ADD ANALYSIS INCREMENT UTILITY

1. PURPOSE:

The objective of the AddAnalysisIncrement (AAI) utility is to add a previously calcualted analysis increment to the corresponding tria field (background field) to produce the analysis field. The utility endeavours to do this as painlessly as possible for the user.

2. ON THE SURFACE, WHAT DOES THE AAI UTILITY DO?:

To make life easy for the user, the utility reads the analysis-increment and trial fields directly from FSTD files, and writes the result, the analysis field, to a new FSTD file. The utility matches fields from each of the two input files, one pair at a time. The two fields are added together according to the kind of field that it is; i.e. the UU and VV fields are added vectorially, the HU field is paired with the LQ increment field, the GZ is calculated from the resulting HU field.

3. INTERFACE TO THE AAI UTILITY:

Assuming that a potential user of the AAI utility understands what the utility does, this section specifies all that the user needs to know in order to control the AAI utility.

The two input files are named on the command line. The output file is given a name based on the current date, or the file name ma be specified on the command line. Other parameters controlling the utility are read from a namelist in the file, AaiCtrl.dat, unless a different file is specified on the command line.

The default name for the (output) analysis file is of the form, anlmYYYYMMDDHH_000, where:

anl	is a reminder that this file contains analysis fields
m	has the value 'm', unless the vertical grid type is pressure in which case the value is 'p'
YYYYMMDD	is the date that AddAnalysisIncrement is run, in the implied format
HH	is the hour that AddAnalysisIncrement is run
_000	is a constant suffix

The command line has the form:

AddAnalysisIncrement -idelxa AnalysisIncFilename -ixb BackgroundFileName -inml NameListFileName -oxa AnalysisFileName

where the 'i' or 'o' prefix on each key word indicates whether it is an input or output file.

The namelist contains the following possibilities (default values are given):

&AddAnalInc_control

	The first four keys are used as key words in searching the FSTD files:
s_etiket=' '	, where ' ' means "don't care" for this 12-character field. This value applies only to the field in the delxa file. No particular etiket is sought in the xb file.
n_datev=-1	, where -1 means "don't care" for this integer field. This is the date of validity (as opposed to the date of origin) of the field sought. This value applies to both the delxa and xb fields. It is in a format that can be understood by the FSTD routine, fstinl.
n_ip3=-1	, where -1 means "don't care" for this integer field. This is a user-defined FSTD search key. If the value is "don't care", then the field used is the one carrying the largest IP3 value for a given set of (non-dont-care) values for the other search keys. This key applies only to the delxa field; it is ignored for the xb field.
s_nomvars	There is no default. This is a LIST of nomvar's (e.g. 'ES','UU') that should be treated by the utility. If a nomvar is not found in both input files, it is ignored.
n_nbits=value in xb file	This is the number of significant bits (base 10) that are written to the xa file. This affects both the precision of the data written to the file, and the size of the file.
r_humidityMin=2.5e-6	If the humidity field is below this value (or negative), it is set to zero.
s_vertExtrapType='clamped'	The vertical extrapolation algorithm to be used. The only choice is 'clamped'.
s_vertInterpTypeToHigh='linear'	The vertical interpolation algorithm to be used in the direction toward the grid of the xb field. The choices are 'nearestneighbour', 'linear', and 'cubiclagrange'.
s_vertInterpTypeToLow='linear'	The vertical interpolation algorithm to be used in the direction toward the grid of the delx field. The choices are the same as those for vertInterpTypeToHigh.
s_horizInterpTypeToHigh='cubic'	The horizontal interpolation algorithm to be used in the direction toward the grid of the xb

	field. The choices are those of the ezscint package (ezsetopt routine): 'nearest', 'linear' and 'cubic'.
s_horizInterpTypeToLow='linear'	The horizontal interpolation algorithm to be used in the direction toward the grid of the delxa field. The choices are the same as those for horizinterpTypeToHigh.
s_specialNomvar=' '	This key allows the user to specify one nomvar that is to be interpolated using a special algorithm (see below). If this key is to have any effect, the nomvar given to this key must also appear in the nomvars key.
l_writeHighResDelxa=.false.	Setting this key to .true. causes the delxa fields to be written to a file after they have been interpolated to the grid of the xb fields. The file name used is the same as that used for the delxa file, but with the suffix 'HighRes'.
l_writeOldIp1=.false.	When this key is .true., IP1 values are written to the FSTD files using old-style encoding This possibility is useful for directly comparing results with existing FSTD files.
l_useIncrementedPSurf=.false.	When this key is .false., the delxa fields are deemed to be defined on a grid with the surface pressure as it is read from the xb file. When this key is .true., the delxa fields are deemed to be defined on a grid with the incremented surface pressure, where the increment is read from the delxa file.

