HEASP Guide

Version 2.8

Keith A. Arnaud

HEASARC
Code 662
Goddard Space Flight Center
Greenbelt, MD 20771
USA

March 2025

Contents

Chapter 1

Introduction

HEASP is a C/C++/Python library to manipulate files associated with high energy astrophysics spectroscopic analysis. Currently this handles PHA, RMF, ARF and xspec table model files. The eventual plan is to be able to provide at least the functionality available in current ftools but from a single library.

The main code is written in C++ with classes for each file type. These classes and associated methods have been run through SWIG to produce Python code. C wrappers are provided for many of the C++ methods allowing simple use from C programs.

These C wrappers are compatible with an earlier C only version of HEASP with two exceptions: a) the include file required is now Cheasp.h instead of heasp.h; b) all routines which used to require a FITS file pointer now just require the filename. One consequence of b) is that files no longer need to be opened and closed by the calling program.

There are separate chapters describing the Python, C++, and C interfaces although users of Python should consult the C++ chapter for description of the classes.

Highlights of HEASP are :

- Read and write spectra, responses, arfs and table model files.

- Rebin spectra using grouping arrays created using a variety of methods.

- Compress responses by removing all elements below some value. Rebin responses in either energy or channel space based on a grouping array.

- Use a response to generate random channel numbers for a photon of a given energy.

- Multiply a response by a model.

- Sum both rmfs and arfs and multiply rmfs by arfs.

- Construct type II PHA or ARF from sets of spectra or arfs. Also extract individual spectra or arfs from type II files.

- Create table model files and interpolate on them.

Note that version 1.1 changed the definition of the Real type from float to double for consistency with its use in the xspec code however most FITS data read by these routines are single-precision.

Version 2.0 introduced set and get functions for all internal class members and made the members themselves private. The Python module now uses numpy arrays for C++ vector quantities.

Version 2.2 changed the storage of the rmf members which are potentially variable length vector columns in the FITS file from a single vector to a vector of vectors i.e. a single vector for each energy bin. This requires less contiguous memory (useful for very large matrices and simplifies the code.

Version 2.3 added to the table class units for parameters, the option to scale energies, and the option to have multiple spectra at each parameter grid point with the one to use selected by xspec filter expressions.

Version 2.4 was a minor update adding a new binning method to pha and fixing bugs.

Version 2.5 changed the heasp.h file so it doesn't include using namespace directives for std and CCfits and it places all its own variables and functions in a namespace called heasp. This meant that any programs using heasp and using std and/or CCfits routines may need changing either by explicitly using std::, CCfits::, or heasp:: for the appropriate routines or by adding their own using namespace directives. This version also added support for spectropolarimetry keywords.

Version 2.6 is a minor update with changes to some arf, pha, and rmf methods to remove confusion about extension numbers and version numbers.

Version 2.7 is a minor update with changes to the grouping, pha, and rmf methods mainly to fix unusual cases.

Version 2.8 is a minor update with changes to the arf, pha, phaII, grouping, and rmf classes as well as adding a couple of useful routines to SPutils. The read method in pha and phaII now uses any extension suffix on the filename to read a specific extension.

Chapter 2

Python module

2.1 Getting startedin

The standard HEAsoft build makes a Python 3 module based on the HEASP C++ classes described in the next chapter. The standard HEAsoft initialization script adds the location of the HEASP Python module to PYTHONPATH. To load the module use

```
UNIX> python3
>>>from heasp import *
```

If this produces name conflicts then loading the module by

```
UNIX> python3
>>>import heasp
```

requires heasp. in front of all commands. Since heasp C++ vectors are mapped to numpy arrays it is also a good idea to load the numpy module by

```
>>>from numpy import *
```

To set up an HEASP object first give a simple command such as

```
>>>sp = pha()
```

for a spectrum. This can then be read in and information about it displayed by

```
>>>sp.read("file1.pha")
>>>sp.disp()
```

Alternatively, the object can be defined directly from a file by

```
>>>sp = pha("file1.pha")
>>>sp.disp()
```

Individual components of the object can be accessed by get methods. For instance:

```
>>>first = sp.getFirstChannel()
```

will place the first channel number in the variable first. There are corresponding set methods. For instance:

```
>>>sp.setFirstChannel(0)
```

The components of each class are given in the appropriate chapter describing the C++ interface.

Components which are arrays can be loaded into numpy arrays. For instance:

```
>>>counts = sp.getPha()
```

Alternatively, a single element can be accessed:

```
>>>sp.getPhaElement(5)
```

Going the other way, a single element of the current contents of the PHA column can be replaced by e.g.:

```
>>>sp.setPhaElement(5,0.4)
```

and the entire array using e.g.:

```
sp.setPha(array([0.1,0.2,0.3,0.4,0.5]))
```

Setting an individual element does not resize the array but setting the entire array does.

2.2   HEASP features not (yet) supported in Python

At present the HEASP Python module does not support binary operations defined in the C++ classes such as the addition of two responses by resp = resp1 + resp2. What are supported are the unary equivalents so responses can be added by resp1 += resp2.

2.3   Spectrum example

The following Python example reads a type II PHA file, rebins the channels in each spectrum by a factor of 2, multiplies the spectrum values by 1.5, and writes out the result.

```
import os
from heasp import *

# read the spectrum
spectra = phaII("testin.pha")

# loop round the spectra in the file
# rebinning by a factor of 2 then placing
# in the output spectra
output = phaII()
Nspectra = spectra.getNumberSpectra()
groupInfo = grouping()

for i in xrange(Nspectra):
  spectrum = spectra.getphasElement(i)
  groupInfo.load(2,spectrum.getNumberChannels())
  status = spectrum.rebinChannels(groupInfo, "PROPAGATE")
  spectrum *= 1.5
  output.push(spectrum)

# write out the spectrum
phafile = "testout.pha"
if (os.path.exists(phafile)): os.remove(phafile)
status = output.write(phafile)
if status != 0: print("Failed to write testout.pha: status = ", status)
```

2.4 Response example

The following Python example reads RMF and ARF files, removes all elements smaller than 10^{-6} from the RMF, multiplies the compressed RMF and the ARF, and writes out the result.

```
import os
from heasp import *

# read RMF and ARF
inputRMF = rmf("testin.rmf")
inputARF = arf("testin.arf")

# compress the RMF
inputRMF.compress(1.0e-6)

# if the RMF and ARF are compatible then multiply them and write
# the result adding extra keywords and extensions from testin.rmf.
if inputRMF.checkCompatibility(inputARF) == 0:
```

```
inputRMF *= inputARF
rmffile = "testout.rsp"
if (os.path.exists(rmffile)): os.remove(rmffile)
status = inputRMF.write(rmffile, "testin.rmf")
if status != 0: print("Failed to write testout.rsp: status = ", status)
```

2.5   Ancillary response file example

The following Python example reads in three ARF files and writes out a type II ARF file containing.

```
import os
from heap import *

# read the three type I ARF files
arf1 = arf(''first.arf'')
arf2 = arf(''second.arf'')
arf3 = arf(''third.arf'')

# load the the individual ARF objects into a type II ARF object
arfout = arfII()
arfout.push(arf1)
arfout.push(arf2)
arfout.push(arf3)

# and write it out
arffile = "output.arf"
if (os.path.exists(arffile)): os.remove(arffile)
status = arfout.write(arffile)
if status != 0: print("Failed to write output.arf: status = ", status)
```

2.6   Table model example

The following Python example sets up a table model grid with two parameters. The parameters, energies and fluxes are given arbitrary values, in practice these could be read from text files.

```
import os
from heasp import *
import numpy as np

test = table()

# set table descriptors and the energy array
```

```
test.setModelName("Test")
test.setModelUnits(" ")
test.setisRedshift(True)
test.setisAdditive(True)
test.setisError(False)

# set up the energies. note that the size is one greater
# than that of the array for the model fluxes
test.setEnergies(np.linspace(0.1,10.0,100))

test.setNumIntParams(2)
test.setNumAddParams(0)

# define first parameter and give it 11 values ranging from
# 0.0 to 2.0 in steps of 0.2.

testpar = tableParameter()
testpar.setName("param1")
testpar.setInterpolationMethod(0)
testpar.setInitialValue(1.0)
testpar.setDelta(0.1)
testpar.setMinimum(0.0)
testpar.setBottom(0.0)
testpar.setTop(2.0)
testpar.setMaximum(2.0)
testpar.setTabulatedValues(np.linspace(0.0,2.0,11))

# and push it onto the vector of parameters
test.pushParameter(testpar)

# define the second parameter and give it 5 values ranging from
# 4.6 to 5.4 in steps of 0.2. As an example this parameter is set up
# using a create method which allows information to be passed in one call

testpar2 = tableParameter("param2", 0, 5.0, 0.1, 4.6, 4.6, 5.4, 5.4)
testpar2.setTabulatedValues(np.linspace(4.6,5.4,5)

# and push it onto the vector of parameters
test.pushParameter(testpar2);

# now set up the spectra. these are arbitrarily calculated, in a real program
# this step would read a file or call a routine.

for i1 in range(11):
  for i2 in range(5):
```

```
    flux = empty((99))
    for j in range(99): flux[j] = 0.2*i1+10*(4.6+0.2*i2)+j*0.1
    testspec = tableSpectrum()
    testspec.setParameterValues(np.array([0.2*i1,4.6+0.2*i2]))
    testspec.setFlux(flux)
    test.pushSpectrum(testspec)

# now write out the table.
tablefile = "test.mod"
if (os.path.exists(tablefile)): os.remove(tablefile)
status = test.write(tablefile);
if status != 0: print("Failed to write test.mod: status = ", status)
```

2.7   Table model for spectropolarimetry example

Writing a table for a spectropolarimetric table requires providing three spectra for each point in
the table. The initial set-up of the table descriptors now becomes:

```
test.setModelName("SPTest")
test.setModelUnits(" ")
test.setisRedshift(True)
test.setisAdditive(True)
test.setisError(False)
filters = ["Stokes:0","Stokes:1","Stokes:2"]
test.setFiltExps(filters)
```

and the loops setting the example spectra:

```
for i1 in range(11):
  for i2 in range(5):
    flux = empty((99))
    for k in range(3):
      for j in range(99): flux[j] = 0.2*i1+10*(4.6+0.2*i2)+j*0.1+k/3.0
      testspec = tableSpectrum()
      testspec.setParameterValues(np.array([0.2*i1,4.6+0.2*i2]))
      testspec.setFlux(flux)
      test.pushSpectrum(testspec)
```

2.8   An example combining heasp and pyxspec

The following example uses pyxspec to calculate a set of fluxes given the input response, multiplies
the result by the response to give predicted counts and writes the result out as a spectrum.

```
import os
from heasp import *
import xspec as xsp
import nump as np

responseName = "response.rsp"
inputRSP = rmf(responseName)

# energies needs to be the standard internal xspec energy array
numEnergies = inputRSP.NumberEnergyBins() + 1
energies = np.empty((numEnergies))
energies[0] = inputRSP.getLowEnergyElement(0)
for j in range(1,numEnergies): energies[j] =
                inputRSP.getHighEnergyElement(j-1)

# set up the parameters array as a list
params = [ 1.7 ]

# calculate the model and output flux as a list
fluxlist = []
xsp.callModelFunction("powerlaw", energies.tolist(), params, fluxlist)

# convert the flux to a numpy array and multiply by the response
flux = np.array(fluxlist)
phaValues = inputRSP.multiplyByModel(flux)

# set up the spectrum channel array
channel = np.arange(phaValues.size).astype(np.int32)

# pick an arbitrary exposure time of 10000
exposure = 10000.0
phaValues *= exposure

phaOut = pha()
phaOut.setFirstChannel(0)
phaOut.setPha(phaValues)
phaOut.setChannel(channel)
phaOut.setExposure(exposure)
phaOut.setDetChans(phaValues.size)
phaOut.setPoisserr(True)
phaOut.setDatatype("COUNT")
phaOut.setSpectrumtype("TOTAL")
phaOut.setResponseFile(responseName)
phaOut.setTelescope(inputRSP.getTelescope())
phaOut.setInstrument(inputRSP.getInstrument())
```

```
phaOut.setFilter(inputRSP.getFilter())

phafile = "outputspectrum.pha"
if (os.path.exists(phafile)): os.remove(phafile)
status = phaOut.write(phafile)
if status != 0: print("Failed to write outputspectrum.pha: status = ", status)
```

