A new framework for working with observational data in IFS

Peter Lean, Alan Geer, Deborah Salmond

ECMWF

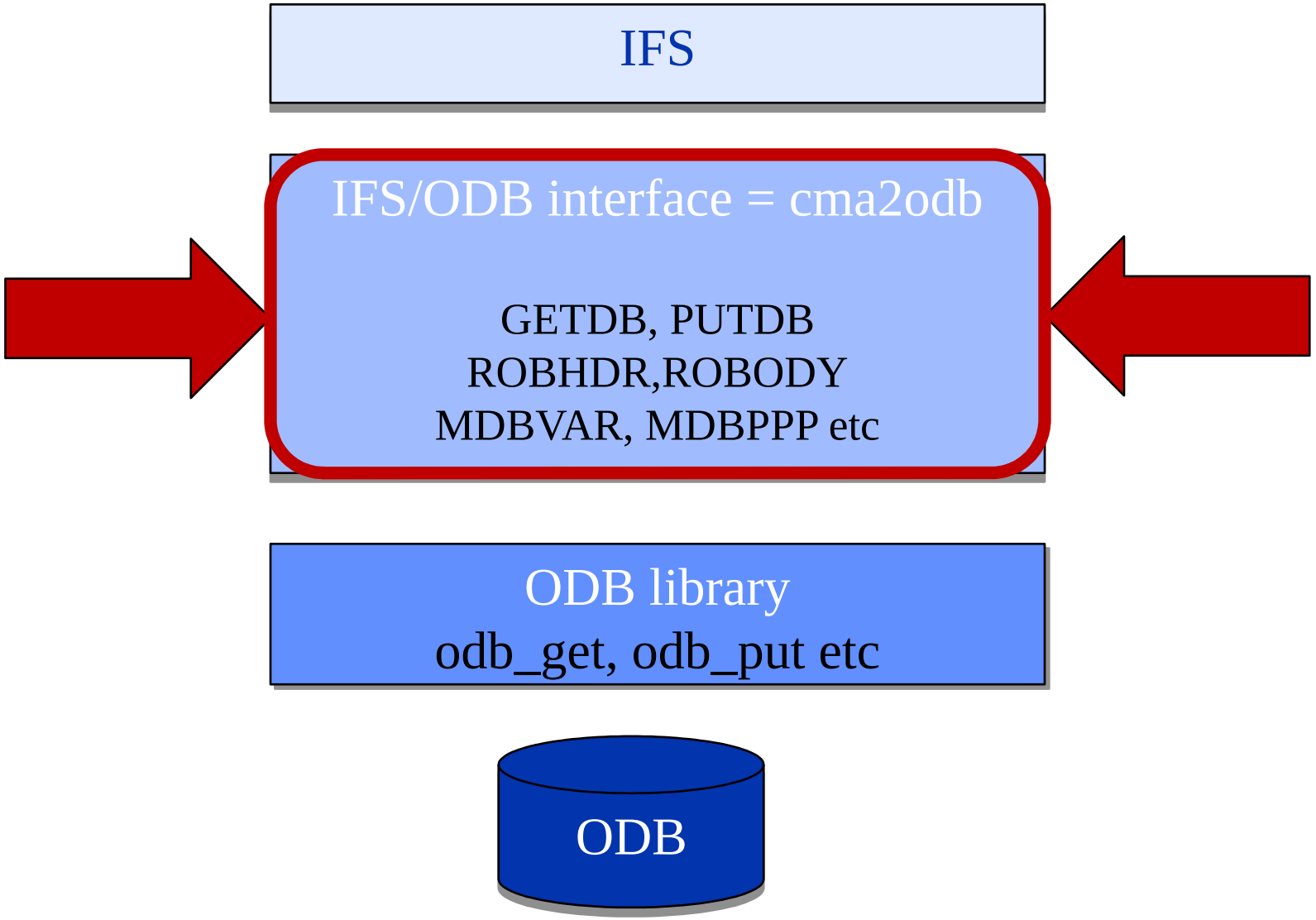
Outline

1. Why the IFS/ODB interface needed upgrading.

2. Description of new object-oriented framework to interact with observation databases.

3. How the observation code will look at 43r1.

Usage of ODB by IFS



Issues with old IFS ODB usage

GETDB hides which SQLs are actually run.
(need to see ctxinitdb.F90 and getdb.F90 to find out)
+ often other ad-hoc processing is done by GETDB…