/

The special algorithm with which the specialNomvar will be interpolated is supplied by the user. When the specialNomvar is encountered by the AAI utility, two user-supplied fortran subroutines are called: M_InitSpecialInterpolation and M_SpecialInterpolation. The first thing to note is that, in order to satisfy the linker, the two routines, M_InitSpecialInterpolation and M_SpecialInterpolation, must be supplied by the user, even if they are not used (As a service, a version of the executable that is already linked with stubs for these two routines is supplied.); if they are not used they may, of course, do nothing. The M_SpecialInterplation routine must accept as input the delxa (low resolution) field and return the same field in high resolution, ready to be added to the xb field. Prior to that, the M_InitSpecialInterpolation routine must accept as input the low- and high-resolution grids (horizontal and vertical components) as well as the surface and ceiling pressures on the high-resolution grid. This is the exact interface to those two routines (If there is any doubt whether this documentation is up to date, the user should consult the source file, SpecialInterpolationDummy.ftn90, which is used during debugging of the AAI utility, or for an absolute reference the user should consult the interface blocks to those two routines in the AddAnalysisIncrement.ftn90 and FieldLogic.ftn90 files.):

```
module VerticalGrid_class
  type T_VerticalGrid
    ! information capable of specifying particular grid types is placed in
    !  variables here:
    integer N_gridType                  ! representation used for vertical levels
    integer N_numVLevels                ! number of vertical points in this grid

                                        ! vertical levels defined for this grid,
                                        ! in units that are implied by gridType
                                        ! (IP1 values, decoded from FSTD format)
    real, pointer, dimension(:) :: R_vLevel_p

    ! Other parameters that may be necessary, depending on gridType
    ! (surface pressure is not explicitly a part of the vertical grid)

    ! hybrid parameters
    real :: R_pTopAvg                   ! pressure (mb) at the model top(ceiling)
    real :: R_pRef                      ! reference pressure (mb)
    real :: R_rCoef                     ! known as 'expansion co-efficient'
  end type T_VerticalGrid

  type VerticalGridPtr
    type(T_VerticalGrid), pointer :: Ptr
  end type T_VerticalGridPtr
end module VerticalGrid_class

module HorizontalGrid_class
  type T_PositionalRecord               ! the '^^' or '>>' record of a horiz grid
    ! ip1, ip2, ip3 of this record match n_ig1, n_ig2, n_ig3
    !   of the containing HorizontalGrid
    integer :: N_ni, N_nj               ! dimensions of this field
    integer :: N_nbits
    character :: S_gridType             ! type on which this record is based
    integer :: N_ig1,N_ig2,N_ig3,N_ig4  ! encoded parameters of the positions
    real, dimension(:,:), pointer :: R_position_p
  end type T_PositionalRecord

  type T_PositionalRecordPtr
    type(T_PositionalRecord), pointer :: Ptr
  end type T_PositionalRecordPtr
```

```
  type T_HorizontalGrid
    character :: S_gridType            ! as defined in the FSTD
    integer   :: N_ig1, &              ! The ig's are as defined in the FSTD
                 N_ig2, &
                 N_ig3, &
                 N_ig4
    integer   :: N_ni, N_nj            ! grid dimensions
    type(T_PositionalRecord) :: O_horizPosn, & ! for 'Y', 'Z', and '#' grid types
                                O_vertPosn
    integer   :: N_ezscintHandle       ! same data in ezscint format
  end type T_HorizontalGrid

  type T_HorizontalGridPtr
    type(THorizontalGrid), pointer :: Ptr
  end type T_HorizontalGridPtr
end module HorizontalGrid_class

subroutine M_InitSpecialInterpolation(o_hGridDestPtr, o_hGridSrcPtr, &
                                      o_vGridDestPtr, o_vGridSrcPtr, r_pSurf, &
                                      r_pTop)
  use VerticalGrid_class
  use HorizontalGrid_class
  type(T_HorizontalGridPtr), intent(in) :: o_hGridDestPtr, &
                                           o_hGridSrcPtr
  type(T_VerticalGridPtr), intent(in) :: o_vGridDestPtr, &
                                         o_vGridSrcPtr
  real, dimension(:,:), intent(in) :: r_pSurf ! pressure values at the surface
  real, dimension(:,:), intent(in) :: r_pTop  ! pressure values at the top
  ! Receive and store the input data ...
end subroutine M_InitSpecialInterpolation

subroutine M_SpecialInterpolation(r_analIncH, r_analIncL)
  real, dimension(:,:,:), intent(out) :: r_analIncH
  real, dimension(:,:,:), intent(in) :: r_analIncL
  ! Perform the interpolation ...
end subroutine M_SpecialInterpolation
```