Chapter 3

C++ classes and methods

3.1   Global defines

HEASP uses the include file "heasp.h" to set a number of global definitions.

```
typedef int Integer;
typedef double Real;
```

set Integer and Real which are used in all classes. It also sets some helpful definitions of vectors and arrays

```
typedef std::valarray<Real> RealArray;
typedef std::vector<Real> RealVector;
typedef std::vector<Integer> IntegerVector;
typedef std::vector<string> StringVector;
```

This include file also contains the enumeration all error statuses.

```
enum{OK, NoSuchFile, NoData, NoChannelData, NoStatError, CannotCreate,
     NoEnergLo, NoEnergHi, NoSpecresp, NoEboundsExt, NoEmin, NoEmax,
     NoMatrixExt, NoNgrp, NoFchan, NoNchan, NoMatrix, CannotCreateMatrixExt,
     CannotCreateEboundsExt, InconsistentGrouping, InconsistentEnergies,
     InconsistentChannels, InconsistentUnits, UnknownXUnits, UnknownYUnits,
     InconsistentNumelt, InconsistentNumgrp, InconsistentNumTableParams,
     TableParamValueOutsideRange, VectorIndexOutsideRange,
     InconsistentKeywordValues, CannotCopyColumn, CannotWriteMatrix,
     InconsistentTableFilter, NoChannels, InconsistentChannelMin,
     InconsistentFChan, InconsistentNChan};
```

3.2   Spectra

3.2.1   Introduction and example

Spectrum files can be manipulated using the pha and phaII classes. The latter is simply a vector of pha classes and is useful for handling type II PHA files. The grouping class and utility routines are also useful for some tasks. As an example the code below reads in a type II PHA file, bins up all the spectra by a factor of 2, then writes out the result. Note that the data types Integer and Real are defined in heasp.h.

```
#include "grouping.h"
#include "phaII.h"

using namespace std;
using namespace heasp;

int main(int argc, char* argv[])
{
  const string infile("testin.pha");
  const string outfile("testout.pha");

  Integer Status(0);

  // read in all the spectra

  phaII inputSpectra(infile);

  Integer Nspectra = inputSpectra.getNumberSpectra();

  // loop round the spectra

  for (size_t i=0; i<(size_t)Nspectra; i++) {

    // set up the grouping object to rebin by a factor of 2

    grouping groupInfo;
    groupInfo.load(2, inputSpectra.phas[i].NumberChannels());

    // rebin this spectrum

    pha thisSpectrum = inputSpectra.get(i);
    Status = thisSpectrum.rebinChannels(groupInfo,"PROPAGATE");
    Status = inputSpectra.setphasElement(i, thisSpectrum);
```

```
  }

  // write the new spectra out copying the primary HDU, other HDUs,
  // keywords and columns

  Status = inputSpectra.write(outfile, infile);

  exit(Status);
}
```

3.2.2   pha class private members

```
class pha{
 private:

  Integer m_FirstChannel;     // First legal channel number

  RealVector m_Pha;           // PHA data
  RealVector m_StatError;      // Statistical error
  RealVector m_SysError;       // Statistical error

  IntegerVector m_Channel;     // Channel number
  IntegerVector m_Quality;     // Data quality (0=good, 1=bad, 2=dubious,
                               //                 5=set bad by user)
  IntegerVector m_Group;       // Data grouping (1=start of bin,
                               //                 -1=continuation of bin)

  RealVector m_AreaScaling;    // Area scaling factor
  RealVector m_BackScaling;    // Background scaling factor

  Real m_Exposure;            // Exposure time
  Real m_CorrectionScaling;    // Correction file scale factor

  Integer m_DetChans;         // Total legal number of channels
  bool m_Poisserr;            // If true, errors are Poisson
  string m_Datatype;          // "COUNT" for count data and "RATE" for count/sec
  string m_PHAVersion;        // PHA extension format version

  string m_Spectrumtype;      // "TOTAL", "NET", or "BKG"

  string m_ResponseFile;      // Response filename
  string m_AncillaryFile;     // Ancillary filename
  string m_BackgroundFile;    // Background filename
```

```
string m_CorrectionFile;    // Correction filename

string m_FluxUnits;         // Units for Pha and StatError

string m_ChannelType;       // Value of CHANTYPE keyword
string m_Telescope;
string m_Instrument;
string m_Detector;
string m_Filter;
string m_Datamode;

string m_StokesParameter;   // For spectropolarimetry
                            //       (e.g. 'Ixx', 'xQx', 'xxU') //
string m_StokesWeighting;   // Stokes parameter weighting scheme

StringVector m_XSPECFilter; // Filter keywords
```

3.2.3   pha class public methods

- `pha()`

  `pha(const pha& a)`

  `pha(const string filename, const Integer PHAnumber = 1,`
  `    const Integer SpectrumNumber = 1)`

  `pha(const RealVector& inPha, const RealVector& inStatError,`
  `    const RealVector& inSysError, const IntegerVector& inChannel,`
  `    const IntegerVector& inQuality, const IntegerVector& inGroup,`
  `    const RealVector& inAreaScaling, const RealVector& inBackScaling,`
  `    const Integer inFirstChannel, const Real inExposure,`
  `    const Real inCorrectionScaling, const Integer inDetChans,`
  `    const bool inPoisserr, const std::map<string,string>& inKeys,`
  `    const StringVector& inXSPECFilter = StringVector())`

  Constructors. The first is the default constructor, the others are constructor versions of the copy, read, and load methods described below.

- `~pha()`

  Default destructor.

- `Integer read(string filename, Integer PHAnumber = 1,`
  `               Integer SpectrumNumber = 1)`

  Read file into object. If PHAnumber is given then look for the SPECTRUM extension with EXTVER=PHAnumber. The third argument is to read the pha from the SpectrumNumber row of a type II file.

- `pha(const RealVector& inPha, const RealVector& inStatError,`
    `const RealVector& inSysError, const IntegerVector& inChannel,`
    `const IntegerVector& inQuality, const IntegerVector& inGroup,`
    `const RealVector& inAreaScaling, const RealVector& inBackScaling,`
    `const Integer inFirstChannel, const Real inExposure,`
    `const Real inCorrectionScaling, const Integer inDetChans,`
    `const bool inPoisserr, const std::map<string,string>& inKeys,`
    `const StringVector& inXSPECFilter = StringVector())`

  Load all the required information into the pha object.

- `pha& operator= (const pha&)`

  `pha& copy(const pha&)`

  Deep copy.

- `string disp() const`

  Display information about the spectrum - return as a string.

- `void clear()`

  Clear information from the spectrum

- `string check() const`

  Check completeness and consistency of information in spectrum, if there is a problem then return diagnostic in string.

- `Integer write(const string filename, const string sourceFilename="",`
    `const Integer extVer=-1) const`

  Write spectrum as type I file. If sourceFilename is given then the primary HDU and any other extra HDUs are copied into the output file as well as any extra keywords or columns in the spectrum extension. If extVer is given then the SPECTRUM extension is given that version number.

- `pha& operator*= (const Real)`

  Multiply by a constant.

- `pha& operator+= (const pha&)`

  Add to another pha. If one or both of the pha have non-Poisson errors then they are added in quadrature. If systematic errors are present then the resulting pha has systematic errors equal to the maximum of those from the input pha. The exposures are added.

- `Integer checkCompatibility(const pha&) const`

  Check compatibility with another pha. This should be done before attempting to sum them.

- `Integer selectChannels(IntegerVector& Start, IntegerVector& End)`

  Select a subset of the channels

- `Integer setGrouping(grouping&)`

  Set the pha grouping array from a grouping object.

- `grouping getGrouping() const`

  Get a grouping object from the pha grouping array

- `grouping getMinCountsGrouping(const Integer MinCounts,`
  `                              const Integer StartChannel,`
  `                              const Integer EndChannel) const`

  Get grouping (optionally between channels StartChannel and EndChannel) using a minimum number of counts per bin. If StartChannel and EndChannel are both zero then all channels are included.

- `grouping getMinSNGrouping(const Real SignalToNoise,`
  `                          const pha& Background,`
  `                          const Integer StartChannel,`
  `                          const Integer EndChannel) const`

  Get grouping (optionally between channels StartChannel and EndChannel) using a minimum S/N. Optionally includes a background file as well. If StartChannel and EndChannel are both zero then all channels are included. If Background = pha() then the background is not included.

- `grouping getMinSNOptimalGrouping(const RealVector FWHM,`
  `                                 const Real SignalToNoise,`
  `                                 const pha& Background,`
  `                                 const Integer StartChannel,`
  `                                 const Integer EndChannel) const`

  Get grouping (optionally between channels StartChannel and EndChannel) using a optimal binning with a minimum S/N. Optionally includes a background file as well. If StartChannel and EndChannel are both zero then all channels are included. If Background = pha() then the background is not included. If SignalToNoise is negative then it will be ignored and only optimal binning is used.

- `Integer rebinChannels(grouping& groupInfo,`
  `                      const string errorType)`

  Rebin spectrum channels based on a grouping object. If no string is input then the resulting StatError is calculated by summing in quadrature. If a string is input then the resulting StatError is calculated based on the value of the input string. If it is PROPAGATE then summing in quadrature is used. If not, then GAUSS uses 1/sqrt(Pha), POISS-0 sets Poisserr to true, POISS-1 uses $1 + \mathrm{sqrt}(N+0.75)$, POISS-2 uses sqrt(N-0.25), and POISS-3 uses the arithmetic mean of POISS-1 and POISS-2. In general, only the options PROPAGATE or POISS-0 should be used.

- `Integer shiftChannels(const Integer Start, const Integer End,`
  `                      const Real Shift, const Real Factor = 1.0)`

```
Integer shiftChannels(const IntegerVector& Start,
                      const IntegerVector& End, const RealVector& Shift,
                      const RealVector& Factor)


Integer shiftChannels(const RealVector& ChannelLowEnergy,
                      const RealVector& ChannelHighEnergy,
                      const Integer Start, const Integer End,
                      const Real Shift, const Real Factor = 1.0)


Integer shiftChannels(const RealVector& ChannelLowEnergy,
                      const RealVector& ChannelHighEnergy,
                      const IntegerVector& Start, const IntegerVector& End,
                      const RealVector& Shift, const RealVector& Factor)
```

Shift counts in channels. Those counts between channels Start and End will be shifted by Shift channels and stretched by Factor. If the channel energies are given then Shift is assumed to be in energy, otherwise in channels.Total counts are conserved.

- `Integer convertUnits(const RealVector& ChannelLowEnergy,`
  `const RealVector& ChannelHighEnergy,`
  `const string EnergyUnits)`

Convert flux units from whatever they are currently to $ph/cm^2/s$. This requires as input the channel energy arrays from the rmf object and the string specifying their units. The allowed flux units are: $ph/cm^2/s/X$, where X is one of MeV, GeV, Hz, A, cm, um, nm; $ergs/cm^2/s$; $ergs/cm^2/s/X$, where X is one of Hz, A, cm, um, nm; Jy. The allowed energy units are keV, GeV, Hz, angstrom, cm, micron or nm.

- `Integer NumberChannels() const`

  `Integer getNumberChannels() const`

Return the size of RealVectors.

