Foreword

This document, designed for experienced users, is based on the second part of 2014 Gridpoint Statistical Interpolation (GSI) User’s Guide and updated with the 2015 community GSI version 3.4 release. It 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 first released with the community GSI version 3.3 in 2014. 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. It include 10 Chapters and one appendix:

- Chapter 1: Overview
- Chapter 2: Software Installation
- Chapter 3: Advanced Topics on Run and Diagnosis
- Chapter 4: GSI Theory
- Chapter 5: GSI Code Structure
- Chapter 6: Static Background Error Covariance
- Chapter 7 Observations
- Chapter 8: Satellite Radiance Data Assimilation
- Chapter 9 Radar Data Assimilation
- Chapter 10 GSI Applications
- Appendix A: GSI Namelist: Name, Default value, Explanation

DTC will update the content of this advanced User’s Guide if needed after the release. For the latest version of this document, please visit the GSI User’s Website at


For referencing this document, please use:

Please send questions and comments to gsi_help@ucar.edu.

Editors: Ming Hu, Hui Shao, Donald Stark, Kathryn Newman, Chunhua Zhou

Contributors to this guide:
NOAA/NCEP/EMC: John Derber, Russ Treadon, Mike Lueken, Wan-Shu Wu, Andrew Collard, Ed Safford
NCAR: Xiang-Yu Huang, Syed Rizvi, Zhiquan Liu, Tom Auligne, Arthur Mizzi
NOAA/ESRL/GSD: Steve Weygandt, Dezso Devenyi, Joseph Olson.
Shanghai Central Meteorological Observatory, China: Min Sun
Acknowledgements:
We thank the U.S. Air Force Weather Agency and the National Oceanic and Atmospheric Administration for their support of this work. This work is also facilitated by National Center for Atmospheric Research (NCAR). NCAR is supported by the National Science Foundation (NSF).
## Table of Contents

Chapter 1: Overview ......................................................................................................................... 1

Chapter 2: Software Installation ................................................................................................. 3
  2.1 Modifying the GSI Build Environment ............................................................................. 3
  2.2 Understanding the Build System ..................................................................................... 3
    2.2.1 Configuration Resource File ....................................................................................... 4
    2.2.2 Modification Example ................................................................................................. 6

Chapter 3: Advanced Topics on Run and Diagnosis ............................................................... 8
  3.1 Convergence Information from file fort.220 ................................................................. 8
  3.2 Use bundle to configure control, state variables and background fields ..................... 11
  3.3 Using observations station use list and rejection list in GSI ........................................ 12
    3.3.1 surface observation rejection and use list ................................................................. 13
    3.3.2 aircraft observation rejection .................................................................................... 14

Chapter 4: GSI Theory .................................................................................................................. 16
  4.1 3DVAR equations: ............................................................................................................ 16
  4.2 Iterations to find the optimal results ................................................................................ 17
  4.3 Analysis variables ............................................................................................................. 18

Chapter 5: GSI Code Structure .................................................................................................. 19
  5.1 Main process .................................................................................................................... 19
  5.2 GSI background IO (for 3DVAR) ..................................................................................... 22
  5.3 Observation ingestion ...................................................................................................... 23
  5.4 Observation innovation calculation ................................................................................... 24
  5.5 Inner iteration ................................................................................................................... 25

Chapter 6: Static Background Error Covariance ....................................................................... 27
  6.1 what is background error covariance .............................................................................. 27
  6.2 Processing of background error matrix .......................................................................... 28
  6.3 Apply background error covariance ................................................................................. 30

Chapter 7 Observations .............................................................................................................. 32
  7.1 Process BUFR/PrepBUFR Files ....................................................................................... 32
    7.1.1 introduction ............................................................................................................... 32
    7.1.2 Encode, Decode, Append a simple BUFR file ......................................................... 34
      7.1.2.1 Decoding/reading data from a simple BUFR file ............................................. 34
      7.1.2.2 Encoding/writing data into a simple BUFR file ................................................. 42
      7.1.2.3 Appending data to a simple BUFR file ............................................................. 45
    7.1.3 Encode, Decode, Append the PrepBUFR file .......................................................... 47
      7.1.3.1 Decoding/reading data from a PrepBUFR file ................................................. 47
      7.1.3.2 More examples on processing PrepBUFR files ............................................... 49
  7.3 GSI BUFR interface ........................................................................................................... 50
    7.3.1 GSI observation data ingest and process procedure ............................................... 50
    7.3.2 The BUFR decoding in GSI read files ..................................................................... 54
  7.4 NCEP generated BUFR files .......................................................................................... 55
    7.4.1 Knowledge on NCEP BUFR/PrepBUFR files ....................................................... 55
    7.4.2 BUFR/PrepBUFR Data Resources for Community Users .................................... 58
  7.5 Observation error adjustment ......................................................................................... 59
# Table of Contents

Chapter 8: Satellite Radiance Data Assimilation ..........................................................60
  8.1. Satellite radiance data ingest and distribution ......................................................60
    8.1.1 link radiance BUFR files to GSI recognized names .....................................60
    8.1.2 GSI Code to ingest radiance data ..................................................................63
    8.1.3 information on ingesting and distribution....................................................65
  8.2. Radiance observation operator .............................................................................66
  8.3. Radiance observation quality control .................................................................67
  8.4. Bias correction for radiance observations ............................................................69
    8.4.1. Bias correction for satellite observations .....................................................69
    8.4.2. The GSI Bias correction procedure and configurations .............................70
    8.4.3 Namelist, satinfo, and coefficients for bias correction.................................72
    8.4.4 enhanced radiance bias correction .................................................................75
    8.4.5. Utility for Angle Bias Correction outside GSI ..........................................76
    8.4.6. Discussion of FAQ. .......................................................................................79
  8.5. Radiance data analysis monitoring ...................................................................81

Chapter 9 Radar Data Assimilation ............................................................................82
  9.1 Prepare Radar Data Files for GSI ......................................................................82
    9.1.1 Introduction. ..................................................................................................82
    9.1.2. GSI interface to Level II radar velocity ..................................................83
      9.1.2.1 Read observations from Level II radar radial velocity BUFR files ....83
      9.1.2.2 Write Level II radar radial velocity observations to BUFR files .......85
    9.1.3 GSI interface to radar reflectivity ...............................................................85
      9.1.3.1 Radar reflectivity preprocess code ......................................................85
      9.1.3.2 Radar reflectivity interface: content and structure ...........................86
      9.1.3.3 Check the results .................................................................................87
  9.2 Analyze Radar Radial Velocity with GSI ............................................................88
    9.2.1 Data Preprocessing of Radar Radial Velocity Assimilation within GSI ....89
    9.2.2 The Processes of the read_radar.f90 code ...............................................96
  9.3 Analyze Radar Reflectivity with GSI .................................................................104
    9.4 information on radar data quality control .......................................................106

Chapter 10 GSI Applications .......................................................................................107
  10.1 Introduction to Global GSI analysis .................................................................107
    10.1.1 The difference between GSI global and regional .....................................107
    10.1.2 Global GFS scripts ..................................................................................108
    10.1.3 Sample resuls. .........................................................................................112
  10.2 Introduction to RTMA analysis .........................................................................113
    10.2.1. Prepare first guess file ...........................................................................113
    10.2.2. Run GSI RTMA analysis .......................................................................117
    10.2.3. post-process .........................................................................................120
    10.2.4. Notes on this RTMA section ..................................................................123
  10.3 GSI hybrid .........................................................................................................123
  10.4 GSI 4DVAR and FSO .......................................................................................123
  10.5 GSI Chem .........................................................................................................123

Appendix A: GSI Namelist: Name, Default value, Explanation ..............................125
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 fundamental GSI User’s Guide, released together with this document.

While the fundamental 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 User’s Guide and would like to apply GSI for specific research topics that need more advanced knowledge and skills.

Unlike the fundamental 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:

<table>
<thead>
<tr>
<th>Version</th>
<th>Release time</th>
<th>Modifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.0.0</td>
<td>06/20/2014</td>
<td>Initial release with official release 3.3</td>
</tr>
<tr>
<td>3.3.0.1</td>
<td>07/07/2014</td>
<td>Fix typos in Equation 1-5 in Chapter 4</td>
</tr>
<tr>
<td>3.3.0.2</td>
<td>08/12/2014</td>
<td>Fix typos in Table in Section 5.5. Add step 5 in radar reflectivity analysis in section 9.3</td>
</tr>
<tr>
<td>3.4.0.0</td>
<td>08/09/2015</td>
<td>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</td>
</tr>
</tbody>
</table>
Update in section 10.1 GSI global analysis
Update in the namelist (Appendix A) based on version 3.4

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.

Appendix A contains a complete list of the GSI namelist with explanations and default values.
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 sort 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.

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

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
Software Installation

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.

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preamble</td>
<td>Uniform requirements for the code. Currently only contains shell information and comments.</td>
</tr>
<tr>
<td>configure.defaults</td>
<td>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.</td>
</tr>
<tr>
<td>postamble</td>
<td>Standard compilation (“make”) rules and dependencies</td>
</tr>
</tbody>
</table>

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
Software Installation

\[ \text{INC_DIR} = \$(\text{COREDIR})/\text{include} \]

Compiler definitions
- Intel ifort Fortran compiler
- GNU gcc C compiler

\[
\begin{align*}
\text{SFC} &= \text{ifort} \\
\text{SF90} &= \text{ifort } -\text{free} \\
\text{SCC} &= \text{gcc}
\end{align*}
\]

The include paths for GSI source code and NetCDF:

\[ \text{INC_FLAGS} = \text{-module } \$(\text{INC_DIR}) \text{ -I } \$(\text{INC_DIR}) \text{ -I } /\text{usr/local/netcdf3-}\text{ifort/include} \]

The default Fortran compiler flags for the main source code:

\[
\begin{align*}
\text{FFLAGS_DEFAULT} &= \text{-fp-model precise } -\text{assume byterecl } -\text{convert big_endian} \\
\text{FFLAGS_FULOPT} &= \text{-O3} \\
\text{FFLAGS} &= \text{\$(FFLAGS_OPT) } \$(\text{FFLAGS_DEFAULT}) \text{ \$(INC_FLAGS) \$(LDFLAGS) -DLINUX}
\end{align*}
\]

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 thereby maintaining compatibility with legacy NOAA output.

The default Fortran compiler flags for the external libraries:

\[
\begin{align*}
\text{FFLAGS_BACIO} &= \text{-O3 } \$(\text{FFLAGS_DEFAULT}) \\
\text{FFLAGS_BUFR} &= \text{-O3 } \$(\text{FFLAGS_DEFAULT}) \text{ \$(FFLAGS_i4r8)} \\
\text{FFLAGS_CLOUD} &= \text{-O3 } \$(\text{FFLAGS_DEFAULT}) \\
\text{FFLAGS_CRTM} &= \text{-O2 } \$(\text{FFLAGS_DEFAULT}) \\
\text{FFLAGS_GFSIO} &= \text{-O3 } \$(\text{FFLAGS_DEFAULT}) \text{ \$(FFLAGS_i4r4)} \\
\text{FFLAGS_SFCIO} &= \text{-O3 } \$(\text{FFLAGS_DEFAULT}) \text{ \$(FFLAGS_i4r4)} \\
\text{FFLAGS_SIGIO} &= \text{-O3 } \$(\text{FFLAGS_DEFAULT}) \text{ \$(FFLAGS_i4r4)} \\
\text{FFLAGS_SP} &= \text{-O3 } \$(\text{FFLAGS_DEFAULT}) \text{ \$(FFLAGS_i4r8)} \\
\text{FFLAGS_W3} &= \text{-O3 } \$(\text{FFLAGS_DEFAULT})
\end{align*}
\]

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

\[
\begin{align*}
\text{CPP} &= \text{cpp} \\
\text{CPP_FLAGS} &= \text{-C } -\text{P } -\text{D}$(\text{BYTE_ORDER}) \text{ -D}\text{REAL8 } -\text{DWRF } -\text{DLINUX} \\
\text{CPP_F90FLAGS} &= \text{-traditional-cpp } -\text{langfortran}
\end{align*}
\]

The MPI compiler definitions:

\[
\begin{align*}
\text{DM_FC} &= \text{mpif90 } -\text{f90=}$\$(\text{SFC}) \\
\text{DM_F90} &= \text{mpif90 } -\text{free } -\text{f90=}$\$(\text{SFC}) \\
\text{DM_CC} &= \text{gcc}
\end{align*}
\]

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 vendors 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:

\[
\begin{align*}
\text{DM\_FC} & \quad = \quad \text{ifort} \\
\text{DM\_F90} & \quad = \quad \text{ifort} \quad \text{-free}
\end{align*}
\]

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

The default C compiler flags:

\[
\begin{align*}
\text{CFLAGS} & \quad = \quad \text{-O0 -DLINUX -DUNDERSCORE} \\
\text{CFLAGS2} & \quad = \quad \text{-DLINUX -Dfunder -DFortranByte=char -DFortranInt=int -DFortranLlong='long'}
\end{align*}
\]

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

\[
\begin{align*}
\text{MYLIBsys} & \quad = \quad \text{-L$(\text{LAPACK\_PATH})$ -mkl=sequential}
\end{align*}
\]

NetCDF path information

- Older versions of NetCDF only have the single library \(-\text{lncdf}\). If you are using an older version you may need to remove the first library name.

\[
\begin{align*}
\text{NETCDFPATH} & \quad = \quad /\text{usr/local/netcdf3-dot-fortran} \\
\text{NETCDFLIBS} & \quad = \quad \text{-lncdf -lncdf $(\text{NETCDF\_PATH})}$
\end{align*}
\]

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 vendor-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 \(-\text{lm}p\text{i}\) flag to specify an MPI build. Therefore the DM compiler definitions become:

\[
\begin{align*}
\text{DM\_FC} & \quad = \quad \text{ifort} \\
\text{DM\_F90} & \quad = \quad \text{ifort} \quad \text{-free} \\
\text{DM\_CC} & \quad = \quad \text{gcc}
\end{align*}
\]

Next, additional link flags for MPI are needed. These are in \textbf{bold}.

\[
\begin{align*}
\text{LDFLAGS} & \quad = \quad \text{-Wl,-rpath,}/\text{usr/local/netcdf3-dot-fortran/lib} \quad \text{-L$\text{MPI\_ROOT}/\text{lib}$ -lm\text{pi} \quad \text{-openmp}
\end{align*}
\]
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 iterations. The following example uses the first two iterations to explain the meaning of each value:

<table>
<thead>
<tr>
<th>Minimization iteration</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td></td>
</tr>
<tr>
<td>( J )</td>
<td>0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
<tr>
<td>( b )</td>
<td>-0.310927744401462716E+04 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
<tr>
<td>( c )</td>
<td>0.310927744401462659E+08 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
<tr>
<td>( EJ )</td>
<td>0.277433945264109695E+02 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Minimization iteration</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>2)</td>
<td></td>
</tr>
<tr>
<td>( J )</td>
<td>0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
<tr>
<td>( b )</td>
<td>-0.310927744401462716E+04 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
<tr>
<td>( c )</td>
<td>0.310927744401462659E+08 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
<tr>
<td>( EJ )</td>
<td>0.277433945264109695E+02 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Minimization iteration</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3)</td>
<td></td>
</tr>
<tr>
<td>( J )</td>
<td>0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
<tr>
<td>( b )</td>
<td>0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
<tr>
<td>( c )</td>
<td>0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
<tr>
<td>( EJ )</td>
<td>0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Minimization iteration</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>4)</td>
<td></td>
</tr>
<tr>
<td>( J )</td>
<td>0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
<tr>
<td>( b )</td>
<td>0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
<tr>
<td>( c )</td>
<td>0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
<tr>
<td>( EJ )</td>
<td>0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00 0.000000000000000000E+00</td>
</tr>
</tbody>
</table>
Advanced Topics on Run and Diagnosis

0.000000000000000000000E+00 0.000000000000000000000E+00 0.000000000000000000000E+00
0.000000000000000000000E+00 0.000000000000000000000E+00 0.000000000000000000000E+00
5)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>stepsize estimates</td>
<td>0.944604610006952448E-03</td>
</tr>
<tr>
<td>stepsize stprat</td>
<td>0.944604610006930960E-03</td>
</tr>
<tr>
<td>stepsize guesses</td>
<td>0.944604610006930960E-03</td>
</tr>
<tr>
<td>penalties</td>
<td>0.944604610006930960E-03</td>
</tr>
<tr>
<td>cost,grad,b,step?</td>
<td>1 0 1.946084290880794579E+05</td>
</tr>
<tr>
<td>cost,grad,b,step?</td>
<td>1 0 1.946084290880794579E+05</td>
</tr>
<tr>
<td>estimated penalty reduction this iteration</td>
<td>1 0 2.937037807046784850E+04</td>
</tr>
</tbody>
</table>

Minimization iteration 1

J= 0.277433945264109712E+02
b= 0.103454706240715466E+06

c= 0.744926644276673208E+08

EJ= 0.970421173071280315E+02

penalty and grad reduction WRT outer and initial iter= 1 0 1.0000000000000000E+00
1.0000000000000000E+00
1.0000000000000000E+00

0.234780E+05 -0.234780E+05 -0.234780E+05
0.394604610006930960E-03 -0.394604610006930960E-03 -0.394604610006930960E-03
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)**: \( 1 = (\text{norm of the gradient})^2, 2 = (\text{norm of the gradient})^2 \)
- **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 anavinfo 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::

