beta})].$$

For more details on the bias correction, please see the references listed below:

1. Auligne T., A. P. McNally and D. P. Dee. 2007. Adaptive bias correction for satellite data in a numerical weather prediction system. *Q. J. R. Meteorol. Soc.* 133: 631-642.
2. Derber JC, Wu W-S. 1998. The use of TOVS cloud-cleared radiances in the NCEP SSI analysis system. *Mon. Weather Rev.* 126: 2287–2299.
3. Harris BA, Kelly G. 2001. A satellite radiance-bias correction scheme for data assimilation. *Q. J. R. Meteorol. Soc.* 127: 1453–1468.
4. Dee, D. P., 2004: Variational bias correction of radiance data in the ECMWF system. Pp. 97–112 in Proceedings of the workshop on assimilation of high-spectral-resolution sounders in NWP. 28 June–1 July 2004, ECMWF, Reading, UK.
5. Dee, D. P. and S. M. Uppala, 2009, Variational bias correction of satellite radiance data in the ERA-Interim reanalysis. Q. J. R. Meteorol. Soc. 135, 1830–1841.

8.4.2. The GSI Bias Correction Procedure And Configurations

In GSI, the bias correction for satellite radiance has two parts: one part is air mass bias correction, also called the variational part of the bias correction; another part is angle dependent bias correction. Each part of bias correction has its own bias correction coefficient file:

- The *satbias_angle* file contains the angle dependent part of the brightness temperature bias for each channel/instrument/satellite. Also included in this file is the mean temperature lapse rate for each channel weighted by the weighting function for the given channel/instrument.
- The *satbias_in* file contains the coefficients for the variational part of the bias correction.

GSI will read in the coefficients from both *satbias_angle* and *satbias_in* files, combine them together with predictors to generate a system bias value for each channel, and then subtract this system bias from the observation innovation during the radiance observation operator calculation. During the minimization process, GSI will calculate the updated coefficients for the predictive part of the bias correction and save the updated coefficients in another file called "*satbias_out*". The angle dependent bias coefficients are updated outside of GSI using a utility named *gsi_angleupdate* in the release package. These new mass and angle dependent bias coefficients should be used for the bias correction in the next cycle of the GSI analysis.

To set up the bias correction for satellite radiance in the GSI system, users need to link the right coefficient files in the run directory and keep the coefficient files updated in cycles:

Step 1, Link coefficient files for both air bias correction and angle dependent bias into the GSI run directory before running the GSI executable.

The coefficient files should come from the previous data assimilation cycle. However, if there is no previous data analysis cycle, the sample coefficient files can be copied from the directory *./fix* within the community release version as a cold start. When using the run script with the released version, the following lines in the run script copy the coefficient files:

```
SATANGL=${FIX_ROOT}/global_satangbias.txt
SATINFO=${FIX_ROOT}/global_satinfo.txt
...
 cp $SATANGL  satbias_angle
 cp $SATINFO  satinfo

# for satellite bias correction
cp ${FIX_ROOT}/sample.satbias ./satbias_in
```

Within the directory *./fix*, the sample angle dependent bias correction coefficients file is called `global_satangbias.txt`, and the file for mass bias correction coefficients is `sample.satbias`. Here, we also include the copy to the *satinfo* file because the bias correction needs information from the *satinfo* file.

Step 2, Run GSI and save the output from the mass bias correction for next cycle

After running the GSI, an updated coefficient file for the mass bias correction is generated in the run directory. This file is called "*satbias_out*", which should be saved for the next cycle of the GSI analysis. There is a line commented out in the released GSI run script reserved for this purpose. The user should choose how to save the file for the next cycle:

```
#   GSI updating satbias_in
#
# cp ./satbias_out ${FIX_ROOT}/sample.satbias
```

Step 3, Run the angle dependent bias correction utility after GSI runs and save updated coefficients of angle dependent bias correction for use in the next cycle

The update of the coefficients for angle dependent bias correction is done by a stand-alone application named *gsi_angleupdate* located under the directory *./util*, but outside the GSI itself. This application reads in the diag files from the GSI analysis results and the old angle dependent bias coefficients, updates the coefficients and saves them as a new file called "*satbias_ang.out*". We will introduce how to apply this utility in the next section.

Please note the cycling of the coefficients to let the bias information accumulate during the data assimilation cycle is the key to getting the right bias correction.

8.4.3 Namelist, Satinfo, And Coefficients For Bias Correction

To conduct the bias correction, GSI needs several pieces of information from different files:

- The satellite platform information from the GSI namelist
- The usage information for each channel from the *satinfo* file
- The coefficients from both mass and angle dependent bias correction coefficient files

The following is a brief introduction to these files to help the user to understand the contents of each file and know how to check if the user interested satellite channels are correctly configured in these files.

- The satellite platform information from the GSI namelist

The complete explanation of the GSI run script and most often used namelist options can be found in Chapter 3 of this guide. More details of setting up radiance data analysis in the run script are described in section 1 of this chapter. Users should make sure that required satellite instruments and platforms are in the list in &OBS_INPUT and have been correctly linked to the BUFR files.

Also, the following is a list of GSI namelist options related to the bias correction:

Variable name	Default value	Description
`diag_rad`	.true.	logical to turn off or on the diagnostic radiance file (true=on)
`passive_bc`	.false.	logical to turn off or on radiance bias correction for monitored channels
`adp_anglebc`	.false.	option to perform variational angle bias correction

- The usage information for each channel from the *satinfo* file

The GSI uses an information file called "*satinfo*" to control how to use each radiance channel. Detailed information about *satinfo* can be found in the GSI User's Guide Section 4.3. The following is an example:

```
!sensor/instr/sat      chan iuse   error   error_cld   ermax   var_b    var_pg  icld_det
 amsua_n15               1   1     3.000     9.100     4.500   10.000    0.000      1
 amsua_n15               2   1     2.000    13.500     4.500   10.000    0.000      1
 amsua_n15               3   1     2.000     7.100     4.500   10.000    0.000      1
 amsua_n15               4   1     0.600     1.300     2.500   10.000    0.000      1
    ●●●●●
 amsua_n15              14  -1     2.000     1.400     4.500   10.000    0.000     -1
 amsua_n15              15   1     3.000    10.000     4.500   10.000    0.000      1
 hirs3_n17               1  -1     2.000     0.000     4.500   10.000    0.000     -1
 hirs3_n17               2  -1     0.600     0.000     2.500   10.000    0.000     -1
 hirs3_n17               3  -1     0.530     0.000     2.500   10.000    0.000     -1
    ●●●●●
```

Users can easily understand the first 2 columns are sensor/instrument/satellite and channel number information. The 3rd column is the usage information, which has the following meanings:

iuse	Channel usage in GSI
-2	do not use
-1	monitor if diagnostics produced
0	monitor and use in QC only
1	use data with complete bias correction
2	use data with no air mass bias correction
3	use data with no angle dependent bias correction
4	use data with no bias correction

For bias correction purposes, please make sure user interested channels are listed in the *satinfo* file and have been set to the correct usage flag.

- The coefficients from both mass and angle dependent bias correction coefficient files

As previously introduced in this section, there are two bias correction coefficient files. These files include the bias correction coefficients for each channel:

1) *satbias_in*

This file contains the coefficients for the predictive part of the bias correction (air mass bias correction coefficients). There is a sample for this file named "*sample.satbias*" in the GSI release package under the directory *./fix*. All coefficients in this sample file are 0.

Here, we use NOAA-15 AMSU-A from the *satbias_out* file from the radiance application case in Chapter 5 as example:

```
    1 amsua_n15              1    0.472353   -0.231512    0.291223    0.000634   -0.148959
    2 amsua_n15              2   -0.677697    0.382025    1.424922   -0.000061    0.016514
    3 amsua_n15              3   -2.631062    0.134578    2.968469   -0.004946    1.213581
    4 amsua_n15              4   -0.470401    2.121855    5.764014    0.006496    1.333609
    5 amsua_n15              5   10.996354   -0.762965    1.787372    0.082404   -1.661531
    6 amsua_n15              6  -22.026905   -1.543174   -1.397403    0.175626  -11.384948
    7 amsua_n15              7   -1.954080   -0.293421    0.029899   -0.064129    3.039958
    8 amsua_n15              8   -9.468913   -1.490995   -0.856006   -0.013090   -0.945916
    9 amsua_n15              9  -22.737061   -2.195735    0.247890   -0.357354  -15.298422
   10 amsua_n15             10   -0.875332    2.212551   -0.392323   -0.337414   -8.785395
   11 amsua_n15             11    0.000000    0.000000    0.000000    0.000000    0.000000
   12 amsua_n15             12    2.800800   -4.042608    0.060067    0.913834   15.980004
   13 amsua_n15             13    0.000000    0.000000    0.000000    0.000000    0.000000
   14 amsua_n15             14    0.000000    0.000000    0.000000    0.000000    0.000000
   15 amsua_n15             15   -0.439501    0.539856    0.412582   -0.001741    0.158646
```

The first 3 columns are series number, the sensor/instrument/satellite, and channel number of each instrument. Columns 4 through 8 are 5 coefficients for the predictive (air mass) part of the bias correction, which has 5 predictors.

2) *satbias_angle*

The satbias_angle contains the angle dependent part of the brightness temperature bias. There are two sample files for this in the GSI release package under the directory *./fix*: *global_satangbias.txt* and *nam_global_satangbias.txt*. Here we only give two channels as examples from the file *global_satangbias.txt*:

```
1 amsua_n15              1   0.528768E-02
      0.063 -0.200 -0.411 -0.588 -0.638 -0.523 -0.493 -0.466 -0.482 -0.475
     -0.666 -0.587 -0.593 -0.602 -0.766 -0.955 -1.080 -1.218 -1.149 -1.374
     -1.553 -1.635 -1.715 -1.783 -1.689 -1.507 -1.473 -1.244 -1.233 -1.259
      0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
      0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
      0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
      0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
      0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
      0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
2 amsua_n15              2   0.290661E-02
     -2.769 -2.880 -2.583 -2.449 -2.218 -1.810 -1.536 -1.242 -0.882 -0.788
     -0.676 -0.697 -0.508 -0.464 -0.544 -0.790 -0.945 -1.108 -1.002 -1.364
     -1.404 -1.315 -1.318 -1.151 -0.826 -0.219  0.086  0.631  1.121  1.807
      0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
      0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
      0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
      0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
      0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
      0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
```

The first line for each channel looks like:

```
    1 amsua_n15              1   0.528768E-02
    2 amsua_n15              2   0.290661E-02
```

The columns are series number, the sensor/instrument/satellite, channel number of each instrument, and "T lap mean", respectively. The next 90 numbers are coefficients for angle dependent bias correction. These numbers correspond to the number of the FOV per scan. For example, AMSU-A has 30 FOV per scan, which is using the first 30 numbers to represents the bias correction coefficients for each FOV position, while the AMSU-B has 90 FOV per scan, all 90 numbers are used to do bias correction.

If there are some instruments in *satinfo* but not in *satbias_in* and *satbias_angle*, GSI will set 0 as the initial value for these instruments and write out updated coefficients for these new instruments in coefficient results files.

8.4.4 Enhanced Radiance Bias Correction

Since comGSIv3.3, the enhanced radiance bias correction is available to improve the radiance bias correction and simplify the bias corrections cycles. In the enhanced radiance bias correction, the angle bias is also calculated inside GSI instead of outside GSI like previous versions. This section is tailor based on an email from Yunqiu Zhu on how to setup the enhance bias correction in GSI , for more details on this enhanced radiance bias correction, please check the following published paper:

> Zhu, Y., Derber, J., Collard, A., Dee, D., Treadon, R., Gayno, G. and Jung, J. A. (2013), Enhanced radiance bias correction in the National Centers for Environmental Prediction's Gridpoint Statistical Interpolation data assimilation system. Q.J.R. Meteorol. Soc..
> doi: 10.1002/qj.2233

The steps to get the enhanced radiance bias correction running are summarized as follows.

1. Add namelist options to turn on in the SETUP:

   In ./run/ run_gsi.ksh, add the following namelist options in section SETUP

   ```
   newpc4pred=.true.,adp_anglebc=.true.,angord=4,
   passive_bc=.true.,use_edges=.false.,emiss_bc=.true.,
   diag_precon=.true.,step_start=1.e-3,
   ```

   You may set the option passive_be=.true. if you want to do bias correction for the passive channels as well.

2. Link bias files and diag files from previous cycle

   Angle bias *satbias_angle* file and the separate angle bias correction step are no longer needed. The files required at each analysis cycle are *satbias_in*, *satbias_pc*, and *diag* files from previous analysis cycle. User can copy *satbias_out* , *satbias_pc.out*. in prevous cycle to current GSI run directory and rename the files as *satbias_in* and *satbias_pc*. Please make sure that *diag* files is available to be used for the first analysis cycle. The diag file for guess are used here and the time tag is removed when used in the bias correction, for example, previous cycle has diag file called: *diag_amsua_n18_ges.2014061915* In this cycle for bias correction, this file should be called: *diag_amsua_n18*.

Since the format and units of the bias file are changed, at the very first time when you start to use the enhanced radiance bias correction, please use the released sample files in fix directory to start:

- rap_satbias_in_enhanced
- rap_satbias_pc_enhanced

3. script changes

Please make sure the GSI run scripts has code to save the diag files and bias files for the next cycle bias correction.