3.2.4 Public pha methods to get and set internal data

- `Integer getFirstChannel() const`

  `void setFirstChannel(const Integer value)`

- `const RealVector& getPha() const`

  `Integer setPha(const RealVector& values)`

  `Real getPhaElement(const Integer i) const`

```
Integer setPhaElement(const Integer i, const Real value)
```

- `const RealVector& getStatError() const`

  ```
  Integer setStatError(const RealVector& values)
  ```

  ```
  Real getStatErrorElement(const Integer i) const
  ```

  ```
  Integer setStatErrorElement(const Integer i, const Real value)
  ```

- `const RealVector& getSysError() const`

  ```
  Integer setSysError(const RealVector& values)
  ```

  ```
  Real getSysErrorElement(const Integer i) const
  ```

  ```
  Integer setSysErrorElement(const Integer i, const Real value)
  ```

- `const IntegerVector& getChannel() const`

  ```
  Integer setChannel(const IntegerVector& values)
  ```

  ```
  Integer getChannelElement(const Integer i) const
  ```

  ```
  Integer setChannelElement(const Integer i, const Integer value)
  ```

- `const IntegerVector& getQuality() const`

  ```
  Integer setQuality(const IntegerVector& values)
  ```

  ```
  Integer getQualityElement(const Integer i) const
  ```

  ```
  Integer setQualityElement(const Integer i, const Integer value)
  ```

- `const IntegerVector& getGroup() const`

  ```
  Integer setGroup(const IntegerVector& values)
  ```

  ```
  Integer getGroupElement(const Integer i) const
  ```

  ```
  Integer setGroupElement(const Integer i, const Integer value)
  ```

- `const RealVector& getAreaScaling() const`

  `Integer setAreaScaling(const RealVector& values)`

  `Real getAreaScalingElement(const Integer i) const`

  `Integer setAreaScalingElement(const Integer i, const Real value)`

- `const RealVector& getBackScaling() const`

  `Integer setBackScaling(const RealVector& values)`

  `Real getBackScalingElement(const Integer i) const`

  `Integer setBackScalingElement(const Integer i, const Real value)`

- `Real getExposure() const`

  `void setExposure(const Real value)`

- `Real getCorrectionScaling() const`

  `void setCorrectionScaling(const Real value)`

- `Integer getDetChans() const`

  `void setDetChans(const Integer value)`

- `bool getPoisserr() const`

  `void setPoisserr(const bool value)`

- `string getDatatype() const`

  `void setDatatype(const string value)`

- `string getSpectrumtype() const`

  `void setSpectrumtype(const string value)`

- `string getResponseFile() const`

```
     void setResponseFile(const string value)
```

- ```
  string getAncillaryFile() const
  ```

  ```
  void setAncillaryFile(const string value)
  ```

- ```
  string getBackgroundFile() const
  ```

  ```
  void setBackgroundFile(const string value)
  ```

- ```
  string getCorrectionFile() const
  ```

  ```
  void setCorrectionFile(const string value)
  ```

- ```
  string getFluxUnits() const
  ```

  ```
  void setFluxUnits(const string value)
  ```

- ```
  string getChannelType() const
  ```

  ```
  void setChannelType(const string value)
  ```

- ```
  string getTelescope() const
  ```

  ```
  void setTelescope(const string value)
  ```

- ```
  string getInstrument() const
  ```

  ```
  void setInstrument(const string value)
  ```

- ```
  string getDetector() const
  ```

  ```
  void setDetector(const string value)
  ```

- ```
  string getFilter() const
  ```

  ```
  void setFilter(const string value)
  ```

- ```
  string getDatamode() const
  ```

  ```
  void setDatamode(const string value)
  ```

- ```
  string getStokesParameter() const
  ```

  ```
  void setStokesParameter(const string value)
  ```

- ```
  string getStokesWeighting() const
  ```

  ```
  void setStokesWeighting(const string value)
  ```

- ```
  const StringVector& getXSPECFilter() const
  ```

  ```
  Integer setXSPECFilter(const StringVector& values)
  ```

  ```
  string getXSPECFilterElement(const Integer i) const
  ```

  ```
  Integer setXSPECFilterElement(const Integer i, const string value)
  ```

3.2.5   Other pha routines

- ```
  pha operator+ (const pha& a, const pha& b)
  ```
  Add pha a and b. If one or both of the pha have non-Poisson errors then they are added
  in quadrature. If systematic errors are present then the resulting pha has systematic errors
  equal to the maximum of those from the input pha. The exposures are added.

- ```
  bool isSpectrumValid(const string filename, const Integer extNumber)
  ```
  Simple test for whether the specified extension is a valid spectrum. This just tests for
  HDUCLAS1='SPECTRUM' and the presence of a COUNTS or RATE column.

- ```
  Integer PHAtype(const string filename, const Integer PHAnumber)
  ```

  ```
  Integer PHAtype(const string filename, const Integer PHAnumber,
                  Integer& Status)
  ```

  Return the type of a SPECTRUM extension.

- ```
  bool IsPHAcounts(const string filename, const Integer PHAnumber)
  ```

  ```
  bool IsPHAcounts(const string filename, const Integer PHAnumber,
                   Integer& Status)
  ```

  Return true if the COUNTS column exists and is integer.

- ```
  Integer NumberofSpectra(const string filename,
                          const Integer PHAnumber)
  ```

```
    Integer NumberofSpectra(const string filename,
                            const Integer PHAnumber, Integer& Status)
```

Return the number of spectra in a type II SPECTRUM extension.

- ```
  IntegerVector SpectrumExtensionNumbers(const string filename)
  ```

  ```
  IntegerVector SpectrumExtensionNumbers(const string filename,
                                         Integer& Status)
  ```

  Return the numbers of any spectrum extensions

- ```
  IntegerVector SpectrumVersionNumbers(const string filename)
  ```

  ```
  IntegerVector SpectrumVersionNumbers(const string filename,
                                       Integer& Status)
  ```

  Return the versions of any spectrum extensions

3.2.6   phaII class private members

```
class phaII{
 private:

  std::vector<pha> m_phas;        // vector of pha objects
```

3.2.7   phaII class public methods

- ```
  phaII()
  ```

  ```
  phaII(const phaII& a)
  ```

  ```
  phaII(const string filename, const Integer PHAnumber = 1,
        const IntegerVector& SpectrumNumber = IntegerVector())
  ```

  Constructors. First is the default constructor, other two are constructor versions of the copy and read methods described below.

- ```
  ~phaII()
  ```
  Default destructor.

- `Integer read(const string filename, const Integer PHAnumber = 1,`
  `const IntegerVector& SpectrumNumber = IntegerVector())`

  Read a PHA type II file into an object. If PHAnumber is given then read from the SPEC-TRUM extension with EXTVER=PHAnumber. If the SpectrumNumber array is given then read those rows in the extension otherwise read all spectra.

- `phaII& copy(const phaII&)`

  `phaII& operator= (const phaII&)`

  Deep copy.

- `pha get(Integer number) const`

  Get pha object (counts from zero).

- `pha push(pha sp)`

  Push pha object into phaII object

- `string disp() const`

  Display information about the spectra - return as a string.

- `void clear()`

  Clear information about the spectra

- `string check() const`

  Check completeness and consistency of information in spectrum, if there is a problem then return diagnostic in string.

- `Integer write(const string filename, const string sourceFilename="",`
  `const Integer extVer=-1) const`

  Write spectra as type II file. Note that if the output filename exists and already has a SPECTRUM extension then this method will write an additional SPECTRUM extension. If sourceFilename is given then the primary HDU and any other extra HDUs are copied into the output file as well as any extra keywords or columns in the spectrum extension. If extVer is given then the SPECTRUM extension is given that version number.

- `Integer NumberSpectra()`

  `Integer getNumberSpectra()`

  Return the number of spectra in the object.

3.2.8   Public phaII methods to get and set internal data

- `const std::vector<pha>& getphas() const`

  `Integer setphas(const std::vector<pha>& values)`

  `pha getphasElement(const Integer i) const`

  `Integer setphasElement(const Integer i, const pha& value)`

3.3   Responses

3.3.1   Introduction and examples

Response files come in two varieties: RMFs and ARFs. The former contain the response matrices describing the probability of a photon of a given energy being registered in a given channel of the spectrum. The latter describes the effective area versus energy. The rmf class is used for manipulating RMFs and the arf and arfII classes for manipulating ARFs. The arfII class is an analog of the phaII class and is useful for the case where an ARF file contains many individual effective area curves. The rmft class handles the transposed response matrix and is of limited use at present.

The example code below shows a program to read in an RMF file, to compress the matrix to remove any element below 1.0e-6, to multiply the result by an ARF, and write a new RMF file.

```
#include "rmf.h"
#include "arf.h"

using namespace std;
using namespace heasp;

int main(int argc, char* argv[])
{
  string rmffile("testin.rmf");
  string arffile("testin.arf");
  string outfile("testout.rmf");

  rmf inputRMF, outputRMF;
  arf inputARF;

  Integer Status(0);

  // read in the RMF and the ARF
```

```
  Status = inputRMF.read(rmffile);
  Status = inputARF.read(arffile);

  // remove elements from the RMF with values < 1.0e-6

  Real threshold(1.0e-6);
  inputRMF.compress(threshold);

  // multiply the compressed RMF and the ARF to make an output RMF

  if ( inputRMF.checkCompatibility(inputARF) == OK ) {

    outputRMF = inputRMF * inputARF;

    // and write out the result copying any extra HDUs and keywords from
    // the input RMF

    Status = outputRMF.write(outfile, rmffile);

  }

  exit(0);
}
```

The next example shows an example program to create an RMF file.

```
#include "rmf.h"
using namespace std;
using namespace heasp;

int main(int argc, char* argv[])
{

  const size_t Nenergies(1024), Nchannels(1024);
  const Real responseThreshold(1.0e-6);

  rmf Response;

  Response.setAreaScaling(1.0);
  Response.setResponseThreshold(responseThreshold);
  Response.setEnergyUnits("keV");
  Response.setRMFUnits("cm^2");
```

```
Response.setChannelType("PI");
Response.setTelescope("MyTelescope");
Response.setInstrument("MyInstrument");
Response.setDetector("NONE");
Response.setFilter("NONE");
Response.setRMFType("FULL");
Response.setRMFExtensionName("MATRIX");
Response.setEBDExtensionName("EBOUNDS");

// code to load the response energy ranges

RealVector lowE(Nenergies), highE(Nenergies);
for(size_t i=0; i<Nenergies; i++) {
  lowE[i] = 0.1 + i*0.01;
  highE[i] = 0.1 + (i+1)*0.01;
}
Response.setLowEnergy(lowE);
Response.setHighEnergy(highE);

// code to load the channel energy ranges. in this
// case the channel energy ranges are identical to
// the response energy ranges but they need not be

RealVector chlowE(Nenergies), chhighE(Nenergies);
for(size_t i=0; i<Nchannels; i++) {
  chlowE[i] = 0.1 + i*0.01;
  chhighE[i] = 0.1 + (i+1)*0.01;
}
Response.setChannelLowEnergy(chlowE);
Response.setChannelHighEnergy(chhighE);

// loop over the energy ranges creating the response. In this
// example the response is assumed to be a simple gaussian

for (size_t i=0; i<Nenergies; i++) {

  Real centroid = (lowE[i] + highE[i])/2.0;
  Real sigma = sqrt(centroid);

  RealVector work(Nchannels);
  calcGaussResp(sigma, centroid, responseThreshold, chlowE,
                chhighE, work);

  // add to the response matrix
  Response.addRow(work, lowE[i], highE[i]);
```

```
  }

  // Convert to correct units (not required in this particular example)
  Integer status = Response.convertUnits();

  // Check for errors
  string diagnostic = Response.check();

  // Write to an output FITS file
  status = Response.write("MyResponse.rsp");

  return(0);

}
```

The following example shows how to read in three type I ARFs and write out a single type II ARF.

```
#include "arf.h"
using namespace heasp;

int main(int argc, char* argv[])
{

  arf arf1, arf2, arf3;
  arfII outarf;

  int status(0);

  // Read in the ARFs

  status = arf1.read(''first.arf'');
  status = arf2.read(''second.arf'');
  status = arf3.read(''third.arf'');

  // Load into the output type II ARF

  outarf.push(arf1);
  outarf.push(arf2);
  outarf.push(arf3);

  // Write to an output FITS file. Include any extra keywords
  // or extensions from the first arf file.
  status = outarf.write("output.arf", "first.arf");
```

```
  return(status);

}
```

3.3.2   rmf class private members

```
class rmf{
 private:

  Integer m_FirstChannel;              // First channel number

  IntegerVector m_NumberGroups;        // Number of response groups for this
                                       //  energy bin

  std::vector<IntegerVector> m_FirstChannelGroup; // First channel in group
  std::vector<IntegerVector> m_NumberChannelsGroup; // Number channels in group
  std::vector<IntegerVector> m_OrderGroup;         // Grating order of group

  RealVector m_LowEnergy;              // Start energy of bin
  RealVector m_HighEnergy;             // End energy of bin

  std::vector<RealVector> m_Matrix;    // Matrix elements

  RealVector m_ChannelLowEnergy;       // Start energy of channel
  RealVector m_ChannelHighEnergy;      // End energy of channel

  Real m_AreaScaling;                  // Value of EFFAREA keyword
  Real m_ResponseThreshold;            // Minimum value in response

  string m_EnergyUnits;                // Energy units used
  string m_RMFUnits;                   // Units for RMF values

  string m_ChannelType;                // Value of CHANTYPE keyword
  string m_Telescope;
  string m_Instrument;
  string m_Detector;
  string m_Filter;
  string m_RMFType;                    // HDUCLAS3 keyword in MATRIX extension
  string m_RMFExtensionName;           // EXTNAME keyword in MATRIX extension
  string m_EBDExtensionName;           // EXTNAME keyword in EBOUNDS extension

  string m_StokesParameter;            // For spectropolarimetry
```

```
                                        (e.g. 'Ixx', 'xQx', 'xxU') //
  string m_StokesWeighting;            // Stokes parameter weighting scheme
```

3.3.3 rmf class public methods

- `rmf()`

  `rmf(const rmf& a)`

  `rmf(const string filename, const Integer RMFnumber = 1)`

  ```
  rmf(const IntegerVector& inNumberGroups,
      const std::vector<IntegerVector>& inFirstChannelGroup,
      const std::vector<IntegerVector>& inNumberChannelsGroup,
      const std::vector<IntegerVector>& inOrderGroup,
      const RealVector& inLowEnergy, const RealVector& inHighEnergy,
      const std::vector<RealVector>& inMatrix,
      const RealVector& inChannelLowEnergy,
      const RealVector& inChannelHighEnergy,
      const Integer inFirstChannel, const Real inAreaScaling,
      const Real inResponseThreshold, const std::map<string,string>& inKeys)
  ```

  Constructors. The first is the default constructor. The rest are constructor versions of the copy, read, and load methods described below.