<table>
<thead>
<tr>
<th>!var</th>
<th>level</th>
<th>itracer</th>
<th>amedge</th>
<th>source</th>
<th>funcof</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>30</td>
<td>0</td>
<td>no</td>
<td>met_guess</td>
<td>u</td>
</tr>
<tr>
<td>v</td>
<td>30</td>
<td>0</td>
<td>no</td>
<td>met_guess</td>
<td>v</td>
</tr>
<tr>
<td>tv</td>
<td>30</td>
<td>0</td>
<td>no</td>
<td>met_guess</td>
<td>tv</td>
</tr>
<tr>
<td>tsen</td>
<td>30</td>
<td>0</td>
<td>no</td>
<td>met_guess</td>
<td>tv,q</td>
</tr>
<tr>
<td>q</td>
<td>30</td>
<td>1</td>
<td>no</td>
<td>met_guess</td>
<td>q</td>
</tr>
<tr>
<td>oz</td>
<td>30</td>
<td>1</td>
<td>no</td>
<td>met_guess</td>
<td>oz</td>
</tr>
<tr>
<td>cw</td>
<td>30</td>
<td>1</td>
<td>no</td>
<td>met_guess</td>
<td>cw</td>
</tr>
<tr>
<td>p3d</td>
<td>31</td>
<td>0</td>
<td>yes</td>
<td>met_guess</td>
<td>p3d</td>
</tr>
<tr>
<td>ps</td>
<td>1</td>
<td>0</td>
<td>no</td>
<td>met_guess</td>
<td>p3d</td>
</tr>
<tr>
<td>sst</td>
<td>1</td>
<td>0</td>
<td>no</td>
<td>met_guess</td>
<td>sst</td>
</tr>
</tbody>
</table>

### control_vector::

<table>
<thead>
<tr>
<th>!var</th>
<th>level</th>
<th>itracer</th>
<th>as/tsfc_sdv</th>
<th>an_amp0</th>
<th>source</th>
<th>funcof</th>
</tr>
</thead>
<tbody>
<tr>
<td>sf</td>
<td>30</td>
<td>0</td>
<td>1.00</td>
<td>-1.0</td>
<td>state</td>
<td>u,v</td>
</tr>
<tr>
<td>vp</td>
<td>30</td>
<td>0</td>
<td>1.00</td>
<td>-1.0</td>
<td>state</td>
<td>u,v</td>
</tr>
<tr>
<td>ps</td>
<td>1</td>
<td>0</td>
<td>0.50</td>
<td>-1.0</td>
<td>state</td>
<td>p3d</td>
</tr>
<tr>
<td>t</td>
<td>30</td>
<td>0</td>
<td>0.70</td>
<td>-1.0</td>
<td>state</td>
<td>tv</td>
</tr>
<tr>
<td>q</td>
<td>30</td>
<td>1</td>
<td>0.70</td>
<td>-1.0</td>
<td>state</td>
<td>q</td>
</tr>
<tr>
<td>oz</td>
<td>30</td>
<td>1</td>
<td>0.50</td>
<td>-1.0</td>
<td>state</td>
<td>oz</td>
</tr>
<tr>
<td>sst</td>
<td>1</td>
<td>0</td>
<td>1.00</td>
<td>-1.0</td>
<td>state</td>
<td>sst</td>
</tr>
<tr>
<td>cw</td>
<td>30</td>
<td>1</td>
<td>1.00</td>
<td>-1.0</td>
<td>state</td>
<td>cw</td>
</tr>
<tr>
<td>stl</td>
<td>1</td>
<td>0</td>
<td>1.00</td>
<td>-1.0</td>
<td>motley</td>
<td>sst</td>
</tr>
<tr>
<td>sti</td>
<td>1</td>
<td>0</td>
<td>1.00</td>
<td>-1.0</td>
<td>motley</td>
<td>sst</td>
</tr>
</tbody>
</table>

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.

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

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:

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

### 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.
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:

\[ J = 12x_a - xbTB - 1xa - xb + 12Hxa - ooTO - 1Hxa - oo + Jc \]  \hspace{1cm} (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 (\( \Delta x = x - x_b \)), then equation (1) becomes:

\[ J = 12xTB - 1x + 12H(xb + x) - ooTO - 1H(xb + x) - oo + Jc \]  \hspace{1cm} (2)

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

\[ J = 12xTB - 1x + 12Hx - (oo - Hxb)TO - 1Hx - (oo - Hxb) + Jc \]  \hspace{1cm} (3)

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

\[ J = 12xTB - 1x + 12Hx - oTO - 1Hx - o + Jc \]  \hspace{1cm} (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:

\[
J = 12y^TBy + 12HBy - oTO - 1HBy - o + Jc
\]  

(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:

\[
\nabla_x J = B^{-1}x + H^T O^{-1} (Hx - o) \]  

(6)

\[
\nabla_y J = B^T y + B^T H^T O^{-1} (HBy - o) = B\nabla_y J \]  

(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 \( n \):

\[
\nabla x J_n = B^{-1}x_n + HTO - 1Hx_{n-1} - o = y_{n-1} + HTO - 1Hx_{n-1} - o
\]

\[
\nabla y J_n = B\nabla x J_n
\]

\[
Dir \cdot x^n = \nabla_y J^n + \beta Dir \cdot x^{n-1}
\]

\[
Dir \cdot y^n = \nabla_x J^n + \beta Dir \cdot y^{n-1}
\]

\[
x^n = x^{n-1} + aDir \cdot x^n
\]

\[
y^n = y^{n-1} + aDir \cdot y^n
\]

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

During the above iteration, the \( \beta \) is calculated in subroutine \texttt{pcgsoi} and the stepsize (\( \alpha \)) is calculated in subroutine \texttt{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 \texttt{sqrt(B)} preconditioner when namelist variable \texttt{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 ($\psi$)
- Unbalanced velocity potential ($\chi$)
- 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.
### GSI Code Structure

<table>
<thead>
<tr>
<th>gsimain.f90</th>
<th>main steps in each call</th>
</tr>
</thead>
</table>
| **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 Code Structure

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
  ```fortran
  if (lobserver) then
    if initial pass: call observer_init
    call observer_run
    if last pass: call observer_finalize
  else
    call glbsoi(mype)
  endif
  end if
  ```
- If *last pass*: Deallocate arrays

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

**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

```fortran
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 pegsoi (details in section 6.2.5)
    endif
    endif
  Set 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)

Read background

<table>
<thead>
<tr>
<th>Background files</th>
<th>Convert to internal format (regional_io.f90)</th>
<th>Read in and distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMM NetCDF</td>
<td>convert_regional_guess</td>
<td>read_guess (read_guess.F90)</td>
</tr>
<tr>
<td>NMM binary</td>
<td>convert_netcdf_nmm</td>
<td>read_wrf_nmm_netcdf_guess</td>
</tr>
<tr>
<td>ARW NetCDF</td>
<td>convert_binary_nmm</td>
<td>read_wrf_nmm_binary_guess</td>
</tr>
<tr>
<td>ARW binary</td>
<td>convert_netcdf_mass</td>
<td>read_wrf_mass_netcdf_guess</td>
</tr>
<tr>
<td>RTMA(twodvar)</td>
<td>convert_binary_mass</td>
<td>read_wrf_mass_binary_guess</td>
</tr>
<tr>
<td>nems_nmmb</td>
<td>convert_binary_2d</td>
<td>read_2d_guess</td>
</tr>
<tr>
<td>CMAQ</td>
<td>convert_nems_nmmb</td>
<td>read_nems_nmmb_guess</td>
</tr>
<tr>
<td>global GFS</td>
<td></td>
<td>read_cmaq_guess</td>
</tr>
</tbody>
</table>

Note: this chart doesn’t include ensemble member ingest for hybrid

Output analysis result (regional_io.f90)

write_all (write_all.F90)
write_regional_analysis

write_regional_analysis

if ( use_gfs_nemsio )
  write_nems
else
  write_gfs
write_bias (bias correction)

write_regional_analysis

write_cmaq

Analysis results file

NMM NetCDF
NMM binary
ARW NetCDF
ARW binary
RTMA(twodvar)
nems_nmmb
CMAQ
global GFS
## 5.3 Observation ingestion

<table>
<thead>
<tr>
<th>Data type (ditype)</th>
<th>Observation type (obstype)</th>
<th>Subroutine that reads data</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv</td>
<td>t, q, ps, pw, spd, mta_cl, gos_cpt, gust, vis</td>
<td>read_prepbufr</td>
</tr>
<tr>
<td></td>
<td>from satwnd</td>
<td>read_satwnd</td>
</tr>
<tr>
<td></td>
<td>Not from satwnd</td>
<td>read_prepbufr</td>
</tr>
<tr>
<td></td>
<td>sst</td>
<td>from mods</td>
</tr>
<tr>
<td></td>
<td>not from mods</td>
<td>read_modsbufr</td>
</tr>
<tr>
<td></td>
<td>srw</td>
<td>read_superwinds</td>
</tr>
<tr>
<td></td>
<td>tcp</td>
<td>read_tcps</td>
</tr>
<tr>
<td></td>
<td>lag</td>
<td>read_lag</td>
</tr>
<tr>
<td></td>
<td>rw (radar winds Level-2)</td>
<td>read_radar</td>
</tr>
<tr>
<td></td>
<td>dw (lidar winds)</td>
<td>read_lidar</td>
</tr>
<tr>
<td></td>
<td>rad_ref</td>
<td>read_RadarRef_mosaic</td>
</tr>
<tr>
<td></td>
<td>lghtn</td>
<td>read_lightning</td>
</tr>
<tr>
<td></td>
<td>larccld</td>
<td>read_NASA_LaRC</td>
</tr>
<tr>
<td></td>
<td>pm2_5</td>
<td>read_anowbufr</td>
</tr>
<tr>
<td></td>
<td>pbil</td>
<td>read_pbil</td>
</tr>
<tr>
<td>rad (satellite radiances)</td>
<td>(platform) not AQUA</td>
<td>read_bufrtovs (TOVS 1b data)</td>
</tr>
<tr>
<td></td>
<td>(platform) AQUA</td>
<td>read_airs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(airs data)</td>
</tr>
<tr>
<td></td>
<td>atms</td>
<td>read_atms</td>
</tr>
<tr>
<td></td>
<td>iasi</td>
<td>read_iasi</td>
</tr>
<tr>
<td></td>
<td>cris</td>
<td>read_cris</td>
</tr>
<tr>
<td></td>
<td>sndr, sndrd1/2/3/4</td>
<td>read_goesndr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(GOES sounder data)</td>
</tr>
<tr>
<td></td>
<td>ssmi</td>
<td>read_ssmi</td>
</tr>
<tr>
<td></td>
<td>amsre_low/mid/hig</td>
<td>read_amsre</td>
</tr>
<tr>
<td></td>
<td>ssmis, ssmis_las/ias/uss/ve</td>
<td>read_ssmis</td>
</tr>
<tr>
<td></td>
<td>goes_img</td>
<td>read_goesimg</td>
</tr>
<tr>
<td></td>
<td>seviri</td>
<td>read_seviri</td>
</tr>
<tr>
<td></td>
<td>avhrr_navy</td>
<td>read_avhrr_navy</td>
</tr>
<tr>
<td></td>
<td>avhrr</td>
<td>read_avhrr</td>
</tr>
<tr>
<td>ozone</td>
<td>subuv2, omi, gome, o3lev, mls</td>
<td>read_ozone</td>
</tr>
<tr>
<td>co</td>
<td>mopitt</td>
<td>read_co</td>
</tr>
<tr>
<td>pcp</td>
<td>pcp_ssmi, pcp_tm, pcp_amsu, pcp_stage3</td>
<td>read_pcp</td>
</tr>
<tr>
<td>gps</td>
<td>gps_ref, gps_bnd</td>
<td>read_gps</td>
</tr>
<tr>
<td>aero</td>
<td>aod</td>
<td>read_aerosol</td>
</tr>
</tbody>
</table>

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

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

Each observation type uses one or more processors to read in the data and then write the data into an 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_outter-loop, where * is 4 digital processor ID.
### 5.4 Observation innovation calculation

<table>
<thead>
<tr>
<th>Data type (ditype)</th>
<th>Observation type (obstype)</th>
<th>Subroutine calculate innovation</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>setupt</td>
<td></td>
</tr>
<tr>
<td>uv</td>
<td>setupw</td>
<td></td>
</tr>
<tr>
<td>q</td>
<td>setupq</td>
<td></td>
</tr>
<tr>
<td>ps</td>
<td>setupps</td>
<td></td>
</tr>
<tr>
<td>pw</td>
<td>setuppw</td>
<td></td>
</tr>
<tr>
<td>spd</td>
<td>setupsdp</td>
<td></td>
</tr>
<tr>
<td>sst</td>
<td>setupst</td>
<td></td>
</tr>
<tr>
<td>srw</td>
<td>setupsrw</td>
<td></td>
</tr>
<tr>
<td>tcp</td>
<td>setuptcp</td>
<td></td>
</tr>
<tr>
<td>lag</td>
<td>setuplag</td>
<td></td>
</tr>
<tr>
<td>rw (radar winds Level-2)</td>
<td>setuprw</td>
<td></td>
</tr>
<tr>
<td>dw (lidar winds)</td>
<td>setupdw</td>
<td></td>
</tr>
<tr>
<td>pm2_5</td>
<td>setuppm2_5</td>
<td></td>
</tr>
<tr>
<td>gust</td>
<td>setupgust</td>
<td></td>
</tr>
<tr>
<td>vis</td>
<td>setupvis</td>
<td></td>
</tr>
<tr>
<td>pblh</td>
<td>setuppblh</td>
<td></td>
</tr>
<tr>
<td>amsua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>amsub</td>
<td></td>
<td></td>
</tr>
<tr>
<td>msu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mhs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hirs4,3,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ssu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>airs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>amsua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hsb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atms</td>
<td>setuprad</td>
<td></td>
</tr>
<tr>
<td>iasi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>crisi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sndr, snrd1, snrd2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>snrd3, snrd4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ssmi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>amsre_low/mid/hig</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ssmis ssmis_*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>goes_img</td>
<td></td>
<td></td>
</tr>
<tr>
<td>seviri</td>
<td></td>
<td></td>
</tr>
<tr>
<td>avhrr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>avhrr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subuv2, omi, gome,</td>
<td>setupozlay</td>
<td></td>
</tr>
<tr>
<td>ozone</td>
<td>o3lev, mls</td>
<td></td>
</tr>
<tr>
<td>pcp</td>
<td>setuppcp</td>
<td></td>
</tr>
<tr>
<td>mopitt, subuv2</td>
<td>setupco</td>
<td></td>
</tr>
<tr>
<td>co</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gps</td>
<td>setupref</td>
<td></td>
</tr>
<tr>
<td>gps_bnd</td>
<td>setupbnd</td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td>data level category</td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icat</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 pegsoi (pcgsoi.f90), inside the following loop:

```plaintext
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.

<table>
<thead>
<tr>
<th>Steps in inner iteration</th>
<th>Code in pcgsoi.f90</th>
<th>Corresponding equations in Chapter 4 (variables are defined in Chapter 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient of observation term</td>
<td><code>call intall</code></td>
<td>$HTO-1H_{xn-1-o}$</td>
</tr>
<tr>
<td>Add gradient of background term</td>
<td><code>gradx(i)=gradx(i)+yhatsave(i)</code></td>
<td>$\nabla x_Jn=HTO-1H_{xn-1-o+y_n-1}$</td>
</tr>
<tr>
<td>Apply background error covariance</td>
<td><code>call bkerror(gradx,grady)</code></td>
<td>$\nabla y_Jn=B\nabla x_Jn$</td>
</tr>
</tbody>
</table>
| Calculate norm of gradients | | $b=\beta$
$\beta=\nabla x_Jn-\nabla x_Jn-1\nabla y_Jn\nabla x_Jn-\nabla x_Jn-1TD_{xr} \cdot x_n$ |
| Calculate new search direction | `dirx(i)=-grady(i)+b*dirx(i)`
`diry(i)=-gradx(i)+b*diry(i)` | $Dir \cdot x^n = \nabla J^n + \beta Dir \cdot x^{n-1}$
$Dir \cdot y^n = \nabla J^n + \beta Dir \cdot y^{n-1}$ |
| Calculate stepsize | `call stpcalc` | $stp=\alpha=ab$ |
| 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 are model forecasts from a prior time step, the background error covariance matrix (B) can be defined as the error covariance of model forecasts:

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

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 $10^6 \times 10^6$, 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 cover 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>
<thead>
<tr>
<th>Category</th>
<th>Array name</th>
<th>Dimension</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balance (Horizontal regression coefficients)</td>
<td>agvi</td>
<td>0:mlat+1,1:nsig,1:nsig</td>
<td>Regression coefficients for stream function and temperature</td>
</tr>
<tr>
<td></td>
<td>wgvi</td>
<td>0:mlat+1,1:nsig</td>
<td>Regression coefficients for stream function and surface pressure</td>
</tr>
<tr>
<td></td>
<td>bvi</td>
<td>0:mlat+1,1:nsig</td>
<td>Regression coefficients for stream function and velocity potential</td>
</tr>
<tr>
<td>Horizontal and vertical influence scale</td>
<td>hwll</td>
<td>0:mlat+1,1:nsig,1:nc3d</td>
<td>Horizontal lengthscales for stream function, unbalanced velocity potential, unbalanced temperature, and relative humidity</td>
</tr>
<tr>
<td></td>
<td>hwllp</td>
<td>0:mlat+1, nc2d</td>
<td>Horizontal lengthscale for unbalanced surface pressure</td>
</tr>
<tr>
<td></td>
<td>vz</td>
<td>1:nsig, 0:mlat+1, 1:nc3d</td>
<td>Vertical lengthscale for stream function, unbalanced velocity potential, unbalanced temperature, and relative humidity</td>
</tr>
<tr>
<td>Variance</td>
<td>corz</td>
<td>1:mlat,1:nsig,1:nc3d</td>
<td>Square root of variance for stream function, unbalanced velocity potential, unbalanced temperature, and relative humidity</td>
</tr>
<tr>
<td></td>
<td>corp</td>
<td>1:mlat,nc2d</td>
<td>Square root of variance for unbalanced surface pressure</td>
</tr>
</tbody>
</table>