Gives no indication what arrays are returned.
ROBHDR,ROBODY,ROBSU,SATHDR,SATBODY??

```
CALL GETDB('HRETR', NUPTRA, IRET, INFO, 0, ZINFO,0, &
         & KSET, ITSLOT, -1, IOBSTYPE, ICDTYP_TOVS, ISENSOR_GETDB)


DO JOBS = 1,ILEN
  DO JBODY=1,ICMBDY(JOBS)
   IBODY=MLNKH2B(JOBS)+(JBODY-1)
   IVNM=ROBODY(IBODY,MDBVNM)

   IF ( ( IVNM == NVNUMB( 7) .OR. IVNM == NVNUMB(10) .OR.&
     & IVNM == NVNUMB(61) .OR. IVNM == NVNUMB(62) .OR.&
     & IVNM == NVNUMB(56) .OR. IVNM == NVNUMB(57) )   ) THEN

     IF(ROBODY(IBODY,MDBPPP) <= 1.0_JPRB)
ROBODY(IBODY,MDBPPP)=ZPSOLPPP
   ENDIF

 ENDDO
ENDDO

CALL PUTDB('HRETR', NUPTRA, IRET, INFO, 0,ZINFO,0)
```

Looping over hdr and body level data is complex ;
Easy to make mistakes

ROBHDR, ROBODY etc are globals

What column is MDBPPP?

Need to look in initmdb.F90 to find out that it is :
vertco_reference_1@body

Actually,
#define ROBODY o_(it)%robody

2D arrays provide no metadata about what information they contain.

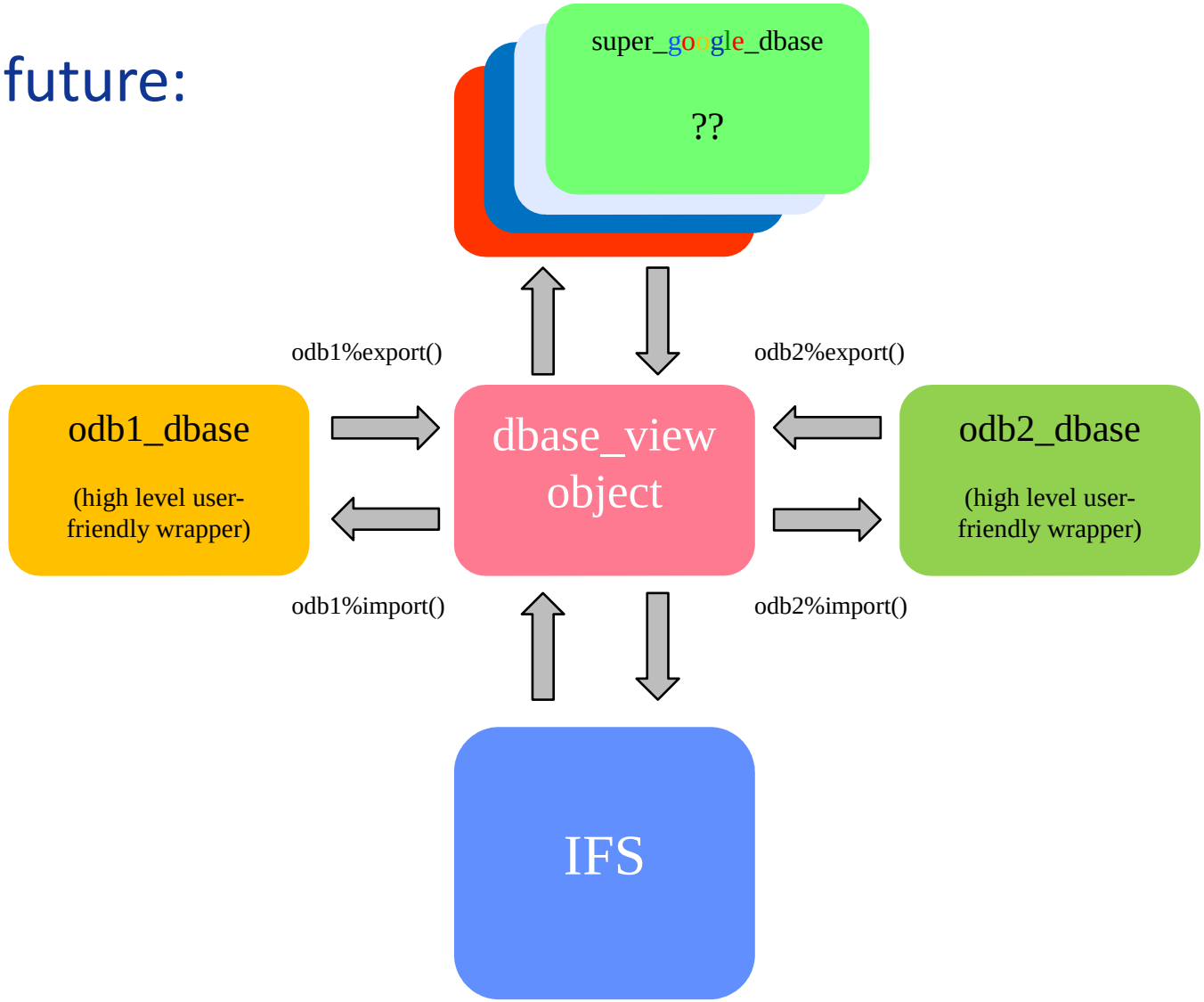
EASY TO GO OUTSIDE ARRAY BOUNDS BY MISTAKE