- `~rmf()`

  Default destructor.

- `Integer read(string filename, Integer RMFnumber = 1)`

  Read the RMF file into an object. If RMFnumber is given read from the MATRIX (or SPECRESP MATRIX) and EBOUNDS extensions with EXTVER=RMFnumber. If there is only one EBOUNDS extension then that will be used.

- `Integer readMatrix(string filename, Integer RMFnumber = 1)`

  Read the MATRIX (or SPECRESP MATRIX) extension from an RMF file into an object. If RMFnumber is given read from the MATRIX (or SPECRESP MATRIX) extension with EXTVER=RMFnumber.

- `Integer readChannelBounds(string filename, Integer RMFnumber = 1)`

  Read the EBOUNDS extension from an RMF file into an object. If RMFnumber is given read from the EBOUNDS extension with EXTVER=RMFnumber.

- ```
  Integer load(const IntegerVector& inNumberGroups,
               const std::vector<IntegerVector>& inFirstChannelGroup,
               const std::vector<IntegerVector>& inNumberChannelsGroup,
               const std::vector<IntegerVector>& inOrderGroup,
               const RealVector& inLowEnergy,
               const RealVector& inHighEnergy,
               const std::vector<RealVector>& inMatrix,
               const RealVector& inChannelLowEnergy,
               const RealVector& inChannelHighEnergy,
               const Integer inFirstChannel, const Real inAreaScaling,
               const Real inResponseThreshold,
               const std::map<string,string>& inKeys)
  ```

  Load the rmf object with the required information. The inKeys map is used to set the string members with the first element being the FITS keyword name and the second the values.

- ```
  void initialize(const arf&)
  ```

  Initialize from an arf object. Copies members in common between arfs and rmfs

- ```
  void loadDiagonalResponse(const RealVector& eLow,
                            const RealVector& eHigh,
                            const RealVector& rspVals,
                            const Integer firstChan)
  ```

  Load the simple case of a diagonal response where the response energy and channel energy ranges are assumed to be the same and the only non-zero response values are along the diagonal and are set using the rspVals argument.

- ```
  rmf& operator= (const rmf&)

  rmf& copy(const rmf&)
  ```

  Deep copy.

- ```
  Real ElementValue(Integer Channel, Integer EnergyBin,
                    Integer GratingOrder = -999)
  ```

  Return the value for a particular channel, energy, and grating order. If the grating order argument is set to -999 then it is ignored.

- ```
  RealVector RowValues(Integer EnergyBin,
                       Integer GratingOrder = -999)
  ```

  Return the response array for a particular energy and grating order. If the grating order argument is set to -999 then it is ignored.

- ```
  IntegerVector RandomChannels(const Real energy,
                               const Integer NumberPhotons,
                               const RealVector& RandomNumber,
                               const Integer GratingOrder = -999)
  ```

```
IntegerVector RandomChannels(const RealVector& energy,
                             const IntegerVector& NumberPhotons,
                             const std::vector<RealVector>& RandomNumber,
                             Integer GratingOrder = -999)
```

Use the response matrix to generate random channel numbers for photons of given energy and grating order. When using a vector of energies, NumberPhotons[i] events are generated for energy[i]. The RandomNumber input should be uniformly distributed between 0 and 1. There should be NumberPhotons[i] random numbers for energy[i]. If the grating order argument is set to -999 then it is ignored.

- `string disp() const`

  Display information about the response. - return as a string.

- `void clear()`

  Clear information from the response.

- `void clearMatrix()`

  Clear only the matrix information from the response.

- `string check() const`

  Check completeness and consistency of information in the rmf, if there is a problem then return diagnostic in string.

- `void normalize()`

  Normalize the rmf so it sums to 1.0 for each energy bin.

- `void compress(const Real threshold)`

  Compress the rmf to remove all elements below the threshold value.

- `void uncompress()`

  Uncompress the rmf i.e. turn it into a full rectangular matrix.

- `Integer rebinChannels(grouping&)`

  Rebin in channel space using the specified grouping object.

- `Integer rebinEnergies(grouping&)`

  Rebin in energy space using the specified grouping object.

- `Integer shiftChannels(const Integer Start,
                         const Integer End, const Real Shift,
                         const Real Factor,
                         bool useEnergyBounds)`

```
    Integer shiftChannels(const IntegerVector& vStart,
                          const IntegerVector& vEnd,
                          const RealVector& vShift,
                          const RealVector& vFactor,
                          bool useEnergyBounds)
```

Remap the response up or down in channels. Moves response between channels Start and End by Shift channels and stretches by Factor. If useEnergyBounds is true then Shift is assumed to be in energy, otherwise in channels.

- `Integer shiftEnergies(const Integer Start, const Integer End,`
  `                      const Real Shift, const Real Factor)`


```
    Integer shiftEnergies(const IntegerVector& vStart,
                          const IntegerVector& vEnd,
                          const RealVector& vShift,
                          const RealVector& vFactor)
```

Remap the response up or down in energies. Moves response between energy bins Start and End by Shift energy and stretches by Factor. Shift is assumed to be in energy.

- `Integer remapEnergies(const RealVector& targetEnergies)`

  Remap the response onto an arbitrary energy grid (which should be coarser than the original).

- `Integer interpolateAndMultiply(const RealVector& energies,`
  `                                const RealVector& factors)`

  Multiply by a vector which may not have the same energy binning as the response.

- `Integer write(const string filename,`
  `              const string sourceFilename="") const`

  Write response to a RMF file. Note that if the output filename exists and already has MATRIX and EBOUNDS extensions then this method will write additional extensions provided that no sourceFilename is given. If sourceFilename is given then the primary HDU and any other extra HDUs are copied into the output file as well as any extra keywords or columns in the EBOUNDS and MATRIX extensions.

- `Integer writeMatrix(const string filename) const`

  Write the MATRIX extension to a RMF file. Note that if the output filename exists and already has a MATRIX extension then this method will write an additional extension.

- `Integer writeChannelBounds(const string filename) const`

  Write the EBOUNDS extension to a RMF file. Note that if the output filename exists and already has a EBOUNDS extension then these methods will write an additional extension.

- `Integer append(const string filename, const bool doMatrix=true,`
  `const bool doEbounds=true) const`

  Append the MATRIX and/or the EBOUNDS extensions to a file.

- `rmf& operator*=(const arf&)`

  Multiply current rmf by an arf.

- `rmf& operator*=(const Real&)`

  Multiply current rmf by a factor.

- `rmf& operator+=(const rmf&)`

  Add another rmf to the current rmf.

- `Integer checkCompatibility(const rmf&)`

  Check compatibility with another rmf.

- `Integer checkCompatibility(const arf&)`

  Check compatibility with an arf.

- `Integer convertUnits()`

  Convert energy units from current units to keV. Valid options for the current units are keV, MeV, GeV, Hz, angstrom, cm, micron, nm.

- `void reverseRows()`

  Reverse the rows. This is useful if the rows are not in increasing order of energy, which xspec requires.

- `void addRow(const RealVector& Response,`
  `const Real eLow, const Real eHigh)`

  `void addRow(const std::vector<RealVector>& Response, const Real eLow,`
  `const Real eHigh, const IntegerVector& GratingOrder)`

  Add a row to the response using an input response vector and energy range.

- `void addRow(const IntegerVector& fChan, const IntegerVector& nChan,`
  `const RealVector& Response, const Real eLow, const Real eHigh)`

  Add a row to the response using channel group format

- `void substituteRow(const Integer RowNumber,`
  `const RealVector& Response)`

  `void substituteRow(const Integer RowNumber,`
  `const std::vector<RealVector>& Response,`
  `const IntegerVector& GratingOrder)`

Substitute a row into the response using an input response vector and energy range.

- `void substituteRow(const IntegerVector& fChan,`
  `                    const IntegerVector& nChan, const RealVector& Response,`
  `                    const Real eLow, const Real eHigh)`

Substitute a row into the response using channel group format

- `RealVector multiplyByModel(const RealVector& model)`

Multiply a response by a vector and output a vector of pha values. The input vector is assumed to be on the energy binning.

- `RealVector estimatedFWHM() const`

Return a vector containing the FWHM in channels for each energy. This does assume that the response has a well-defined main peak.

- `RealVector estimatedFWHMperChannel() const`

Return a vector containing the FWHM in channels for each channel. This does assume that the response has a well-defined main peak.

- `Integer NumberChannels() const`

  `Integer getNumberChannels() const`

Return the number of spectrum channels.

- `Integer NumberEnergyBins() const`

  `Integer getNumberEnergyBins() const`

Return the number of response energies.

- `Integer NumberTotalGroups() const`

  `Integer getNumberTotalGroups() const`

Return the number of response groups.

- `Integer NumberTotalElements() const`

  `Integer getNumberTotalElements() const\end{verbateim}`

```
        Return the number of response elements.
```

  `\end{itemize}`

  `\subsection{Public rmf methods to get and set internal data}`

```
\begin{itemize}

\item  \begin{verbatim}  Integer getFirstChannel() const

   void setFirstChannel(const Integer value)
```

- const IntegerVector& getNumberGroups() const

  Integer setNumberGroups(const IntegerVector& values)

  Integer getNumberGroupsForEnergyBin(const Integer i) const

  Integer setNumberGroupsForEnergyBin(const Integer i, const Integer value)

- const std::vector<IntegerVector>& getFirstChannelGroup() const

  Integer setFirstChannelGroup(const std::vector<IntegerVector>& values)

  IntegerVector& getFirstChannelGroupForEnergyBin(const Integer i) const

  Integer setFirstChannelGroupForEnergyBin(const IntegerVector i,
                                           const Integer value)

- const std::vector<IntegerVector>& getNumberChannelsGroup() const

  Integer setNumberChannelsGroup(const std::vector<IntegerVector>& values)

  IntegerVector& getNumberChannelsGroupForEnergyBin(const Integer i) const

  Integer setNumberChannelsGroupForEnergyBin(const Integer i,
                                             const IntegerVector value)

- const std::vector<IntegerVector>& Getordergroup() const

  Integer setOrderGroup(const std::vector<IntegerVector>& values)

  IntegerVector& getOrderGroupForEnergyBin(const Integer i) const

  Integer setOrderGroupForEnergyBin(const Integer i, const Integer value)