8.4.5. Utility For Angle Bias Correction Outside GSI

Before the enhanced radiance bias correction available, the coefficients for correcting the angle dependent part of the brightness temperature bias in the GSI are calculated after each GSI run. The NCEP has developed a tool to calculate these coefficients and, the community GSI release v3.1 started to include this tool as a part of the release package. This tool is released as a directory named *./gsi_angupdate* within *./comGSIv3.3/util* . This way is still working if users don't want to use enhanced bias correction.

Please note the community GSI release package version 3.0 doesn't have this tool. If users are using the GSI release 3.0 and need this tool, please download the tar file "*gsi_angupdate.tar*" from the GSI download page on-line. Once untarred, under the GSI directory "./comGSIv3/util", you will see a new directory: *./gsi_angupdate*, which includes the tool to update coefficients for radiance angle dependent bias correction.

- Compile

Inside the directory *./gsi_angupdate*, type the following command:

```
./make
```

then check if executable "*gsi_angupdate.exe*" exists in the same directory.

Please note that before compiling this utility, the community GSI should already be compiled successfully. Please refer to Chapter 2 of this User's Guide on how to configure and compile the community GSI.

- Run

Before running "*gsi_angupdate.exe*", make sure that GSI with radiance data have finished successfully and the diagnostic files that hold O-B information have been generated in the GSI run directory. The executable ""*gsi_angupdate.exe*" will read in O-B information from the diagnostic files for each sensor to update coefficients for angle dependent bias correction of the sensor.

To help users easily run this tool, a sample run script named *run_gsi_angupdate.ksh* is provided within the *./comGSIv3*.1.run directory. If a user uses the tar file downloaded separately on-line, a similar run script can be found in the directory *"./comGSI/util*" with the code.

This script is modified based on the GSI run script. The script has a similar structure. Please check section 3.2.2.1 for instructions on setting up the machine environment, and section 3.2.2.2 for setting up the run environment. The run environment portion is illustrated below:

```
#####################################################
# machine set up (users should change this part)
#####################################################
#
# GSIPROC = processor number used for GSI analysis
#------------------------------------------------
  GSIPROC=1
  ARCH='LINUX_PBS'
# Supported configurations:
            # IBM_LSF,
            # LINUX, LINUX_LSF, LINUX_PBS,
            # DARWIN_PGI
```

In this script, only four parameters need to be set for a case study. These parameters have been explained clearly in the run script and illustrated below:

```
#
#####################################################
# case set up (users should change this part)
#####################################################
#
# ANAL_TIME= analysis time  (YYYYMMDDHH)
# WORK_ROOT= working directory, where angupdate executable runs
# GSI_WORK_ROOT= GSI working directory, where GSI runs
# GSI_ANGUPDATE_EXE  = path and name of the gsi angupdate executable
  ANAL_TIME=2011032212
  WORK_ROOT=./comGSIv3.1/run/angupdate_${ANAL_TIME}
  GSI_WORK_ROOT=./comGSIv3.1/run/arw_2011032212
  GSI_ANGUPDATE_EXE=./comGSIv3.1/util/gsi_angupdate/gsi_angupdate.exe
#
```

These parameters tell the analysis case time, where to find the GSI run directory and gsi_angupdate.exe, and where to run gsi_angupdate.exe.

The run time information can be found in the *stdout* file. A successful run should end with the following information:

```
   PROGRAM GLOBAL_ANGUPDATE HAS ENDED.  IBM RS/6000 SP
```

After a successful run, an updated coefficients file named "*satbias_ang.out*" should be found in the run directory.

- Namelist

The namelist for gsi_angupdate.exe has two sections: `setup` and `obs_input`. Here, we only show and illustrate part of the namelist as an example.

```
&setup
 jpch=2680,nstep=90,nsize=20,wgtang=0.008333333,wgtlap=0.0,
 iuseqc=1,dtmax=1.0,
 iyy1=${iy},imm1=${im},idd1=${id},ihh1=${ih},
 iyy2=${iy},imm2=${im},idd2=${id},ihh2=${ih},
 dth=01,ndat=50
/
&obs_input
 dtype(01)='hirs3',     dplat(01)='n17',       dsis(01)='hirs3_n17',
 dtype(02)='hirs4',     dplat(02)='metop-a',   dsis(02)='hirs4_metop-a',
 dtype(03)='goes_img',  dplat(03)='g11',       dsis(03)='imgr_g11',
 dtype(04)='goes_img',  dplat(04)='g12',       dsis(04)='imgr_g12',
 dtype(05)='airs',      dplat(05)='aqua',      dsis(05)='airs281SUBSET_aqua',
 dtype(06)='amsua',     dplat(06)='n15',       dsis(06)='amsua_n15',
```

The section `obs_input` only has three columns, which have the same meaning as their counterparts in the GSI namelist, i.e., *dtype* and *dplat* specify the radiance instrument and the satellite name, respectively, and *dsis* indicates the radiance observation type with a name combining both the instrument and the satellite names.

Most of the parameters in the section `setup` are different from the section `setup` in GSI namelist. We will explain these parameters below:

`jpch`:	total channel number in coefficients file : *satbias_ang.in*
`nstep`:	maximum number of FOV per scan
`iyy1,imm1,idd1,ihh1`:	start date: year, month, day, and hour
`iyy2,imm2,idd2,ihh2`:	end date: year, month, day, and hour
`dth`:	time interval between start and end date. If start date is not equal to end date, the code will loop based on *dth* through the period to process multiple cycles.
`ndat`:	Number of radiance observation types that can be processed, which is the dimension for parameters in section: `obs_input`
`iuseqc`:	>0 (i.e., 1), check variance. If *errinv*= (1 /(obs error)) is small (small = less than 1.e-6), the observation did not pass quality control. In this case, do not use this observation in computing the update to the angle dependent bias.

	<=0 (i.e., 0 or -1), ensure (o-g)<*dtmax*. If the user says to ignore the qc flag, check that the o-g difference falls within the user specified maximum allowable difference. If the o-g lies outside this bound, do not use this observation in computing the update to the angle dependent bias.
`dtmax`:	user specified maximum allowable difference for o-g difference
`nsize`:	the sample size number. If sample size is less than this number, the updating weight will be reduced based on sample size
`wgtang`:	weight for updating the mean temperature lapse rate
`wgtlap`:	weight for updating angle dependent bias coefficients. The update will be faster as this number gets bigger.

8.4.6. Discussion of FAQ

In this section, we will discuss some frequently asked questions on satellite radiance bias correction.

- Where to get bias correction coefficient files for the NCEP operational system.

  The real-time satellite bias correction coefficients used for the NCEP operational system is available on-line from the same website that holds observation BUFR/PrepBUFR files:

  For GDAS: http://nomads.ncep.noaa.gov/ pub/data/n ccf/com/gfs/prod
  Once in the sub-directory, look for files with name similar to:

  gdas1.t00z.abias for coefficients of mass bias correction.

  For NAM: http://nomads.ncep.noaa.gov/ pub/data/n ccf/com/nam/prod
  Once in the sub-directory, look for files with name similar to:

  nam.t00z.satbias.tm00 for coefficients of mass bias correction.

  Right now, the coefficient files for angle dependent bias correction are not available in these web sites.

- Notes on released *satbias_in* and *satbias_angle*

  As mentioned in this section, the released version provides sample files for these coefficients under the directory *./fix*:

  satbias_in: *sample.satbias*
  satbias_angle: *global_satangbias.txt* and *nam_global_satangbias.txt*

These files are provided as a sample only. Users need to generate their own coefficients based on their experiments. Usually, these coefficients need to be cycled for a period (weeks or months) to get to a stage to do the right bias correction.

- What if the user has no bias correction coefficients and only runs short experiments (e.g., a week) for radiance data assimilation?

  Following the suggestion from NCEP experts, the following may help some users to improve their radiance data assimilation experiments:
  1) Start with coefficient files for a date as close as possible to your cases.
  2) Run a single GSI analysis with mass bias and angle dependent bias correction. You can get updated mass bias and angle dependent bias correction coefficient files.
  3) Run the same GSI analysis as step 2 using the same background and observations but supply GSI with updated mass and angle dependent bias correction coefficient files.
  4) Repeat step 3 about 10 times to spin up the mass and angle dependent coefficients.
  5) Move on to the next cycle or analysis time and repeat steps 2 to 4.
  6) After one or two days, the mass coefficient should be ready for the real case test. Angle dependent bias correction will spin up slowly.

  By starting two days prior to your real case period to spin up the coefficients, you should be able to get better bias correction results.

- Channel lists in *satinfo*, *satbias_in* and *satbias_angle* do not match

  The radiance channels in *satinfo* should match the channels in *satbias_in* and *satbias_angle*. If they do not match, GSI will match *satbias_in* based on channels in *satinfo*:

  If radiance channels only exist in *satinfo* but not in *satbias_in*, these channels will be added to the updated coefficient files with 0 as the initial values.

  If radiance channels are not in *satinfo* but are in *satbias_in*, the extra channels in *satbias_in* will be removed from the updated files.

  If channels in *satinfo* and *satbias_angle* do not match, GSI will use the channels in both files, but the angle dependent update tool will crash due to the mismatch. Therefore, users need to make sure the channels in *satinfo* and *satbias_angle* match.

- How to select suitable satellite radiance channels when assimilating radiance data with GSI:
  This question is not only for bias correction.
  1) Model top and instrument weighting functions:

Each channel has its own weighting function. If part of the weighting function is above the model top, you may need to exclude this channel because your model cannot obtain the correct simulated radiance from background.

2) Bias correction:
   If a particular channel cannot be bias-corrected, for reasons such as the channel is not correctly calibrated or due to instrument failure, you need to turn that channel off. You may be able to check the time-series of bias for a certain channel to get an idea of the status of the channel bias correction.
3) Test:
   Try to view the data impact of each channel on the forecast to decide which channel(s) are best for your application. You can monitor and perform bias correction on each channel for a certain period and then turn that particular channel from monitoring to usage in order to check the impact of the channel.

8.5. Radiance Data Analysis Monitoring

The NCEP operational GSI system includes a Radiance Monitoring Package to extract certain radiance data from the GSI radiance diagnostic files and produce images as an aid to monitor GSI radiance data assimilation performance and diagnose assimilation problems. This package has been used at NCEP to support the following Radiance Assimilation Monitoring web site:

http://www.emc.ncep.noaa.gov/gmb/gdas/radiance/index.html

As discussed in the previous sections of this Chapter, radiance data assimilation is a complex process, in which data quality control and bias correction are key steps for a successful GSI application with radiance observations. To help users to monitor their radiance data assimilation with the GSI system, the DTC ported this useful package into the community GSI system for the Linux platform and included it as one of the utility tools in the release version 3.1.

NCEP has updated this package since release 3.1. In this release, the Radaince Monitoring package has been taken out of the official community GSI release to give DTC more time to port and test the new package. The code and instructions to the Radaince Monitoring will be available on-line as a separate package. Please send gsi_help@ucar.edu for latest update on this package.

Chapter 9 Radar Data Assimilation

The community GSI release version 3.2 and later includes functions for both radar radial velocity and reflectivity analysis. The radial velocity observations in each bin are used in variational process with other wind observations to improve wind field. The reflectivity data are not used in variational process. Instead, they are used by GSD cloud analysis package inside GSI to improve precipitation hydrometeor analysis and provide temperature tendency in storm to enhance the storm initialization through WRF DDFI. Currently, the radial velocity observations are used in NAM operation and reflectivity observations are used in RAP and NAM operation.

9.1 Prepare Radar Data Files for GSI

9.1.1 Introduction

Real time data feeding for operational radar data analysis with GSI is complex, involving many steps of data quality control and format converting. But in research, these steps can be simplified so that community users can generate their own radar data files to feed GSI for radar data analysis as long as they understand the GSI radar data interface. Since release version 3.2, a new tool is available to help users understand the GSI radar data interface, it includes:

- This section to explain the content and structure of the radial velocity and reflectivity BUFR files used by the GSI.
- Sample code to learn how to encode and decode NCEP Level II radial velocity BUFR files based on the NCEP radar data preprocess code
- Sample code to read NSSL MRMS mosaics tiles and to interpolate the mosaic to analysis grid based on the RAP reflectivity preprocess.

Users should already be familiar with the basic BUFR process skills. If not, please visit the DTC BUFR webpage:

http://www.dtcenter.org/com-GSI/BUFR/index.php

In the comGSIv3.2 package, The new sample code for GSI radar data interface is released separately from the official package. Users can download it from the same download page as the comGSIv3.2 package. It is named as "*comGSI_v3.2_radar_process.tar.gz*" and need to be placed in directory *./util* and un-tared before use. After comGSI_v3.3 release, this tool is already under ./util directory.

9.1.2. GSI Interface To Level II Radar Velocity

To add your own radar level II radial velocity data into GSI analysis, the first thing is to understand how GSI reads the radial velocity from the radar Level II radial velocity BUFR files. In current GSI code and run script, the Level II radial velocity BUFR file is named as "*l2rwbufr*" and reads in through a subroutine called "*radar_bufr_read_all"* (in file *read_l2bufr_mod.f90*). The main functions of this subroutine are:

- decodes the BUFR file to read in the radial wind observations
- does "super-obbing" to get radar velocity super obs
- write out the new super obs to a binary file called "*radar_supobs_from_level2*"

Based on this subroutine and the BUFR output interface code from the NCEP radar Level II radial wind process, we generated two sample codes to illustrate the content and the structure of the radar level II radial velocity BUFR file used by GSI. Users can find these two samples under directory *./util/radar_process/radialwind,*

- *bufr_decode_l2rwbufr.f90* : sample code to decode (read) the radial velocity from BUFR file "*l2rwbufr*" and write radial velocity observations in a binary file.
- *bufr_encode_l2rwbufr.f90* : sample code to read in radial velocity from the binary file generated by *bufr_decode_l2rwbufr.f90* and then encode (write) the radial velocity to the BUFR file "*l2rwbufr*".