4. MINIMAL INTERFACE TO THE AAI UTILITY

Thus, in addition to linking to an exact copy of the two 'special' routines, a minimum interface to the AAI utility would look like this:

In a file called AaiCtrl.dat that is located in the current directory:

```
&AddAnalInc_control
nomvars='TT','HU','GZ','ES','UU','VV'
/
```

On the command line, enter:

```
AddAnalInc_Linux.Abs -idelxa delxafile -ixb xbfile
```

5. LINK INSTRUCTIONS

Unless you want to use the SpecialInterpolation capability of the software, there is no need to link with it; use the executable (/usr/local/env/armnlib/modeles/ANAL/v_addinc0.1.0/bin/Linux/AddAnalInc_Linux.Abs) for your architecture. If you do want to use the SpecialInterpolation capability, remember to write you own M_InitSpecialInterpolation and M_SpecialInterpolation routines. Then, compile and link. This is a suggested makefile, assuming that your two routines are located in the file, SpecialInterpolation.ftn90:

```
.SUFFIXES:

.SUFFIXES : .o .ftn90

SHELL = /bin/sh

COMPILE = compile

FFLAGS =

CFLAGS =

OPTIMIZ = -O 2

default: absolu
```

```
.ftn90.o:
        r.compile -arch $(ARCH) -abi $(ABI) $(OPTIMIZ) -opt "=$(FFLAGS)" \
        -includes $(ARMNLIB)/modeles/ANAL/v_addinc0.1.0/lib/$(ARCH) \
        $(ARMNLIB)/modeles/ANAL/v_vrtint0.1.0/lib/$(ARCH) -src $<

OBJECTS= SpecialInterpolation.o

absolu: $(OBJECTS)
        r.build -bidon -main main_bidon -o AddAnalInc_$(ARCH).Abs -obj $(OBJECTS) -arch $(ARCH) \
        -abi$(ABI) -libpath $(ARMNLIB)/modeles/ANAL/v_addinc0.1.0/lib/$(ARCH) \
        $(ARMNLIB)/modeles/ANAL/v_vrtint0.1.0/lib/$(ARCH) \
        -libappl AddAnalInc VertInterp -librmn rmnbeta
```

This make file contains the following features:

- the compile line contains an 'includes' directive, with two paths, so that the compiler knows where to look for the modules that you have 'use'd in your two routines.
- the link line uses the 'bidon' directive so as to automatically generate a main program that calls main_bidon, the function where entry into the AddAnalysisIncrement package begins.
- the link line uses a 'libpath' directive, with two paths, so that the linker knows where to find the libraries
- the link line uses a 'libappl' directive so that the linker knows to link with both the AddAnalInc and VertInterp libraries
- the link line specifies the rmnbeta revision of the rmn library (at the time of writing: December, 2002)

6. REQUIREMENTS PLACED ON THE USER

If someone uses the AAI utility, he agrees to abide by these rules:

1. For a given field (i.e. set of FSTD keys), it is assumed that the grid on all records is the same. Failure to meet this criterion will produce unexpected results.
2. Within a single run of the utility, it is assumed that all low-resolution grids have the same dimensions, and that all high-resolution grids have the same dimensions. If this criterion is not met, the utility will refuse to continue if it doesn't crash first.
3. The horizontal grid is independent of the date. If a file contains more that one horizontal grid, they had better be distinguished by the etiket or IP3 keys. If they are distinguished only by the date, there will be unexpected results.
4. Ditto (see the preceding point) for the hybrid record of the vertical grid.
5. It is assumed that each FSTD record contains exactly one, and not multiple, vertical levels.