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”, “hzc”, “as3d” and “as2d”) supplied through gsiparm.anl and anavinfo.txt. Desired information is finally processed and transformed to new arrays such as “slw”, “slf”, “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
Static Background Error Covariance

“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:

\[
\nabla_y J = B \nabla_x J \quad \text{(subroutine berror(gradx, grady))}
\]

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:

\[
B = B_{balance}VB_z(B_xB_yB_yB_x)B_zVB_{balance}^T
\]

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

<table>
<thead>
<tr>
<th>Sub-matrix of ( B )</th>
<th>Function</th>
<th>Subroutine</th>
<th>GSI files</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_{balance} )</td>
<td>balance among different variables</td>
<td>balance</td>
<td>balmod.f90</td>
</tr>
<tr>
<td>( B_{balance}^T )</td>
<td>adjoint of balance equation</td>
<td>tbalance</td>
<td>balmod.f90</td>
</tr>
<tr>
<td>( V )</td>
<td>Square root of variance</td>
<td>bkgvar</td>
<td>bkgvar.f90</td>
</tr>
<tr>
<td>( B_z )</td>
<td>vertical smoother</td>
<td>frfhvo</td>
<td>smoothzrf.f90</td>
</tr>
<tr>
<td>( B_xB_yB_yB_x )</td>
<td>Self-adjoint smoothers in West-East ((B_x)) and South-North ((B_y)) direction</td>
<td>smoothrf</td>
<td>smoothzrf.f90</td>
</tr>
</tbody>
</table>

The composition of \( B \) is achieved by calling \textit{bkeror} in following three steps:

Step 1. Adjoint of balance equation \((B_{balance}^T)\) is done by calling \textit{tbalance}

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

- Multiply by square root of background error variances \((V)\) by calling \textit{bkgvar};
- Apply vertical smoother \((B_z)\) by calling \textit{frfhvo};
- Convert from subdomain to the full horizontal field distributed among processors by calling \textit{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:

\[
\begin{align*}
\text{velocity potential} & = \text{unbalanced velocity potential} + B_{Balance(st \rightarrow vp)} \text{ stream function} \\
\text{temperature} & = \text{unbalanced temperature} + B_{Balance(st \rightarrow t)} \text{ stream function} \\
\text{surface pressure} & = \text{unbalanced surface pressure} + B_{Balance(st \rightarrow p)} \text{ stream function}
\end{align*}
\]
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 to 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 to 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 problems understanding 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:


7.1 Process BUFR/PrepBUFR Files

7.1.1 Introduction

BUFR (Binary Universal Form for the Representation 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.

```fortran
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
  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)
  end do
end program bufr_decode_sample
```
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:

1. The 1\textsuperscript{st} Level: the three RED lines are the first level (file level) statements, which open/close a BUFR file for decoding.
2. The 2\textsuperscript{nd} 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.
3. The 3\textsuperscript{rd} 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.
4. The 4\textsuperscript{th} 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.
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

```fortran
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**:

  ```fortran
  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:

  ```fortran
  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:

**2\text{nd} level (message level): read in messages**

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

- **Function ireadmg:**

  ```fortran
  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. The BUFR table and mnemonics is discussed in detail by Chapter 3 of the BUFR User’s
Observations

Guide. 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):

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>NUMBER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADPUPA</td>
<td>A48102</td>
<td>UPPER-AIR (RAOB, PIBAL, RECCO, DROPS) REPORTS</td>
</tr>
<tr>
<td>AIRCAR</td>
<td>A48103</td>
<td>MDCRS ACARS AIRCRAFT REPORTS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>SEQUENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADPUPA</td>
<td>HEADR</td>
</tr>
<tr>
<td></td>
<td>SID</td>
</tr>
<tr>
<td></td>
<td>XOB</td>
</tr>
<tr>
<td></td>
<td>YOB</td>
</tr>
<tr>
<td></td>
<td>DHR</td>
</tr>
<tr>
<td></td>
<td>ELV</td>
</tr>
<tr>
<td></td>
<td>TYP</td>
</tr>
<tr>
<td></td>
<td>T29</td>
</tr>
<tr>
<td></td>
<td>TSB</td>
</tr>
<tr>
<td></td>
<td>ITP</td>
</tr>
<tr>
<td></td>
<td>SQN</td>
</tr>
<tr>
<td></td>
<td>PROCN</td>
</tr>
<tr>
<td></td>
<td>RPT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>SCAL</th>
<th>REFERENCE</th>
<th>BIT</th>
<th>UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>0</td>
<td>0</td>
<td>64</td>
<td>CCITT IA5</td>
</tr>
<tr>
<td>XOB</td>
<td>2</td>
<td>-18000</td>
<td>16</td>
<td>DEG E</td>
</tr>
<tr>
<td>YOB</td>
<td>2</td>
<td>-9000</td>
<td>15</td>
<td>DEG N</td>
</tr>
<tr>
<td>DHR</td>
<td>3</td>
<td>-24000</td>
<td>16</td>
<td>HOURS</td>
</tr>
<tr>
<td>ELV</td>
<td>0</td>
<td>-1000</td>
<td>17</td>
<td>METER</td>
</tr>
<tr>
<td>TYP</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>CODE TABL</td>
</tr>
</tbody>
</table>

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 `HEADR`. 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:

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

● ubfint
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 README, 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 README.

### 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.

```fortran
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 ufbinp(unit_out, hdr, 3, 1, iret, hdstr)
call ufbinp(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 )
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 CLOSMG ( 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 $IREADMG$ 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 \texttt{writsb}.

### 7.1.2.3 Appending data to a simple BUFR file

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

```fortran
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 \textit{sample.bufr}. 

45
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:

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

  ```fortran
  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.

```fortran
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.
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.

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

```fortran
integer :: i, k, iret
open(unit_table, file='prepobs_prep.bufrtable')
open(unit_in, file='prepbufr', form='unformatted', status='old')
call openbf(unit_in, 'IN', unit_in)
call dxdump(unit_in, unit_table)
call datelen(10)
```

Specifies the date format as YYYYMMDDHH.

```fortran
nmsg=0
msg_report: do while (ireadmg(unit_in, subset, idate) == 0)
nmsg=nmsg+1
ntb = 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.

```fortran
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 the 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 been read in even though **mxlv**=250.

```fortran
    rstation_id=hdr(1)
    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
```

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 Guiders, 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.
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 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:

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

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

<table>
<thead>
<tr>
<th>GSI Name</th>
<th>Content</th>
<th>Example file names</th>
</tr>
</thead>
<tbody>
<tr>
<td>prepbufr</td>
<td>Conventional observations, including ps, t, q, pw, uv, spd, dw, sst, from observation platforms such as METAR, sounding, et al.</td>
<td>gdas1t12z.prepbufr</td>
</tr>
<tr>
<td>amsaubufr</td>
<td>AMSU-A 1b radiance (brightness temperatures) from satellites NOAA-15, 16, 17, 18, 19 and METOP-A</td>
<td>gdas1t12z.1bamua.tm00.bufr_d</td>
</tr>
<tr>
<td>amsubbufr</td>
<td>AMSU-B 1b radiance (brightness temperatures) from satellites NOAA15, 16,17</td>
<td>gdas1t12z.1bamub.tm00.bufr_d</td>
</tr>
<tr>
<td>radarbufr</td>
<td>Radar radial velocity Level 2.5 data</td>
<td>ndas.t12z_radwnd.tm12.bufr_d</td>
</tr>
<tr>
<td>gspsofbufr</td>
<td>GPS radio occultation observation</td>
<td>gdas1t12z.gpsro.tm00.bufr_d</td>
</tr>
<tr>
<td>ssmirrbufr</td>
<td>Precipitation rate observations from SSM/I</td>
<td>gdas1t12z.spssmi.tm00.bufr_d</td>
</tr>
<tr>
<td>hirs4bufr</td>
<td>HIRS4 1b radiance observation from satellite NOAA 18, 19 and METOP-A</td>
<td>gdas1t12z.1bhrs4.tm00.bufr_d</td>
</tr>
<tr>
<td>msubufr</td>
<td>MSU observation from satellite NOAA 14</td>
<td>gdas1t12z.1bmsu.tm00.bufr_d</td>
</tr>
</tbody>
</table>

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)="amsaubufr", dtype(28)="amsu", dplat(28)="n15", dsis(28)="amsu_n15", dval(28)=10.0, dthin(28)=2,
  dfile(29)="amsaubufr", dtype(29)="amsu", dplat(29)="n16", dsis(29)="amsu_n16", dval(29)=0.0, dthin(29)=2,
```

Users may notice that the first column, \texttt{dfile}, is the GSI recognized file names listed in the section 3.1 of the GSI User’s Guider. The 2\textsuperscript{nd} column, \texttt{dtype}, is the observation type. The 3\textsuperscript{rd} column, \texttt{dplat}, is satellite platform ID. And the 4\textsuperscript{th} column, \texttt{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, \texttt{dtype}, that is used to decide which routine to call for reading the data from the corresponding input file defined by \texttt{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 ‘amsubufr’ (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.

**Example 1: Process conventional (prepbufr) data**

```fortran
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 == 'gosctp' ) then
    call read_prepbufr(nread,npuse,nouse,infile,obstype,lunout,twind,sis,&
                        prsl_full)
    string='READ_PREPBUIFR'
  endif()
endif()
```

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’, ‘gosctp’ from PrepBUFR file saved in “infile”.

**Example 2: Process TOVS 1b data**

```fortran
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
  llib=1
  lll=1
  if((obstype == 'amsua' .or. obstype == 'amsub' .or. obstype == 'mhs') .and. &
     (platid /= 'metop-a' .or. platid /= 'metop-b' .or. platid /= 'metop-c'))llll=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'
```

52
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:

\[
\begin{align*}
\text{infile} &= \text{dfile} \text{ of the namelist section } &\& \text{OBS}_\text{INOUT} \\
\text{obstype} &= \text{dtype} \text{ of the namelist section } &\& \text{OBS}_\text{INOUT} \\
\text{sis} &= \text{dsis} \text{ of the namelist section } &\& \text{OBS}_\text{INOUT}
\end{align*}
\]

- **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_prebufr.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 FWO 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 metarclidstr /'CLAM HOCB' ! cloud amount and cloud base height 
data metarwthstr /'FRWE' ! present weather 
data metarvisstr /'HOVI' ! visibility 
data geosclidstr /'CDTF TOCC GCDTT CDTF_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 downloaded 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.

```fortran
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 a 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:
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/
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_prepbufr.f90), 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:

<table>
<thead>
<tr>
<th>GSI Name</th>
<th>Content</th>
<th>Example file names</th>
</tr>
</thead>
<tbody>
<tr>
<td>amsuabufr</td>
<td>AMSU-A 1b radiance (brightness temperatures) from satellites NOAA-15, 16, 17, 18, 19 and METOP-A</td>
<td>gdas1.t12z.1bamua.tm00.bufr_d</td>
</tr>
<tr>
<td>amsubbufr</td>
<td>AMSU-B 1b radiance (brightness temperatures) from satellites Noaa15, 16, 17</td>
<td>gdas1.t12z.1bamub.tm00.bufr_d</td>
</tr>
</tbody>
</table>

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)='amsuabufr', dtype(33)='amsua', dplat(33)='aqua',      dsis(33)='amsua_aqua',
dfile(34)='amsuabufr', dtype(34)='amsub',  dplat(34)='n15',      dsis(34)='amsub_n15',
dfile(35)='amsuabufr', dtype(35)='amsub',  dplat(35)='n16',      dsis(35)='amsub_n16',
dfile(36)='amsuabufr', 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:

```plaintext
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:

```bash
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:

```plaintext
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 \texttt{read_*} 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:

<table>
<thead>
<tr>
<th>Data type (ditype)</th>
<th>Observation type (obstype)</th>
<th>Subroutine that reads data</th>
</tr>
</thead>
<tbody>
<tr>
<td>rad (satellite radiances)</td>
<td>amsub</td>
<td>read_bufrtos (TOVS 1b data)</td>
</tr>
<tr>
<td>(platform) not AQUA</td>
<td>amsua</td>
<td></td>
</tr>
<tr>
<td></td>
<td>msu</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mhs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hirs4,3,2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ssu</td>
<td></td>
</tr>
<tr>
<td>sndr, sndrd1, sndrd2</td>
<td>read_goesndr (GOES sounder data)</td>
<td></td>
</tr>
<tr>
<td>sndrd3, sndrd4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ssmi</td>
<td>read_ssmi</td>
<td></td>
</tr>
</tbody>
</table>

From this table, we can see TOVS 1b observations from the NOAA and METOP satellites are read in by subroutine \texttt{read_bufrtos} and the GEOS sounder and SSMI observations are read in by subroutine \texttt{read_goesndr} and \texttt{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 \texttt{read_bufrtos (read_bufrtos.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 (\texttt{read_subset}) from the BUFR file and then read in all observations (\texttt{read_loop}) from each message. In the subroutine \texttt{read_bufrtos}, the two loops are marked by the following code:
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:

```fortran
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_bufrtosv.

- 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_bufrtosv 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:

<table>
<thead>
<tr>
<th>OBS_PARA: ps</th>
<th>2352</th>
<th>2572</th>
<th>8367</th>
<th>2673</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBS_PARA: t</td>
<td>4617</td>
<td>4331</td>
<td>12418</td>
<td>4852</td>
</tr>
<tr>
<td>OBS_PARA: q</td>
<td>3828</td>
<td>3908</td>
<td>11096</td>
<td>3632</td>
</tr>
<tr>
<td>OBS_PARA: pw</td>
<td>89</td>
<td>31</td>
<td>141</td>
<td>23</td>
</tr>
<tr>
<td>OBS_PARA: ssu</td>
<td>5704</td>
<td>4835</td>
<td>15025</td>
<td>4900</td>
</tr>
<tr>
<td>OBS_PARA: hirs4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBS_PARA: n15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBS_PARA: n18</td>
<td>1002</td>
<td>2119</td>
<td>0</td>
<td>390</td>
</tr>
<tr>
<td>OBS_PARA: n19</td>
<td>244</td>
<td>1093</td>
<td>0</td>
<td>235</td>
</tr>
<tr>
<td>OBS_PARA: n19</td>
<td>651</td>
<td>3486</td>
<td>0</td>
<td>469</td>
</tr>
</tbody>
</table>

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, \( \text{CO}_2 \) 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_bufrtos.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 qcmo_90 and are listed as follows for reference:

<table>
<thead>
<tr>
<th>subroutine name</th>
<th>Quality Control for</th>
</tr>
</thead>
<tbody>
<tr>
<td>qc_ssmi</td>
<td>ssmi, amsre, and ssmis</td>
</tr>
</tbody>
</table>
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:

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

Using the same example as section 4.5.2:

```plaintext
sat   type       penalty        nobs   iland  isnoice  icoast ireduce   ivarl nlgross
n15    amusa      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 \(\frac{1}{4}\) 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 \(P_i(x)\), with associated coefficients \(\beta_i\):
Since the bias correction is applied to the radiance departures, this is equivalent to using the modified definition of the observation operator:

\[ \tilde{H}(x, \beta) = H(x) + BC(\beta, x) \]

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

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

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


### 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
Satellite Radiance Data Assimilation

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:

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>diag_rad</td>
<td>.true.</td>
<td>logical to turn off or on the diagnostic radiance file (true=on)</td>
</tr>
<tr>
<td>passive_bc</td>
<td>.false.</td>
<td>logical to turn off or on radiance bias correction for monitored channels</td>
</tr>
<tr>
<td>adp_anglebc</td>
<td>.false.</td>
<td>option to perform variational angle bias correction</td>
</tr>
</tbody>
</table>

- 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:

<table>
<thead>
<tr>
<th>sensor/instr/sat</th>
<th>chan</th>
<th>iuse</th>
<th>error</th>
<th>error_cld</th>
<th>ermax</th>
<th>var_b</th>
<th>var_pg</th>
<th>icld_det</th>
</tr>
</thead>
<tbody>
<tr>
<td>amsua_n15</td>
<td>1</td>
<td>1</td>
<td>3.000</td>
<td>9.100</td>
<td>4.500</td>
<td>10.000</td>
<td>0.000</td>
<td>1</td>
</tr>
<tr>
<td>amsua_n15</td>
<td>2</td>
<td>1</td>
<td>2.000</td>
<td>13.500</td>
<td>4.500</td>
<td>10.000</td>
<td>0.000</td>
<td>1</td>
</tr>
<tr>
<td>amsua_n15</td>
<td>3</td>
<td>1</td>
<td>2.000</td>
<td>7.100</td>
<td>4.500</td>
<td>10.000</td>
<td>0.000</td>
<td>1</td>
</tr>
<tr>
<td>amsua_n15</td>
<td>4</td>
<td>1</td>
<td>0.600</td>
<td>1.300</td>
<td>2.500</td>
<td>10.000</td>
<td>0.000</td>
<td>1</td>
</tr>
<tr>
<td>hirs3_n17</td>
<td>1</td>
<td>-1</td>
<td>2.000</td>
<td>0.000</td>
<td>4.500</td>
<td>10.000</td>
<td>0.000</td>
<td>-1</td>
</tr>
<tr>
<td>hirs3_n17</td>
<td>2</td>
<td>-1</td>
<td>0.600</td>
<td>0.000</td>
<td>2.500</td>
<td>10.000</td>
<td>0.000</td>
<td>-1</td>
</tr>
<tr>
<td>hirs3_n17</td>
<td>3</td>
<td>-1</td>
<td>0.530</td>
<td>0.000</td>
<td>2.500</td>
<td>10.000</td>
<td>0.000</td>
<td>-1</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>iuse</th>
<th>Channel usage in GSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>do not use</td>
</tr>
<tr>
<td>-1</td>
<td>monitor if diagnostics produced</td>
</tr>
<tr>
<td>0</td>
<td>monitor and use in QC only</td>
</tr>
<tr>
<td>1</td>
<td>use data with complete bias correction</td>
</tr>
<tr>
<td>2</td>
<td>use data with no air mass bias correction</td>
</tr>
<tr>
<td>3</td>
<td>use data with no angle dependent bias correction</td>
</tr>
<tr>
<td>4</td>
<td>use data with no bias correction</td>
</tr>
</tbody>
</table>

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 1.335609
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 -9.468913 -1.490995 -0.856006 -0.013090 0.945916
8 amsua_n15 8 -22.737061 -2.195735 -0.357354 0.158646 15.298422
9 amsua_n15 9 0.000000 0.000000 0.000000 0.000000 0.000000
10 amsua_n15 10 0.000000 0.000000 0.000000 0.000000 0.000000
11 amsua_n15 11 0.000000 0.000000 0.000000 0.000000 0.000000
12 amsua_n15 12 0.000000 0.000000 0.000000 0.000000 0.000000
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 0.000000 0.000000 0.000000 0.000000 0.000000

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 -0.411 -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 -1.869 -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
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

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 represent 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 enhanced bias correction in GSI, for more details on this enhanced radiance bias correction, please check the following published paper:


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 previous 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: \textit{diag\_amsua\_n18\_ges.2014061915} In this cycle for bias correction, this file should be called: \textit{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.

\textbf{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 \texttt{./gsi\_angupdate} within \texttt{./comGSIv3\_3/\texttt{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 \texttt{“gsi\_angupdate.tar”} from the GSI download page on-line. Once untarred, under the GSI directory \texttt{“./comGSIv3/\texttt{util}”}, you will see a new directory: \texttt{./gsi\_angupdate}, which includes the tool to update coefficients for radiance angle dependent bias correction.

- Compile

Inside the directory \texttt{./gsi\_angupdate}, type the following command:

\input{./make}

then check if executable \texttt{“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
# GSIRW_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}
GSIRW_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.

```plaintext
&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.

\[ \text{dtmax: } \] user specified maximum allowable difference for o-g difference
\[ \text{nsize: } \] the sample size number. If sample size is less than this number, the updating weight will be reduced based on sample size
\[ \text{wgtang: } \] weight for updating the mean temperature lapse rate
\[ \text{wgtlap: } \] 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/ccf/com/gfs/prod](http://nomads.ncep.noaa.gov/pub/data/ccf/com/gfs/prod)

Once in the sub-directory, look for files with name similar to:

\[ \text{gdas1.t00z.abias } \] for coefficients of mass bias correction.

For NAM: [http://nomads.ncep.noaa.gov/pub/data/ccf/com/nam/prod](http://nomads.ncep.noaa.gov/pub/data/ccf/com/nam/prod)

Once in the sub-directory, look for files with name similar to:

\[ \text{nam.t00z.satbias.tm00 } \] 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 \textit{satbias\_in} and \textit{satbias\_angle}

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

\[ \text{satbias\_in: sample.satbias } \] \[ \text{satbias\_angle: global\_satangbias.txt and nam\_global\_satangbias.txt } \]

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.
Satellite Radiance Data Assimilation

- 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 Radiance 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 Radiance 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:


In the comGSI_v3.2 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 comGSI_v3.2 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
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

<table>
<thead>
<tr>
<th>mnemonic</th>
<th>meaning</th>
<th>dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>RADAR STATION IDENTIFIER (not used in GSI)</td>
<td>1</td>
</tr>
<tr>
<td>XOB</td>
<td>X-index for grid coordinate of reflectivity column</td>
<td>1</td>
</tr>
<tr>
<td>YOB</td>
<td>Y-index for grid coordinate of reflectivity column</td>
<td>1</td>
</tr>
<tr>
<td>DHR</td>
<td>OBSERVATION TIME MINUS CYCLE TIME (not used in GSI)</td>
<td>1</td>
</tr>
<tr>
<td>TYP</td>
<td>PREPBUFR REPORT TYPE (not used in GSI)</td>
<td>1</td>
</tr>
<tr>
<td>HREF</td>
<td>Horizontal reflectivity</td>
<td>31</td>
</tr>
</tbody>
</table>

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 1\textsuperscript{st} 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:

\texttt{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 soverage, 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:
Radar Data Assimilation

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:

\[ \text{dfile}(09)='\text{radarbufr}', \text{dtype}(09)=\text{'rw'}, \quad \text{dplat}(09)=\text{''}, \quad \text{dsis}(09)=\text{'rw'}, \quad \text{dval}(09)=1.0, \quad \text{dthin}(09)=0, \quad \text{dsfcalc}(09)=0, \]

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:

\[
\&\text{SUPEROB\_RADAR} \\
\quad \text{del\_azimuth}=5., \quad \text{del\_elev}=.25, \quad \text{del\_range}=5000., \quad \text{del\_time}=.5, \quad \text{elev\_angle\_max}=5., \quad \text{minnum}=50, \quad \text{range\_max}=100000., \\
\quad \text{l2superob\_only}=\text{.false.}., \\
\]

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:

\[
\text{rw} \quad 999 \quad 0 \quad 1 \quad 2.5 \quad 0 \quad 0 \quad 0 \quad 10.0 \quad 10.0 \quad 2.0 \quad 10.0 \quad 0.00000 \quad 0 \quad 0. \quad 0. \quad 0
\]

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., minnum=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 MPTH 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 \textit{stdout} file:

\begin{verbatim}
create superobs only, radar file date = 'oyear' 'omonth' 'oday' 'ohour'
RADAR_BUFR_READ_ALL: analysis time is 'oyear' 'omonth' 'oday' 'ohour'
\end{verbatim}

If the parameter \texttt{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 \textit{stdout} file:

\begin{verbatim}
using restart file date = 'byear' 'bmonth' 'bday' 'bhour'
RADAR_BUFR_READ_ALL: analysis time is 'oyear' 'omonth' 'oday' 'ohour'
\end{verbatim}

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 \textquote{DIST125M} data is used to determine the multiplying factor for radial distance. If the minimum difference of \textquote{DIST125M} between the adjacent gates is 1, the factor is set to 250. If the minimum difference of \textquote{DIST125M} between the adjacent gates is 2, the factor is set to 125. If users see the following message in the \textit{stdout} file:

\begin{verbatim}
RADAR_BUFR_READ_ALL: problem with level 2 bufr file, gate distance scale factor undetermined, going with 125
\end{verbatim}

which means the minimum difference of \textquote{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 \textquote{DIST125M} data.

- The GSI counts the radar number according to the radar station identifier \textquote{SSTN}
  - The default maximum number of radars that GSI would deal with is 150, if users see the following message in the \textit{stdout} file:

\begin{verbatim}
RADAR_BUFR_READ_ALL: stop processing level 2 radar bufr file--increase parameter max_num_radars
\end{verbatim}

which means the radar numbers in the BUFR file exceed 150, if so, the parameter \texttt{max\_num\_radars} should be changed in the file \texttt{read\_l2bufr\_mod.f90}:

\begin{verbatim}
integer(i_kind),parameter:: max_num_radars=150
\end{verbatim}

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

\begin{verbatim}
RADAR_BUFR_READ_ALL: NO RADARS KEPT IN radar_bufr_read_all, continue without level 2 data
\end{verbatim}

will be printed in the \textit{stdout} file and the \textquote{super-obbing} process will not be done.

  - 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 \textit{stdout} file:
The unique master tables of all radar station identifier, latitude, longitude and height are created.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Dimension</th>
<th>Content</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>master_stn_table</td>
<td>total radar numbers in l2rwbufr file</td>
<td>radar station identifier</td>
<td>SSTN</td>
</tr>
<tr>
<td>master_lat_table</td>
<td></td>
<td>radar station latitude</td>
<td>CLAT</td>
</tr>
<tr>
<td>master_lon_table</td>
<td></td>
<td>radar station longitude</td>
<td>CLON</td>
</tr>
<tr>
<td>master_hgt_table</td>
<td></td>
<td>radar station height</td>
<td>HSMSL+HSALG</td>
</tr>
</tbody>
</table>

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 \textit{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 \textit{range\_max}, the radar data in this gate will not be used. The parameter \textit{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 \textit{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 \textit{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 \textit{irbin} which ranges from 1 to \textit{nrbin}, if the distance index is out of this range, the radar data in this gate will not be used. The parameter \textit{nobs\_lrbin} is used to counter the number of gates which have the distance index less than 1 and \textit{nobs\_hrbin} is used to counter the number of gates which have the distance index greater than \textit{nrbin}.

Step 5, the three-dimensional coordinate (\textit{izabin}, \textit{ielbin}, \textit{irbin}) is transferred into one-dimensional coordinate \textit{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 \textit{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 \textit{minnum}, the data of this super-ob box will not be used.

After doing these steps, all the statistical information is listed in the \textit{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
                     44.00    4372
master list radar   2 stn id,lat,lon,hgt,num = SHQP     31.08    120.96
                     42.00    6408
RADAR_BUFR_READ_ALL:  ddiffmin,distfact,idups=   2.00000000000000
                       125.000000000000                0
nthisrad=       28800
nthisbins=      172800
timemin,max= -3.05555555555555E-002  2.52777777777778E-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
```
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_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 except for the radars near the polar (radar station latitude is higher than 89.5 degrees). The variables include `thisrange` (radial range),
Radar Data Assimilation

**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:

\[
thiserr = \sqrt{V_r^2 - \overline{V_r}^2}
\]

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 **thison** (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**.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>this_staid</strong></td>
<td>radar station identifier</td>
</tr>
<tr>
<td><strong>this_atalat</strong></td>
<td>radar station latitude</td>
</tr>
<tr>
<td><strong>this_stalon</strong></td>
<td>radar station longitude</td>
</tr>
<tr>
<td><strong>this_stahgt</strong></td>
<td>radar station height</td>
</tr>
<tr>
<td><strong>thistime</strong></td>
<td>time difference between observation and background</td>
</tr>
<tr>
<td><strong>thislat</strong></td>
<td>super-ob box latitude</td>
</tr>
<tr>
<td><strong>thison</strong></td>
<td>super-ob box longitude</td>
</tr>
<tr>
<td><strong>thishgt</strong></td>
<td>super-ob box height</td>
</tr>
<tr>
<td><strong>thisvr</strong></td>
<td>mean radial velocity</td>
</tr>
<tr>
<td><strong>corrected_azimuth</strong></td>
<td>corrected azimuth</td>
</tr>
<tr>
<td><strong>thiserr</strong></td>
<td>mean radial velocity errorr</td>
</tr>
<tr>
<td><strong>corrected_tilt</strong></td>
<td>corrected elevation angle</td>
</tr>
</tbody>
</table>

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

```plaintext
for radar RSHI nsuper= 4372  delazmmax= 0.531495369516961
  vrmin,max= -20.9300000000000  20.9700000000000  errmin,max=
  0.309294787065859  14.6110084866894
deltiltmin,max= 2.2395414549778E-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=
  0.298890185862377  21.7435944590585
deltiltmin,max= 2.10245505536138E-002  0.6658190756920249
  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 -20.93m/s and 20.97m/s respectively
- ‘**errmin,max= 0.309294787065859  14.6110084866894**’ means the minimum and maximum values of the mean radial velocity error are 0.309294787065859m/s and
14.6110084866894m/s respectively

- ‘deltlmin,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 -326.328815013534m and -0.148169009099547m respectively

<table>
<thead>
<tr>
<th>total number of superobs written</th>
<th>10780</th>
</tr>
</thead>
<tbody>
<tr>
<td>vrmin,maxall= -24.4200000000000</td>
<td>22.9650000000000</td>
</tr>
<tr>
<td>errmin,maxall= 0.298880185862377</td>
<td>21.7435944590585</td>
</tr>
<tr>
<td>delazmaxall= 0.531495369516961</td>
<td></td>
</tr>
<tr>
<td>deltltmin,maxall= 2.102455505536138E-002 0.665819075690249</td>
<td></td>
</tr>
<tr>
<td>deldistmin,maxall= -326.328815013534 -9.66676340202685E-002</td>
<td></td>
</tr>
</tbody>
</table>

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 -24.42m/s and 22.965m/s respectively

- ‘errmin,maxall=0.298880185862377 21.7435944590585’ means the totally minimum and maximum values of the mean radial velocity error are 0.298880185862377m/s and 21.7435944590585m/s respectively

- ‘delazmaxall=0.531495369516961’ means the totally maximum corrected value of the azimuth is 0.531495369516961 degrees

- ‘deltlmin,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 -9.66676340202685E-002’ means the totally minimum and maximum corrected values of the radial distance are -326.328815013534m and -9.66676340202685E-002m respectively

Reference

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, ‘tlddlbrbuf’ and ‘tlddlps’ files
2. Set some parameters:

\[
\begin{align*}
\text{vad\_leash} &= 0.3 \text{ (used in VAD QC)} \\
\text{xscale} &= 20000 \text{ (horizontal scale, unit: meters)} \\
\text{maxvadbins} &= 15 \text{ (the maximum of VAD levels)} \\
\text{dzvad} &= 304.8 \text{ (vad reports are every 1000 ft = 304.8 meters)}
\end{align*}
\]

The information of these parameters will be listed in \textit{stdout} file:

\[
\begin{align*}
\text{READ\_RADAR: } &\text{ set } \text{vad\_leash},\text{xscale} = 0.300000000000000 \ 20000.000000000000 \\
\text{READ\_RADAR: } &\text{ set } \text{maxvadbins},\text{maxbadbins}\text{dzvad} = 15 \ 4572.000000000000
\end{align*}
\]

3. Open BUFR file ‘\textit{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 ‘\textit{vadfile}’ file does not exist the program will still go on and users will see the information in \textit{stdout} file:

\[
\begin{align*}
\text{READ\_RADAR: } &\text{nsuper2\_in},\text{nsuper2\_kept} = 12482 \ 0 \\
\text{READ\_RADAR: } &\# \text{ no vad match} = 12482
\end{align*}
\]

It tells you that all the observations have no VAD wind to match, and no observation is kept.