A makefile in the same directory is provided for users to compile the code. The sample code has to be compiled after successful compile the GSI. It can be compiled with both Intel and PGI compilers.

9.1.2.1 Read observations from Level II radar radial velocity BUFR files

The sample code *bufr_decode_l2rwbufr.f90* only has 87 lines. It has the same structure as the other BUFR decoding code released by DTC as samples for users to learn BUFR file decoding. After users know the general BUFR file decoding steps, the key to understand the radar radial velocity BUFR file decode process is to know all the mnemonics used in the code and the meanings of these mnemonics. Users can get explanations on each mnemonic from a BUFR table called "*bufr_radar.table*", which is a text file generated during decoding sample BUFR file "l2rwbufr" using *bufr_decode_l2rwbufr.f90*.
In this document, we provide the following table to explain the meanings of the mnemonics used in GSI Level II radial velocity interface. Please refer to the BUFR table itself for more details.

The mnemonics and their meanings for radar Level II radial velocity

mnemonic	Meaning	dimension
`SSTN`	RADAR STATION IDENTIFIER (SHORT)	1
`CLAT`	RADAR STATION LATITUDE (COARSE ACCURACY)	1
`CLON`	RADAR STATION LONGITUDE (COARSE ACCURACY)	1
`HSMSL`	HEIGHT OF RADAR STATION GROUND ABOVE MSL	1
`HSALG`	HEIGHT OF ANTENNA ABOVE GROUND	1
`ANEL`	ANTENNA ELEVATION ANGLE	1
`ANAZ`	ANTENNA AZIMUTH ANGLE	1
`QCRW`	QUALITY MARK FOR WINDS ALONG RADIAL LINE	1
`YEAR`	YEAR OF OBSERVATION BEAM	1
`MNTH`	MONTH OF OBSERVATION BEAM	1
`DAYS`	DAY OF OBSERVATION BEAM	1
`HOUR`	HOUR OF OBSERVATION BEAM	1
`MINU`	MINUTE OF OBSERVATION BEAM	1
`SECO`	SECOND OF OBSERVATION BEAM	1
`DIST125M`	DISTANCE FROM ANTENNA TO GATE CENTER IN UNITS OF 125M	Beam
`DMVR`	DOPPLER MEAN RADIAL VELOCITY	Beam
`DVSW`	DOPPLER VELOCITY SPECTRAL WIDTH	Beam
`SCID`	RADAR SCAN ID (RANGE 1-21)	1
`HNQV`	HIGH NYQUIST VELOCITY	1
`VOCP`	VOLUME COVERAGE PATTERN	1
`VOID`	RADAR VOLUME ID (IN THE FORM DDHHMM)	1

In NCEP Level II radar radial velocity BUFR file, radar observations are organized and saved as radial observation beams. Each subset includes observations from one beam. Two parts of information are available in each subset about the beam:

- Head mnemonics (Single variables) describe the beam features:

  ```
  SSTN CLAT CLON HSMSL HSALG ANEL ANAZ QCRW
  YEAR MNTH DAYS HOUR MINU SECO
  SCID HNQV VOCP VOID
  ```
- Arrays content the observation location (`DIST125M`), mean radial wind (`DMVR`), and velocity spectral width (`DVSW`) along the beam

In our sample decoding file *bufr_decode_l2rwbufr.f90*, the above information of each beam is read in beam by beam (subset by subset) until all the beams have been processed. If this beam includes valid radial wind or velocity width observations, it will be saved to a binary file: *l2rwbufr.bin*. We currently commented out most of the standard output information in the file, but leave the final count on the total subsets that have valid observations.

9.1.2.2 Write Level II radar radial velocity observations to BUFR files

After familiar with the NCEP radar Level II radial wind BUFR file structure and content, users can easily understand the sample encoding code *bufr_encode_l2rwbufr.f90* in the same directory. Based on this file, users can encode their own observations into a BUFR file for GSI to do radial wind analysis.

The encoding shares the same mnemonics and structure as decoding. So, after run decode sample, users can run encode sample to read in the radar observations from *l2rwbufr.bin* and encode them into a new BUFR file called: l2rwbufr_new. Users may notice that the file size of *l2rwbufr_new* is smaller than the size of *l2rwbufr*. This is because the *l2rwbufr_new* only includes radial beam with valid observations while the *l2rwbufr* includes beams with missing observations.

Another possible operation is to append some new radial wind observations to a exiting NCEP Level II radial wind BUFR file. A little changes to the encoding sample will do the job. Please refer to the BUFR user's guide from DTC BUFR website for how to append the observations.

Based on the NCEP radar data interface code, there are 4 variables, `SCID HNQV VOCP VOID`, are in Level II BUFR file but not read in by GSI. Our sample codes keeps these 4 variables for reference only.

9.1.3 GSI Interface To Radar Reflectivity

The GSI interface to radar reflectivity is different from the one to Level II radar radial wind introduced above. Before GSI, the radar reflectivity observations in certain height level have to be horizontally interpolated into analysis grid points and saved into a BUFR file called "*refInGSI*". Then the GSI reads in these reflectivity columns over each grid point from the BUFR to feed the reflectivity into the GSD cloud analysis package to improve the precipitation analysis and storm forecast.

9.1.3.1 Radar reflectivity preprocess code

The GSD has developed an application package to preprocess both the NSSL radar reflectivity mosaics and the NCEP radar reflectivity mosaics for RAP GSI cloud analysis. DTC simplified that package to only preprocess NSSL new 4 tiles MRMS mosaics in binary format. We will use this simplified package as an example to illustrate how to prepare radar reflectivity BUFR for the community GSI release version 3.2 and later.

The package is under *"./util/radar_process/reflectivity"*. It includes fortran code, a namelist "*mosaic.namelist*" for running the code, and a BUFR table "*prepobs_prep.bufrtable*" for encoding the reflectivity BUFR files. The fortran code can be compiled with Intel compiler only with the makefile under the same directory. After compile, an executable named as "*process_NSSL_mosaic.exe*" should show up in the same directory.

There are three steps to set up running environment for this executable:

1. The sample code will read the NSSL new 4 tiles MRMS mosaics in binary format. The 4 tiles should be renamed as:

   mosaic_t1 mosaic_t2 mosaic_t3 mosaic_t4

   The sample code can only process 4 tiles MRMS mosaics binary files available from NSSL since summer 2013. The code for processing old 8 tiles mosaic netcdf files is not included in this package.

2. Configure namelist file, *mosaic.namelist*:

   ```
   &setup
     tversion=4,
     analysis_time = 2013111518,
     dataPath = '../data/',
     bkfile = '../data/wrfinput_d01',
    /
   ```
   where *tversion* is always set to 4. The `analysis_time` have format YYYYMMDDHH; the `dataPath` is the directory that includes 4 mosaic tiles (mosaic_t1-4); the `bkfile` is the path and WRF background file used for GSI analysis.

3. Run *process_NSSL_mosaic.exe* with 4 cores.
   Please note the code has to be run by at least 4 cores because each tile needs one core to process. The namelist (*mosaic.namelist*) and BUFR table file (*prepobs_prep.bufrtable*) should be in the same directory as the executable.

After run, the radar reflectivity BUFR file named as "NSSLRefInGSI.bufr" should show up in run directory.

9.1.3.2 Radar reflectivity interface: content and structure

In this package, the file "*write_bufr_ref.f90*" is to write reflectivity into the BUFR file. From this file, we can learn the structure and content of the reflectivity BUFR file.
The radar reflectivity observations are written column by column. Each subset includes the information from one column. In each subset, there are only 6 mnemonics:

The mnemonics and their meanings for radar reflectivity

mnemonic	meaning	dimension
`SID`	RADAR STATION IDENTIFIER (not used in GSI)	1
`XOB`	X-index for grid coordinate of reflectivity column	1
`YOB`	Y-index for grid coordinate of reflectivity column	1
`DHR`	OBSERVATION TIME MINUS CYCLE TIME (not used in GSI)	1
`TYP`	PREPBUFR REPORT TYPE (not used in GSI)	1
`HREF`	Horizontal reflectivity	31

Only XOB, YOB, and HREF are used by GSI, if users can wire their only reflectivity observations over analysis grid with columns that has vertical level list below (in km):

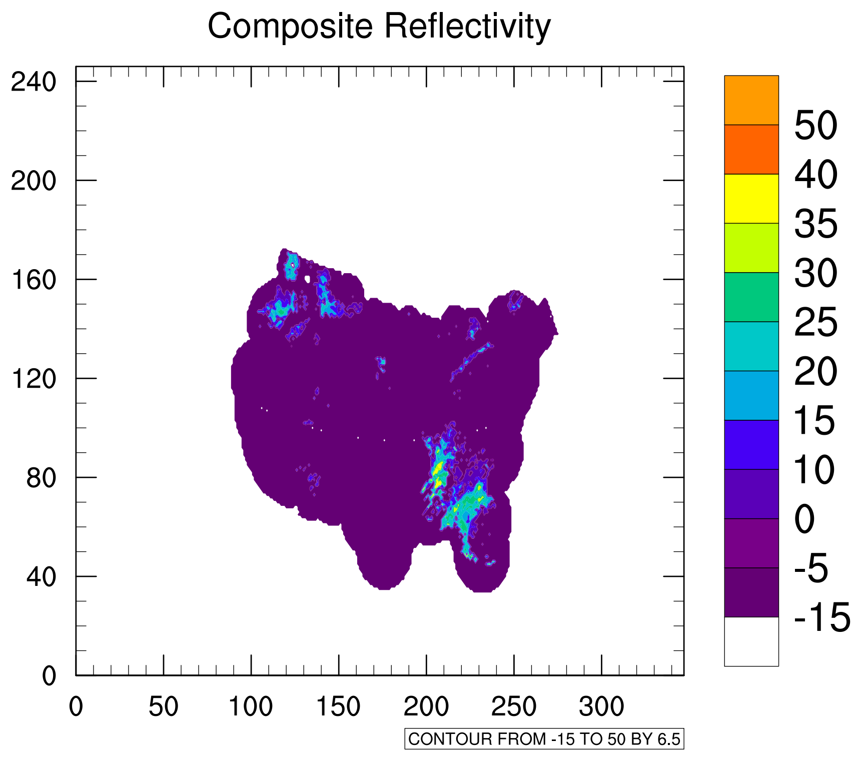
0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 8.5, 9, 10, 11, 12, 13, 14, 15, 16, 18

Then, users can use "“*write_bufr_ref.f90*” directory to encode BUFR file for GSI. If user’s radar reflectivity column has different vertical levels, please contact DTC GSI help desk for how to change the cloud analysis code for the new vertical levels.

9.1.3.3 Check the results

When generate the radar reflectivity BUFR file for GSI, the sample preprocess also write out the composite reflectivity (*compref.bin*) based on reflectivity columns over the analysis grid. This composite reflectivity can be used to check if the preprocess is process reflectivity mosaic successfully.

In the release package under the reflectivity directory, we provide a NCL script, *plot_compositeRef.ncl*, to help user plot the composite reflectivity. The result figure is called *compsiteRef.pdf* and the figure from the sample data we provided on-line is shown below.



Composite Reflectivity from the sample reflectivity preprocess, which is based on NSSL MRMS reflectivity observations at 18Z, November 11, 2013

9.2 Analyze Radar Radial Velocity With GSI

After get the radar level II radial velocity BUFR file ready for GSI, users need to go through the following steps to setup GSI radial velocity analysis.

1. Link the radial velocity BUFR file to GSI run directory in run scripts

GSI code has hardwired the BUFR file name for Level II radial velocity observations. So the 1st step to use the radial velocity is to add a link in the GSI run scripts to link the radial velocity BUFR file to GSI working directory with this hardwired name:

```
ln -s "the patch and name of level II radial velocity BUFR file" l2rwbufr
```

GSI can also analyze the level-III and level-2.5 radar velocity, which is available for NAM application for many years. When both Level-II and Level-III/2.5 available, level-II will be used over the III/2.5, but outside the Level-II radar coverage, Level-III/2.5 will be used. The Level-II/2.5 BUFR file can be linked through the following line in the runs scripts:

```
ln -s "the patch and name of level III/2.5 radial velocity BUFR file" radarbufr
```

2. Setup GSI namelist for radial velocity analysis

In GSI namelist, only level III/2.5 radial velocity need to be set as the following sample:

```
dfile(09)='radarbufr', dtype(09)='rw',       dplat(09)=' ',      dsis(09)='rw',              dval(09)=1.0, dthin(09)=0, dsfcalc(09)=0,
```

To apply high-resolution radial velocity to regional GSI analysis, radar observations need to be thinned with superobs method. This superobs method is controlled by the following namelist section:

```
&SUPEROB_RADAR

del_azimuth=5.,del_elev=.25,del_range=5000.,del_time=.5,elev_
angle_max=5.,minnum=50,range_max=100000.,
   l2superob_only=.false.,
 /
```

Please check Appendix A for the detailed explanation of the options in the SUPEROB_RADAR section.