7. WHAT DOES THE AAI UTILITY REALLY DO TO THE DATA?:

Having such a utility to use is nice, but one must have confidence that it does what one expects. Therefore, this section outlines exactly what the AAI utility does to the data.

7.1. Which fields does it choose?

Based on the etiket, datev, IP3, and nomvar that are specified in the namelist, the utility attempts to process every field that matches these keys.

It can be noted that four keys are listed here, whereas the FSTD's support 7 keys. The missing keys are typvar, IP1, and IP2. Typvar is fixed for each type of file and is not under user control. The various values of IP1 constitute the various vertical levels in the field; therefore, all IP1's are used. IP2 is a part of the specification of datev; the FSTD key is actually date.

If the user sets IP3 to "don't care", the only field processed is the one with the largest IP3 value for the same values of the other three keys.

Of the fields that meet the criteria so far, a field can be processed only if it has a matching xb field. A matching xb field is sought based on the nomvar and datev of the delxa field. If one is found, these two fields (xb and delxa) are processed together.

7.2. Which FSTD keys are significant?

Consider the four interesting (see the previous section) FSTD search keys one at a time:

nomvar - only one nomvar is considered at a time, taken from the possibilities in the namelist

etiket - could have multiple values
datev - could have multiple values
ip3 - with all other keys the same, only the largest IP3 is used

Thus, it is possible to treat several fields, all with the same nomvar. These multiple fields would be distinguished by the values of the etiket and datev (dateo) keys. [There are some cases (those with dependent nomvar's -- see the section about special treatments) where multiple fields with the same nomvar cannot be processed on the same run. The AAI utility will detect this condition, tell you that it "can't handle this", and abort.]

Thus, all records that are used together are required to have matching typvar, etiket, datev (i.e. dateo and ip2, combined), and ip3 keys. The nomvar and IP1 identify the particular record.

When processing a particular nomvar, only the datev, etiket, and ip3 keys could possibly be different. They merit further discussion. The Po field (read only from the xb file) is required only to have a matching datev key; if possible, it is chosen to have matching etiket and ip3 keys as well. The delta-Po field (read only from the delxa file) is required to have matching datev, etiket and ip3 keys. The Pt field is required only to have a matching datev key. A hybrid record (i.e. HY record) is required to match all three keys. Positional records (nomvar = '^^' or '>>') are not required to match the datev or etiket; it is required only that their ip1/ip2/ip3 keys match the ig1/ig2/ig3 parameters of the grid that they describe.

While datev and nomvar must match between all files involved, etiket and ip3 do not need to match; the latter two are specific to each file and must match within the file. The etiket and ip3 read from the delxa file are used to write to the output files.

7.3. Can the user be sure that the vertical and horizontal grids match each field that is read in?

As each field is read in, its horizontal and vertical grids are also read. These grids are compared with the ones already in memory. If any one grid is different, all of the old grids are completely flushed and replaced by the new grids. In particular, the horizontal and vertical interpolation routines are re-initialized with the new grids.

The previous paragraph applies to independent nomvar's. In the case of dependent nomvar's (see the section on special treatments), if a grid is not the same for all fields in a group of nomvar's, the AAI utility aborts with an error message.

Whenever the grids are revised, the new horizontal grid(s) are written to the output file so as to ensure that the information there is self-contained. In doing so, the 'overwrite' flag is set. This means that any field with the same six FSTD keys (the FSTD routines ignore the date when doing this) will be over-written with the new field. This is desireable in order to avoid having duplicate records in the file.

Each level of the vertical grid is explicitly attached to each record. Thus, a new vertical grid is necessarily written to the file, unless is of the hybrid type. In this case, the hybrid record is explicitly written to the output file. The Po and Pt arrays are similarly written.