ECMWF

Problems with current IFS/ODB usage

- Opaque code; hard to read and understand
- Extensive use of global variables
- Easy to misuse and cause memory overwrite – SEGV
- Hard to debug

The future:

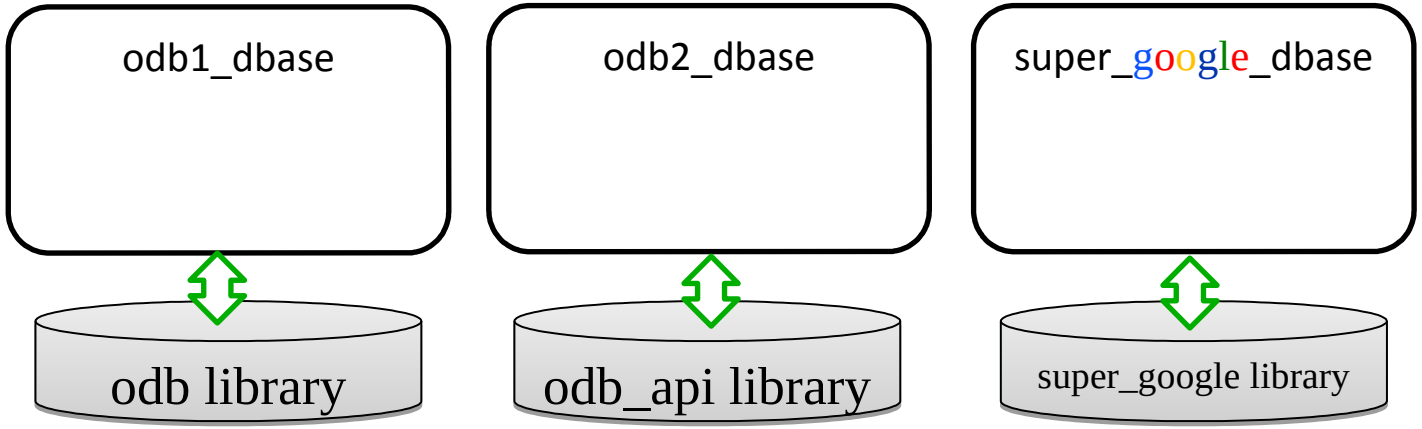

super_google_dbase
??
odb1%export()
odb2%export()
odb1_dbase
(high level user-friendly wrapper)
dbase_view object
odb2_dbase
(high level user-friendly wrapper)
odb1%import()
odb2%import()
IFS