3. Setup convinfo for radial velocity

As other conventional observations, GSI uses "*convinfo*" file to control the data usage of each observation type. Please check GSI user's guide for details of "convinfo", here is an example of the line to control the radial velocity:

```
rw      999   0   1    2.5     0     0     0  10.0  10.0   2.0  10.0  0.000000    0   0.    0.    0
```

4. Check the radial velocity results

The fit of the analysis results to radial velocity is recorded in fort.209. We have introduced how to check the fit (fort) files in the GSI User's Guide. Here we suggest user to check fort.209 file to get detailed information on bias, rms, and observation numbers for analysis.

9.2.1 Data Preprocessing Of Radar Radial Velocity Assimilation Within GSI

This section, drafted by Ming Sun, discusses how GSI does "super-obbing" to get radar velocity super-obs after reading the level II radial velocity BUFR file named as "*l2rwbufr*" and generates the new binary file called "*radar_supobs_from_level2*".

1. Introduction

A significant characteristic of radar observation is its high spatial and temporal resolution, which would also produce redundant information. Therefore, it is desirable to maximize whatever data compression the ensemble of radar observations allows, while minimizing

any degradation of the information content. The term for a surrogate datum that replaces several partially redundant actual data is a "super-observation" or "super-ob" (Alpert et al., 2006).
In GSI source code directory (./src/main), the file *read_l2bufr_mod.f90* reads the radar radial velocity data from the BUFR file *l2rwbufr*, does "super-obbing" and writes out the new super-obs to a new binary file named *radar_supobs_from_level2*.

2. Adaptable "Super-Ob" Parameters

In the GSI namelist, section *&SUPEROB_RADAR* is used to setup the spatial and temporal sizes of a super-ob box, the minimum number of samples needed to make a super-ob, the range of data used to construct super-obs and the logical flag to do "super-obbing" only. The following is a sample of the namelist section *&SUPEROB_RADAR*:

```
&SUPEROB_RADAR
   del_azimuth=5.,del_elev=.25,del_range=5000.,del_time=.5,elev_angle_max=5.,minnu
m=50,range_max=100000.,
    l2superob_only=.false.,
```

where *del_azimuth* is the azimuth range for super-ob box in units of degrees (default 5 degrees); the *del_elev* is the elevation angle range for super-ob box in units of degrees (default 0.25 degrees); the *del_range* is the radial range for super-ob box in units of meters (default 5km); the *del_time* is half of the time range for super-ob box in units of hours (default 0.5h); the *elev_angle_max* is the maximum elevation angle in units of degrees and the radar radial wind data above this elevation angle will not be used (default 5 degrees); the *minnum* is the minimum number of samples in a super-ob box needed to make a super-ob (default 50); the *range_max* is the maximum radial range to use in constructing super-obs in units of meters and the radar radial wind data out of this range will not be used (default 100km); the *l2superob_only* is the logical flag to do "super-obbing" only if set to true (default false).

The super-obs are still in the radar polar coordinate, and the values of the parameters above can define the bin numbers in azimuthal, radial and elevation directions. The bin number in the azimuthal direction (*nazbin*) is the nearest integer to 360 divided by *del_azimuth*, the bin number in the radial direction (*nrbin*) is the nearest integer to *range_max* devided by *del_range*, and the bin number in the elevation direction (*nelbin*) is the nearest integer to *elev_angle_max* devided by *del_elev*, so the total number of super-ob boxes for one radar *nthisrad* is *nrbin**nazbin*nelbin*.

3. Create A Radar Information Table

The GSI does an initial decoding of the BUFR file *l2rwbufr* to read in '`DIST125M`' and '`SSTN CLAT CLON HSMSL HSALG ANEL YEAR MNTH DAYS HOUR MINU SECO`' (refer to the BUFR table itself for more details).

- If the parameter *l2superob_only* is set to true, the radar observation time will be printed out into *stdout* file:

```
 create superobs only, radar file date = 'oyear' 'omonth' 'oday' 'ohour'
RADAR_BUFR_READ_ALL: analysis time is 'oyear' 'omonth' 'oday' 'ohour'
```

If the parameter *l2superob_only* is set to false (default), the analysis time of the background file and the radar observation time will be both printed out into *stdout* file:

```
 using restart file date = 'byear' 'bmonth' 'bday' 'bhour'
RADAR_BUFR_READ_ALL: analysis time is 'oyear' 'omonth' 'oday' 'ohour'
```

Users can use this information to check if the observation time is right and if the times match between the background and observation files.

- The 'DIST125M' data is used to determine the multiplying factor for radial distance. If the minimum difference of '`DIST125M`' between the adjacent gates is 1, the factor is set to 250
If the minimum difference of '`DIST125M`' between the adjacent gates is 2, the factor is set to 125.
If users see the following message in the *stdout* file:

```
RADAR_BUFR_READ_ALL:  problem with level 2 bufr file, gate distance scale
factor undetermined, going with 125
```

which means the minimum difference of '`DIST125M`' between the adjacent gates is neither 1 nor 2, but the multiplying factor for radial distance is still set to 125 and the process will still go on. This message only reminds the users that there might be some wrong '`DIST125M`' data.

- The GSI counts the radar number according to the radar station identifier '`SSTN`'

o The default maximum number of radars that GSI would deal with is 150, if users see the following message in the *stdout* file:

```
RADAR_BUFR_READ_ALL:  stop processing level 2 radar bufr file--increase
parameter max_num_radars
```

which means the radar numbers in the BUFR file exceed 150, if so, the parameter *max_num_radars* should be changed in the file *read_l2bufr_mod.f90*:

```
integer(i_kind),parameter:: max_num_radars=150
```

o If the radar number is less than or equal to zero, the message:

```
RADAR_BUFR_READ_ALL:  NO RADARS KEPT IN radar_bufr_read_all, continue without
level 2 data
```

will be printed in the *stdout* file and the "super-obbing" process will not be done.

o Because the reading process runs in a parallel mode, if the total radar number is greater than zero meanwhile less than the defined maximum radar number, the information of the minimum and maximum radar numbers processed by each core can be found in the *stdout* file:

```
 min,max num_radars=num_radars_min num_radars_max
```

- The unique master tables of all radar station identifier, latitude, longitude and height are created.

Table Name	Dimension	Content	Mnemonic
master_stn_table	total radar numbers in *l2rwbufr* file	radar station identifier	`SSTN`
master_lat_table		radar station latitude	`CLAT`
master_lon_table		radar station longitude	`CLON`
master_hgt_table		radar station height	`HSMSL+HSALG`

4. "Super-Obbing " Preprocessing

The GSI reopens and rereads the BUFR file *l2rwbufr* to read in '`SSTN YEAR MNTH DAYS HOUR MINU SECO ANAZ ANEL QCRW`' of a subset, and does following checks:

- If the elevation angle '`ANEL`' is higher than the defined maximum elevation angle *elev_angle_max*, the data '`DIST125M DMVR DVSW`' of this subset will not be read in. The parameter *nradials_fail_angmax* is used to counter the number of subsets which are above the maximum elevation angle.
- If the absolute value of the difference between the radar observation time and analysis time of the background file is larger than the defined time range *del_time*. The parameter *nradials_fail_time* is used to counter the number of subsets which are out of time range.
- The azimuth '`ANAZ`' is transferred into azimuth index *iazbin* which ranges from 1 to *nazbin*. If the calculated index is out of this range, the program will be stopped and the following message can be found in the *stdout* file:

  ```
  RADAR_BUFR_READ_ALL:  error in getting iazbin, program stops
  ```

  which means there must be some wrong azimuth data in the BUFR file *l2rwbufr*.
- The elevation angle '`ANEL`' will be transferred into elevation angle index *ielbin* which ranges from 1 to *nelbin*. If the calculated index is out of this range, the data '`DIST125M DMVR DVSW`' of this subset will not be read in. The parameter *nradials_fail_elb* is used to counter the number of subsets which are out of the elevation angle index range.
- The radar station identifier '`SSTN`' will be compared with the created radar information table *master_stn_table*. If there is no match station, the program will be stopped and the following message can be found in the *stdout* file:

  ```
  index error in radar_bufr_read_all -- program stops - 0 'stn_id'
  ```
  where *stn_id* is the wrong radar station identifier.

If the subset goes through all the checks above, GSI will read the '`DIST125M DMVR DVSW`' data which contain all the radar observations (number of gates) in a radial direction, and do the following 5 steps:

Step 1, the distance from antenna to gate center is calculated by the multiplying factor for radial distance multiplied by '`DIST125M`'. If the distance is greater than the maximum radial range *range_max*, the radar data in this gate will not be used. The parameter *nrange_max* is used to counter the number of gates which are out of the maximum radial range.

Step 2, if the radial velocity '`DMVR`' is greater than 100000, the radar data in this gate will not be used. The parameter *nobs_badvr* is used to counter the number of gates which have bad radial velocity data.

Step 3, if the velocity spectral width '`DVSW`' is greater than 100000, the radar data in this gate will not be used. The parameter *nobs_badsr* is used to counter the number of gates which have bad velocity spectral width data.

Step 4, the distance is transferred into distance index *irbin* which ranges from 1 to *nrbin*, if the distance index is out of this range, the radar data in this gate will not be used. The parameter *nobs_lrbin* is used to counter the number of gates which have the distance index less than 1 and nobs_hrbin is used to counter the number of gates which have the distance index greater than *nrbin*.

Step 5, the three-dimensional coordinate (*izabin*, *ielbin*, *irbin*) is transferred into one-dimensional coordinate *iloc*, using the formula:
iloc = nrbin(nazbin*(ielbin-1)+(iazbin-1))+irbin*
All the observations from the same radar at the same one-dimensional coordinate *iloc*, which means in the same super-ob box, are added up and the number of observations in the same super-ob box is counted. If the number of samples in a super-ob box is less than the defined minimum number *minnum*, the data of this super-ob box will not be used.

After doing these steps, all the statistical information is listed in the *stdout* file as shown in the following example:

```
RADAR_BUFR_READ_ALL:  num_radars_0 =            2
 master list radar   1 stn id,lat,lon,hgt,num = RSHI     31.01    121.89
44.0      4372
 master list radar   2 stn id,lat,lon,hgt,num = SHQP     31.08    120.96
42.0      6408
 RADAR_BUFR_READ_ALL:  ddiffmin,distfact,idups=   2.00000000000000
   125.000000000000                0
  nthisrad=       28800
  nthisbins=      172800
  timemin,max= -3.055555555555555E-002  2.527777777777778E-002
  nradials_in=        6554
  nradials_fail_angmax=        2897
  nradials_fail_time=           0
  nradials_fail_elb=          0
  nobs_in=     1469686
  nobs_badvr=          0
  nobs_badsr=         12
  nobs_lrbin=          0
  nobs_hrbin=          0
  nrange_max=      392524
 ielbin,histo_el=     1                  0
 ielbin,histo_el=     2             142854
```

```
    ielbin,histo_el=     3             85071
    ielbin,histo_el=     4                 0
    ielbin,histo_el=     5                 0
    ielbin,histo_el=     6            217218
    ielbin,histo_el=     7              8899
    ielbin,histo_el=     8                 0
    ielbin,histo_el=     9              6365
    ielbin,histo_el=    10            209058
    ielbin,histo_el=    11                 0
    ielbin,histo_el=    12                 0
    ielbin,histo_el=    13            125610
    ielbin,histo_el=    14             77976
    ielbin,histo_el=    15              3795
    ielbin,histo_el=    16                 0
    ielbin,histo_el=    17            131792
    ielbin,histo_el=    18             68512
    ielbin,histo_el=    19                 0
    ielbin,histo_el=    20                 0
```

where *num_radars_0* is the total number of radars, in the example there are radars in the BUFR file *l2rwbufr*; in this example, the 2 lines below list the detail information from each radar, which include radar station identifier (*RSHI* and *SHQP* respectively), latitude (*31.01°N* and *31.08°N* respectively), longitude (*121.89°E* and *120.96°E* respectively), height (*44.0m* and *42.0m*) and the number of useful super-ob boxes (*4372* and *6408* respectively); *ddiffmin* means the minimum difference of '`DIST125M`' between the adjacent gates, in this example *ddiffmin* is 2, so the multiplying factor for radial distance *distfact* is 125, and *idups* means the number of observations that the minimum difference of '`DIST125M`' between the adjacent gates equals zero (normally 0 as shown in this example); '*nthisrad = 28800*' means he total number of super-ob boxes for one radar is 28800 and *nthisbins* equals *nthisrad* multiplied by 6 (in this example *28800*6=172800*); '*timemin,max= -3.055555555555555E-002 2.527777777777778E-002*' means the minimum and maximum difference between the observation time and the analysis time of the background in units of hour; '*nradials_in=6554*' means the total number of subsets read from the BUFR file *l2rwbufr* is *6554*; '*nradials_fail_angmax=2897*' means there are *2897* subsets above the maximum elevation angle; '*nradials_fail_time=0*' means there is no subset out of the time range; '*nradials_fail_elb=0*' means there is no subset out of the elevation angle index range; '*nobs_in=1469686*' means the total number of gates read from the subsets is 1469686; '*nobs_badvr=0*' means there is no gate having bad radial velocity data; '*nobs_badsr=12*' means there are 12 gates having bad velocity spectral width data; '*nobs_lrbin= 0*' means there is no gate having distance index less than 1; '*nobs_hrbin=0*' means there is no gate having distance index greater than *nrbin*; '*nrange_max=392524*' means there are *392524* gates out of the maximum radial range; *ielbin* and *histo_el* are the elevation angle index and the total gates number of the elevation index respectively, in this example there are totally 20 elevation angle indexes and *217218* gates in the elevation angle index *6* (*ielbin,histo_el=6 217218*).