- const RealVector& getLowEnergy() const

```
        Integer setLowEnergy(const RealVector& values)

        Real getLowEnergyElement(const Integer i) const

        Integer setLowEnergyElement(const Integer i, const Real value)
```

- ```
  const RealVector& getHighEnergy() const
  ```

  ```
        Integer setHighEnergy(const RealVector& values)

        Real getHighEnergyElement(const Integer i) const

        Integer setHighEnergyElement(const Integer i, const Real value)
  ```

- ```
  const std::vector<RealVector>& getMatrix() const
  ```

  ```
        Integer setMatrix(const std::vector<RealVector>& values)

        RealVector& getMatrixForEnergyBin(const Integer i) const

        Integer setMatrixElement(const Integer i, const RealVector value)
  ```

- ```
  const RealVector& getChannelLowEnergy() const
  ```

  ```
        Integer setChannelLowEnergy(const RealVector& values)

        Real getChannelLowEnergyElement(const Integer i) const

        Integer setChannelLowEnergyElement(const Integer i, const Real value)
  ```

- ```
  const RealVector& getChannelHighEnergy() const
  ```

  ```
        Integer setChannelHighEnergy(const RealVector& values)

        Real getChannelHighEnergyElement(const Integer i) const

        Integer setChannelHighEnergyElement(const Integer i, const Real value)
  ```

- ```
  Real getAreaScaling() const
  ```

  ```
        void setAreaScaling(const Real value)
  ```

- Real getResponseThreshold() const

  void setResponseThreshold(const Real value)

- string getEnergyUnits() const

  void setEnergyUnits(const string value)

- string getRMFUnits() const

  void setRMFUnits(const string value)

- string getChannelType() const

  void setChannelType(const string value)

- string getTelescope() const

  void setTelescope(const string value)

- string getInstrument() const

  void setInstrument(const string value)

- string getDetector() const

  void setDetector(const string value)

- string getFilter() const

  void setFilter(const string value)

- string getRMFType() const

  void setRMFType(const string value)

- string getRMFExtensionName() const

  void setRMFExtensionName(const string value)

- string getEBDExtensionName() const

  void setEBDExtensionName(const string value)

- string getStokesParameter() const

  void setStokesParameter(const string value)

- string getStokesWeighting() const

  void setStokesWeighting(const string value)

3.3.4   Other rmf routines

- `rmf operator* (const rmf&, const arf&)`

  `rmf operator* (const arf&, const rmf&)`

  Multiply an rmf by an arf.

- `rmf operator* (const rmf&, const Real&)`

  `rmf operator* (const Real&, const rmf&)`

  Multiply an rmf by a factor.

- `rmf operator+ (const rmf&, const rmf&)`
  Add two rmfs.

- `IntegerVector RMFeboundsExtensionNumbers(const string filename,`
  `                                const string extname="")`
  Return a vector containing the extension numbers of all EBOUNDS extensions in the file.

- `IntegerVector RMFmatrixExtensionNumbers(const string filename,`
  `                                const string extname="")`
  Return a vector containing the extension numbers of all MATRIX extensions in the file.

- `IntegerVector RMFeboundsVersionNumbers(const string filename,`
  `                                const string extname="")`
  Return a vector containing the version numbers of all EBOUNDS extensions in the file.

- `IntegerVector RMFmatrixVersionNumbers(const string filename,`
  `                                const string extname="")`
  Return a vector containing the version numbers of all MATRIX extensions in the file.

- `void calcGaussResp(const Real width, const Real energ,`
  `                   const Real threshold,`
  `                   const RealVector& ChannelLowEnergy,`
  `                   const RealVector& ChannelHighEnergy,`
  `                   RealVector& ResponseVector)`
  Calculate the response vector for some energy give a gaussian width. The gaussian is assumed to be in the units of energy, ChannelLowEnergy and ChannelHighEnergy. The resulting response vector can be added into a response using addRow.

- `size\_t binarySearch(const Real energy, const RealVector& lowEnergy,`
  `                     const RealVector& highEnergy)`
  Return the index in the energy array containing the input energy using binary search.

3.3.5   rmft class private members

```
class rmft{
 private:

  Integer m_FirstChannel;              // First channel number

  IntegerVector m_NumberGroups;        // Number of response groups for this
                                       // channel bin

  std::vector<IntegerVector> m_FirstEnergyGroup; // First energy bin in group
  std::vector<IntegerVector> m_NumberEnergiesGroup;// Number energy bins in group
  std::vector<IntegerVector> m_OrderGroup; // Grating order of group

  RealVector m_LowEnergy;              // Start energy of bin
  RealVector m_HighEnergy;             // End energy of bin

  std::vector<RealVector> m_Matrix;    // Matrix elements

  RealVector m_ChannelLowEnergy;       // Start energy of channel
  RealVector m_ChannelHighEnergy;      // End energy of channel

  Real m_AreaScaling;                  // Value of EFFAREA keyword
  Real m_ResponseThreshold;            // Minimum value in response

  string m_EnergyUnits;                // Energy units
  string m_RMFUnits;                   // RMF units

  string m_ChannelType;                // Value of CHANTYPE keyword
  string m_Telescope;
  string m_Instrument;
  string m_Detector;
  string m_Filter;
  string m_RMFType;                    // HDUCLAS3 keyword in MATRIX extension
  string m_RMFExtensionName;           // EXTNAME keyword in MATRIX extension
  string m_EBDExtensionName;           // EXTNAME keyword in EBOUNDS extension
```

3.3.6   rmft class public methods

- `void load(rmf&)`

  Load object from a standard rmf object.

- `void update`

Update the FirstGroup and FirstElement arrays from NumberGroups and NumberEnergies-Group, respectively.

- `rmft& operator= (const rmft&)`

  Deep copy.

- `Real ElementValue(Integer Channel, Integer EnergyBin) const`

  Return the value for a particular channel and energy.

- `RealVector RowValues(Integer Channel) const`

  Return the array for a particular channel.

- `string disp() const`

  Display information about the object. - return as a string.

- `void clear()`

  Clear information from the object.

- `Integer NumberChannels() const`

  `Integer getNumberChannels() const`

  Number of spectrum channels

- `Integer NumberEnergyBins() const`

  `Integer getNumberEnergyBins() const`

  Number of response energies

- `Integer NumberTotalGroups() const`

  `Integer getNumberTotalGroups() const`

  Total number of response groups

- `Integer NumberTotalElements() const`

  `Integer getNumberTotalElements() const`

  Total number of response elements

3.3.7   Public rmft methods to get and set internal data

- `Integer getFirstChannel() const`

  `void setFirstChannel(const Integer value)`

- `const IntegerVector& getNumberGroups() const`

  `Integer setNumberGroups(const IntegerVector& values)`

  `Integer getNumberGroupsElement(const Integer i) const`

  `Integer setNumberGroupsElement(const Integer i, const Integer value)`

- `const std::vector<IntegerVector>& getFirstEnergyGroup() const`

  `Integer setFirstEnergyGroup(const std::vector<IntegerVector>& values)`

  `IntegerVector getFirstEnergyGroupForChannel(const Integer i) const`

  `Integer setFirstEnergyGroupForChannel(const Integer i,`
  `                                      const IntegerVector value)`

- `const std::vector<IntegerVector>& getNumberEnergiesGroup() const`

  `Integer setNumberEnergiesGroup(const std::vector<IntegerVector>& values)`

  `IntegerVector& getNumberEnergiesGroupForChannel(const Integer i) const`

  `Integer setNumberEnergiesGroupForChannel(const Integer i,`
  `                                         const IntegerVector value)`

- `const std::vector<IntegerVector>& getOrderGroup() const`

  `Integer setOrderGroup(const std::vector<IntegerVector>& values)`

  `IntegerVector getOrderGroupForChannel(const Integer i) const`

  `Integer setOrderGroupForChannel(const Integer i,`
  `                                const IntegerVector value)`

- `const RealVector& getLowEnergy() const`

```
        Integer setLowEnergy(const RealVector& values)

        Real getLowEnergyElement(const Integer i) const

        Integer setLowEnergyElement(const Integer i, const Real value)
```

- ```
  const RealVector& getHighEnergy() const
  ```

  ```
        Integer setHighEnergy(const RealVector& values)

        Real getHighEnergyElement(const Integer i) const

        Integer setHighEnergyElement(const Integer i, const Real value)
  ```

- ```
  const std::vector<RealVector>& getMatrix() const
  ```

  ```
        Integer setMatrix(const std::vector<RealVector>& values)

        RealVector getMatrixForChannel(const Integer i) const

        Integer setMatrixForChannel(const Integer i, const RealVector value)
  ```

- ```
  const RealVector& getChannelLowEnergy() const
  ```

  ```
        Integer setChannelLowEnergy(const RealVector& values)

        Real getChannelLowEnergyElement(const Integer i) const

        Integer setChannelLowEnergyElement(const Integer i, const Real value)
  ```

- ```
  const RealVector& getChannelHighEnergy() const
  ```

  ```
        Integer setChannelHighEnergy(const RealVector& values)

        Real getChannelHighEnergyElement(const Integer i) const

        Integer setChannelHighEnergyElement(const Integer i, const Real value)
  ```

- ```
  Real getAreaScaling() const
  ```

  ```
        void setAreaScaling(const Real value)
  ```

- Real getResponseThreshold() const

  void setResponseThreshold(const Real value)

- string getEnergyUnits() const

  void setEnergyUnits(const string value)

- string getRMFUnits() const

  void setRMFUnits(const string value)

- string getChannelType() const

  void setChannelType(const string value)

- string getTelescope() const

  void setTelescope(const string value)

- string getInstrument() const

  void setInstrument(const string value)

- string getDetector() const

  void setDetector(const string value)

- string getFilter() const

  void setFilter(const string value)

- string getRMFType() const

  void setRMFType(const string value)

- string getRMFExtensionName() const

  void setRMFExtensionName(const string value)

- string getEBDExtensionName() const

  void setEBDExtensionName(const string value)

3.3.8   arf class private members

```
class arf{
 private:

  RealVector m_LowEnergy;   // Start energy of bin
  RealVector m_HighEnergy;  // End energy of bin

  RealVector m_EffArea;     // Effective areas

  string m_EnergyUnits;     // Units for energies
  string m_arfUnits;        // Units for effective areas

  string m_Telescope;
  string m_Instrument;
  string m_Detector;
  string m_Filter;

  string m_StokesParameter; // For spectropolarimetry
                                (e.g. 'Ixx', 'xQx', 'xxU') //
  string m_StokesWeighting; // Stokes parameter weighting scheme

  string m_ExtensionName;   // EXTNAME keyword in SPECRESP extension
```

3.3.9   arf class public methods

- `arf()`

  `arf(const arf& a)`

  `arf(const string filename, const Integer ARFnumber = 1,`
  `    const Integer RowNumber = 1)`

  Constructors. The first is the default constructor, the other two are constructor versions of the copy and read methods described below.

- `~arf()`

  Default destructor.

- `Integer read(string filename, Integer ARFnumber = 1,`
  `              Integer RowNumber = 0)`

  Read file into an object. If ARFnumber is given read from the SPECRESP extension with EXTVER=ARFnumber. The third option is to read an arf from the RowNumber row of a type II file.

- `arf& copy(const arf&)`

  `arf& operator= (const arf&)`

  Deep copy.

- `string disp() const`

  Display information about the arf. - return as a string.

- `void clear()`

  Clear information from the arf.

- `string check() const`

  Check completeness and consistency of information in the arf, if there is a problem then return diagnostic in string.

- `rebin(grouping&)`

  Rebin the arf using the input grouping.

- `remapEnergies(const RealVector& targetEnergies)`

  Remap the arf onto an arbitrary energy grid (which should be coarser than the original).

- `Integer write(const string filename, const string sourceFilename="")`

  Write arf as type I file. Note that if the output filename exists and already has a SPECRESP extension then this method will write an additional SPECRESP extension provided that no sourceFilename is given. If sourceFilename is given then the primary HDU and any other extra HDUs are copied into the output file as well as any extra keywords or columns in the spectrum extension.

- `arf& operator*=(const Real&)`

  Multiply by a constant.

- `arf& operator+=(const arf&)`

  Add another arf.

- `Integer checkCompatibility(const arf&) const`

  Check compatibility with another arf.

- `Integer convertUnits()`

  Convert the energies on which the arf is calculated from the current units to keV. The allowed units are keV, MeV, GeV, Hz, angstrom, cm, micron, nm.

- `Integer NumberEnergyBins() const`

  `Integer getNumberEnergyBins() const`

  Return size of RealVectors.