If the file ‘\textit{vadfile}’ exists and reads the first message correctly, a line will be found in \textit{stdout} file:

\[
\begin{align*}
\text{READ\_RADAR: } &\text{first read vad winds}\ --\ \text{use vad quality marks to qc 2.5/3 radar winds}
\end{align*}
\]

4. Find out whether the VAD data is in the BUFR file according to subtype(224) or type in the ‘\textit{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 ‘\textit{twindow}’ in the ‘\textit{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 \textit{nvad} counts the number of VAD wind stations, if it exceeds the \textit{maxvad} (default value is 500) defined in the program, the program will stop and the error will be printed in the \textit{stdout} file:

\[
\begin{align*}
\text{READ\_RADAR: } &\text{***ERROR*** MORE THAN 500 RADARS: PROGRAM STOPS}
\end{align*}
\]

which means the VAD wind station numbers in the BUFR file exceed 500, if so, the parameter \textit{maxvad} should be changed in the file \textit{read\_radar.f90}:

\[
\begin{align*}
\text{integer(i\_kind), parameter:: maxvad=500}
\end{align*}
\]
The VAD wind information table is created

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Dimension</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>vadlon</td>
<td>nvd</td>
<td>VAD station longitude</td>
</tr>
<tr>
<td>vadlat</td>
<td></td>
<td>VAD station latitude</td>
</tr>
<tr>
<td>vadid</td>
<td></td>
<td>VAD station identifier</td>
</tr>
</tbody>
</table>

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).

\[
\text{diffuu} = U_{vad} - U_{bk} \\
\text{diffvv} = V_{vad} - V_{bk}
\]

If \(\sqrt{\text{diffuu}^2 + \text{diffvv}^2} > 10.0\), the VAD data will not be used.
If \(|\text{diffvv}| > 8.0\), the VAD data will not be used.
If \(|\text{diffvv}| > 5.0\) 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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Dimension</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>errzmax</td>
<td>1</td>
<td>The maximum difference between observation height and the nearest VAD level height</td>
</tr>
<tr>
<td>vadqm</td>
<td>(nvad, levs)</td>
<td>The maximum value of WQM (VAD U-, V-component wind quality marker) of a VAD wind station at a</td>
</tr>
</tbody>
</table>
### Radar Data Assimilation

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>vadqmmmax</td>
<td>The maximum value of the vadqm array</td>
</tr>
<tr>
<td>vadqmmmin</td>
<td>The minimum value of the vadqm array</td>
</tr>
<tr>
<td>vadu</td>
<td>Add all the VAD U-component wind up at the same VAD station and the same level</td>
</tr>
<tr>
<td>vadv</td>
<td>Add all the VAD V-component wind up at the same VAD station and the same level</td>
</tr>
<tr>
<td>vadcount</td>
<td>Count the numbers at the same VAD station and the same level</td>
</tr>
</tbody>
</table>

#### 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
 errzmax = 48.00000000000000
```

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>this_staid</td>
<td>radar station identifier</td>
</tr>
<tr>
<td>this_atalat</td>
<td>radar station latitude</td>
</tr>
<tr>
<td>this_stalon</td>
<td>radar station longitude</td>
</tr>
<tr>
<td>this_stahgt</td>
<td>radar station height</td>
</tr>
<tr>
<td>thistime</td>
<td>time difference between observation and background</td>
</tr>
<tr>
<td>thislat</td>
<td>super-ob box latitude</td>
</tr>
<tr>
<td>thislon</td>
<td>super-ob box longitude</td>
</tr>
<tr>
<td>thishgt</td>
<td>super-ob box height</td>
</tr>
<tr>
<td>thisvr</td>
<td>mean radial wind</td>
</tr>
<tr>
<td>corrected_azimuth</td>
<td>corrected azimuth</td>
</tr>
<tr>
<td>thiserr</td>
<td>mean radial wind error</td>
</tr>
<tr>
<td>corrected_tilt</td>
<td>corrected elevation angle</td>
</tr>
</tbody>
</table>
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 irr 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 irr equals 1, and 16 parts when irr equals 2, 24 parts when irr equals 3, and so on.

\[\text{iaaamax and iaaamin are the maximum and minimum of the observation azimuth index respectively.}\]
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 qcmd.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. ibadheighbelowsta 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:
    \[ \text{thiswgt} = \frac{1}{\max(4.0, \text{thiserr}^2)} \], where thiserr is mean radial wind error of the super-ob
  - square of the radial wind difference between VAD and super-ob observation:
    \[ \text{thisfit2} = (\text{VAD}_{vr} - \text{this}_{vr})^2 \], where \( VAR_{vr} \) is the radial wind calculated from VAD U,V-component wind and thisvr is mean radial wind of the super-ob
Radar Data Assimilation

- square root of this fit:
  \[ thisfit = \sqrt{thisfit^2} \]
- 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 \( thiswt^2 \) up
- \( vadcount2 \) is used to count the number
- \( vadwgt2 \) is used to add all the \( thiswt \) 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 \( iirrr \), 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.

<table>
<thead>
<tr>
<th>Output Array Member</th>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cdata(1)</td>
<td>error</td>
<td>wind observation error (m/s)</td>
</tr>
<tr>
<td>cdata(2)</td>
<td>dlon</td>
<td>grid relative longitude</td>
</tr>
<tr>
<td>cdata(3)</td>
<td>dlat</td>
<td>grid relative latitude</td>
</tr>
<tr>
<td>cdata(4)</td>
<td>height</td>
<td>observation absolute height (m)</td>
</tr>
<tr>
<td>cdata(5)</td>
<td>rwnd</td>
<td>radial wind observation (m/s)</td>
</tr>
<tr>
<td>cdata(6)</td>
<td>azm*deg2rad</td>
<td>azimuth angle (radians)</td>
</tr>
<tr>
<td>cdata(7)</td>
<td>t4dv</td>
<td>observation time (hour)</td>
</tr>
<tr>
<td>cdata(8)</td>
<td>ikx</td>
<td>observation type</td>
</tr>
</tbody>
</table>
### Radar Data Assimilation

<table>
<thead>
<tr>
<th>cdata(9)</th>
<th>tiltangle</th>
<th>tilt angle (radians)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cdata(10)</td>
<td>staheight</td>
<td>station elevation (m)</td>
</tr>
<tr>
<td>cdata(11)</td>
<td>rstation_id</td>
<td>station id</td>
</tr>
<tr>
<td>cdata(12)</td>
<td>usage</td>
<td>usage parameter</td>
</tr>
<tr>
<td>cdata(13)</td>
<td>idomsfc</td>
<td>dominate surface type</td>
</tr>
<tr>
<td>cdata(14)</td>
<td>skint</td>
<td>skin temperature</td>
</tr>
<tr>
<td>cdata(15)</td>
<td>ff10</td>
<td>10 meter wind factor</td>
</tr>
<tr>
<td>cdata(16)</td>
<td>sfcr</td>
<td>surface roughness</td>
</tr>
<tr>
<td>cdata(17)</td>
<td>dlon_earth*rad2deg</td>
<td>earth relative longitude (degrees)</td>
</tr>
<tr>
<td>cdata(18)</td>
<td>dlat_earth*rad2deg</td>
<td>earth relative latitude (degrees)</td>
</tr>
<tr>
<td>cdata(19)</td>
<td>dist</td>
<td>range from radar in km (used to estimate beam spread)</td>
</tr>
<tr>
<td>cdata(20)</td>
<td>zsges</td>
<td>model elevation at radar site</td>
</tr>
<tr>
<td>cdata(21)</td>
<td>thiserr</td>
<td>mean radial wind error</td>
</tr>
<tr>
<td>cdata(22)</td>
<td>two</td>
<td></td>
</tr>
</tbody>
</table>

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 vadrang= 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 vadwind = 0
READ_RADAR: # bad fit = 0
READ_RADAR: # num thinned = 0
READ_RADAR: # not good0 = 372
READ_RADAR: # not good = 0
READ_RADAR: # hgt below sta=0
READ_RADAR: timemin,max = 4.940656458412465E-324 4.940656458412465E-324
READ_RADAR: ermin,max = 0.198997487421342 26.2045158189491
READ_RADAR: dlatmin,max,dionmin,max= 200.880606608956 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 vadrang= 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 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 = 0.19897487421342 26.2045158189491’ means the minimum and
  maximum of the observation error respectively.
• ‘dlatmin,max,dlonmin,max = 200.88060608956 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.
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’, dpplat(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
```

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 control the reflectivity:

```
rad_ref 999 0 1.5 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
```
6. Check the reflectivity analysis results

Because the reflectivity is not analyzed 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 Global GSI analysis

The Global Forecast System (GFS) is a global numerical weather prediction system containing a global computer model and variational analysis run by the U.S. National Weather Service (NWS). As of February 2015, the mathematical model is run four times a day, and produces forecasts for up to 16 days in advance, with decreased spatial resolution after 10 days. The model is a spectral model with a resolution of T1534 from 0 to 240 hours (0-10 days) and T574 from 240 to 384 hours (10-16 days). In the vertical, the model is divided into 64 layers and temporally, it produces forecast output every hour for the first 12 hours, every 3 hours out to 10 days, and every 12 hours after that. Its data assimilation system runs 6-hourly continuous cycles using the GSI-hybrid.

GSI has many functions specially designed and suited for GFS. Although the release version of the community GSI includes all the functions used by the operational systems, the DTC can only support the GSI regional applications because the DTC is not able to run GFS on community computers. Beginning with release version 3.2, the DTC began to introduce the use of GSI for global applications, assuming users can obtain the GFS background through the NCEP data hub or by running GFS themselves.

10.1.1 The difference between GSI global and regional

As mentioned previously, all NCEP operational systems are using GSI as the analysis system, but there is only one set of GSI code. The majority of the GSI code is shared by these operational systems with only a small amount of code that is specific for a particular operational system. The major differences among those operational systems in terms of GSI applications are configuring the run scripts to set up the special run environment and namelist parameters.

Different GSI applications need different backgrounds, observations, and fixed files. For the GFS system, GSI needs:

- GFS Backgrounds: usually, GSI uses 6-h GFS forecasts as the background. GFS 3-h and 9-h forecasts are also needed for the FGAT function in the GSI analysis. Both surface and atmosphere forecasts are needed.

- Observations: NCEP has several sets of BUFR/PrepBUFR observations files with global coverage for global systems. The files that start with “GDAS” are for the 6-hourly global data assimilation system, which has more data available for the analysis but has to wait longer for use in real-time. The files that start with “gfs” are for the GFS forecast 4 times a day. The different operational systems need different observation data files because they require different
kinds of observations with different coverage, cut-off times, and quality control processes. All these observation files are read in and processed in GSI by the same code. Therefore, there is no problem using GFS observation data files for regional GSI applications, as done in the practices examples using the community GSI in the fundamental User’s Guide. Using regional BUFR files for global applications will cause scientific problems because the data only covers part of the analysis domain, although GSI can still read in observations and perform the analysis.

- Fixed files: In Section 3.1 of the fundamental User’s Guide, we introduced that different operational systems have their own fixed files. These global fixed files can be downloaded as a stand-alone tar ball from the GSI user’s website (http://www.dtcenter.org/com-GSI/users/downloads/index.php). For the GFS GSI application, the big difference is the background error covariance (BE). Different resolutions of the GFS backgrounds use their matched BE files, which are different from the BE files used by the regional GSI applications. In release version 3.4, we provide ten BE files (in addition to the two for release version 3.3):
  - global_berror.l64y386.f77
  - global_berror.l64y96.f77
  - global_berror.l64y1154.f77
  - global_berror.l64y290.f77
  - global_berror.l64y882.f77
  - global_berror.l64y130.f77
  - global_berror.l64y192.f77
  - global_berror.l64y578.f77
  - global_berror.l64y258.f77
  - global_berror.l64y674.f77

### 10.1.2 Global GFS scripts

Starting release version 3.3, we began to help community users to run GSI global applications through a sample run script under directory ./run. The script run_gsi_global.ksh is set based on GSI GFS regression tests. This script has a similar structure to the regional run script run_gsi_regional.ksh, but include a couple of different details.

The first piece of the run script is to set up the computer environment and case configuration. This part is the similar to the regional runs except specifying the choice of GFS case and the namelist for global application:

```
GFSCASE=T126
GSI_NAMELIST=${GSI_ROOT}/run/comgsi_namelist_gfs.sh
```

Different from regional, instead of selecting background and BE files, the global needs to know the background resolution setup by the following parameters.
Similar to the regional run script, this global run script will also double check the needed parameters. Then it creates a run directory and generates the namelist in the directory and copies the background, observations, and fixed files into the run directory. Certainly, many details are different between global and regional applications, which is shown in several steps:

1. Specify the values of LATA, LONA, DELTIME, resol based on the choice of JCAP:

   # Given the requested resolution, set dependent resolution parameters
   if ["$JCAP" = "382"] then
     LONA=768
     LATA=384
     DELTIM=180
     resol=1
   elif ["$JCAP" = "574"] then
     LONA=1152
     LATA=576
     DELTIM=1200
     resol=2
   elif ["$JCAP" = "254"] then
     LONA=512
     LATA=256
     DELTIM=1200
     resol=2
   elif ["$JCAP" = "126"] then
     LONA=256
     LATA=128
     DELTIM=1200
     resol=2
   elif ["$JCAP" = "62"] then
     LONA=192
     LATA=94
   fi
DELTIM=1200
resol=2
else
  echo "INVALID JCAP = $JCAP"
  exit
fi
NLAT=` expr $LATA + 2`

2. Set up CO2, CH4, N2O, CO file decisions:

# CO2 namelist and file decisions
ICO2=${ICO2:-0}
if [ $ICO2 -gt 0 ] ; then
  # Copy co2 files to $workdir
  co2dir=${FIX_ROOT}/global
  yyyy=`echo $ANAL_TIME | cut -c1-4`
  rm ./global_co2_data.txt
  co2=${co2dir}/global_co2.gcmscl_$yyyy.txt
  while [ ! -s $co2 ] ; do
    ((yyyy-=1))
    co2=${co2dir}/global_co2.gcmscl_$yyyy.txt
  done
  if [ -s $co2 ] ; then
    cp $co2 ./global_co2_data.txt
  fi
  if [ ! -s ./global_co2_data.txt ] ; then
    echo "/.global_co2_data.txt" not created
    exit 1
  fi
fi

#CH4 file decision
ICH4=${ICH4:-0}
if [ $ICH4 -gt 0 ] ; then
  # Copy ch4 files to $workdir
  ch4dir=${FIX_ROOT}/global
  ...

3. Set up the namelist parameters and generate the namelist:

  # Set some parameters for use by the GSI executable and to build the namelist
  echo " Build the namelist "
  vs_op='0.7,'
  hzscl_op='1.7,0.8,0.5,'
  ...

  # Build the GSI namelist on-the-fly
  . $GSI_NAMELIST
cat << EOF > gsiparm.anl
$comgsi_namelist
EOF

4. Multiple time level backgrounds are needed:

  cp $BK_ROOT/gdas$(resol).t$h$h$z.bf03 ./sfcf03
Both surface and atmosphere files at 03, 06, 09 hour forecasts are needed. In our example data, these files are:

```
gdas1.t00z.sgesprep  gdas1.t18z.bf03
gdas1.t00z.sgm3prep  gdas1.t18z.bf06
gdas1.t00z.sgp3prep  gdas1.t18z.bf09
```

5. More radiance observations files are available

In the sample run script, many more radiance observations are listed for use:

```
# Link to the radiance data
suffix=tm00.bufr_d
ln -s ${OBS_ROOT}/gdas1.t00z.satwnd.${suffix} ./satwndbufr
ln -s ${OBS_ROOT}/gdas1.t00z.gpsro.${suffix} ./gpsrobufr
#ln -s ${OBS_ROOT}/gdas1.t00z.ssmi.${suffix} ./ssmirbufr
ln -s ${OBS_ROOT}/gdas1.t00z.sptrmm.${suffix} ./tmirrbufr
ln -s ${OBS_ROOT}/gdas1.t00z.osbuv8.${suffix} ./sbuvbufr
##ln -s ${OBS_ROOT}/gdas1.t00z.goesfv.${suffix} ./gsndlbufr
ln -s ${OBS_ROOT}/gdas1.t00z.ibamaua.${suffix} ./amsuabufr
ln -s ${OBS_ROOT}/gdas1.t00z.ibamub.${suffix} ./amsubbufr
#ln -s ${OBS_ROOT}/gdas1.t00z.1bhrs2.${suffix} ./hirs2bufr
ln -s ${OBS_ROOT}/gdas1.t00z.1bhrs3.${suffix} ./hirs3bufr
ln -s ${OBS_ROOT}/gdas1.t00z.1bhrs4.${suffix} ./hirs4bufr
ln -s ${OBS_ROOT}/gdas1.t00z.1bmhs.${suffix} ./mhsbufr
#ln -s ${OBS_ROOT}/gdas1.t00z.1bmhs.${suffix} ./mhsbufr
##ln -s ${OBS_ROOT}/gdas1.t00z.airsev.${suffix} ./airsbufr
ln -s ${OBS_ROOT}/gdas1.t00z.sevcr.${suffix} ./sevirbufr
## ln -s ${OBS_ROOT}/gdas1.t00z.mtiasi.${suffix} ./iasibufr
## ln -s ${OBS_ROOT}/gdas1.t00z.smit.${suffix} ./smitbufr
ln -s ${OBS_ROOT}/gdas1.t00z.amsre.${suffix} ./amsrebfr
##ln -s ${OBS_ROOT}/gdas1.t00z.ssms.${suffix} ./ssmsbufr
ln -s ${OBS_ROOT}/gdas1.t00z.rome.${suffix} ./romebufr
ln -s ${OBS_ROOT}/gdas1.t00z.omi.${suffix} ./omibufr
#ln -s ${OBS_ROOT}/gdas1.t00z.mlsbufr.${suffix} ./mlsbufr
ln -s ${OBS_ROOT}/gdas1.t00z.syndata.tcivitals.tm00 ./tcivitl
```

Below is a table of the grid dimensions for GFS of different resolutions from “Running Global Model Parallel Experiments Version 6.0” from NCEP/EMC, which may provide users more information on the above mentioned resolution parameters:

<table>
<thead>
<tr>
<th>SPECTRAL RESOLUTION</th>
<th>EULERIAN LONB</th>
<th>EULERIAN LATB</th>
<th>SEMI-LAGRANGIAN LONB</th>
<th>SEMI-LAGRANGIAN LATB</th>
</tr>
</thead>
<tbody>
<tr>
<td>T62</td>
<td>192</td>
<td>94</td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>T126</td>
<td>384</td>
<td>190</td>
<td>256</td>
<td>128</td>
</tr>
<tr>
<td>T170</td>
<td>512</td>
<td>256</td>
<td>352</td>
<td>176</td>
</tr>
</tbody>
</table>
10.1.3 Sample results.