5. Create Super-Obs And Generate The radar_supobs_from_level2 File

The accumulated values of the same radar in the same super-ob box are divided by the number of samples in the super-ob box expect for the radars near the polar (radar station latitude is higher than 89.5 degrees). The variables include *thisrange* (radial range), *thisazimuth* (azimuth), *thistilt* (elevation angle), *thisvr* (radial velocity), *thisvr2* (the square of radial velocity), *thistime* (time difference between observation and background). An additional variable *thiserr* is calculated according to the following formula:

$$thiserr = \sqrt{\left|\overline{V_r^2} - \overline{V_r}^2\right|}$$

The variable *thishgt* (height of the super-obs box) is also calculated. Then the elevation angle, radial distance and azimuth are corrected and written into *corrected_tilt*, *gamma* and *corrected_azimuth* respectively. Meanwhile, *thislat* and *thislon* (the latitude and longitude of the super-obs box) are calculated.
So all the variables listed below for each super-ob box are written into a new binary file named *radar_supobs_from_level2*.

Variable	Meaning
this_staid	radar station identifier
this_atalat	radar station latitude
this_stalon	radar station longitude
this_stahgt	radar station height
thistime	time difference between observation and background
thislat	super-ob box latitude
thislon	super-ob box longitude
thishgt	super-ob box height
thisvr	mean radial velocity
corrected_azimuth	corrected azimuth
thiserr	mean radial velocity errorr
corrected_tilt	corrected elevation angle

Finally, some information can be found in the *stdout* file. Below is an example using the same data in Section 4:

```
for radar RSHI nsuper=        4372  delazmmax=  0.531495369516961
vrmin,max=  -20.9300000000000        20.9700000000000       errmin,max=
0.309294787065859        14.6110084866894
deltiltmin,max=  2.239541454977478E-002  0.663221332668083
deldistmin,max=  -326.328815013534      -0.148169009099547
for radar SHQP nsuper=        6408  delazmmax=  0.530711027335997
vrmin,max=  -24.4200000000000        22.9650000000000       errmin,max=
0.298880185862377        21.7435944590585
deltiltmin,max=  2.102455505536138E-002  0.665819075690249
deldistmin,max=  -303.381730881694      -9.666676340202685E-002
```

Because there are two radars in this case, the statistical information of each radar is listed. Take *RSHI* radar as an example:

- '*nsuper=4372*' means there are *4372* super-ob boxes from *RSHI* radar used in this example
- '*delazmmax=0.531495369516961*' means the maximum corrected value of the azimuth is *0.531495369516961* degrees
- '*vrmin,max= -20.9300000000000   20.9700000000000*' means the minimum and maximum values of the mean radial velocity are *-20.93*m/s and *20.97*m/s respectively
- '*errmin,max= 0.309294787065859   14.6110084866894*' means the minimum and maximum values of the mean radial velocity error are *0.309294787065859*m/s and *14.6110084866894m/s* respectively

- '*deltiltmin,max= 2.239541454977478E-002 0.663221332668083*' means the minimum and maximum corrected values of the elevation angle are *2.239541454977478E-002* degrees and *0.663221332668083* degrees respectively
- '*deldistmin,max= -326.328815013534   -0.148169009099547*' means the minimum and maximum corrected values of the radial distance are *-326.328815013534*m and -*0.148169009099547*m respectively

```
total number of superobs written=       10780
 vrmin,maxall=  -24.4200000000000        22.9650000000000
errmin,maxall=  0.298880185862377        21.7435944590585
delazmmaxall=  0.531495369516961
deltiltmin,maxall=  2.102455505536138E-002  0.665819075690249
deldistmin,maxall=  -326.328815013534      -9.666676340202685E-002
```

The statistical information of all the radars are also listed in the *stdout* file:

- '*total number of superobs written=10780*' means there are totally *10780* super-ob boxes used in this example
- '*vrmin,maxall=-24.4200000000000 22.9650000000000*' means the totally minimum and maximum values of the mean radial velocity are *-24.42*m/s and *22.965*m/s respectively
- '*errmin,maxall=0.298880185862377 21.7435944590585*' means the totally minimum and maximum values of the mean radial velocity error are *0.298880185862377*m/s and *21.7435944590585m/s* respectively
- '*delazmmaxall=0.531495369516961*' means the totally maximum corrected value of the azimuth is *0.531495369516961* degrees
- '*deltiltmin,maxall=2.102455505536138E-002 0.665819075690249*' means the totally minimum and maximum corrected values of the elevation angle are *2.102455505536138E-002* degrees and *0.665819075690249* degrees respectively
- '*deldistmin,maxall=-326.328815013534 -9.666676340202685E-002*' means the totally minimum and maximum corrected values of the radial distance are -*326.328815013534*m and *-9.666676340202685E-002*m respectively

Reference

Alpert J C, Kumar V K. Radial wind super-obs from the WSR-88D radars in the NCEP operational assimilation system[J]. Monthly weather review, 2007, 135(3): 1090-1109.

9.2.2 The Processes Of The read_radar.f90 Code

This section was drafted by Ming Sun.

1. Check if radar wind files exist. If none exist, exit this routine.
   The files include *'radar_supobs_from_level2'*, the level 2.5 and 3 super-obs files, *'tldplrbufr'* and *'tldplrso'* files

2. Set some parameters:

```
vad_leash=0.3 (used in VAD QC)
xscale=20000 (horizontal scale, unit: meters)
maxvadbins=15 (the maximum of VAD levels)
dzvad=304.8 (vad reports are every 1000 ft = 304.8 meters)
```

   The information of these parameters will be listed in *stdout* file:

```
READ_RADAR:  set vad_leash,xscale=  0.300000000000000   20000.0000000000
READ_RADAR:  set maxvadbins,maxbadbins*dzvad=     15   4572.00000000000
```

3. Open BURF file *'vadfile'*(which is given in GSI namelist under &OBSQC section) which includes VAD winds and read in all VAD winds so that radar data can be decided to keep or not using VAD wind quality marks
   If the *'vadfile'* file does not exit the program will still go on and users will see the information in *stdout* file:

```
READ_RADAR: nsuper2_in,nsuper2_kept=       12482       0
READ_RADAR: # no vad match   =           12482
```

   It tells you that all the observations have no VAD wind to match, and no observation is kept.
   If the file *'vadfile'* exists and reads the first message correctly, a line will be found in stdout file:

```
READ_RADAR:  first read vad winds--use vad quality marks to qc 2.5/3 radar
winds
```

4. Find out whether the VAD data is in the BUFR file according to subtype(224) or type in the *'convinfo'* file and only read VAD wind data in the BUFR file.
   There is also a time check, the VAD wind data outside the time window will not be read.
   For 3DVAR, the time window is set by both *'twindow'* in the *'convinfo'* file and half an hour.

5. Create VAD wind information table
   If the latitude and longitude of a new VAD wind station is less than 0.1 degrees away from a VAD wind station reading before, then it will be considered as the same VAD wind station, otherwise, the information of a new VAD wind station will be stored.
   The parameter *nvad* counts the number of VAD wind stations, if it exceeds the *maxvad*(default value is 500) defined in the program, the program will stop and the error will be printed in the *stdout* file:

```
READ_RADAR:  ***ERROR*** MORE THAN 500 RADARS:  PROGRAM STOPS
```

   which means the VAD wind station numbers in the BUFR file exceed 500, if so, the parameter *maxvad* should be changed in the file *read_radar.f90*:

```
integer(i_kind),parameter:: maxvad=500
```

The VAD wind information table is created

Table Name	Dimension	Content
vadlon	nvad	VAD station longitude
vadlat		VAD station latitude
vadid		VAD station identifier

6. Update vadqm table
   If levels of the VAD data (*levs*) are greater than *maxlevs*(default value is 1500) defined by the program, the program will stop and the error will be printed in the *stdout* file:

```
READ_RADAR:  ***ERROR*** increase read_radar bufr size since number of
levs='levs' > maxlevs=1500
```

which means the VAD wind levels beyond 1500, if so, the parameter *maxlevs* should be changed in the file read_radar.f90:

```
integer(i_kind),parameter:: maxlevs=1500
```

If it is a new VAD wind station (the logical flag *'newvad'* from read_prepbufr.f90), the vadqm table will be updated according to the difference (diffuu, diffvv) between VAD wind observation (U_{vad},V_{vad}) and background wind (U_{bk},V_{bk}).

$$diffuu = U_{vad} - U_{bk}$$
$$diffvv = V_{vad} - V_{bk}$$

If $\sqrt{diffuu^2 + diffvv^2} > 10.0$, the VAD data will not be used.
If $|diffvv| > 8.0$, the VAD data will not be used.
If $|diffvv| > 5.0$ and $zob < 5000.0$, the VAD data will not be used (*zob* is the height of VAD wind observation).
If $zob > 7000.0$, the VAD data will not be used.

Translate *zob* (the height of VAD wind observation) into index *ivadz* (the height divided by *dzvad* which defined before, default value is *304.8* meters). If *ivadz* is less than 1 or greater than *maxvadbins* (default value is *15*), the VAD data will not be used.

Varible	Dimension	Content
errzmax	*1*	The maximum difference between observation height and the nearest VAD level height
vadqm	*(nvad, levs)*	The maximum value of *WQM* (VAD U-, V-component wind quality marker) of a VAD wind station at a level

vadqmmax	*1*	The maximum value of the *vadqm* array
vadqmmin	*1*	The minimum value of the *vadqm* array
vadu	*(nvad, levs)*	Add all the VAD U-component wind up at the same VAD station and the same level
vadv	*(nvad, levs)*	Add all the VAD V-component wind up at the same VAD station and the same level
vadcount	*(nvad, levs)*	Count the numbers at the same VAD station and the same level

7. Print vadwnd table
vadu and *vadv* are divided by *vadcount* at the same VAD station and the same level so that the average U-, V-component wind at every level of every station are obtained.
The VAD wind table will be printed in *stdout* file as follows:

```
n,lat,lon,qm=  1   31.08  120.96   -9 -9  2  2  2  2  2  2  2  2  2  2  2  2
2
……
errzmax=   48.0000000000000
```

where *n* is the serial number of VAD wind station, *lat* and *lon* are the latitude and longitude of the VAD wind station, *qm* is the maximum value of *WQM* of this VAD wind station at every level. There should be *nvad* lines in the *stdout* file.
The maximum difference between observation height and the nearest VAD level height *errzmax* in the unit of meters is also listed in the *stdout* file.

8. Open and read the binary file '*radar_supobs_from_level2*' which contains super-obs

All the variables in '*radar_supobs_from_level2*' file are listed below

Variable	Meaning
this_staid	radar station identifier
this_atalat	radar station latitude
this_stalon	radar station longitude
this_stahgt	radar station height
thistime	time difference between observation and background
thislat	super-ob box latitude
thislon	super-ob box longitude
thishgt	super-ob box height
thisvr	mean radial wind
corrected_azimuth	corrected azimuth
thiserr	mean radial wind errorr
corrected_tilt	corrected elevation angle

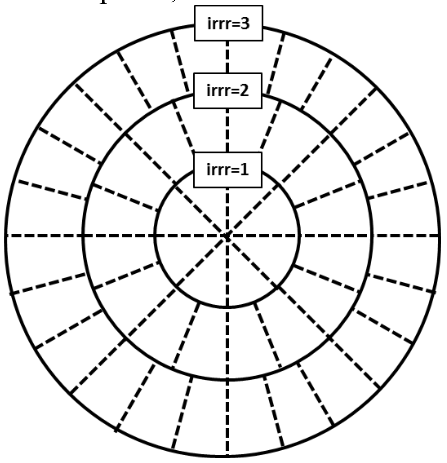
nsuper2_in is used to count the total number of the super-obs read from the binary file.

If the GSI is run under regional mode and the location of the radar is outside the region, the super-obs will not be read but this super-ob is still counted in *nsuper2_in*. *dlatmax*, *dlonmax*, *dlatmin* and *dlonmin* are used to store the maximum and minimum grid-relative latitude and longitude of all the radar stations.

9. Find match VAD wind station for every super-ob radar station according to the distance between the two stations.
   If the distance between the VAD wind station and the super-ob radar station is less than 0.2 degrees, they are matched up.
   numhits (dimension of *nvad*)counts the number of super-obs matched for every VAD wind station.
   novadmatch counts the number of super-obs which have no match VAD wind station and if a super-ob has no matched VAD wind station, the super-ob data will not be used.
   If the GSI is run under regional mode and the location of the matched VAD wind station is outside the region, the super-ob data will not be used.
   If the time difference between observation and background of the super-ob is larger than the time window, it will not be used. For 3DVAR, the time window is half an hour.

10. If the GSI is run under regional mode and the location of the super-ob observation is outside the region, the super-ob data will not be used.
    Compute the distance between the super-ob observation and the radar station, and transform it into the distance index (*irrr*) according to *xscale* defined before (the default value is *20000* meters). If *irrr* less than one or greater than *max_rrr*, which is the integer of *100000.0* devided by *xscale*, the data will not be used, which means the super-ob observations should be within 100km away from the radar station.