7.4. How are the interpolations performed?

The horizontal interpolations are performed with the well-known ezscint routines. The vertical interpolations follow the horizontal one, and are performed using the new ezscint-like vertical interpolation routines. The vertical interpolation routines perform the interpolation linearly in ln(P), unless the vertical-grid type is 'generic'.

In the special case of the humidity, where an inverse interpolation must be performed (see the section on special treatments for the fields), the inverse vertical interpolation is performed before the inverse horizontal interpolation.

Depending on a switch in the namelist, the (forward) interpolation can be performed using a user-supplied routine.

7.5. What special treatments do the various fields receive (Dependent Nomvar's)?

The UU and VV field are processed together so that they can be interpolated horizontally as a vector.

The TT, LQ, HU, and GZ fields are interdependent. GZ requires HU. HU requires TT and LQ. The AAI utility arranges that the required fields are processed even if the user doesn't request them. For example, if the user requests the GZ field, all four will be read and processed, but only the GZ field will be written to the output file.

The HU field is not subjected to the same mathematical treatment as the other fields, although in a perfect world the result would be the same. The analysis increment field is given as ln(HU) and bears the name LQ. This is combined with the background HU, but not as one would think. For historical reasons (which are subject to debate), the background HU is reverse-interpolated to the analysis-increment grid, where the log of the result is added to the analysis-increment LQ field. The resulting log of the 'analysis field' is then exponentiated to obtain an 'analysis field' (still based on the analysis-increment grid) whose difference from the reverse-interplated HU field is obtained. This new analysis-increment field (now not a log) is the input to the same treatment as all other fields receive. The resulting humidity field is adjusted by the well-established ajhum routine. In addition, negative (or near-negative) values are clipped from the humitidy field at several points in its processing.

The GZ field is special in that it is not calculated from the background GZ and the analysis-increment GZ as are all other fields (unless the vertical grid is of the pressure type, in which case GZ is calculated in the normal way). Instead it is rederived from the (already incremented) HU and 'virtual temperature', plus the background GZ only at the ground level, using the well-established calgz routine.

7.6. In what order are the nomvar's processed?

The nomvar's are processed in the same order as is supplied in the namelist, except where there are dependencies between multiple nomvar's. Dependent nomvar's are processed after all others, in the order dictated by the dependencies.

7.7. How is datev obtained from the delxa file?

Since it is known a priori that deet and npas (in the FSTD) are zero in a delxa file, datev is equal to dateo. Thus, the dateo field is read as if it were datev, without actually checking that deet and npas are zero.

7.8. Are there any assumptions about the grids on which the pressure arrays (Po, Pt) are defined?

No. They can be supplied on any grid. They are interpolated to the grid of the xb field before using them. See also the section about the significance of the FSTD keys.

It is assumed that the Pt arrays are supposed to represent the same physical pressures in both the xb and delxa files. This equality is tested (assuming that a difference of less than 2e-6 is negligible), and the AAI utility aborts with an error message if there is any difference.

7.9. Is the order of the IP1 values important?

The AAI utility makes no assumption about the order of the IP1 values. On reading them and the associated arrays, they are put in order from the top down for processing.

7.10. What FSTD format is used?

FSTD routines are backwards compatible for reading. They write the new style only.

7.11. What CONVIP format is used?

Convip is backwards compatible for reading; it can accept input from and FSTD file in the old or new format. The AAI utility writes convip in the new format. (There is, however, a namelist switch that causes convip to be written in the old format.)

7.12. How is the vertical-grid type identified?

It is identified based on the opinion of convip() and the presence of HY and PT records. More specifically, if convip believes that the grid is type sigma, the file is checked for the presence of a matching HY record. If found, the grid is hybrid type. If not the file is checked for the presence of a matching PT record. If found, the grid is eta type. If not, in the case of the delxa file for backwards compatibility, the xb file is checked for the presence of a PT record. If found, the grid is eta type. If not, it is sigma type. However, in the case of the delxa file, the sigma type will never be used (because it is an old type); in this case, the program aborts with an error message. It can be noted that a HY record will never be sought in a file other than the present one.

7.13. Memory management

In order to avoid repeated allocation and deallocation, the arrays are allocated once at the beginning. This is the reason for the requirement that the array dimensions not change during a run.

Author (code and documentation): Jeffrey W. Blezius