After a successful GSI GFS analysis, the run directory may look like (with the clean option turned on):

```
amsrebufr                         diag_mhs_n18_anl.2011080100.gz       fort.213
amsuabufr                         diag_mhs_n18_ges.2011080100.gz       fort.214
amsubbufr                         diag_mhs_n19_anl.2011080100.gz       fort.215
anavinfo                           diag_mhs_n19_ges.2011080100.gz       fort.217
atma_beamwidth.txt               diag_omi_aura_anl.2011080100.gz      fort.218
beamr_stats                        diag_omi_aura_ges.2011080100.gz     fort.219
berror_stats                       diag_pcp_tmi_trmm_anl.2011080100.gz  fort.220
convinfo                           diag_pcp_tmi_trmm_ges.2011080100.gz  fort.221
diag_amsre_hig_aqua_anl.2011080100.gz
                                 diag_amsre_hig_aqua_ges.2011080100.gz
                                 diag_amsre_low_aqua_anl.2011080100.gz
                                 diag_amsre_low_aqua_ges.2011080100.gz
                                 diag_amsre_mid_aqua_anl.2011080100.gz
                                 diag_amsre_mid_aqua_ges.2011080100.gz
                                 diag_amsua_metop-a_anl.2011080100.gz
                                 diag_amsua_metop-a_ges.2011080100.gz
                                 diag_amsua_n15_anl.2011080100.gz
                                 diag_amsua_n15_ges.2011080100.gz
                                 diag_amsua_n18_anl.2011080100.gz
                                 diag_amsua_n18_ges.2011080100.gz
                                 diag_amsua_n19_anl.2011080100.gz
                                 diag_amsua_n19_ges.2011080100.gz
                                 diag_conv_anl.2011080100.gz
                                 diag_conv_ges.2011080100.gz
                                 diag_gome_metop-a_anl.2011080100.gz
                                 diag_gome_metop-a_ges.2011080100.gz
                                 diag_hirs3_n17_anl.2011080100.gz
                                 diag_hirs3_n17_ges.2011080100.gz
                                 diag_hirs4_meteop-a_anl.2011080100.gz
                                 diag_hirs4_meteop-a_ges.2011080100.gz
                                 diag_hirs4_n19_anl.2011080100.gz
                                 diag_hirs4_n19_ges.2011080100.gz
                                 diag_mhs_metop-a_anl.2011080100.gz
                                 diag_mhs_metop-a_ges.2011080100.gz
```
instead of only one “wrf_inout”. A quick check of the standard output file stdout will also show information similar to what was shown in previous regional runs with respect to the namelist, data ingest, minimization, but quite different with respect to background IO.

Please visit our online tutorial for more details regarding how to conduct a global GSI run.

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 \texttt{./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 (e.g., 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 \texttt{./util/RTMA/rtma\_firstguess};
2) open file “\texttt{param.incl}”; 
3) find the following lines (starts from line 94):

\begin{verbatim}
!==>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
\end{verbatim}

4) modify the values to fit the user’s specific domain:

\begin{itemize}
  \item \texttt{nx\_dtc}: analysis domain dimension in X direction
  \item \texttt{ny\_dtc}: analysis domain dimension in Y direction
  \item \texttt{alat1\_dtc}: analysis domain latitude of southwestern most point
  \item \texttt{elon1\_dtc}: analysis domain longitude of southwestern most point
  \item \texttt{da\_dtc}: analysis grid space in meters
\end{itemize}

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:

\texttt{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:

\textbf{For Intel compiler, pick:}
\begin{verbatim}
FC=ifort
FFLAGS=-nofixed -convert big\_endian
\end{verbatim}

\textbf{For PGI compiler, pick:}
\begin{verbatim}
FC=pgf90
FFLAGS= -Mfree -byteswapio
\end{verbatim}

Then, in the same directory, compile the code using the command:
GSI Application

./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/releaseV3.3_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 ./util/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
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:

```plaintext
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:

```fortran
write(88) ihdrbuf
write(88) iyear,imonth,iday,ihour,iminute,isecond,nx,ny,nsig
write(88) dx,dy
write(88) glat
cycle
write(88) glon
write(88) psfcgrid ! psfc0
write(88) phbgrid ! PHB (zsfc*g)
write(88) tgrid ! T(k) ! TEMP (sensible)
write(88) ggrid ! 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
```
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:

```fortran
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 fundamental 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.

```bash
BK_DIR=comGSI/releaseV33/util/RTMA/rtmagus
ROOTDIR= comGSI/releaseV33/util/RTMA
```
GSI Application

- **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:

```fortran
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
```
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, temperature, and specific humidity, respectively.

• sclpsi, sclchi, sclpsfc, sctltemp, 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):

```plaintext
!==>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

```plaintext
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 format.
- `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

10.3 GSI hybrid

10.4 GSI 4DVAR and FSO

10.5 GSI Chem
References


Appendix A: GSI Namelist: Name, Default value, Explanation

The following are lists and explanations of the GSI namelist variables. You can also find them in the source code `gsimod.F90`.

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>&amp;SETUP</strong></td>
<td></td>
<td><strong>General control namelist</strong></td>
</tr>
<tr>
<td>gencode</td>
<td>80</td>
<td>source generation code</td>
</tr>
<tr>
<td>factqmin</td>
<td>1</td>
<td>weighting factor for negative moisture constraint</td>
</tr>
<tr>
<td>factqmax</td>
<td>1</td>
<td>weighting factor for supersaturated moisture constraint</td>
</tr>
<tr>
<td>clip_supersaturation</td>
<td>.false.</td>
<td>flag to remove supersaturation during each outer loop</td>
</tr>
<tr>
<td>factv</td>
<td>1</td>
<td>weighting factor for negative visibility constraint</td>
</tr>
<tr>
<td>deltim</td>
<td>1200</td>
<td>model timestep</td>
</tr>
<tr>
<td>dtphys</td>
<td>3600</td>
<td>physics timestep</td>
</tr>
<tr>
<td>biascor</td>
<td>-1</td>
<td>background error bias correction coefficient</td>
</tr>
<tr>
<td>bcoption</td>
<td>1</td>
<td>0=ibc (no bias correction to bkg); 1=sbc(original implementation)</td>
</tr>
<tr>
<td>diurnalbc</td>
<td>0</td>
<td>1= diurnal bias; 0= persistent bias</td>
</tr>
<tr>
<td>niter(0:50)</td>
<td>0, …</td>
<td>Maximum number of inner loop iterations for each outer loop</td>
</tr>
<tr>
<td>niter_no_qc</td>
<td>1000000</td>
<td>Inner loop iteration at which to turn on variational quality control</td>
</tr>
<tr>
<td>(0:50)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>miter</td>
<td>1</td>
<td>number of outer loops</td>
</tr>
<tr>
<td>qoption</td>
<td>1</td>
<td>option for moisture analysis variable; 1:q/qsatg 2:normalized RH</td>
</tr>
<tr>
<td>pseudo_q2</td>
<td>.false.</td>
<td>breed between q1/q2 options, that is, (q1/sig(q))</td>
</tr>
<tr>
<td>nhr_assimilation</td>
<td>6</td>
<td>assimilation time interval (currently 6 hours for global, 3 hours for regional)</td>
</tr>
<tr>
<td>min_offset</td>
<td>3</td>
<td>time of analysis in assimilation window</td>
</tr>
<tr>
<td>iout_iter</td>
<td>220</td>
<td>output file number for iteration information</td>
</tr>
<tr>
<td>npredp</td>
<td>6</td>
<td>number of predictors for precipitation bias</td>
</tr>
</tbody>
</table>
GSI Namelist

retrieval  .false.  logical to turn off or on the SST physical retrieval
nst_gsi  0  indicator to control the Tr Analysis mode:
          0 = no nst info input at all;
          1 = input nst info, but used for monitoring only
          2 = input nst info, and used in CRTM simulation, but no Tr analysis
          3 = input nst info, and used in CRTM simulation and Tr analysis is on
nst_tzr  0  indicator to control the Tzr_QC mode:
          0 = no Tz retrieval;
          1 = Do Tz retrieval and applied to QC
nstinfo  0  number of nst variables
fac_dtl  0  index to apply diurnal thermocline layer or not: 0 = no; 1 = yes
fac_tsl  0  index to apply thermal skin layer or not: 0 = no; 1 = yes.
tzr_bufrsave  .false.  logical to turn off or on the bufr Tz retrieval file true=on
diag_rad  .true.  logical to turn off or on the diagnostic radiance file (true=on)
diag_pcp  .true.  logical to turn off or on the diagnostic precipitation file (true=on)
diag_conv  .true.  logical to turn off or on the diagnostic conventional file (true=on)
diag_ozone  .true.  logical to turn off or on the diagnostic ozone file (true=on)
diag_aero  .false.  logical to turn off or on the diagnostic aerosol file (true=on)
diag_co  .false.  logical to turn off or on the diagnostic carbon monoxide file (true=on)
iguess  1  flag for guess solution (currently not working)
        -1  do not use guess file
            0  write only guess file
            1  read and write guess file
            2  read only guess file
write_diag  .false., ...  logical to write out diagnostic files for outer iteration
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reduce_diag</td>
<td>.false.</td>
<td>Namelist logical to produce reduced radiance diagnostic files</td>
</tr>
<tr>
<td>oneobtest</td>
<td>.false.</td>
<td>One observation test flag true=on</td>
</tr>
<tr>
<td>sfcmodel</td>
<td>.false.</td>
<td>If true, then use boundary layer forward model for surface temperature data.</td>
</tr>
<tr>
<td>dtbduv_on</td>
<td>.true.</td>
<td>Logical for switching on (.true.) sensitivity of uv winds to microwave brightness temperatures</td>
</tr>
<tr>
<td>ifact10</td>
<td>0</td>
<td>Flag for recomputing 10m wind factor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 1 compute using GFS surface physics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 2 compute using MM5 surface physics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 0 or any other value - DO NOT recompute - use value from guess file</td>
</tr>
<tr>
<td>l_foto</td>
<td>.false.</td>
<td>Option for First-Order Time extrapolation to observation</td>
</tr>
<tr>
<td>offtime_data</td>
<td>.false.</td>
<td>If true, then allow use of obs files with ref time different from analysis time. Default value = .false., in which case analysis fails if observation file reference time is different from analysis time.</td>
</tr>
<tr>
<td>npred_conv_max</td>
<td>0</td>
<td>Maximum number of conventional observation bias correction coefficients</td>
</tr>
<tr>
<td>id_bias_ps</td>
<td>0</td>
<td>Prepbufr id to have conv_bias added for testing</td>
</tr>
<tr>
<td>id_bias_t</td>
<td>0</td>
<td>Prepbufr id to have conv_bias added for testing</td>
</tr>
<tr>
<td>id_bias_spd</td>
<td>120</td>
<td>Prepbufr id to have conv_bias added for testing</td>
</tr>
<tr>
<td>conv_bias_ps</td>
<td>0</td>
<td>Magnitude of ps bias(mb)</td>
</tr>
<tr>
<td>conv_bias_t</td>
<td>0</td>
<td>Magnitude of t bias(deg K)</td>
</tr>
<tr>
<td>conv_bias_spd</td>
<td>0</td>
<td>Magnitude of spd bias(m/sec)</td>
</tr>
<tr>
<td>stndev_conv_ps</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>stndev_conv_t</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>stndev_conv_spd</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>use_pbl</td>
<td>.false.</td>
<td>Logical flag to include PBL effects in tendency model.</td>
</tr>
<tr>
<td>use_compress</td>
<td>.false.</td>
<td>Option to turn on the use of compressibility factors in geopotential heights</td>
</tr>
<tr>
<td>nsig_ext</td>
<td>13</td>
<td>Number of layers above the model top which are necessary to compute the bending angle for gpsro</td>
</tr>
</tbody>
</table>
### GSI Namelist

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpstop</td>
<td>30.0</td>
<td>maximum height for gpsro data assimilation. Reject anything above this height. (km)</td>
</tr>
<tr>
<td>perturb_obs</td>
<td>.false</td>
<td>logical flag to perturb observation (true=on)</td>
</tr>
<tr>
<td>perturb_fact</td>
<td>1</td>
<td>magnitude factor for observation perturbation</td>
</tr>
<tr>
<td>oberror_tune</td>
<td>.false</td>
<td>logical to tune (=true) oberror</td>
</tr>
<tr>
<td>preserve_restart_date</td>
<td>.false</td>
<td>if true, then do not update regional restart file date.</td>
</tr>
<tr>
<td>crtm_coeffs_path</td>
<td>./</td>
<td>path of directory w/ CRTM coeffs files</td>
</tr>
<tr>
<td>berror_stats</td>
<td>berror_stats</td>
<td>filename if other than &quot;berror_stats&quot;</td>
</tr>
<tr>
<td>newpc4pred</td>
<td>.false</td>
<td>option for additional preconditioning for pred coeff</td>
</tr>
<tr>
<td>adp_anglebc</td>
<td>.false</td>
<td>option to perform variational angle bias correction</td>
</tr>
<tr>
<td>angord</td>
<td>0</td>
<td>order of polynomial for variational angle bias correction</td>
</tr>
<tr>
<td>passive_bc</td>
<td>.false</td>
<td>option to turn on bias correction for passive (monitored) channels</td>
</tr>
<tr>
<td>use_edges</td>
<td>.true.</td>
<td>option to exclude radiance data on scan edges</td>
</tr>
<tr>
<td>biaspredvar</td>
<td>0.1</td>
<td>set background error variance for radiance bias coeffs</td>
</tr>
<tr>
<td>lobsdiagsave</td>
<td>.false</td>
<td>write out additional observation diagnostics</td>
</tr>
<tr>
<td>l4dvar</td>
<td>.false</td>
<td>turn 4D-Var on/off (default=off=3D-Var)</td>
</tr>
<tr>
<td>lbicg</td>
<td>.false</td>
<td>use B-precond w/ bi-conjugate gradient for minimization</td>
</tr>
<tr>
<td>lsqrtb</td>
<td>.false</td>
<td>Use sqrt(B) preconditioning</td>
</tr>
<tr>
<td>lcongrad</td>
<td>.false</td>
<td>Use conjugate gradient/Lanczos minimizer</td>
</tr>
<tr>
<td>lbfgsmin</td>
<td>.false</td>
<td>Use L-BFGS minimizer</td>
</tr>
<tr>
<td>llint</td>
<td>.false</td>
<td>Use TL inner loop (ie TL intall)</td>
</tr>
<tr>
<td>nhr_obsbin</td>
<td>-1</td>
<td>length of observation bins</td>
</tr>
<tr>
<td>nhr_subwin</td>
<td>-1</td>
<td>length of weak constraint 4d-Var sub-window intervals</td>
</tr>
<tr>
<td>nwrvecs</td>
<td>-1</td>
<td>Number of precond vectors (Lanczos) or pairs of vectors (QN) being saved</td>
</tr>
<tr>
<td>iorthomax</td>
<td>0</td>
<td>max number of vectors used for orthogonalization of various CG options</td>
</tr>
<tr>
<td>ladjtest</td>
<td>.false</td>
<td>Run adjoint test</td>
</tr>
<tr>
<td>Namelist Name</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ladtest_obs</td>
<td>.false.</td>
<td>if true, doing the adjoint check for the observation operators</td>
</tr>
<tr>
<td>lgrtest</td>
<td>.false.</td>
<td>Run gradient test</td>
</tr>
<tr>
<td>lobsskeep</td>
<td>.false.</td>
<td>keep obs from first outer loop for subsequent OL</td>
</tr>
<tr>
<td>lsensrecompute</td>
<td>.false.</td>
<td>does adjoint by recomputing forward solution</td>
</tr>
<tr>
<td>jsiga</td>
<td>-1</td>
<td>calculate approximate analysis errors from lanczos for jiter=jsiga</td>
</tr>
<tr>
<td>ltcost</td>
<td>.false.</td>
<td>calculate true cost when using Lanczos (this is very expensive)</td>
</tr>
<tr>
<td>lobssensfc</td>
<td>.false.</td>
<td>compute forecast sensitivity to observations</td>
</tr>
<tr>
<td>lobssensjb</td>
<td>.false.</td>
<td>compute Jb sensitivity to observations</td>
</tr>
<tr>
<td>lobssensincr</td>
<td>.false.</td>
<td>compute increment sensitivity to observations</td>
</tr>
<tr>
<td>lobssensadj</td>
<td>.false.</td>
<td>use adjoint of approx. Hessian to compute obs sensitivity</td>
</tr>
<tr>
<td>lobssensmin</td>
<td>.false.</td>
<td>use minimisation to compute obs sensitivity</td>
</tr>
<tr>
<td>iobsconv</td>
<td>0</td>
<td>compute convergence test in observation space</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=1 at final point, =2 at every iteration</td>
</tr>
<tr>
<td>idmodel</td>
<td>.false.</td>
<td>uses identity model when running 4D-Var (test purposes)</td>
</tr>
<tr>
<td>iwrtinc</td>
<td>.false.</td>
<td>when .t., writes out increments instead of analysis</td>
</tr>
<tr>
<td>jiterstart</td>
<td>1</td>
<td>first outloop iteration number</td>
</tr>
<tr>
<td>jiterend</td>
<td>1</td>
<td>last outloop iteration number</td>
</tr>
<tr>
<td>lobserver</td>
<td>.false.</td>
<td>when .t., calculate departure vectors only</td>
</tr>
<tr>
<td>lanczosave</td>
<td>.false.</td>
<td>save lanczos vectors for forecast sensitivity computation</td>
</tr>
<tr>
<td>llancdone</td>
<td>.false.</td>
<td>use to tell adjoint that Lanczos vecs have been pre-computed</td>
</tr>
<tr>
<td>lferrscale</td>
<td>.false.</td>
<td>Something related to forecast error</td>
</tr>
<tr>
<td>print_diag_pcg</td>
<td>.false.</td>
<td>logical turn on of printing of GMAO diagnostics in pcgsoi.f90</td>
</tr>
<tr>
<td>tsensible</td>
<td>.false.</td>
<td>option to use sensible temperature as the analysis variable. Works only for twodvar_regional=.true.</td>
</tr>
<tr>
<td>lgschmidt</td>
<td>.false.</td>
<td>option for re-biorthogonalization of the {gradx} and {grady} set from pcgsoi when twodvar_regional=.true.</td>
</tr>
<tr>
<td>Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>lread_obs_save</code></td>
<td>.false.</td>
<td>option to write out collective obs selection info</td>
</tr>
<tr>
<td><code>lread_obs_skip</code></td>
<td>.false.</td>
<td>option to read in collective obs selection info</td>
</tr>
<tr>
<td><code>use_gfs_ozone</code></td>
<td>.false.</td>
<td>option to read in gfs ozone and interpolate to regional model domain</td>
</tr>
<tr>
<td><code>check_gfs_ozone_date</code></td>
<td>.false.</td>
<td>option to date check gfs ozone before interpolating to regional model domain</td>
</tr>
<tr>
<td><code>regional_ozone</code></td>
<td>.false.</td>
<td>option to turn on ozone in regional analysis</td>
</tr>
<tr>
<td><code>lwrite_predterms</code></td>
<td>.false.</td>
<td>option to write out actual predictor terms instead of predicted bias to the radiance diagnostic files</td>
</tr>
<tr>
<td><code>lwrite_peakwt</code></td>
<td>.false.</td>
<td>option to write the approximate pressure of the peak of the weighting function for satellite data to the radiance diagnostic files</td>
</tr>
<tr>
<td><code>use_gfs_nemsio</code></td>
<td>.false.</td>
<td>option to use nemsio to read global model NEMS/GFS first guess</td>
</tr>
<tr>
<td><code>liauon</code></td>
<td>.false.</td>
<td>treat 4dvar CV as tendency perturbation (default=false)</td>
</tr>
<tr>
<td><code>use_prepb_satwnd</code></td>
<td>.false.</td>
<td>allow using satwnd's from prepbufr (historical) file</td>
</tr>
<tr>
<td><code>l4densvar</code></td>
<td>.false.</td>
<td>logical to turn on ensemble 4dvar</td>
</tr>
<tr>
<td><code>ens4d_nstarthr</code></td>
<td>3</td>
<td>start hour for ensemble perturbations (generally should match min_offset)</td>
</tr>
<tr>
<td><code>use_gfs_stratosphere</code></td>
<td></td>
<td>When true, a guess gfs valid at the same time as the nems-nmmb guess is used to replace the upper levels with gfs values. The purpose of this is to allow direct use of gdas derived sat radiance bias correction coefs.</td>
</tr>
<tr>
<td><code>pblend0</code></td>
<td>152</td>
<td>The nems-nmmb vertical coordinate is smoothly merged with gfs above this level. Below this level, is original nems-nmmb.</td>
</tr>
<tr>
<td><code>pblend1</code></td>
<td>79.0</td>
<td>The nems-nmmb vertical coordinate is smoothly merged with gfs below this level. Above this level, is gfs.</td>
</tr>
<tr>
<td><code>step_start</code></td>
<td>1.e-4</td>
<td>initial stepsize in minimization</td>
</tr>
<tr>
<td><code>diag_precon</code></td>
<td>.false.</td>
<td>if true do preconditioning</td>
</tr>
<tr>
<td><code>lrun_subdirs</code></td>
<td>.false.</td>
<td>logical to toggle use of subdirectories at runtime for per specific files</td>
</tr>
<tr>
<td><code>emiss_bc</code></td>
<td>.false.</td>
<td>option to turn on emissivity bias predictor</td>
</tr>
<tr>
<td><code>upd_pred</code></td>
<td>1</td>
<td>bias update indicator for radiance bias correction; 1.0=bias correction coefficients evolve</td>
</tr>
</tbody>
</table>
GSI Namelist