3.3.10   Public arf methods to get and set internal data

- `const RealVector& getLowEnergy() const`

  `Integer setLowEnergy(const RealVector& values)`

  `Real getLowEnergyElement(const Integer i) const`

  `Integer setLowEnergyElement(const Integer i, const Real value)`

- `const RealVector& getHighEnergy() const`

  `Integer setHighEnergy(const RealVector& values)`

  `Real getHighEnergyElement(const Integer i) const`

  `Integer setHighEnergyElement(const Integer i, const Real value)`

- `const RealVector& getEffArea() const`

  `Integer setEffArea(const RealVector& values)`

  `Real getEffAreaElement(const Integer i) const`

  `Integer setEffAreaElement(const Integer i, const Real value)`

- `string getEnergyUnits() const`

  `void setEnergyUnits(const string& value)`

- `string getEffAreaUnits() const`

  `void setEffAreaUnits(const string& value)`

- `string getTelescope() const`

  `void setTelescope(const string& value)`

- `string getInstrument() const`

  `void setInstrument(const string& value)`

- `string getDetector() const`

```
    void setDetector(const string& value)
```

- `string getFilter() const`

  ```
  void setFilter(const string& value)
  ```

- `string getStokesParameter() const`

  ```
  void setStokesParameter(const string& value)
  ```

- `string getStokesWeighting() const`

  ```
  void setStokesWeighting(const string& value)
  ```

- `string getExtensionName() const`

  ```
  void setExtensionName(const string& value)
  ```

3.3.11   Other arf routines

- `arf operator+ (const arf&, const arf&)`
  Add two arfs.

- `arf operator* (const arf&, const Real&)`

  ```
  arf operator* (const Real&, const arf&)
  ```

  Multiply by a constant.

- `IntegerVector ARFExtensionNumbers(const string filename,`
                                   `const string extname="")`
  Return a vector of extension numbers of all SPECRESP extensions.

- `IntegerVector ARFVersionNumbers(const string filename,`
                                 `const string extname="")`
  Return a vector of version numbers of all SPECRESP extensions.

- `Integer ARFtype(const string filename,`
                 `const Integer ARFnumber, Integer& Status)`

  ```
  Integer ARFtype(const string filename,
                  const Integer ARFnumber)
  ```

  Return the type of a SPECRESP extension.

3.3.12   arfII class private members

```
class arfII{
 private:

  std::vector<arf> m_arfs;              // vector of arf objects
```

3.3.13   arfII class public methods

- `arfII()`

  `arfII(const arfII& a)`

  `arfII(const string filename, const Integer ARFnumber = 1,`
  `        const IntegerVector& RowNumber = IntegerVector())`

  Constructors. The first is the default constructor and the other two are constructor versions of the copy and read methods described below.

- `~arfII()`

  Default destructor.

- `Integer read(string filename, Integer ARFnumber = 1,`
  `                IntegerVector& RowNumber = IntegerVector())`

  Read an ARF type II file into an object. If ARFnumber is given then read from the SPECRESP extension with EXTVER=ARFnumber. If the RowNumber array is given then read those in the extension otherwise read all the arfs.

- `arfII& copy(const arfII&)`

  `arfII& operator= (const arfII&)`

  Deep copy.

- `arf get(Integer number) const`

  Get arf object (counts from zero).

- `void push(arf ea)`

  Push arf object into arfII object

- `string disp() const`

  Display information about the ARFs. - return as a string.

- `void clear()`

  Clear information from the ARFs.

- `string check() const`

  Check completeness and consistency of information in the arfs, if there is a problem then return diagnostic in string.

- `Integer write(const string filename, const string sourceFilename="")`

  Write ARFs as type II file. Note that if the output filename exists and already has a SPECRESP extension then this method will write an additional SPECRESP extension provided no sourceFilename is given. If sourceFilename is given then the primary HDU and any other extra HDUs are copied into the output file as well as any extra keywords or columns in the spectrum extension.

- `Integer NumberARFs() const`

- `Integer getNumberARFs() const`

  Return the number of ARFs in the object.

3.3.14 Public arfII methods to get and set internal data

- `const std::vector<arf>& getarfs() const`

  `Integer setarfs(const std::vector<arf>& values)`

  `arf getarfsElement(const Integer i) const`

  `Integer setarfsElement(const Integer i, const arf& value)`

3.3.15 Other arfII routines

- `Integer NumberofARFs(string filename, Integer HDUumber)`

  `Integer NumberofARFs(string filename, Integer HDUumber,`
  `                     Integer& Status)`

  Return the number of ARFS in the type II SPECRESP extension.

3.4   Table Models

3.4.1   Introduction and example

The table model file is used in xspec to provide grids of model calculations on which to interpolate when fitting a model to data. The table class can be used to create these files. The example code below sets up a grid with two parameters.

```
#include "table.h"

using namespace std;
using namespace heasp;

int main(int argc, char* argv[])
{

  table test;

  // set table descriptors and the energy array

  test.setModelName("Test");
  test.setModelUnits(" ");
  test.setisRedshift(true);
  test.setisAdditive(true);
  test.setisError(false);

  RealVector energy(100);
  for (size_t i=0; i<100; i++) energy[i] = 0.1+i*0.1;
  test.setEnergies(energy);
  test.setEnergyUnits("keV");

  test.setNumIntParams(2);
  test.setNumAddParams(0);


  // define first parameter and give it 11 values ranging from
  // 0.0 to 2.0 in steps of 0.2.

  tableParameter testpar;

  testpar.setName("param1");
  testpar.setInterpolationMethod(0);
  testpar.setInitialValue(1.0);
  testpar.setDelta(0.1);
  testpar.setMinimum(0.0);
```

```
  testpar.setBottom(0.0);
  testpar.setTop(2.0);
  testpar.setMaximum(2.0);

  RealVector tabVals(11);
  for (size_t i=0; i<11; i++) tabVals[i] = 0.2*i;
  testpar.setTabulatedValues(tabVals);

  // and push it onto the vector of parameters

  test.pushParameters(testpar);

  // define the second parameter and give it 5 values ranging from
  // 4.6 to 5.4 in steps of 0.2. Use the load constructor as an
  // illustration in this case

  tabVals.resize(5);
  for (size_t i=0; i<5; i++) tabVals[i] = 4.6+0.2*i;

  tableParameter testpar2("param2", 0, 5.0, 0.1, 4.6, 4.6, 5.4, 5.4, tabVals);

  // and push it onto the vector of parameters

  test.pushParameters(testpar);

  // now set up the spectra. these are arbitrarily calculated, in a real program
  // this step would read a file or call a routine.

  RealVector flux(99);
  tabVals.resize(2);

  for (size_t i1=0; i1<11; i1++) {
    for (size_t i2=0; i2<5; i2++) {
      tabVals[0] = 0.2*i1;
      tabVals[1] = 4.6+0.2*i2;
      for (size_t j=0; j<99; j++) {
        flux[j] = tabVals[0]+10*tabVals[1];
      }
      tableSpectrum testspec(tabVals, flux);
      test.Spectra.push_back(testspec);
    }
  }

  // now write out the table.
```

```
  test.write("test.mod");

  exit(0);
}
```

3.4.2   table class private members

```
class table{
 private:

  vector <tableParameter> m_Parameters; // Parameter information
  vector <tableSpectrum> m_Spectra;     // Tabulated model spectra
  string m_ModelName;                   // Name to use in xspec
  string m_ModelUnits;                  // Units (not used at present)
  int m_NumIntParams;                   // Dimension of interpolation grid
  int m_NumAddParams;                   // Number of additional parameters
  bool m_isError;                       // If true then model errors included
  bool m_isRedshift;                    // If true include redshift
  bool m_isAdditive;                    // If true model is additive
  bool m_isEscale;                      // If true include an energy scaling
  RealVector m_Energies;                // Energy bins on which model is
                                        // calculated. The size should be one
                                        // larger than that of the spectrum
                                        // array
  string m_EnergyUnits;                 // Units of energy bins
  Real m_LowEnergyLimit;                // Value to use for energies
                                        // below those tabulated
  Real m_HighEnergyLimi;                // Value to use for energies
                                        // above those tabulated
  string m_Filename;                    // Name of file from which table was read
  StringVector m_FiltExps;              // Filter expressions
```

3.4.3   table class public methods

- `table()`

  `table(const table& a)`

  `table(const string infilename, bool loadAll = true)`

  `table(const string modName, const string modUnits,`
  `      const Integer nInt, const Integer nAdd, const bool isz,`
  `      const bool isAdd, const string eUnits, const Real lowElim,`
  `      const Real highElim, const string filename,`

```
            const RealVector& energies = RealVector(),
            const std::vector<tableParameter>& paramObjects
                        = std::vector<tableParameter>(),
            const std::vector<tableSpectrum>& spectrumObjects
                        = std::vector<tableSpectrum>(),
            const bool isEscale = false, const bool isErr = false)
```

Constructors. The first is the default constructor and the others are constructor versions of the copy, read, and load methods described below.

- ~table()

  Default destructor.

- void load(const string modName, const string modUnits, const Integer nInt,
            const Integer nAdd, const bool isz, const bool isAdd,
            const string eUnits,
            const Real lowElim, const Real highElim,
            const string filename,
            const RealVector& energies = RealVector(),
            const std::vector<tableParameter>& paramObjects
                                    = std::vector<tableParameter>(),
            const std::vector<tableSpectrum>& spectrumObjects
                                    = std::vector<tableSpectrum>(),
            const bool isEscale = false, const bool isErr = false)

  Loads required information into the table object.

- Integer read(const string filename, bool loadAll = true)

  Read the table model object from filename. If loadAll is false then don't actually read in the Spectra but set up the objects.

- template <class T> Integer readSpectra(T& spectrumList)

  Read in the listed Spectra if they have not already been. The class T can be IntegerVector, std::valarray<Integer>, std::vector<size_t>, or std::valarray<size_t>.

- void pushParameter(tableParameter paramObject)

  Push a table Parameter object

- void pushSpectrum(tableSpectrum spectrumObject)

  Push a table spectrum object

- tableParameter getParameter(const Integer Number) const

  Get a table Parameter object

- tableSpectrum getSpectrum(const Integer Number) const

  Get a table Spectrum object

- `table& copy(const table& a)`

  `table& operator= (const table& a)`

  Deep copy.

- `string disp() const`

  Display information about the table - return as a string.

- `void clear()`

  `void clearParameters()`

  `void clearSpectra()`

  Clear contents of the table, clear the parameters only, and clear the spectra only.

- `string check() const`

  Check completeness and consistency of information in table, if there is a problem then return diagnostic in string.

- `Integer convertUnits()`

  Convert the table to standard units (keV and $ph/cm^2/s$). The original units should be set using setEnergyUnits and setModelUnits. The allowed model units are: $ph/cm^2/s/X$, where X is one of MeV, GeV, Hz, A, cm, um, nm; $ergs/cm^2/s$; $ergs/cm^2/s/X$, where X is one of Hz, A, cm, um, nm; Jy. The allowed energy units are keV, GeV, Hz, angstrom, cm, micron or nm.

- `void reverseRows()`

  Reverse the spectra if energies are not increasing (this can occur after using convertUnits to convert from wavelength).

- `write(string filename)`

  Write to a FITS file

- `template <class T> Integer getValues(const T& parameterValues,`
  `                                      const Real minEnergy,`
  `                                      const Real maxEnergy,`
  `                                      T& tableEnergyBins,`
  `                                      T& tableValues,`
  `                                      T& tableErrors)`

  Interpolate on the table using the input parameters in parameterValues and return the resulting spectrum between the input minimum and maximum energies. The actual energies on which the spectrum is evaluated are returned in tableEnergyBins with the spectrum values as tableValues and, if included, their errors as tableErrors. The class T can be either RealVector or RealArray.

- ```
  template <class T> Integer getValues(const T& parameterValues,
                                       const std::map<string,Real>& xfltInfo,
                                       const Real minEnergy,
                                       const Real maxEnergy,
                                       T& tableEnergyBins,
                                       T& tableValues,
                                       T& tableErrors)
  ```

  Interpolate on the table using the input parameters in parameterValues and if there are multiple model spectra per parameter grid point selected the correct one using the xfltInfo. Return the resulting spectrum between the input minimum and maximum energies. The actual energies on which the spectrum is evaluated are returned in tableEnergyBins with the spectrum values as tableValues and, if included, their errors as tableErrors. The class T can be either RealVector or RealArray.

- ```
  Integer NumberParameters() const

  Integer getNumberParameters() const
  ```

  Return the number of parameters in the table.

- ```
  Integer NumberSpectra() const

  Integer getNumberSpectra() const
  ```

  Return the total number of grid points in the table.

- ```
  Integer NumberEnergies() const

  Integer getNumberEnergies() const
  ```

  Return the number of energies in the spectra.