11. Calculate the azimuth index *iaaa*, which depends on the distance index (*irrr*). As shown below, the azimuth is divided into *8* parts when *irrr* equals *1*, and *16* parts when *irrr* equals *2*, *24* parts when *irrr* equal *3*, and so on.



    iaaamax and *iaaamin* are the maximum and minimum of the observation azimuth index respectively.

12. Calculate the observation error (*error*)

Observation error (*error*) equals mean radial wind error (*thiserror*) multiplied by a factor (*erradar_inflate*).
erradar_inflate is defined in the *qcmod.f90* code, meaning radar error inflation factor and the default value is *one*.
errmax and errmin are the maximum and minimum (greater than zero) of the observation error respectively.

13. Perform limited QC based on azimuth angle, radial wind speed, distance from radar site, elevation of radar, height of observation, observation error
    - If the azimuth angle is greater than 400 degrees, the data is considered as a bad data. *ibadazm* is used to count the numbers of bad azimuth angle data.
    - If the radial wind is greater than 200 m/s, the data is considered as a bad data. *ibadwnd* is used to count the numbers of bad radial wind data.
    - If the distance between the super-ob observation and the radar station is greater than 400 meters, the data is considered as a bad data. *ibaddist* is used to count the numbers of bad distance data.
    - If the radar station height is lower than -1000 meters or higher than 50000 meters, the data is considered as a bad data. *ibadstaheight* is used to count the numbers of bad radar station height data.
    - If the super-ob observation height is lower than -1000 meters or higher than 50000 meters, the data is considered as a bad data. *ibadheight* is used to count the numbers of bad observation height data.
    - If the super-ob observation height is lower than the radar station height, the data is considered as a bad data. *iheightbelowsta* is used to count the numbers of observation height lower than radar station height data.
    - If the mean radial wind error is greater than 6 or no more than 0, the data is considered as a bad data. *ibaderror* is used to count the numbers of bad mean radial wind error data.

    notgood0 is used to count the total number of bad data mentioned above. And if the data is a bad data, the checks below will not be done.

14. Check fit to VAD wind and VAD wind quality mark
    - Transform the super-ob observation height into index *ivadz* (the height divided by *dzvad* which defined before, default value is *304.8* meters). If *ivadz* is less than 1 or greater than *maxvadbins* (default value is *15*), the data is considered as a bad data and the checks below will not be done. *ioutofvadrange* is used to count the numbers that out of the VAD height range data.
    - Calculate some variables as follows:
        - weight:
          $thiswgt = \dfrac{1}{\max(4.0, thiserr^2)}$, where *thiserr* is mean radial wind error of the super-ob
        - square of the radial wind difference between VAD and super-ob observation:
          $thisfit2 = (VADvr - thisvr)^2$, where *VARvr* is the radial wind calculated from VAD U,V-component wind and *thisvr* is mean radial wind of the super-ob
        - square root of thisfit2:

$thisfit = \sqrt{thisfit2}$

- speed of VAD wind:
  $thisvadspd = \sqrt{VADu^2 + VADv^2}$, where $VADu$ and $VADv$ are the VAD U,V-component wind respectively
- *vadfit2* is used to add all the *thiswgt***thisfit2* up
- *vadcount2* is used to count the number
- *vadwgt2* is used to add all the *thiswgt* up

If the ratio $\frac{thisfit}{\max(1, thisvadspd)}$ is larger than *vad_leash* defined before (the default value is 0.3), the data is considered as a bad data. *ibadfit* is used to count the number of these bad fit data.
This check is commented out in comGSI_v3.3!

- Thin out the data
  For the same distance index (*irrr*), azimuth angle index (*iaaa*), height index (*ivadz*) and VAD wind station index (*ivad*), if the number of super-ob observation is more than *nboxmax* (the default value is *one*), the data is thinned out. *kthin* is used to count the number of thinned out data.
- VAD wind quality mark check
  If the maximum value of *WQM* (VAD U-, V-component wind quality marker) of a VAD wind station at a level is greater than 3.5 or less than -1, the data is considered as a bad data. *ibadvad* is used to count the number of the bad VAD wind quality marker data.
  This check is commented out in comGSI_v3.3!

15. If the data passed all the checks above, then load it into output array
    nsuper2_kept is used to count the total number of the kept good data.
    level2 is used to count the number of the kept good data for each VAD wind station.
    nobs_box is used to count the number of the kept good data for each thinned box.
    notgood is used to count the number of the bad data which does not fit to VAD wind and VAD wind quality mark.

Output Array Member	Variable	Meaning
cdata(1)	*error*	wind observation error (m/s)
cdata(2)	*dlon*	grid relative longitude
cdata(3)	*dlat*	grid relative latitude
cdata(4)	*height*	observation absolute height (m)
cdata(5)	*rwnd*	radial wind observation (m/s)
cdata(6)	*azm*deg2rad*	azimuth angle (radians)
cdata(7)	*t4dv*	observation time (hour)
cdata(8)	*ikx*	observation type
cdata(9)	*tiltangle*	tilt angle (radians)

cdata(10)	*staheight*	station elevation (m)
cdata(11)	*rstation_id*	station id
cdata(12)	*usage*	usage parameter
cdata(13)	*idomsfc*	dominate surface type
cdata(14)	*skint*	skin temperature
cdata(15)	*ff10*	10 meter wind factor
cdata(16)	*sfcr*	surface roughness
cdata(17)	*dlon_earth*rad2deg*	earth relative longitude (degrees)
cdata(18)	*dlat_earth*rad2deg*	earth relative latitude (degrees)
cdata(19)	*dist*	range from radar in km (used to estimate beam spread)
cdata(20)	*zsges*	model elevation at radar site
cdata(21)	*thiserr*	mean radial wind errorr
cdata(22)	*two*	

16. Finally, some information can be found in the *stdout* file. Below is an example:

```
READ_RADAR:  level 2 superobs: reached eof on 2/2.5/3 superob radar file
 READ_RADAR: nsuper2_in,nsuper2_kept=       12482       10704
 READ_RADAR: # no vad match   =           0
 READ_RADAR: # out of vadrange=           0
 READ_RADAR: # bad azimuths=          0
 READ_RADAR: # bad winds   =          0
 READ_RADAR: # bad dists   =          0
 READ_RADAR: # bad stahgts =          0
 READ_RADAR: # bad obshgts =          0
 READ_RADAR: # bad errors  =        372
 READ_RADAR: # bad vadwnd  =          0
 READ_RADAR: # bad fit     =          0
 READ_RADAR: # num thinned =          0
 READ_RADAR: # notgood0    =        372
 READ_RADAR: # notgood     =          0
 READ_RADAR: # hgt belowsta=          0
 READ_RADAR: timemin,max   =  4.940656458412465E-324  4.940656458412465E-324
 READ_RADAR: errmin,max    =  0.198997487421342        26.2045158189491
 READ_RADAR: dlatmin,max,dlonmin,max=   200.880606608956
   203.096439643754        230.469249250455        260.028462165266
 READ_RADAR: iaaamin,max,8*max_rrr  =           1           40          40
```

- '*nsuper2_in,nsuper2_kept= 12482 10704*' means there are totally *12482* super-ob boxes reading in this example, and *10704* super-ob boxes are kept after all the checking process.
- *'# no vad match = 0'* tells users how many super-ob boxes have no match VAD wind station (refer to *novadmatch*).
- *'# out of vadrange= 0'* tells users how many super-ob boxes are out of the VAD height range (refer to *ioutofvadrange*).
- *'# bad azimuths= 0'* tells users how many super-ob boxes have bad azimuth angle

(refer to *ibadazm*).

- *'# bad winds = 0'* tells users how many super-ob boxes have bad bad radial wind (refer to *ibadwnd*).
- *'# bad dists = 0'* tells users how many super-ob boxes have bad distance (refer to *ibaddist*).
- *'# bad stahgts = 0'* tells users how many super-ob boxes have bad radar station height (refer to *ibadstaheight*).
- *'# bad obshgts = 0'* tells users how many super-ob boxes have bad observation height (refer to *ibadheight*).
- *'# bad errors = 372'* tells users how many super-ob boxes have bad mean radial wind error (refer to *ibaderror*).
- *'# bad vadwnd = 0'* tells users how many super-ob boxes have bad VAD wind quality marker (refer to *ibadvad*).
- *'# bad fit = 0'* tells users how many super-ob boxes are bad fit data (refer to *ibadfit*)
- *'# num thinned = 0'* tells users how many super-ob boxes are thinned out (refer to *kthin*)
- *'# notgood0 = 372'* tells users how many super-ob boxes have not passed the limited QC based on azimuth angle, radial wind speed, distance from radar site, elevation of radar, height of observation, observation error (refer to *notgood0*)
- *'# notgood = 0'* tells users how many super-ob boxes do not fit to VAD wind and VAD wind quality mark (refer to *notgood*)
- *'# hgt belowsta= 0'* tells users how many super-ob boxes have height lower than radar station height (refer to *iheightbelowsta*)
- *'timemin,max = 4.940656458412465E-324 4.940656458412465E-324'* means the minimum and maximum observation time respectively.
- *'errmin,max = 0.198997487421342 26.2045158189491'* means the minimum and maximum of the observation error respectively
- *'dlatmin,max,dlonmin,max= 200.880606608956 203.096439643754 230.469249250455 260.028462165266'* means the minimum and maximum grid-relative latitude and longitude of all the radar stations
- *'iaaamin,max,8*max_rrr = 1 40 40'* means the minimum, maximum of the observation azimuth index and 8 times the maximum azimuth index (refer to *iaaamin*, *iaaamax*, *max_rrr*)

9.3 Analyze Radar Reflectivity With GSI

After get the radar reflectivity BUFR file ready for GSI, users need to go through the following steps to setup GSI reflectivity analysis.

1. Compile with GSD cloud analysis

Reflectivity observations are used with the GSI in GSD cloud analysis. To open the cloud analysis in the GSI, users need to add the following bold conditional compiling option in "*configure.gsi*" file:

```
CPP_FLAGS      =  -C -P -D_REAL8_ -DWRF -DLINUX -DRR_CLOUDANALYSIS
```

2. Setup GSI namelist for radial velocity analysis

In GSI namelist section "*OBS_INPUT*", a line needs to be set to let GSI know the name of the radar reflectivity:

```
dfile(88)='refInGSI',  dtype(88)='rad_ref',   dplat(88)=' ',      dsis(88)='rad_ref',          dval(88)=1.0, dthin(88)=0, dsfcalc(88)=0,
```

Please note the total observation files number in namelist section SETUP need to be add 1:

```
ndat=original number + 1
```

The namelist options to control GSD cloud analysis, including the reflectivity analysis, are in section RAPIDREFRESH_CLDSURF. Please check Appendix A for the detailed explanation of the options in the RAPIDREFRESH_CLDSURF section.

3. Link the radial velocity BUFR file to GSI run directory in run scripts

After add GSI namelist for reflectivity, a new link need to be added in the GSI run scripts to link the reflectivity BUFR file to GSI working directory with the name setup in the GSI OBS_INPUT section:

```
ln –s "the patch and name of reflectivity BUFR file" refInGSI'
```

4. Setup convinfo for reflectivity

As other conventional observations and radial velocity, GSI uses "*convinfo*" file to control the data usage of each observation type. Please check GSI user's guide for details of "convinfo", here is an example of the line to contral the reflectivity:

```
rad_ref 999   0   1    1.5     0     0     0  7.0   5.6   1.3  10.0  0.000000     0   0.    0.     0
```

5. Setup anavinfo for reflectivity

Reflectivity is analyzed as part of the GSD cloud analysis. To open the GSD cloud analysis, users also need to make the following changes to the met_guess section of the anavinfo_arw_netcdf in fix files:

```
met_guess::
!var     level    crtm_use    desc              orig_name
  cw       30      10         cloud_condensate  cw
  ql       30      10         cloud_liquid      ql
  qi       30      10         cloud_ice         qi
  qr       30      10         rain              qr
```

```
    qs      30      10          snow            qs
    qg      30      10          graupel         qg
    qnr     30      10          rain_noconc     qnr
  ::
```

6. Check the reflectivity analysis results

Because the reflectivity is not analzed with the variational method, there is no fit files for the reflectivity. But users still can use the *stdout* file to find if the reflectivity is used in the analysis.

- Check data distribution in stdout to look for line:

```
      OBS_PARA: rad_ref                 ????      ????      ????      ????
```

- Check the line after minimization:

```
========================================
gsdcloudanalysis: Start generalized cloud analysis
========================================
```

- Check analysis increment for rain and snow mixing ratio

9.4 Information On Radar Data Quality Control

Radar data quality control is not discussed in this document because of the complexity of the problem. Users can check Shun Liu's slides in the 2010 Summer Community GSI residential Tutorial on radar data assimilation for quality control steps conducted in radial velocity process.

Chapter 10 GSI Applications

10.1 Introduction To Hybrid 4-Dimensional Ensemble-Variational Analysis

The 4-Dimensional ensemble-variational analysis is the newly implemented feature of the GSI-Hybrid system. It takes advantage of the time varying ensembles and first guess fields so that the GSI analysis can get the flow-dependent background error information of different time levels.