Abstract dbase class

```
type, abstract  :: dbase
      character(len=strlen)          :: filename     = ""
      integer(kind=jpin)             :: handle       = ODB_INT_MDI
      logical                        :: readonly     = .true.
      logical                        :: inuse        = .false.
contains
      procedure(generic_open),       deferred, pass(this)  :: open
      procedure(generic_close),      deferred, pass(this)  :: close
      procedure(generic_import),     deferred, pass(this)  :: import
      procedure(generic_export),     deferred, pass(this)  :: export
      procedure(generic_select),     deferred, pass(this)  :: select
      procedure(generic_put),        deferred, pass(this)  :: put
      procedure(generic_destroy),    deferred, pass(this)  :: destroy
end type dbase
```

What can you do with databases?

~~How do they do it?~~



Example: generic conversion program

Application doesn't know or care about the specific type of database you are using

dbase Factory creates dbase objects of different types (odb1,odb2,ascii etc)

Data extracted from input database into dbase_view object

dbase_view object imported into output database

```
program convert

 use dbase_mod

 implicit none

 type(dbase_factory)              :: db_factory
 class(dbase), pointer            :: in_dbase => null()
 class(dbase), pointer            :: out_dbase => null()
 type(dbase_view)                 :: view

 ! Use a factory to create different flavours of database objects;
 call db_factory%init("odb1")
 in_dbase => db_factory%create_dbase()

 call db_factory%init("odb2")
 out_dbase => db_factory%create_dbase()

 ! Open the input and output databases
 rc = in_dbase%open(infile,"r")
 rc = out_dbase%open(outfile,"new")

 ! Extract all the data from input database into a view object
 rc = in_dbase%export(view)

 ! Import data from the view object into the output database
 rc = out_dbase%import(view)

 ! Close both databases
 rc = in_dbase%close()
 rc = out_dbase%close()

end program convert
```

Example: generic conversion program

Application doesn't know or care about the specific type of database you are using

dbase Factory creates dbase objects of different types (odb1,odb2,ascii etc)

Data extracted from input database into dbase_view object

dbase_view object imported into output database

```
program convert

 use dbase_mod

 implicit none

 type(dbase_factory)              :: db_factory
 class(dbase), pointer            :: in_dbase => null()
 class(dbase), pointer            :: out_dbase => null()
 type(dbase_view)                 :: view

 ! Use a factory to create different flavours of database objects;
 call db_factory%init("ascii")
 in_dbase => db_factory%create_dbase()

 call db_factory%init("odb2")
 out_dbase => db_factory%create_dbase()

 ! Open the input and output databases
 rc = in_dbase%open(infile,"r")
 rc = out_dbase%open(outfile,"new")

 ! Extract all the data from input database into a view object
 rc = in_dbase%export(view)

 ! Import data from the view object into the output database
 rc = out_dbase%import(view)

 ! Close both databases
 rc = in_dbase%close()
 rc = out_dbase%close()

end program convert
```

New classes

- **dbase_view**

- Self describing:

Encapsulate everything needed to describe odb data
-column names, types, bitfield info, data values etc

```
type dbase_view
   character(len=strlen)                :: view_name    = ""
   real(kind=jprl), allocatable         :: data(:,:)
   integer(kind=jpin)                   :: nrows        = 0
   integer(kind=jpin)                   :: ncols        = 0
   character(len=strlen), allocatable   :: colnames(:)
   integer(kind=jpin), allocatable      :: coltypes(:)
   character(len=strlen), allocatable   :: coltables(:)
   type(bitfield), allocatable          :: bitfields(:)
contains
   procedure                  :: get_column_index
   procedure                  :: get_column_ptr
   procedure                  :: copy_column
   procedure                  :: get_value
   procedure                  :: set_value
   procedure                  :: fill_mdi
   procedure                  :: trim_mdi
   procedure                  :: copy_common_columns_from
   procedure                  :: compatibility_mode
   procedure                  :: destroy
end type dbase_view
```