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use_reflectivity</td>
<td>.false.</td>
<td>option of using reflectivity</td>
</tr>
<tr>
<td>lnested_loops</td>
<td>.false.</td>
<td>allow for nested resolution outer/inner loops</td>
</tr>
<tr>
<td>lwrite4danl</td>
<td>.false.</td>
<td>logical to write out 4d analysis states if 4dvar or 4denvar mode</td>
</tr>
<tr>
<td>lsingleradob</td>
<td>.false.</td>
<td>logical for single radiance observation assimilation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Uses existing bufr file and rejects all radiances that don't fall within a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tight threshold around oblat/oblon (SINGLEOB_TEST)</td>
</tr>
<tr>
<td>ssmis_method</td>
<td>1</td>
<td>choose method for SSMIS noise reduction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0=no smoothing 1=default</td>
</tr>
<tr>
<td>ssmis_precond</td>
<td>0.01</td>
<td>weighting factor for SSMIS preconditioning (if not using newpc4pred)</td>
</tr>
<tr>
<td>R_option</td>
<td>.false.</td>
<td>Option to use variable correlation length for lcbas based on data density</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- follows Hayden and Purser (1995) (twodvar_regional only)</td>
</tr>
</tbody>
</table>

 NOTE: for now, if in regional mode, then iguess=-1 is forced internally.

& GRIDOPTS

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jcap</td>
<td>62</td>
<td>spectral resolution of the analysis</td>
</tr>
<tr>
<td>jcap_b</td>
<td>62</td>
<td>spectral resolution of background (model guess field)</td>
</tr>
<tr>
<td>nsig</td>
<td>42</td>
<td>number of sigma levels</td>
</tr>
<tr>
<td>nlat</td>
<td>96</td>
<td>number of latitudes</td>
</tr>
<tr>
<td>nlon</td>
<td>384</td>
<td>number of longitudes</td>
</tr>
<tr>
<td>hybrid</td>
<td></td>
<td>logical hybrid data file flag true=hybrid</td>
</tr>
<tr>
<td>nlat_regional</td>
<td>0</td>
<td>Number of y grid point in whole regional domain</td>
</tr>
<tr>
<td>nlon_regional</td>
<td>0</td>
<td>Number of x grid point in whole regional domain</td>
</tr>
<tr>
<td>diagnostic_reg</td>
<td>.false.</td>
<td>logical for regional debugging</td>
</tr>
<tr>
<td>update_regsfc</td>
<td>.false.</td>
<td>logical to write out updated surface fields to the regional analysis file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(default = false)</td>
</tr>
<tr>
<td>netcdf</td>
<td>.false.</td>
<td>if true, then wrf files are in netcdf format, otherwise wrf files are in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>binary format.</td>
</tr>
<tr>
<td>regional</td>
<td>.false.</td>
<td>logical for regional GSI run</td>
</tr>
<tr>
<td>wrf_nmm_regional</td>
<td>.false.</td>
<td>logical for input from WRF NMM</td>
</tr>
<tr>
<td>nems_nmmmb_regional</td>
<td>.false.</td>
<td>logical for input from NEMS NMMMB</td>
</tr>
<tr>
<td>wrf_mass_regional</td>
<td>.false.</td>
<td>logical for input from WRF MASS-CORE (ARW)</td>
</tr>
</tbody>
</table>
### GSI Namelist

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>twodvar_regional</td>
<td>.false.</td>
<td>logical for regional 2d-var analysis</td>
</tr>
<tr>
<td>filled_grid</td>
<td>.false.</td>
<td>logical to fill in points on WRF-NMM E-grid</td>
</tr>
<tr>
<td>half_grid</td>
<td>.false.</td>
<td>logical to use every other row of WRF-NMM E-Grid</td>
</tr>
<tr>
<td>nvege_type</td>
<td>24</td>
<td>number of types of vegetation; old=24, IGBP=20</td>
</tr>
<tr>
<td>nlayers(100)</td>
<td>1</td>
<td>number of sub-layers to break indicated model layer into prior to calling radiative transfer model</td>
</tr>
<tr>
<td>cmaq_regional</td>
<td>.false.</td>
<td>Background input is from CMAQ model</td>
</tr>
<tr>
<td>nmmb_reference_grid</td>
<td>H</td>
<td>=‘H’, then analysis grid covers H grid domain</td>
</tr>
<tr>
<td></td>
<td>V</td>
<td>= ‘V’, then analysis grid covers V grid domain</td>
</tr>
<tr>
<td>grid_ratio_nmmb</td>
<td>sqrt(2)</td>
<td>ratio of analysis grid to nmmb model grid in nmmb model grid units.</td>
</tr>
<tr>
<td>grid_ratio_wrfmass</td>
<td>1.0</td>
<td>ratio of analysis grid to wrf mass grid in wrf grid units</td>
</tr>
<tr>
<td>jcap_gfs</td>
<td></td>
<td>spectral truncation used to transform high wavenumber spectral coefficients to a coarser resolution grid, when use_gfs_ozone = .true. or use_gfs_stratosphere = .true.</td>
</tr>
<tr>
<td>use_sp_eqspac</td>
<td>.false.</td>
<td>if .true., then ensemble grid is equal spaced, staggered 1/2 grid unit off poles. if .false., then gaussian grid assumed for ensemble (global only)</td>
</tr>
</tbody>
</table>

#### &BKGERR

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vs</td>
<td>1/1.5</td>
<td>scale factor for vertical correlation lengths for background error</td>
</tr>
<tr>
<td>nhscrf</td>
<td>3</td>
<td>number of horizontal scales for recursive filter</td>
</tr>
<tr>
<td>hzscl(3)</td>
<td>1, 1, 1</td>
<td>scale factor for horizontal smoothing, n=1, number of scales (3 for now)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specifies factor by which to reduce horizontal scales (i.e. 2 would then apply 1/2 of the horizontal scale)</td>
</tr>
<tr>
<td>hswgt(3)</td>
<td>1/3, 1/3, 1/3</td>
<td>empirical weights to apply to each horizontal scale</td>
</tr>
<tr>
<td>norh</td>
<td>2</td>
<td>order of interpolation in smoothing</td>
</tr>
<tr>
<td>ndeg</td>
<td>4</td>
<td>degree of smoothing in recursive filters</td>
</tr>
<tr>
<td>noq</td>
<td>3</td>
<td>1/4 of accuracy in compact finite differencing</td>
</tr>
<tr>
<td>bw</td>
<td>0</td>
<td>factor in background error calculation</td>
</tr>
<tr>
<td>norsp</td>
<td>0</td>
<td>order of interpolation for smooth polar cascade routine default is norsp=0, in which case norh is used with original polar cascade interpolation (global only).</td>
</tr>
<tr>
<td>fstat</td>
<td>.false.</td>
<td>logical to separate f from balance projection</td>
</tr>
</tbody>
</table>
### GSI Namelist

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pert_berr</td>
<td>.false.</td>
<td>logical to turn on random inflation/deflation of background error tuning parameters</td>
</tr>
<tr>
<td>pert_berr_fct</td>
<td>0</td>
<td>factor for increasing/decreasing berror parameters, this is multiplied by random number</td>
</tr>
<tr>
<td>bkgv_flowdep</td>
<td>.false.</td>
<td>flag to turn on flow dependence to background error variances</td>
</tr>
<tr>
<td>bkgv_rewgtfct</td>
<td>0</td>
<td>factor used to perform flow dependent reweighting of error variances</td>
</tr>
<tr>
<td>bkgv_write</td>
<td>.false.</td>
<td>flag to turn on=.true./off=.false. generation of binary file with reweighted variances</td>
</tr>
<tr>
<td>fpsproj</td>
<td>.true.</td>
<td>controls full nsig projection to surface pressure</td>
</tr>
<tr>
<td>fut2ps</td>
<td></td>
<td>controls the projection from unbalance T to surface pressure</td>
</tr>
<tr>
<td>adjustozvar</td>
<td></td>
<td>adjusts ozone variances in the stratosphere based on guess field</td>
</tr>
<tr>
<td>cwcoveqqcov</td>
<td></td>
<td>sets cw Bcov to be the same as B-cov(q) (presently glb default)</td>
</tr>
</tbody>
</table>

### &ANBKGERR

#### Anisotropic background error related variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anisotropic</td>
<td>.false.</td>
<td>if true, then use anisotropic background error covariance</td>
</tr>
<tr>
<td>ancovmdl</td>
<td>0</td>
<td>covariance model settings - 0: pt-based, 1: ensemble based</td>
</tr>
<tr>
<td>triad4</td>
<td>.true.</td>
<td>for 2d variables, if true, use blended triad algorithm</td>
</tr>
<tr>
<td>ifilt_ord</td>
<td>4</td>
<td>filter order for anisotropic filters</td>
</tr>
<tr>
<td>npass</td>
<td>1</td>
<td>$2 \times \text{npass} = \text{number of factors in background error}$</td>
</tr>
<tr>
<td>normal</td>
<td>200</td>
<td>number of random vectors to use for filter normalization (if &lt; 0 then slightly slower, but results independent of number of processors)</td>
</tr>
<tr>
<td>binom</td>
<td>.true.</td>
<td>if true, weight correlation lengths of factors using binomial distribution, with shortest scales on outside, longest scales on inside. This can help to produce smoother correlations in the presence of strong anisotropy</td>
</tr>
<tr>
<td>ngauss</td>
<td>3</td>
<td>number of Gaussians to add together in each factor</td>
</tr>
<tr>
<td>rgauss</td>
<td>0</td>
<td>multipliers on reference aspect tensor for each Gaussian factor</td>
</tr>
<tr>
<td>anhswgt</td>
<td>1.0</td>
<td>empirical weights to apply to each gaussian</td>
</tr>
</tbody>
</table>
## GSI Namelist

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>an_vs</td>
<td>1</td>
<td>scale factor for background error vertical scales (temporary carry over from isotropic inhomogeneous option)</td>
</tr>
<tr>
<td>grid_ratio</td>
<td>2.0</td>
<td>ratio of coarse to fine grid in fine grid units</td>
</tr>
<tr>
<td>grid_ratio_p</td>
<td>0</td>
<td>ratio of coarse to fine grid in fine grid units for polar patches</td>
</tr>
<tr>
<td>nord_f2a</td>
<td>4</td>
<td>order of interpolation for transfer operators between filter grid and analysis grid</td>
</tr>
<tr>
<td>an_flen_u</td>
<td>1</td>
<td>coupling parameter for connecting horizontal wind to background error</td>
</tr>
<tr>
<td>an_flen_t</td>
<td>1</td>
<td>coupling parameter for connecting grad(potential temperature) to background error</td>
</tr>
<tr>
<td>an_flen_z</td>
<td>1</td>
<td>coupling parameter for connecting grad(terrain) to background error</td>
</tr>
<tr>
<td>rtma_subdomain_option</td>
<td>.false.</td>
<td>if true, then call alternative code which calls recursive filter directly from subdomain mode, bypassing transition to/from horizontal slabs. This is mainly to improve efficiency for 2d rtma analysis. at the moment, this only works for twodvar_regional=.true. rtma_subdomain_option will be forced to false when twodvar_regional=.false.</td>
</tr>
<tr>
<td>lreadnorm</td>
<td>.false.</td>
<td>if true, then read normalization from fixed files</td>
</tr>
<tr>
<td>nsmooth</td>
<td>0</td>
<td>number of 1-2-1 smoothing passes before and after background error application</td>
</tr>
<tr>
<td>nsmooth_shapiro</td>
<td>0</td>
<td>number of 2nd moment preserving (shapiro) smoothing passes before and after background error application. NOTE: default for nsmooth and nsmooth_shapiro is 0. if both are &gt; 0, then nsmooth will be forced to zero.</td>
</tr>
<tr>
<td>afact0</td>
<td>0.0</td>
<td>anistropy effect parameter, the range must be in 0.0-1.0.</td>
</tr>
<tr>
<td>covmap</td>
<td>.false.</td>
<td>if true, covariance map would be drawn</td>
</tr>
</tbody>
</table>

### &JCOPTS

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ljcdfi</td>
<td>.false.</td>
<td>if .false., uses original formulation based on wind, temp, and ps tends when .t. uses digital filter initialization of increments (4dvar)</td>
</tr>
<tr>
<td>alphajc</td>
<td>10.0</td>
<td>parameter for digital filter</td>
</tr>
</tbody>
</table>
switch_on_derivatives .false., ... if true, then compute horizontal derivatives of all state variables (to be used eventually for time derivatives, dynamic constraints and observation forward models that need horizontal derivatives)

tendsflag .false. if true, compute time tendencies

ljcpdry .false. when .t. uses dry pressure constraint on increment

bamp_jcpdry 0.0 parameter for pdry_jc

eps_eer -1.0 Errico-Ehrendofer parameter for q-term in energy norm

ljc4tlevs .false. when true and in 4D mode, apply any weak constraints over all time levels instead of just at a single time

&STRONGOPTS

Strong dynamic constraint

reg_tlnmc_type 1 =1 for 1st version of regional strong constraint
          2 =2 for 2nd version of regional strong constraint

tlnmc_option 0 integer flag for strong constraint (various capabilities for hybrid):

          0: no TLNMC
          1: TLNMC for 3DVAR mode
          2: TLNMC on total increment for single time level only (for 3D EnVar) or if 4D EnVar mode, TLNMC applied to increment in center of window
          3: TLNMC on total increment over all time levels (if in 4D EnVar mode)
          4: TLNMC on static contribution to increment ONLY for any EnVar mode

nstrong 0 if > 0, then number of iterations of implicit normal mode initialization to apply for each inner loop iteration

period_max 1000000.0 cutoff period for gravity waves included in implicit normal mode initialization (units = hours)

period_width 1.0 defines width of transition zone from included to excluded gravity waves

nvModes_keep 0 number of vertical modes to use in implicit normal mode initialization

baldiag_full .false. flag to toggle balance diagnostics for the full fields

baldiag_inc .false. flag to toggle balance diagnostics for the analysis increment

&OBSQC

Observation quality control variables

Parameters used for gross error checks are set in file convinfo (ermin, ermax, ratio)
## GSI Namelist