It is an upgrade on the hybrid ensemble-3DVAR analysis. To run the 4-D GSI-Hybrid analysis using the multiple time level GFS ensembles, some additional changes are required from the hybrid ensemble-3DVAR analysis, in both the namelist and the run script:

Change 1: Link the ensemble members to the GSI run directory

This change is to link the GFS ensemble members of different time levels (usually three time levels) to the GSI run directory. The current implementation can only accept the GFS ensemble forecasts, which is corresponding to the namelist variable *regional_ensemble_option=1*. Using an WRF ARW 4-D GSI-hybrid analysis case with three time levels and 6-hour time window as an example, the following lines are needed in the run script to link the GFS ensembles:

```
 # ensemble initial time is 6 hours earlier than the analysis time
 m6date=`${HOME}/bin/da_advance_time.exe ${ANAL_TIME} -6 `

 # set the ensemble path and size
 GFSENS=/where/your/GFS/ensemble/is
 ENSEMBLE_SIZE=80

 ## locate the GFS 6-hour ensemble files for hybrid analysis
 n=1
 m=0
 >filelist06
 while [[ $n -le ${ENSEMBLE_SIZE} ]]; do
 if [[ -s ${GFSENS}/${m6date}/$( printf sfg_${m6date}_fhr06s_mem%03d $n ) ]];
then
   m=$(($m + 1))
   ln -sf ${GFSENS}/${m6date}/$( printf sfg_${m6date}_fhr06s_mem%03d $n )  \
   ./$( printf sfg_${m6date}_fhr06s_mem%03d $m )
   ls ./$( printf sfg_${m6date}_fhr06s_mem%03d $m ) >> filelist06
 fi
 n=$(($n + 1))
 done

## locate the GFS 3-hour ensemble files for hybrid analysis
 n=1
 m=0
 >filelist03
 while [[ $n -le ${ENSEMBLE_SIZE} ]]; do
 if [[ -s ${GFSENS}/${m6date}/$( printf sfg_${m6date}_fhr03s_mem%03d $n ) ]];
then
```

```
   m=$(($m + 1))
   ln -sf ${GFSENS}/${m6date}/$( printf sfg_${m6date}_fhr03s_mem%03d $n )  \
   ./$( printf sfg_${m6date}_fhr03s_mem%03d $m )
   ls ./$( printf sfg_${m6date}_fhr03s_mem%03d $m ) >> filelist03
 fi
 n=$(($n + 1))
 done

## locate the GFS 9-hour ensemble files for hybrid analysis
 n=1
 m=0
 >filelist09
 while [[ $n -le ${ENSEMBLE_SIZE} ]]; do
 if [[ -s ${GFSENS}/${m6date}/$( printf sfg_${m6date}_fhr09s_mem%03d $n ) ]];
then
   m=$(($m + 1))
   ln -sf ${GFSENS}/${m6date}/$( printf sfg_${m6date}_fhr09s_mem%03d $n )  \
   ./$( printf sfg_${m6date}_fhr09s_mem%03d $m )
   ls ./$( printf sfg_${m6date}_fhr09s_mem%03d $m ) >> filelist09
 fi
 n=$(($n + 1))
 done
```

Change 2: Copy the first guess fields to the GSI run directory

This change is to link the first guess fields of different time levels (usually three time levels) to the GSI run directory. Using an WRF ARW 4-D GSI-hybrid analysis centered at 2014080906 with three time levels and 6-hour time window as an example, the flowing lines are used in the run script to locate and copy the first guess fields:

```
ANAL_TIME=2014080906
HH=`echo $ANAL_TIME | cut -c9-10`
WRF_TIME06=`${HOME}/bin/da_advance_time.exe ${ANAL_TIME} 0 -w `
WRF_TIME03=`${HOME}/bin/da_advance_time.exe ${ANAL_TIME} -3 -w `
WRF_TIME09=`${HOME}/bin/da_advance_time.exe ${ANAL_TIME} +3 -w `
     … …
BK_FILE03=${BK_ROOT}/wrfout_d01_${WRF_TIME03}
BK_FILE06=${BK_ROOT}/wrfout_d01_${WRF_TIME06}
BK_FILE09=${BK_ROOT}/wrfout_d01_${WRF_TIME09}
     … …
cp ${BK_FILE06} ./wrf_inout
cp ${BK_FILE03} ./wrf_inou3
cp ${BK_FILE06} ./wrf_inou6
cp ${BK_FILE09} ./wrf_inou9
```

Change 3: Set up the namelist options in section SETUP

Users need to set `l4densvar=.true.,` to turn on 4-D hybrid ensemble analysis. Users also need add `nhr_obsbin=3` if three time levels are used for the analysis.

After setup of the namelist parameters and the path and name of the ensemble members and the first guess fields, GSI can be run following the same way as the GSI hybrid 3-D ensemble-variational analysis. And the same procedures could be followed as in the previous sections to check the run status and diagnose the GSI analysis.

10.2 Introduction to RTMA Analysis

The Real-Time Mesoscale Analysis (RTMA) is a NOAA/NCEP high-spatial and temporal resolution analysis/assimilation system for near-surface weather conditions. Its main component is the NCEP/EMC Gridpoint Statistical Interpolation (GSI) system applied in two-dimensional variational mode to assimilate conventional and satellite-derived observations. The RTMA produces analyses of 2-m temperature, 2-m specific humidity, 2m-dew point temperature, 10-m winds, 10-m wind gust, surface pressure, and surface visibility.

The RTMA was developed to support the National Digital Forecast Database (NDFD) operations and provide field forecasters with high quality analyses for nowcasting, situational awareness, and forecast verification purposes. Presently, the system produces hourly, real-time analyses for the 5-km and 2. 5-km resolution CONUS NDFD grids, 6-km Alaska NDFD grid and 2.5-km Hawaii, Puerto-Rico and Guam NDFD grids.

RTMA fields for the CONUS are displayed at:

http://mag.ncep.noaa.gov/

In this section, we will introduce how to run the RTMA system. The whole RTMA system includes three components:

1. Prepare first guess file
2. Run GSI in RTMA mode
3. RTMA post-process

10.2.1. Prepare First Guess File

The major function of the RTMA is to create a high- resolution 2D near surface analysis. The background file of the RTMA GSI is an unformatted binary file that includes a set of 2 dimensional surface fields. There are no forecast files that can be directly used as its background. For the community RTMA GSI, the background file can be generated using a tool in the release community GSI package, which includes the code under directory *./util/RTMA/rtma_firstguess* and a run script: *./ util/RTMA/ rtma_getguess.sh* .

1. Compile the code

The code in the directory *./util/RTMA/rtma_firstguess* will produce an executable for generating RTMA GSI first guess (background). Because the dimension of the analysis domain, and the needed navigational information (eg., longitude and latitude of the southwestern most point and grid spacing for Lambert-Conformal grids) are hardwired in the code, users need to edit the code for the specific domain:

1) get into directory *./util/RTMA/rtma_firstguess;*
2) open file "*param.incl*" ;
3) find the following lines (starts from line 94):

```
!==>parameter definition for dtc
        integer(4),parameter::nx_dtc=758
        integer(4),parameter::ny_dtc=567

        real(8),parameter::alat1_dtc=21.138000_8
        real(8),parameter::elon1_dtc=237.280000_8
        real(8),parameter::da_dtc=13545.09_8
```

4) modify the values to fit the user's specific domain:

`nx_dtc`:	analysis domain dimension in X direction
`ny_dtc` :	analysis domain dimension in Y direction
`alat1_dtc` :	analysis domain latitude of southwestern most point
`elon1_dtc` :	analysis domain longitude of southwestern most point
`da_dtc` :	analysis grid space in meters

After setting the right analysis grid configuration, edit the "makefile" inside the same directory and put the right location of the GSI root directory in a line:

```
GSIDIR=comGSI/releaseV33/release_V3.3_intel.12-12.0
```

Please note that this tool has to be compiled after the compilation of the community GSI. Users also need to pick the following part for PGI or Intel compiler:

For Intel compiler, pick:
```
FC=ifort
FFLAGS=-nofixed  -convert big_endian
```

For PGI compiler, pick:
```
FC=pgf90
FFLAGS= -Mfree -byteswapio
```

Then, in the same directory, compile the code using the command:

```
./make
```

The successful compilation should give a new executable in the directory named:

`rtma_firstguess.exe`

If user needs to clean the code for recompilation, use command:

```
./make clean
```

2. Using run scripts to generate first guess for RTMA GSI

The generation of background (first guess) files for RTMA is controlled by the script "*rtma_getguess.sh*" in directory "*./util/RTMA*". Users need to setup the following parameters for "rtma_getguess.sh":

```
ROOTDIR= comGSI/releaseV33/release_V3.3_intel.12-12.0/util/RTMA
FGFILE= 2012052811/postprd/wrftwo_rr_01.grib1
work_dir=${ROOTDIR}/rtmagus
CYCLE=2012052811
```

Where

- `ROOTDIR`: full directory for *./uitl/RTMA*
- `FGFILE`: background file, which is a two-dimension grib file from uni-post.
- `work_dir`: work directory
- `CYCLE`: analysis time

This run script can be run in front node directly using:

./rtma_getguess.sh

In this script, command "*wgrib*" is used to extract the surface fields out from the 2D grib file "*wrftwo_rr_01.grib1*" and save these fields into a file called "*slabs.dat*". Then this file and a binary file called "*rtma_dtc_latlon_mpfactor_slmask.data*" under directory *util/RTMA/fix* are read in and processed. Finally, a set of 2D fields are written into a binary file called "*twodvar_input_bi*" to be used as the RTMA background file.

Users should be aware that running a domain other than the Rapid Refresh (RAP) case in the example may require additional modifications to be sure the appropriate surface fields are present in the 2D grib file and the binary files are appropriate for the domain of interest. See the following section (3. Binary file structure) for more information.

After running the script, the run directory *(./RTMA/rtmagus*) for first guess generation should look like:

```
bigrjlist.txt       mass_rjlist.txt_static   slabs2_nobiasc.dat
cycledate           parm_ndfd_time_namelist  slabs.dat
first_guess.grib1   p_rejectlist             stdout.rtma_getguess
fort.20             p_rjlist.txt_static      t_rejectlist
fort.30             q_rejectlist             t_rjlist.txt_static
fort.88             q_rjlist.txt_static      twodvar_input_bi
fort.9              rtma_slmask.dat          w_rejectlist
gridname_input      rtma_terrain.dat         w_rjlist.txt_static
mass_rejectlist     slabs2.dat
```

The following is a list of important files in this run directory:

- *first_guess.grib1*: 2D grib file from uni-post
- *slabs.dat*: binary file including 2D fields extracted from first_guess.grib1 using wgrib command.
- *parm_ndfd_time_namelist*: namelist holding analysis time
- *gridname_input*: namelist holding analysis grid configuration
- *twodvar_input_bi*: RTMA first guess, binary file.
- *stdout.rtma_getguess*: standard output

3. Binary file structure

The binary file "*rtma_dtc_latlon_mpfactor_slmask.data*" is a fix file that includes map factor, grid latitude, grid longitude, and land mask information from the goegrid file. They are 2D real arrays arranged in the following order:

```
mapfac(nx,ny)
glat(nx,ny)
glon(nx,ny)
landmask(nx,ny)
```

Users have to generate "*rtma_dtc_latlon_mpfactor_slmask.data*" for their own analysis domain and save this file in the same location.

If users want to write their own first guess generation code, they can find the content of the binary file "*twodvar_input_bi*" from file "*firstguess.f*" by searching "*write(88)*". Here is a list of these lines. Please check the code for details of each line:

```
write(88) ihdrbuf
write(88) iyear,imonth,iday,ihour,iminute,isecond,nx,ny,nsig
write(88) dx,dy
write(88) glat
write(88) glon
write(88) psfcgrid       !  psfc0
write(88) phbgrid        !  PHB (zsfc*g)
write(88) tgrid          !  T(k)   ! TEMP (sensible)
write(88) qgrid          !  Q(k)
write(88) ugrid          !  U(K)
write(88) vgrid          !  V(K)
write(88) landmask  !  LANDMASK  (0=water and >0.5 for land)
write(88) field          !  XICE
write(88) sst            !  SST
write(88) ifield         !  IVGTYP
write(88) ifield         !  ISLTYP
write(88) field          !  VEGFRA
```

```
        write(88) field          !  SNOW
        write(88) ugrid          !  U10
        write(88) vgrid          !  V10
        write(88) field          !  SMOIS
        write(88) tslb           !  TSLB
        write(88) tsk            !  TSK
        write(88) gust           !  GUST
        write(88) vis            !  VIS
        write(88) pblh           !  PBLH
```

10.2.2. Run GSI RTMA Analysis

The code for GSI RTMA analysis is the same as for other GSI applications, but with different namelist options and environmental setups. In this release, a run script named "*run_gsi_rtma.ksh*" in directory "*./util/RTMA*" is provided to help users set up the RTMA GSI run environments and namelist.

1. Code change for user specific domain

The GSI code also includes hardwired information on the analysis grid. Therefore, users need to add analysis grid information to GSI code for their specific RTMA analysis. This is done by editing the file "*support_2dvar.f90*" in *src/main* to change the following lines:

```
elseif (trim(cgrid) == 'dtc') then
   nx=758
   ny=567
   alat18=21.138_r_kind
   elon18=237.280_r_kind
   da8=13545.09_r_kind
```

After adding this domain configuration, users can compile the GSI the same way as the general community GSI (details see Chapter 2 of the fndamental User's Guide).