Examples: taking advantage of new access methods

Old:

```
DO JOBS = 1,KDLEN
  ICDTYP(JOBS)=ROBHDR(JOBS,MDB_CODETYPE_AT_HDR)
  ZCMALT=ROBHDR(JOBS,MDBALT)
  IF(ZCMALT == RMDI .OR. ABS(ZCMALT) >  10000._JPRB) THEN
    ZOROGOB(JOBS) = RMDI
  ELSE
    ZOROGOB(JOBS) = ZCMALT*RG
  ENDIF
ENDDO
```

New:

```
icdtyp  = robhdr%copy_column('codetype@hdr')
zorogob = robhdr%copy_column('stalt@hdr')

where(abs(zorogob) > 10000._jprb) zorogob = rmdi
where(zorogob /= rmdi)            zorogob = zorogob * rg
```

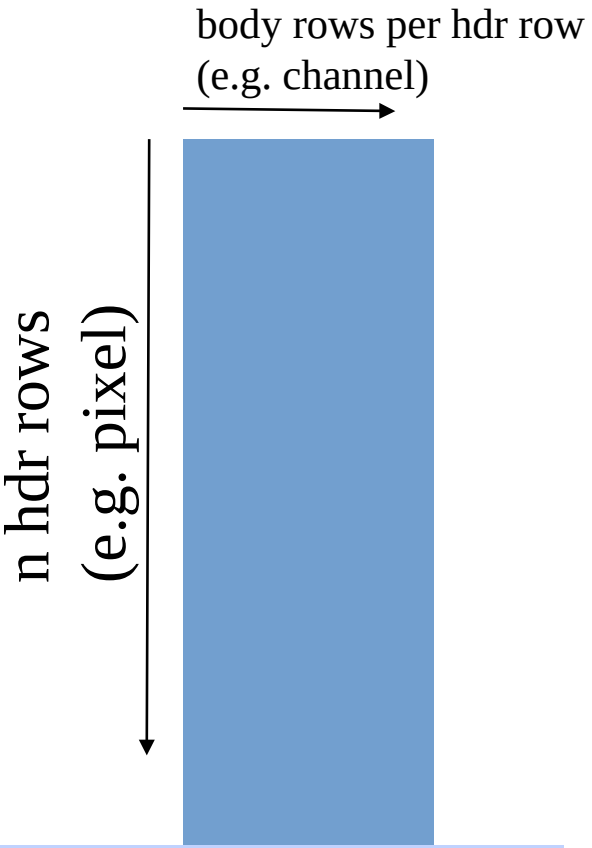
Filling data into 2d arrays

Old:

```
DO JBODY=1,IMXBDY
  DO JOBS = 1,KDLEN
    IF(JBODY > ICMBDY(JOBS)) CYCLE
    IBODY = MLNKH2B(JOBS)+(JBODY-1)

    ZVERTP(JOBS,JBODY)   = ROBODY(IBODY,MDBPPP)
    ZVERTP_L(JOBS,JBODY)= ROBODY(IBODY,MDBPRL)

  ENDDO
ENDDO
```



New:

```
zvertp(:,:)   = robody%copy_column("vertco_reference_1@body",reshape_by="seqno@hdr")
zvertp_l(:,:) = robody%copy_column("vertco_reference_2@body",reshape_by="seqno@hdr")
```

Examples: taking advantage of new access methods

Old:

```
DO JOBS = 1,KDLEN
  ROBODY(JOBS,MDBPPP) = 0.0
ENDDO
```

MDBPPP = index of column containing data for vertco_reference_1@body

What happens if your SQL didn't request vertco_reference_1@body?