### Parameters below used for nonlinear (variational) quality control

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dfact</td>
<td>0</td>
<td>factor for duplicate observation at same location for conventional data</td>
</tr>
<tr>
<td>dfact1</td>
<td>3.0</td>
<td>time factor for duplicate observation at same location for conventional data</td>
</tr>
<tr>
<td>erradar_inflate</td>
<td>1</td>
<td>radar error inflation factor</td>
</tr>
<tr>
<td>tdrerr_inflate</td>
<td>.false.</td>
<td>logical for tdr obs error inflation</td>
</tr>
<tr>
<td>tdrgross_fact</td>
<td>1</td>
<td>factor applied to tdr gross error</td>
</tr>
<tr>
<td>oberrflg</td>
<td>.false.</td>
<td>logical for reading in new observation error table (if set to true)</td>
</tr>
<tr>
<td>vadfile</td>
<td>'none'</td>
<td>character(10) variable holding name of VAD wind bufr file</td>
</tr>
<tr>
<td>noiqc</td>
<td>.false.</td>
<td>logical flag to bypass OI QC (if set to true)</td>
</tr>
<tr>
<td>c_varqc</td>
<td>1</td>
<td>constant number to control variance qc turning on speed</td>
</tr>
<tr>
<td>blacklist</td>
<td>.false.</td>
<td>logical for reading in raob blacklist (if set to true)</td>
</tr>
<tr>
<td>use_poq7</td>
<td>.false.</td>
<td>Logical to toggle accept (.true.) or reject (.false.) SBUV/2 ozone observations flagged with profile ozone quality mark</td>
</tr>
<tr>
<td>hilbert_curve</td>
<td>.false.</td>
<td>option for hilbert-curve based cross-validation. works only with twodvar_regional=.true.</td>
</tr>
<tr>
<td>tcp_refps</td>
<td>1000.0</td>
<td>reference pressure for tcps oberr calculation (mb)</td>
</tr>
<tr>
<td>tcp_width</td>
<td>50.0</td>
<td>parameter for tcps oberr inflation (width, mb)</td>
</tr>
<tr>
<td>tcp_ermin</td>
<td>0.75</td>
<td>parameter for tcps oberr inflation (minimum oberr, mb)</td>
</tr>
<tr>
<td>tcp_ermx</td>
<td>5.0</td>
<td>parameter for tcps oberr inflation (maximum oberr, mb)</td>
</tr>
<tr>
<td>qc_noirjaco3</td>
<td>.false.</td>
<td>controls whether to use O3 Jac from IR instruments</td>
</tr>
<tr>
<td>qc_noirjaco3_pole</td>
<td>.false.</td>
<td>controls wheter to use O3 Jac from IR instruments near poles</td>
</tr>
<tr>
<td>qc_satwnds</td>
<td>.true.</td>
<td>allow bypass sat-winds qc normally removing lots of mid-tropo obs</td>
</tr>
<tr>
<td>aircraft_t_bc_pof</td>
<td>.false.</td>
<td>logical for aircraft temperature bias correction, pof is used for predictor</td>
</tr>
<tr>
<td>aircraft_t_bc</td>
<td>.false.</td>
<td>logical for aircraft temperature bias correction</td>
</tr>
<tr>
<td>aircraft_t_bc_ext</td>
<td>.false.</td>
<td>logical for reading aircraft temperature bias correction from external file</td>
</tr>
<tr>
<td>buddycheck_t</td>
<td>.false.</td>
<td>When true, run buddy check algorithm on temperature observations</td>
</tr>
<tr>
<td>buddydiag_save</td>
<td>.false.</td>
<td>When true, output files containing buddy check QC info for all obs run through the buddy check</td>
</tr>
</tbody>
</table>
GSI Namelist

biaspredt 1  berror var for temperature bias correction coefficients
upd_aircraft .true.  indicator if update bias at 06Z & 18Z
cleanup_tail .false.  logical to remove tail number no longer used

& OBS_INPUT

Controls input data

dfile  " "  input observation file name
dtype  " "  observation type
dplat  " "  satellite (platform) id (for satellite data)
ds Lis  " "  sensor/instrument/satellite flag from satinfo files
dthin  " "  satellite group
dval  " "  relative value of each profile within group relative weight for observation = dval/sum(dval) within grid box

dmesh(max(dthin))  thinning mesh for each group
mesh size (km) for radiance thinning grid (used in satthin)

dsfcalc  " "  specifies method to determine surface fields within a FOV. when equal to one, integrate model fields over FOV. when not one, bilinearly interpolate model fields to FOV center.
time_window_max 3  upper limit on time window for all input data
ext_sonde .false.  logical for extended forward model on sonde data
l_foreaft_thin .false.  separate TDR fore/aft scan for thinning

NOTE: current value for ndatmax is 200.

&SINGLEOB_TEST

Single observation test case setup

maginnov 1  magnitude of innovation for one observation
magoberr 1  magnitude of observational error
oneob_type  " "  observation type (t, u, v, etc.)
obl at 0  observation latitude
obl on 0  observation longitude
obpres 1000.0  observation pressure (hPa)
obdattim 2000010100  observation date (YYYYMMDDHH)
obhourset 0  observation delta time from analysis time
### GSI Namelist

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pctswitch</td>
<td>.false.</td>
<td>if .true. innovation &amp; oberr are relative (%) of background value (level ozone only)</td>
</tr>
<tr>
<td>obchan</td>
<td>0</td>
<td>if &gt; 0, selects the channel number. If &lt;= zero, it will use all channels that pass qc in setuprad.</td>
</tr>
</tbody>
</table>

#### SUPEROB_RADAR

**Level 2 bufr file to radar wind superobs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>del_azimuth</td>
<td>5.0</td>
<td>azimuth range for superob box (default 5 degrees)</td>
</tr>
<tr>
<td>del_elev</td>
<td>0.25</td>
<td>elevation angle range for superob box (default .05 degrees)</td>
</tr>
<tr>
<td>del_range</td>
<td>5000.0</td>
<td>radial range for superob box (default 5 km)</td>
</tr>
<tr>
<td>del_time</td>
<td>0.5</td>
<td>1/2 time range for superob box (default .5 hours)</td>
</tr>
<tr>
<td>elev_angle_max</td>
<td>5.0</td>
<td>max elevation angle (default of 5 deg)</td>
</tr>
<tr>
<td>minnum</td>
<td>50</td>
<td>minimum number of samples needed to make a superob</td>
</tr>
<tr>
<td>range_max</td>
<td>100000.0</td>
<td>max radial range in meters to use in constructing superobs (default 100km)</td>
</tr>
<tr>
<td>l2superob_only</td>
<td>.false.</td>
<td>if true, then process level 2 data creating superobs, then quit. (added for easier retrospective testing, since level 2 bufr files are very large and hard to work with)</td>
</tr>
</tbody>
</table>

#### LAG_DATA

**Lagrangian data assimilation related variables**

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lag_accur</td>
<td>1.0e-6</td>
<td>Accuracy used to decide whether or not a balloon is on the grid</td>
</tr>
<tr>
<td>infile_lag</td>
<td>inistate_lag.dat</td>
<td>File containing the initial position of the balloon</td>
</tr>
<tr>
<td>lag_stepduration</td>
<td>900.0</td>
<td>Duration of one time step for the propagation model</td>
</tr>
<tr>
<td>lag_nmax_bal</td>
<td>1000</td>
<td>Maximum number of balloons at starting time</td>
</tr>
<tr>
<td>lag_vorcore_stderr_a</td>
<td>2.0e3</td>
<td>Observation error for vorcore balloon</td>
</tr>
<tr>
<td>lag_vorcore_stderr_b</td>
<td>0.0</td>
<td>error = b + a*timestep(in hours)</td>
</tr>
</tbody>
</table>

#### HYBRID_ENSEMBLE

**Parameters for use with hybrid ensemble option**

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_hyb_ens</td>
<td>.false.</td>
<td>if true, then turn on hybrid ensemble option</td>
</tr>
<tr>
<td>uv_hyb_ens</td>
<td>.false.</td>
<td>if true, then ensemble perturbation wind variables are u,v, otherwise, ensemble perturbation wind variables are stream, pot. Functions.</td>
</tr>
</tbody>
</table>
GSI Namelist

q_hyb_ens .false. if true, then use specific humidity ensemble perturbations, otherwise, use relative humidity

aniso_a_en .false. if true, then use anisotropic localization of hybrid ensemble control variable a_en

generate_ens .true. if true, then generate internal ensemble based on existing background error

n_ens 0 number of ensemble members.
nlon_ens 0 number of longitudes on ensemble grid (may be different from analysis grid nlon)
nlat_ens 0 number of latitudes on ensemble grid (may be different from analysis grid nlat)
jcap_ens 0 for global spectral model, spectral truncation

pseudo_hybens .false. if true, turn on pseudo ensemble hybrid for HWRF

merge_two_grid_enspers .false. if true, merge ensemble perturbations from two forecast domains to analysis domain (one way to deal with hybrid DA for HWRF moving nest)

regional_ensemble_option 0 integer, used to select type of ensemble to read in for regional application. Currently takes values from 1 to 4

full_ensemble .false. if true, first ensemble perturbation on first guess instead of on ens mean

betaflg .false. if true, use vertical weighting on beta1_inv and beta2_inv, for regional

coef_bw 0.9 fraction of weight given to the vertical boundaries when betaflg is true

pwgtflg .false. if true, use vertical integration function on ensemble contribution of Psfc

jcap_ens_test 0 for global spectral model, test spectral truncation (to test dual resolution)

beta1_inv 1/1/beta1, the default weight given to static background error covariance if (.not. readin_beta)

0 <= beta1_inv <= 1, tuned for optimal performance

=1, then ensemble information turned off

=0, then static background turned off

the weights are applied per vertical level such that:

betas_inv(:,)= beta1_inv , vertically varying weights given to static B ;
GSI Namelist

betae_inv(:) = 1 - beta1_inv, vertically varying weights given ensemble derived covariance.

If (readin_beta) then betas_inv and betae_inv are read from a file and beta1_inv is not used.

s_ens_h 2828 homogeneous isotropic horizontal ensemble localization scale (km)

s_ens_v 30 vertical localization scale (grid units for now)
s_ens_h, s_ens_v, and beta1_inv are tunable parameters.

use_gfs_ens .true. controls use of global ensemble: .t. use GFS (default); .f. uses user-defined ens

readin_localization .false. flag to read (.true.)external localization information file

readin_beta .false. flag to read (.true.) the vertically varying beta parameters betas_inv and betae_inv from a file.

eqspace_ensgrid .false. if .true., then ensemble grid is equal spaced, staggered 1/2 grid unit off ploes.

if .false., then gaussian grid assumed for ensemble (global only)

use_localization_grid .false. if true, then use extra lower res gaussian grid for horizontal localization (global runs only--allows possiblity for non-gaussian ensemble grid)

grid_ratio_ens 1 for regional runs, ratio of ensemble grid resolution to analysis grid resolution

default value = 1 (dual resolution off)

oz_univ_static .false. if true, decouple ozone from other variables and defaults to static B (ozone only)

write_ens_spread .false. writing global ensemble spread in byte addressable format for plotting with grads

enspreproc .false. flag to read (.true.) pre-processed ensemble data already

i_en_perts_io 0 flag to read in ensemble perturbations in ensemble grid.

This is to speed up RAP/HRRR hybrid runs because the same ensemble perturbations are used in 6 cycles

=0: No ensemble perturbations IO (default)

=2: skip get_gefs_for_regional and read in ensemble perturbations from saved files.

l_ens_in_diff_time .false. if use ensembles that are available at different time from analysis time.
=false: only ensembles available at analysis time can be used for hybrid. (default)

=true: ensembles available time can be different from analysis time in hybrid analysis

<table>
<thead>
<tr>
<th>rapidrefresh_cldsurf</th>
<th>Options for cloud analysis and surface enhancement for RR application</th>
</tr>
</thead>
<tbody>
<tr>
<td>dfi_radar_latent_heat_time_period</td>
<td>30.0</td>
</tr>
<tr>
<td>metar_impact_radius</td>
<td>10.0</td>
</tr>
<tr>
<td>metar_impact_radius_lowCloud</td>
<td>4.0</td>
</tr>
<tr>
<td>l_gsdTerrain_match_surfTobs</td>
<td>.false.</td>
</tr>
<tr>
<td>l_sfcobserror_ramp_t</td>
<td>.false.</td>
</tr>
<tr>
<td>l_sfcobserror_ramp_q</td>
<td>.false.</td>
</tr>
<tr>
<td>l_PBL_pseudo_SurfobsT</td>
<td>.false.</td>
</tr>
<tr>
<td>l_PBL_pseudo_SurfobsQ</td>
<td>.false.</td>
</tr>
<tr>
<td>l_PBL_pseudo_SurfobsUV</td>
<td>.false.</td>
</tr>
<tr>
<td>pblH_ratio</td>
<td>0.75</td>
</tr>
<tr>
<td>pps_press_incr</td>
<td>30hPa</td>
</tr>
<tr>
<td>l_gsd_limit_ocean_q</td>
<td>.false.</td>
</tr>
<tr>
<td>l_pw_hgt_adjust</td>
<td>.false.</td>
</tr>
<tr>
<td>l_limit_pw_innov</td>
<td>.false.</td>
</tr>
<tr>
<td>max_innov_pct</td>
<td>0.1</td>
</tr>
<tr>
<td>l_cleanSnow_WarmTs</td>
<td>.false.</td>
</tr>
<tr>
<td>Name</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>l_conserve_thetaV</td>
<td>.false.</td>
</tr>
<tr>
<td>r_cleanSnow_WarmTs_threshold</td>
<td>8.0</td>
</tr>
<tr>
<td>i_conserve_thetaV_iternum</td>
<td>3</td>
</tr>
<tr>
<td>l_gsd_soilTQ_nudge</td>
<td>.false.</td>
</tr>
<tr>
<td>l_cld_bld</td>
<td>.false.</td>
</tr>
<tr>
<td>cld_bld_hgt</td>
<td>1200m</td>
</tr>
<tr>
<td>build_cloud_frac_p</td>
<td>0.95</td>
</tr>
<tr>
<td>clear_cloud_frac_p</td>
<td>0.1</td>
</tr>
<tr>
<td>nesdis_npts_rad</td>
<td>1</td>
</tr>
<tr>
<td>iclean_hydro_withRef</td>
<td>1</td>
</tr>
<tr>
<td>iclean_hydro_withRef_allcol</td>
<td>0</td>
</tr>
<tr>
<td>l_use_2mq4b</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>i_use_2mt4b</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>i_gsdcldanal_type</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>i_gsdscfc_uselist</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>i_lightpcp</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>i_sfct_gross</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### GSI Namelist

into analysis.

**CHEM**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>berror_chem</td>
<td>.false.</td>
<td>if berror file is supplied for chemistry</td>
</tr>
<tr>
<td>oneobtest_chem</td>
<td>.false.</td>
<td>single observation test for chemistry</td>
</tr>
<tr>
<td>maginnov_chem</td>
<td>30.0</td>
<td>if oneobtest_chem=T magnitude of innovation for chemistry</td>
</tr>
<tr>
<td>magoberr_chem</td>
<td>2.0</td>
<td>if oneobtest_chem=T magnitude of observation error for chemistry</td>
</tr>
<tr>
<td>oneob_type_chem</td>
<td>pm2_5</td>
<td>if oneobtest_chem=T type of chemical observation</td>
</tr>
<tr>
<td>oblat_chem</td>
<td>45.0</td>
<td>if oneobtest_chem=T latitude of the observation</td>
</tr>
<tr>
<td>oblon_chem</td>
<td>270.0</td>
<td>if oneobtest_chem=T longitude of the observation</td>
</tr>
<tr>
<td>obpres_chem</td>
<td>1000.0</td>
<td>if oneobtest_chem=T pressure of the observation</td>
</tr>
<tr>
<td>diag_incr</td>
<td>.false.</td>
<td>if user wishes to output to a binary file increment</td>
</tr>
<tr>
<td>elev_tolerance</td>
<td>500.0</td>
<td>for surface chemical observation sometimes elevation (elev_obs) of the measurement is available (sometimes not).</td>
</tr>
<tr>
<td>tunable_error</td>
<td>0.5</td>
<td>tuning parameter to specify representativeness error for in-situ observations</td>
</tr>
<tr>
<td>in_fname</td>
<td>cmaq_input.bin</td>
<td>name of background file for cmaq</td>
</tr>
<tr>
<td>out_fname</td>
<td>cmaq_output.bin</td>
<td>name analysis file for cmaq</td>
</tr>
<tr>
<td>incr_fname</td>
<td>chem_increment.bin</td>
<td>if diag_incr=T name of the binary dump for pm2_5</td>
</tr>
<tr>
<td>laeroana_gocart</td>
<td>.false.</td>
<td>when true, do chem analysis with wrfchem and modis</td>
</tr>
</tbody>
</table>

**Chemistry data assimilation**

- **Chemistry data assimilation**
  - if berror file is supplied for chemistry
  - single observation test for chemistry
  - if oneobtest_chem=T magnitude of innovation for chemistry
  - if oneobtest_chem=T magnitude of observation error for chemistry
  - if oneobtest_chem=T type of chemical observation
  - if oneobtest_chem=T latitude of the observation
  - if oneobtest_chem=T longitude of the observation
  - if oneobtest_chem=T pressure of the observation
  - if user wishes to output to a binary file increment
  - tuning parameter to specify representativeness error for in-situ observations
  - name of background file for cmaq
  - name analysis file for cmaq
  - if diag_incr=T name of the binary dump for pm2_5
  - when true, do chem analysis with wrfchem and modis