2. Run script for RTMA

The sample script "*./ util/RTMA/run_gsi_rtma.ksh*" has a similar structure as the general GSI run script "*./run/run_gsi.ksh*" and needs similar information to set up and run. Here, we only introduce the settings that are different from those in the *run_gsi.ksh*. Please read Chapter 3 of the fundamental User's Guide for instruction on how to set up *run_gsi.ksh*.

```
BK_DIR=comGSI/releaseV33/util/RTMA/rtmagus
ROOTDIR= comGSI/releaseV33/util/RTMA
```

- BK_DIR = path of first guess generation directory
- ROOTDIR = RTMA root directory: *./util/RTMA*

In RTMA GSI, there is no need to set up CRTM and satellite radiance related parameters because RTMA doesn't use satellite radiance observations.

There are two binary files holding geogrid information under: ${ROOTDIR}/fix:

- *rtma_dtc_slmask.dat* : Sea Land mask field
- *rtma_dtc_terrain.dat* : terrain of analysis domain

Users can easily generate these two files from geogrid files based on the following read in code information from GSI:

```
allocate(slmask(nx,ny))
open (55,file='rtma_slmask.dat',form='unformatted')
read(55) slmask
close(55)

allocate(terrain(nx,ny))
open (55,file='rtma_terrain.dat',form='unformatted')
read(55) terrain
close(55)
```

After setting up the run script, users can run the RTMA GSI using the same procedure as that used for the general GSI. Please check Chapter 3 of the fundamental User's Guide for more details.

An important aspect to remember is that the RTMA GSI uses anisotropic recursive filters to model the action of its background error covariances. Therefore, in *run_gsi.ksh*, the namelist variable "anisotropic" under "&ANBKGERR" must be set to ".true." For this tutorial, the background error covariances are mapped to the underlying terrain field to a controlled degree (please see section 4th part of this section for more details).

3. Sample results

The run directory of a successful GSI RTMA run with clean option turned on should look like:

```
anavinfo                  fort.205                     sm_theta.des
bckg_dxdy.dat             fort.209                     sm_z.dat
bckg_psfc.dat             fort.210                     sm_z.des
bckg_qsat.dat             fort.211                     stdout
bckgvar.dat_chi           fort.212                     stdout.anl.2012052811
bckgvar.dat_gust          fort.213                     sub_ps.dat
bckgvar.dat_ps            fort.214                     sub_ps.des
bckgvar.dat_pseudorh      fort.215                     sub_q.dat
bckgvar.dat_psi           fort.218                     sub_q.des
bckgvar.dat_t             fort.219                     sub_sf.dat
bckgvar.dat_vis           fort.220                     sub_sf.des
bckg_z.dat                fort.221                     sub_t.dat
berror_stats              gsi.exe                      sub_t.des
```

```
convinfo                gsiparm.anl                      sub_vp.dat
diag_conv_anl.2012052811  mesonet_stnuselist             sub_vp.des
diag_conv_ges.2012052811  mesonetuselist                 theta.dat
errtable                parmcard_input                   theta.des
fit_p1.2012052811       p_rejectlist                     t_rejectlist
fit_q1.2012052811       prepbufr                         w_rejectlist
fit_t1.2012052811       prepobs_prep.bufrtable           wrfanl.2012052811
fit_w1.2012052811       q_rejectlist                     wrf_inou2
fltnorm.dat_chi         random_flips                     wrf_inou3
fltnorm.dat_gust        rtma_slmask.dat                  wrf_inou4
fltnorm.dat_ps          rtma_terrain.dat                 wrf_inou5
fltnorm.dat_pseudorh    shoreline_obrelocation.dat_000   wrf_inou6
fltnorm.dat_psi         shoreline_obrelocation.dat_001   wrf_inou7
fltnorm.dat_t           shoreline_obrelocation.dat_002   wrf_inou8
fltnorm.dat_vis         shoreline_obrelocation.dat_003   wrf_inou9
fort.201                shoreline_obrelocation.dat_004   wrf_inout
fort.202                shoreline_obrelocation.dat_005   z.dat
fort.203                sigfupdate02                     z.des
fort.204                sm_theta.dat
```

Some files, such as fort.* file (fit files), diag files, stdout, and wrf_inout, are similar to those from the general GSI analysis. Others are specific to the RTMA. Here we introduce some of these specific RTMA files:

wrf_inou2, …, wrf_inou9 are empty and used only when the so-called FGAT option is turned on. FGAT stands for “First guess at the Appropriate Time”. It’s a technique that uses auxiliary first guess files with distinct valid times to improve the time interpolation in the GSI.

random_flips is an input file storing random numbers. It is needed to generate the anisotropic background error covariances.

bckgvar_* contain the square-root of the background error covariances for the various analysis variables. They are used in the RTMA post to aid with the evaluation of the analysis error.

4. Namelist for RTMA

RTMA GSI uses the same namelist as the general GSI, and one additional namelist file:

parmcard_input

The namelist parameters in *parmcard_input* are as follows:

- afact0=1 activates the anisotropic component of the background error covariance model. Use afact0=0 instead to have the anisotropic recursive filter simulate an isotropic analysis.
- `hsteep=500.`: sets an artificial elevation difference of 500m between land and water along the coastlines. The resulting escarpment in the terrain-following

covariances serves to confine the influence of the land (water) observations to the land (water) bodies.

- `lsmoothterrain=.true.` : induces a smoothing of the terrain field before the background error covariances are computed

- `hsmooth_len=1.0` : is the correlation length in grid units used to smooth the terrain field.

- `rltop_wind` : is the function correlation length for streamfunction and velocity potential, and `rltop_temp`, `rltop_q`, `rltop_psfc`, `rltop_gust`, and `rltop_vis` are those for temperature, specific humidity, surface pressure, wind gust, and visibility, respectively. Smaller (larger) values of the function correlation lengths lead to stronger (weaker) anisotropies.

- `svpsi, svchi, svpsfc, svtemp`, and `svshum` are used to adjust the background error variances for streamfunction, velocity potential, surface pressure, temperture, and specific humidity, respectively.

- `sclpsi, sclchi, sclpsfc, scltemp, sclhum, sclgust,` and `sclvis` are used to adjust the spatial correlation lengths for streamfunction, velocity potential, surface pressure, temperature, specific humidity, wind gust, and visibility, respectively.

10.2.3. Post-Process

The analysis result from GSI RTMA is a binary file. It needs to be post-processed to generate GRIB files for easy use. In addition to format conversion, the RTMA post-process also:

- computes an estimate of the analysis error by finding a representation of the inverse of the Hessian matrix of the 2DVar. The analysis error is also made available in GRIB format.

- reads in from the original unformatted gsi observation stats files and writes out formatted, streamlined versions for each observation type.

1. Compile the code

The RTMA post-process code is in directory *./util/RTMA/rtma_post*. Just as with the other components of the RTMA code, the dimensions of the analysis domain, analysis grid spacing, and lat/lon information for the southwestern most point are hardwired in the code. Users need to edit the code for the specific domain:

1) get into directory *./util/RTMA/rtma_post;*
2) open file "*param.incl*" ;
3) find the following lines (starts from line 94):

```
!==>parameter definition for dtc
        integer(4),parameter::nx_dtc=758
        integer(4),parameter::ny_dtc=567

        real(8),parameter::alat1_dtc=47.49000_8
        real(8),parameter::elon1_dtc=256.000000_8
        real(8),parameter::da_dtc=13545.09_8
        real(8),parameter::elonv_dtc=256.000000_8
        real(8),parameter::alatan_dtc=47.490000_8
```

4) modify the values to fit the user's specific domain. For this tutorial, the (Conic Lambert Conformal) navigation parameters are:

`nx_dtc:`	analysis domain dimension in X direction
`ny_dtc :`	analysis domain dimension in Y direction
`alat1_dtc :`	analysis domain latitude of point (1,1)
`elon1_dtc :`	analysis domain longitude of point (1,1)
`da_dtc :`	analysis grid spacing in meters
`elonv_dtc:`	Y-axis is parallel to longitude circle at this longitude
`alatan_dtc:`	Latitude at which the projection intersects the earth

5) open file "*post.f90*" and edit the following two lines
line 955 `if (trim(cgrid)=='dtc')        xn=sin(47.49*dg2rad)`
line 964 `if (trim(cgrid)=='dtc')        elonv=256.0`

Here, `elonv` is the same as `elonv_dtc`, and `xn` is `sin(alatan_dtc *dg2rad)`.

After setting the right analysis grid configuration, edit the "*makfile*" inside the same directory and put the right location of the UPP root directory in a line:

```
UPPDIR= /glade/p/work/mhu/UPP/UPPV2.1
```

Please note that this tool has to be compiled after the compilation of the community UPP because this application needs some UPP libraries.

The location of libraries for grib2 compression also needs to be set in the following line:

```
SRCDIRLIB= /glade/u/home/duda/grib2/lib
```

Please note the that makefile only works for Intel compiler for now.

Then, in the same directory, compile the code using the command:

```
./make all
```

The successful compilation should give a new executable in the directory named:

```
rtma_post.exe
```

If user needs to clean the code for recompilation, use command:

```
./make clean
```

2. Run scripts

The running of the RTMA post process is not straightforward. There are many files from first guess generation, RTMA GSI, and fix directory that need to be copied or linked to the run directory. Here, we provide a sample script "*./ util/RTMA/rtma_post.sh*" to help users run the RTMA post process.

As with all other MPI job scripts, a job control head needs to be at the top of the run script to ask for computer resources to run MPI job. This part can be set in the same way as *run_gsi.ksh* (check Chapter 3 of the Basic User's Guide for more details). Then, the parameters in following section need to be set:

```
ROOTDIR=/glade/p/work/mhu/gsi/rtma/rtma/RTMA
work_dir=/glade/p/work/mhu/gsi/rtma/rtma/RTMA/rpostprd
fixparm=${ROOTDIR}/fix
rtmagsidir=$ROOTDIR/rtmaprd
rtmafgdir=$ROOTDIR/rtmagus

CYCLE=2012052811
RUN_COMMAND="mpirun.lsf"
```

Where:

- ROOTDIR = RTMA root directory: (*util/RTMA*)
- work_dir = working directory for RTMA post
- fixparm = path of RTMA local directory ./fix
- rtmagsidir = run directory of RTMA GSI
- rtmafgdir = run directory of first guess generation directory
- CYCLE = analysis time in YYYYMMDDHH
- RUN_COMMAND = setup MPI run command based on job control system. This is the same as the GSI run command.

After setting up the run script, users can run the RTMA post using the same procedure used for the general GSI. Please check Chapter 3 of the fundamental User's Guide for more details.

3 Results

Although there are many files in the RTMA post run directory, the ones that are most relevant to users are the following:

- *anlerr.grib2, anl.grib2, bckg.grib2*: analysis error, analysis, and background in GRIB2 fromat.
- *t_obs.*, u_obs.*, v_obs.*, q_obs.*, ps_obs.*, spd_obs.*, vis_obs.*, gust_obs.** : lists of observation statistics for each outer loop. Specifically, files carrying the string "iter_01" and "iter_02" display observation statistics for the beginning of the first outer-loop and second outer-loop, respectively. Files carrying the string "*iter_anl*" contain observation statistics valid at the end of the analysis.

10.2.4. Notes on This RTMA Section

In this section, we only briefly introduce how to compile and run each component of the RTMA. This information and code should help users build an initial RTMA system for their own grid configuration. We did not touch some of the other features that the RTMA possesses, such as:

1) Running the RTMA with FGAT activated
2) Using bias correction for the background fields
3) Using the Hilbert-Curve based Cross-validation capability

References

1. Wan-Shu Wu, R. James Purser, and David F. Parrish, 2002: Three-Dimensional Variational Analysis with Spatially Inhomogeneous Covariances. *Monthly Weather Review*, 130, 2905–2916.

2. R. James Purser, Wan-Shu Wu, David F. Parrish, and Nigel M. Roberts, 2003: Numerical Aspects of the Application of Recursive Filters to Variational Statistical Analysis. Part I: Spatially Homogeneous and Isotropic Gaussian Covariances. *Monthly Weather Review*, 131, 1524–1535.

3. R. James Purser, Wan-Shu Wu, David F. Parrish, and Nigel M. Roberts, 2003: Numerical Aspects of the Application of Recursive Filters to Variational Statistical Analysis. Part II: Spatially Inhomogeneous and Anisotropic General Covariances. *Monthly Weather Review*, 131, 1536–1548.

4. David F. Parrish and John C. Derber, 1992: The National Meteorological Center's Spectral Statistical-Interpolation Analysis System. *Monthly Weather Review*, 120, 1747–1763.

5. R. James Purser, 2005: Recursive Filter Basics. 1st GSI User Orientation. 4th-5th January 2005. Camp Springs, MD

6. Russ Treadon, 2005: GSI Compilation, Coding, and Updates. 1st GSI User Orientation. 4th-5th January 2005. Camp Springs, MD

7. John Derber, 2005: minimization and Preconditioning. 1st GSI User Orientation. 4th-5th January 2005. Camp Springs, MD

8. Wu, Wan-Shu, 2005: Background error for NCEP's GSI analysis in regional mode. *Fourth WMO International Symposium on Assimilation of Observations in Meteorology and Oceanography*, 18-22 April 2005, Prague, Czech Republic

9. Seung-Jae Lee, David F. Parrish, and Wan-Shu Wu, 2005: Near-Surface Data Assimilation in the NCEP Gridpoint Statistical Interpolation System: Use of Land Temperature Data and a Comprehensive Forward Model. *NOAA/NCEP Office Note 446*, 46