-memory overwrite

New:

```
zodb_vertco_reference_1 => robody%get_column_ptr('vertco_reference_1@body')
do jobs = 1,robody%nrows
  zodb_vertco_reference_1(jobs) = 0.0
enddo
```

```
Abort: dbase_view % get_column_index :
    vertco_reference_1@body not found in view: robody.
Please consider adding it to the query that created this view.
 e.g. add this column to the sql file : robody.sql
```

Benefits of new framework

- Decouples IFS code from underlying database engine (and file format).
- Makes it easier to change the database engine if required.
- Makes the code more readable (hopefully!).
- Cleans up code that had grown increasingly complex over the years.
- Removes global variables.
- Light-weight; performance critical code sections very similar to before.

Progress

- **42r2 / 42r3: intermediate cycles**
  - **Introduced dbase_view objects into observation operator code**
  - **Still using access methods similar to before:**
    - **ROBODY(IBODY,MDBONM) = …**
    - **ROBODY%DATA(IBODY,ROBODY%SEQNO_AT_HDR) = …**
- **Future cycles: dbase and dbase_view framework more mature**
    - **zodb_seqno => robody%get_column_ptr('seqno@hdr')**
    - **zodb_seqno(ibody) = …**
- **Next year:**
  - **Continue rolling out across IFS**
- **Eventual aim:**
  - **retire cma2odb (ctxinitdb.F90,initmdb.F90 etc)**

Thanks for listening

Varno module:

Old:

```
  IVNM=ROBODY(IBODY,MDBVNM)
  IF ( ( IVNM == NVNUMB( 7) .OR. IVNM == NVNUMB(10) .OR.&
       & IVNM == NVNUMB(61) .OR. IVNM == NVNUMB(62) .OR.&
       & IVNM == NVNUMB(56) .OR. IVNM == NVNUMB(57) )   ) THEN
```

New:

```
use varno_module, only : varno

ivnm = varno_at_body(ibody)
if ( ivnm == varno%rh2m   .or. ivnm == varno%t2m .or. &
     ivnm == varno%u10m   .or. ivnm == varno%v10m .or. &
     ivnm == varno%scatdd .or. ivnm == varno%scatff )then
```

The NVNUMB upgrade python script was used in 42r1

Which SQLs are called when you call GETDB?

hretr.F90

```
CALL GETDB('HRETR', NUPTRA, IRET, INFO, 0, ZINFO,0, &
          & KSET, ITSLOT, -1, IOBSTYPE, ICDTYP_TOVS, ISENSOR_GETDB)
```

ctxinitdb.F90

```
else if (cdretr == 'HRETR') then
      ctx(idctx,it)%view(1)%name = 'robhdr_screen'
      ctx(idctx,it)%view(1)%ncase = JPCASE_ROBHDR
```

getdb.F90

```
SUBROUTINE preproc_hretr()
implicit none
REAL(KIND=JPRB) :: ZHOOK_HANDLE
IF (LHOOK) CALL DR_HOOK('GETDB:PREPROC_HRETR',0,ZHOOK_HANDLE)
LLexecv(3:) = .FALSE.
if (LLRAD1C) then
  LLexecv(3) = .TRUE.    ! for 'sathdr_screen_atovs'
  LLexecv(9) = .TRUE.    ! for 'satbody_screen_atovs'
  LLexecv(10) = .TRUE.    ! for 'sathdr_screen_cloud_sink'
else if (LLSATOB) then
  LLexecv(4) = .TRUE.    ! for 'sathdr_screen_satob'
else if (LLRESAT) then
  LLexecv(5) = .TRUE.    ! for 'sathdr_screen_resat'
else if (LLLRAD) then
  LLexecv(6) = .TRUE.    ! for 'sathdr_screen_lrad'
else if (LLRADAR) then
  LLexecv(7) = .TRUE.    ! for 'sathrd_radar'
  LLexecv(8) = .TRUE.    ! for 'satbody_radar'
else if (LLGPSRO) then
  LLexecv(11) = .TRUE.    ! for 'sathdr_screen_gpsro'
else if (LLCONV) then
  LLexecv(12) = .TRUE.    ! for 'robhdr_screen_conv'
endif
IF (LHOOK) CALL DR_HOOK('GETDB:PREPROC_HRETR',1,ZHOOK_HANDLE)
END SUBROUTINE preproc_hretr
```

```
    enddo ! do it=1,inumt_actual
  endif
```