chipcube

January 12, 2017

Abstract

chipcube is a library of routines for dealing with FITS storage of chip-coordinate images.

1 Description

This library defines the chip cube, which is a way of storing CCD images in FITS files; the library also provides some routines for manipulating these files. A CCD image in the XMM-Newton context is a 2-dimensional image which has the same dimensions as a CCD of one of the EPIC x-ray cameras, and which usually records some quantity explicitly or implicitly related to the x-ray flux detected by the CCD. Because all the CCDs from any given EPIC camera have the same dimensions, their images can be stored as slices of a cube (3-dimensional array). This compact and convenient representation is the format used by the present library. The chip cube is stored in the primary data array of the FITS file.

When I wrote this library, I chose the 1st index (of both the fortran and FITS array representations) to be the CCD index, and the 2nd and 3rd as the CCD x and y coordinates respectively. Now it seems obvious to me that a representation in which the CCD index was the 3rd index would be far more natural. Since it seems unlikely that the present library will ever be widely used I feel reasonably sanguine about changing this order at some future date.

For most quantities measured on EPIC CCDs, an association with a set of Good Time Intervals (GTIs) is important. In the original x-ray event list created as part of the XMM product set, a separate series of GTIs is provided per CCD. Thus in the FITS format for storing chip cubes, an series of binary table extensions for storing GTIs, one per CCD, is an option.

Finally, for maximum generality, a key table is prescribed for the FITS file, for the purpose of relating the CCD number to the relevant plane of the chip cube and also to the GTI extension name which should apply to that CCD. This key table is a mandatory binary table extension with the name `KEY2IMGS`. The table should have as many rows as there are CCD planes in the chip cube, and must have the following columns:

1. `CCDNR` (8-bit integer)
2. `NODE_NR` (8-bit integer)
3. `GTI_NAME` (8-character string). If this is blank or empty, it is assumed that no GTI table is attached for that CCD.

I may some time in the future add the following:

1. a string-valued column SUBMODE to specify the data mode of the CCD.
2. WINDOWX0, WINDOWY0, WINDOWDX and WINDOWDY columns to delineate data windows on the CCD.

There is a corresponding F90 data type defined:

```
type, public :: keyInfoType
  integer(int8) :: ccdNum=0, nodeNum=0
  character(8)  :: gtiTableName=''
end type keyInfoType
```

Row i of the table refers to plane array(i,:,:) (assuming both series start at 0).

The dataset header should contain INSTRUME and DATE-OBS keywords.

2 Library routines

2.1 readCubeData

Reads the chip cube, key-table and GTI data from a FITS table obeying the chip cube format. Note that the type gtiVecType is defined in **ssclib**, module intervals_aux. It is just a 1-d array of pointers, each of which is itself a 1-d array of intervals. If there are no GTIs for a given CCD, that intervals pointer is dimensioned to size 0. None of the pointers are left unassociated.

```
subroutine readCubeDataSingle(cubeSetName, cube, keyInfo, gtiVec)
  character(*),                intent(in) :: cubeSetName
  real(single),                pointer    :: cube(:,:,:)
  type(keyInfoType),           pointer    :: keyInfo(:)
  type(gtiVecType),  optional, pointer    :: gtiVec(:)
end subroutine

subroutine readCubeDataLogical(cubeSetName, cube, keyInfo, gtiVec)
  character(*),                intent(in) :: cubeSetName
  logical,                     pointer    :: cube(:,:,:)
  type(keyInfoType),           pointer    :: keyInfo(:)
  type(gtiVecType),  optional, pointer    :: gtiVec(:)
end subroutine
```

I'll overload this to more data types as I get time.

2.2 writeCubeData

All header keywords in the refSet are copied over to the new cubeSet.

```
subroutine writeCubeDataSingle(cubeSetName, refSetName, cube, keyInfo, gtiVec)
  character(*),      intent(in)           :: cubeSetName, refSetName
```

```
  real(single),      intent(in)           :: cube(:,:,:)
  type(keyInfoType), intent(in)           :: keyInfo(size(cube, 1))
  type(gtiVecType),  intent(in), optional :: gtiVec(size(cube, 1))
end subroutine

subroutine writeCubeDataLogical(cubeSetName, refSetName, cube, keyInfo)
  character(*),      intent(in) :: cubeSetName, refSetName
  logical,           intent(in) :: cube(:,:,:)
  type(keyInfoType), intent(in) :: keyInfo(size(cube, 1))
end subroutine
```

This second routine does not yet accept GTIs - it can be clearly seen that the library is relatively immature.

2.3 readGtisFromCube

```
subroutine readGtisFromCube(chipCubeSetName, gtiVec)
  character(*),     intent(in) :: chipCubeSetName
  type(gtiVecType), pointer    :: gtiVec(:)
end subroutine
```

2.4 writeGtisToCube

```
subroutine writeGtisToCube(chipCubeSetName, gtiVec)
  character(*),     intent(in) :: chipCubeSetName
  type(gtiVecType), intent(in) :: gtiVec(:)
end subroutine
```

2.5 readKeyTable

```
subroutine readKeyTable(set, keyInfo)
  type(DataSetT),    intent(in) :: set
  type(keyInfoType), pointer    :: keyInfo(:)
end subroutine
```

2.6 numberOfImagesInCube

```
integer function numberOfImagesInCube(chipCubeSetName)
  character(*), intent(in) :: chipCubeSetName
end function
```

References