3.4.4 Public table methods to get and set internal data

- ```
  const std::vector<tableParameter>& getParameters() const

  Integer setParameters(const std::vector<tableParameter>& values)

  tableParameter getParametersElement(const Integer i) const

  Integer setParametersElement(const Integer i, const tableParameter& value)
  ```

- ```
  const std::vector<tableSpectrum>& getSpectra() const
  ```

```
Integer setSpectra(const std::vector<tableSpectrum>& values)

tableSpectrum getSpectraElement(const Integer i) const

Integer setSpectraElement(const Integer i, const tableSpectrum& value)
```

- ```
  string getModelName() const
  ```

  ```
  void setModelName(const string value)
  ```

- ```
  string getModelUnits() const
  ```

  ```
  void setModelUnits(const string value)
  ```

- ```
  Integer getNumIntParams() const
  ```

  ```
  void setNumIntParams(const Integer value)
  ```

- ```
  Integer getNumAddParams() const
  ```

  ```
  void setNumAddParams(const Integer value)
  ```

- ```
  bool getisError() const
  ```

  ```
  void setisError(const bool value)
  ```

- ```
  bool getisRedshift() const
  ```

  ```
  void setisRedshift(const bool value)
  ```

- ```
  bool getisAdditive() const
  ```

  ```
  void setisAdditive(const bool value)
  ```

- ```
  bool getisEscale() const
  ```

  ```
  void setisEscale(const bool value)
  ```

- ```
  const RealVector& getEnergies() const
  ```

  ```
  Integer setEnergies(const RealVector& values)
  ```

- Real getEnergiesElement(const Integer i) const

    Integer setEnergiesElement(const Integer i, const Real value)

- string getEnergyUnits() const

    void setEnergyUnits(const string value)

- Real getLowEnergyLimit() const

    void setLowEnergyLimit(const Real
         value)

- Real getHighEnergyLimit() const

    void setHighEnergyLimit(const Real value)

- string getFilename() const

    void setFilename(const string value)

- StringVector getFiltExps() const

    void setFiltExps(const StringVector values)

3.4.5   tableParameter class private members

```
class tableParameter{
 private:

  string m_Name;                 // Parameter name
  int m_InterpolationMethod;     // 0==linear, 1==log,
                                 // -1==additional (non-interp)
  Real m_InitialValue;           // Initial value for fit
  Real m_Delta;                  // Delta for fit
  Real m_Minimum;                // Hard lower-limit
                                 // (should correspond to first tabulated value)
  Real m_Bottom;                 // Soft lower-limit
  Real m_Top;                    // Soft upper-limit
  Real m_Maximum;                // Hard upper-limit
                                 // (should correspond to last tabulated value)
  RealVector m_TabulatedValues;  // Tabulated parameter values
```

3.4.6   tableParameter class public methods

- `tableParameter()`

  ```
  tableParameter(const string name, const string units,
                 const Integer interp, const Real initial,
                 const Real delta, const Real min, const Real bot,
                 const Real top, const Real max,
                 const RealVector& values = RealVector())
  ```

  ```
  tableParameter(const string name, const Integer interp, const Real initial,
                 const Real delta, const Real min, const Real bot,
                 const Real top, const Real max,
                 const RealVector& values = RealVector())
  ```

  Constructors. The first is the default constructor and the second and third are constructor versions of the load methods described below. The third is retained for backwards compatibility without the units string.

- `~tableParameter()`

  Default destructor.

- `void load(const string name, const string units,`
  `           const Integer interp, const Real initial,`
  `           const Real delta, const Real min, const Real bot, const Real top,`
  `           const Real max, const RealVector& values = RealVector())`

  ```
  void load(const string name, const Integer interp, const Real initial,
            const Real delta, const Real min, const Real bot, const Real top,
            const Real max, const RealVector& values = RealVector())
  ```

  Load required content into object. The second version is retained for backwards compatibility without the units string.

- `string disp() const`

  Display information about the table parameter - return as a string.

- `void clear()`

  Clear contents of the table parameter

- `Integer NumberTabulatedValues() const`

  ```
  Integer getNumberTabulatedValues() const
  ```

  Return the number of tabulated values for the parameter.

3.4.7   Public tableParameter methods to get and set internal data

- ```
  string getName() const
  ```

  ```
  void setName(const string value)
  ```

- ```
  string getUnits() const
  ```

  ```
  void setUnits(const string value)
  ```

- ```
  Integer getInterpolationMethod() const
  ```

  ```
  void setInterpolationMethod(const Integer value)
  ```

- ```
  Real getInitialValue() const
  ```

  ```
  void setInitialValue(const Real value)
  ```

- ```
  Real getDelta() const
  ```

  ```
  void setDelta(const Real value)
  ```

- ```
  Real getMinimum() const
  ```

  ```
  void setMinimum(const Real value)
  ```

- ```
  Real getBottom() const
  ```

  ```
  void setBottom(const Real value)
  ```

- ```
  Real getTop() const
  ```

  ```
  void setTop(const Real value)
  ```

- ```
  Real getMaximum() const
  ```

  ```
  void setMaximum(const Real value)
  ```

- ```
  const RealVector& getTabulatedValues() const
  ```

  ```
  Integer setTabulatedValues(const RealVector& values)
  ```

- ```
  Real getTabulatedValuesElement(const Integer i) const
  ```

  ```
  Integer setTabulatedValuesElement(const Integer i, const Real value)
  ```

3.4.8   tableSpectrum class private members

```
class tableSpectrum{
 private:

  RealVector m_Flux;                      // Model flux
  RealVector m_FluxError;                 // Error on Model flux
  RealVector m_ParameterValues;           // Parameter values for this spectrum
  std::vector<RealVector> m_addFlux;      // Model fluxes for any additional
                                          // parameters
  std::vector<RealVector> m_addFluxError; // Error on model fluxes for any
                                          // additional parameters
```

3.4.9   tableSpectrum class public methods

- tableSpectrum()

```
  tableSpectrum(const RealVector& parVals, const RealVector& flux,
               const std::vector<RealVector>& addFlux
                  = std::vector<RealVector>(),
               const RealVector& fluxErr = RealVector(),
               const std::vector<RealVector>& addFluxErr
                  = std::vector<RealVector>())
```

  Constructors. The first is the default constructor and the second is a constructor version of
  the load method described below.

- ~tableSpectrum()

  Default destructor.

- void load(const RealVector& parVals, const RealVector& flux,
           const std::vector<RealVector>& addFlux
              = std::vector<RealVector>(),
           const RealVector& fluxErr = RealVector(),
           const std::vector<RealVector>& addFluxErr
              = std::vector<RealVector>())

  Method to load required content into object.

- void pushaddFlux(RealVector)

  Push an additional parameter spectrum.

- RealVector getaddFlux(Integer Number)

  Get an additional parameter spectrum.

- `string disp() const`

  Display information about the table spectrum - return as a string.

- `void clear()`

  Clear contents of the table spectrum.

- `Integer NumberFluxes() const`

  `Integer getNumberFluxes() const`

  Return the size of the flux array.

- `Integer NumberFluxErrors() const`

  `Integer getNumberFluxErrors() const`

  Return the size of the flux error array.

- `Integer NumberParameterValues() const`

  `Integer getNumberParameterValues() const`

  Return the size of the array of parameter values.

- `Integer NumberAdditiveParameters() const`

  `Integer getNumberAdditiveParameters() const`

  Return the number of additive parameters.

- `Integer NumberAdditiveFluxes(const Integer iaddParam) const`

  `Integer getNumberAdditiveFluxes(const Integer iaddParam) const`

  Return the size of the flux array for this additive parameter.

- `Integer NumberAdditiveFluxErrors(const Integer iaddParam) const`

  `Integer getNumberAdditiveFluxErrors(const Integer iaddParam) const`

  Return the size of the flux error array for this additive parameter.

3.4.10   Public tableSpectrum methods to get and set internal data

- `const RealVector& getFlux() const`

  `Integer setFlux(const RealVector& values)`

  `Real getFluxElement(const Integer i) const`

  `Integer setFluxElement(const Integer i, const Real value)`

- `const RealVector& getFluxError() const`

  `Integer setFluxError(const RealVector& values)`

  `Real getFluxErrorElement(const Integer i) const`

  `Integer setFluxErrorElement(const Integer i, const Real value)`

- `const RealVector& getParameterValues() const`

  `Integer setParameterValues(const RealVector& values)`

  `Real getParameterValuesElement(const Integer i) const`

  `Integer setParameterValuesElement(const Integer i, const Real value)`

- `const std::vector<RealVector>& getaddFlux() const`

  `Integer setaddFlux(const std::vector<RealVector>& values)`

  `const RealVector& getaddFluxElement(const Integer iparam) const`

  `Integer setaddFluxElement(const Integer iparam,`
  `                          const RealVector& valueparam)`

  `Real getaddFluxElementElement(const Integer iparam, const Integer i) const`

  `Integer setaddFluxElementElement(const Integer iparam, const Integer i,`
  `                                 const Real value)`

- `const std::vector<RealVector>& getaddFluxError() const`

  `Integer setaddFluxError(const std::vector<RealVector>& values)`

```
    const RealVector& getaddFluxErrorElement(const Integer iparam) const


    Integer setaddFluxErrorElement(const Integer iparam,
                                   const RealVector& valueparam)


    Real getaddFluxErrorElementElement(const Integer iparam,
                                       const Integer i) const


    Integer setaddFluxErrorElementElement(const Integer iparam,
                                          const Integer i,
                                          const Real value)
```

3.5 Grouping

3.5.1 grouping class private members

```
class grouping{
 private:

  IntegerVector m_groupingFlag; // Grouping flag: 1=start of bin,
                                // -1=continuation of bin
  IntegerVector m_qualityFlag;  // Quality flag: 0=good
```

3.5.2 grouping class public methods

- `grouping()`

  `grouping(IntegerVector inGroup, IntegerVector inQual)`

  Constructors. First is the default and the second loads from an integer arrays of grouping and quality values.

- `~grouping()`

  Default destructor.

- `string disp() const`

  Display grouping information. - return as a string.

- `void clear()`

  Clear grouping information.

- ```
  Integer read(string filename, const Integer Number,
                const Integer First)
  ```

  Read from an ascii file of grouping factors. Each line of the file should have three numbers, the start bin, end bin, and grouping factor. The input bin numbers start at First and there are Number in total.

- ```
  void load(const Integer BinFactor, const Integer Number)
  ```

  Set the grouping flags for Number bins with a binning factor of BinFactor.

- ```
  Integer load(const IntegerVector& StartBin, const IntegerVector& EndBin,
                const IntegerVector& BinFactor, const Integer Number,
                const Integer First)
  ```

  Set grouping flags from an array of binning information in the StartBin, EndBin and BinFactor arrays. The input bin numbers start at First and there are Number in total.

- ```
  Integer loadFromVector(const IntegerVector& QualVector,
                          const IntegerVector GroupVector)
  ```

  Set grouping from quality and grouping vectors from a pha object.

- ```
  template <class T>Integer loadMin(const T Minimum,
                                     const std::vector<T>& values)


  template <class T>Integer loadMin(const T Minimum,
                                     const std::vector<T>& Values,
                                     const Integer StartChannel,
                                     const Integer EndChannel)
  ```

  Set the grouping flags based on requiring a minimum value in each grouped bin when the values in each channel are those in the input Values array. Optionally only do this between the StartChannel and EndChannel channels. If StartChannel and EndChannel are identically zero then they are not used. Any channels not included in a complete bin have grouping flag set to 1 and quality flag set to 2.

- ```
   Integer loadOptimal(const RealVector& FWHM,
                        const IntegerVector& Counts,
                        const Integer StartChannel,
                        const Integer EndChannel,
                        const Integer minCounts)
  ```

  Sets grouping using optimal binning following Kaastra & Bleeker (2016, A&A 587, 151) based on the instrument FWHM. This assumes that the FWHM is that for each energy. If minCounts is non-zero then all bins must also have at least that many counts. Apply binning only between StartChannel and EndChannel unless both are zero. Any channels not included in a complete bin have grouping flag set to 1 and quality flag set to 2.

- `Integer loadOptimalEnergy(const RealVector& FWHM,`
                          `const IntegerVector& Counts)`

  Sets grouping using optimal binning on the energy axis following Kaastra & Bleeker (2016, A&A 587, 151) based on the instrument FWHM. This assumes that the FWHM is that for each energy.

- `bool newBin(const Integer I) const`

  Return whether I is the start of a bin.

- `bool inBin(const Integer I) const`

  Return whether I is in a bin.

- `Integer size() const`

  Return number of bins in grouping object.

- `bool isGroupingOrQuality() const`

  Return true if any grouping flags are not 1 or any quality flags are not 0.

3.5.3 Public grouping methods to get and set internal data

- `const IntegerVector& getGroupingFlag() const`

  `Integer setGroupingFlag(const IntegerVector& values)`

  `const Integer getGroupingFlagElement(const Integer i) const`

  `Integer setGroupingFlagElement(const Integer i, const Integer value)`

- `const IntegerVector& getQualityFlag() const`

  `Integer setQualityFlag(const IntegerVector& values)`

  `const Integer getQualityFlagElement(const Integer i) const`

  `Integer setQualityFlagElement(const Integer i, const Integer value)`

3.5.4 Other grouping routines

- `template <class T> void GroupBin(const std::vector<T>& inArray,`
                              `const Integer mode,`
                              `const grouping& GroupInfo,`
                              `std::vector<T>& outArray)`

```
template <class T> void GroupBin(const std::valarray<T>& inArray,
                                 const Integer mode,
                                 const grouping& GroupInfo,
                                 std::valarray<T>& outArray)
```

This routine applies GroupInfo to the input inArray to create the output outArray. The behavior is determined by mode which can take five values: SumMode which adds the contents of all bins in a group; SumQuadMode which adds the bins in quadrature; MeanMode which returns the arithmetic mean of the all bins in a group; FirstEltMode which returns the value of the first bin in each group; LastEltMode which returns the value of the last bin in each group.

- `Integer readBinFactors(string filename, IntegerVector& StartBin,`
  `IntegerVector& EndBin, IntegerVector& BinFactor)`

Read a file containing grouping information and place into the arrays StartBin, EndBin and BinFactor. Each line of the file should have three numbers, the start bin, end bin, and grouping factor.

3.6   Utility routines

These C++ routines can be found in SPutils.h and SPutils.cxx and are not available from Python.

- `void SPreadColUnits(CCfits::ExtHDU&, const string, string&)`
  Read the units associated with a column.

- `void SPwriteColUnits(CCfits::Table&, const string, const string)`
  Write the units associated with a column.

- `string SPstringTform(const StringVector& Data)`
  Returns the tform string for the longest string in the input vector.

- `Integer SPcopyHDUs(const string infile, const string outfile)`
  Copy from infile to outfile all HDUs which are not manipulated by this library.

- `Integer SPcopySingleHDU(const string infile, const int HDUnummber,`
  `const string outfile)`
  Copy a single HDU from infile to outfile.

- `Integer SPcopyCols(const string infile, const string outfile,`
  `const string HDUname, const Integer HDUnumber = 1);`
  Copy non-critical columns from infile to outfile for the HDUnumber instance of the HDUname HDU.

- ```
  Integer SPcopyKeys(const string infile, const string outfile,
                     const string HDUname, const Integer HDUnumber = 1)

  Integer SPcopyKeys(const string infile, const string outfile,
                     string HDUname, const string outHDUname,
                     const Integer HDUnumber = 1,
                     const Integer outHDUnumber = 1 )
  ```

  Copy non-critical keywords from infile to outfile for HDUname extension with EXTVER HDUnumber.

- ```
  Integer SPsplitFilename(const string infile, string& filename,
                          Integer& extNumber)
  ```

  Split an input filename into the name and extension number in the case when the input filename includes an [#] or a +#.

- ```
  Integer SPwriteCreator(const string filename, const string HDUname,
                         const string creator, const Integer HDUnumber = 1)
  ```

  Write the creating program and version id string into the CREATOR keyword in the specified file.

- ```
  IntegerVector SPfindExtensions(const string filename,
                                 const string keyname, const string value,
                                 Integer& Status);
  ```

  Find the numbers of any extensions containing keyword keyname=keyvalue

- ```
  bool SPisValidXUnits(const string xUnits)
  ```

  Checks whether xUnits are supported.

- ```
  Integer SPcalcXfactor(const string xUnits, bool& isWave, Real& xFactor)
  ```

  Calculates the conversion factor for xUnits energies/wavelength to keV. If xUnits is wavelength then returns isWave as true.

- ```
  bool SPisValidYUnits(const string yUnits)
  ```

  Checks whether yUnits are supported.

- ```
  Integer SPcalcYfactor(const string yUnits, bool& isEnergy, bool& perWave,
                        Real& yFactor)
  ```

  Calculates the conversion factor for yUnits to $ph/cm^2/s$. If yUnits contains an energy numerator (eg ergs) then returns isEnergy as true. If yUnits contains a wavelength denominator (eg micron) returns perWave as true.

- ```
  void SPreportError(const int errorNumber,
                     const string optionalString = ‘‘’’)
  ```

  Add to the error stack errorNumber along with an optional string containing additional information.

- `string SPgetErrorStack()`

  Write the entire error stack into a string.

- `string SPclearErrorStack()`

  Clear the error stack.

- `StringVector SPreadStrings(const string& filename)`

  Read a text file and place each row into its own element of a StringVector.

- `StringVector SPtokenize(const string & str, const string & delim)`

  Divide a string into substrings delimited using delim.

- `string SPmatchString(const string& str,`
  `                     const StringVector& strArray, int& nmatch)`

  Partial match a string from a vector of strings.

- `string SPtrimString(const string& str)`

  Remove leading and trailing blanks from a string.

- `bool SPstring2Real(const StringVector& str,`
  `                   RealVector& value)`

  `bool SPstring2Real(const string& str, Real& value)`

  Convert a string or vector of strings into a Real or vector of Reals.

- `bool SPstring2double(const StringVector& str,`
  `                     std::vector<double>& value)`

  `bool SPstring2double(const string& str, double& value)`

  Convert a string or vector of strings into a double or vector of doubles.

- `bool SPstring2float(const StringVector& str,`
  `                    std::vector<float>& value)`

  `bool SPstring2float(const string& str, float& value)`

  Convert a string or vector of strings into a float or vector of floats.

- `bool SPstring2Integer(const StringVector& str,`
  `                      IntegerVector& value)`

  `bool SPstring2Integer(const string& str, Integer& value)`

Convert a string or vector of strings into an Integer or vector of Integers.

- ```
  bool SPrangeString2IntegerList(const string& str, const string& delim1,
                                 const string& delim2, IntegerVector& list)
  ```

Convert a string of delimited range specifications of form n1"delim2"n2 meaning n1 to n2 inclusive or n3 meaning just n3. Ranges are delimited using delim1.

- ```
  void SPcalcShift(const RealVector& Low,
                   const RealVector& High, const IntegerVector& vStart,
                   const IntegerVector& vEnd, const RealVector& vShift,
                   const RealVector& vFactor,
                   std::vector<std::vector<size\_t> >& fromIndex,
                   std::vector<RealVector>& Fraction)
  ```

Calculate factors for shifting an array

- ```
  template <class T> void SPfind(const T& array, const Real& target,
                                 Integer& index)
  ```

Find the position in the array of the target value. If array is increasing the index is the last element in array $\leq$ target; if target $<$ array[0] then index = -1; if target $\geq$ array[array.size()-1] then index = array.size()-1. If array is decreasing the index is the last element in array $\geq$ target; if target $>$ array[0] then index = -1; if target $\leq$ array[array.size()-1] then index = array.size()-1. The template class T can be either vector or valarray of Real or Integer.

- ```
  template <class T> void SPbisect(Integer& lower, Integer& upper,
                                   const T& array, const Real& target,
                                   bool increasing)
  ```

Do a bisection search between indices lower and upper on array to find target. On exit lower and upper are set to the indices which bracket target.

- ```
  IntegerVector SPcalcOptBins(const RealVector& FWHM,
                              const IntegerVector& counts)
  ```

Calculate the Kaastra & Bleeker optimal binsizes based on input FWHM and counts arrays.

- ```
  bool SPcheckStokesConsistency(const string& stokesP1,
                                const string& stokesW1,
                                const string& stokesP2,
                                const string& stokesW2)
  ```

Check for consistency between two sets of Stokes information

- ```
  bool SPfindFirstBins(const RealVector& currentBins,
                       const RealVector& targetBins, size_t& currentStart,
                       size_t& targetStart)
  ```

Find first bin in targetBins which is contained in the range of currentBins and the bin in currentBins to which it corresponds.

- `bool SPinitializeBins(const RealVector& currentBins,`
  `const RealVector& targetBins,`
  `const size_t& currentStart,`
  `const size_t& targetStart,`
  `IntegerVector& startBin, IntegerVector& endBin,`
  `RealVector& startWeight, RealVector& endWeight)`

  Calculate start and end bins (and weightings) of currentBins for each bin in targetBins.

3.7   I/O routines

These C++ routines can be found in SPio.h and SPio.cxx and are not available from Python.

- `template <class T> T SPreadKey(CCfits::PHDU& primary, const string Keyname,`
  `const T DefValue)`

  Read a keyword value from the primary header.

- `template <class T> T SPreadKey(CCfits::ExtHDU& ext, const string Keyname,`
  `const T DefValue)`

  Read a keyword value from an extension header.

- `template <class T> T SPreadKey(CCfits::ExtHDU& ext, const string Keyname,`
  `const Integer RowNumber, const T DefValue)`

  Read a keyword value which may be in a column for type II files.

- `template <class T> void SPreadCol(CCfits::ExtHDU& ext,`
  `const string ColName,`
  `std::valarray<T>& Data)`

  Read a column into a valarray.

- `template <class T> void SPreadCol(CCfits::ExtHDU& ext,`
  `const string ColName,`
  `std::vector<T>& Data)`

  Read a column into a vector.

- `void SPreadCol(CCfits::ExtHDU& ext, const string ColName,`
  `StringVector& Data)`

  Read a string column into a vector (special case necessary for a workaround in CCfits).

- `template <class T> void SPreadCol(CCfits::ExtHDU& ext,`
  `const string ColName,`
  `const Integer RowNumber,`
  `std::valarray<T>& Data)`

  Read a column from a Type II file into a valarray.

- ```
  template <class T> void SPreadCol(CCfits::ExtHDU& ext,
                                    const string ColName,
                                    const Integer RowNumber,
                                    std::vector<T>& Data)
  ```

  Read a column from a Type II file into a vector.

- ```
  template <class T> void SPreadVectorCol(CCfits::ExtHDU& ext,
                                          const string ColName,
                                std::vector<std::valarray<T> >& Data)
  ```

  Read a vector column into a vector of valarrays.

- ```
  template <class T> void SPreadVectorCol(CCfits::ExtHDU& ext,
                                          const string ColName,
                                          std::vector<std::vector<T> >& Data)
  ```

  Read a vector column into a vector of vectors.

- ```
  template <class T> void SPreadVectorColRow(CCfits::ExtHDU& ext,
                                             const string ColName,
                                             const Integer RowNumber,
                                             std::valarray<T>& Data)
  ```

  Read a single row of a vector column into a valarray.

- ```
  template <class T> void SPreadVectorColRow(CCfits::ExtHDU& ext,
                                             const string ColName,
                                             const Integer RowNumber,
                                             std::vector<T>& Data)
  ```

  Read a single row of a vector column into a vector.

- ```
  template <class T> void SPwriteKey(CCfits::PHDU& primary,
                                     const string Keyname,
                                     const T KeyValue, const string Comment)
  ```

  Write a keyword to the primary header. This uses the CONTINUE convention if KeyValue is a string and longer than 68 characters.

- ```
  template <class T> void SPwriteKey(CCfits::Table& primary,
                                     const string Keyname,
                                     const string ColName,
                                     const std::valarray<T>& Data,
                                     const bool forceCol = false)
  ```

  Write a column from a valarray. If the data size is 1 or all values are the same then just write a keyword unless the forceCol bool is true.

- ```
  template <class T> void SPwriteCol(CCfits::Table& table,
                                     const string ColName,
                                     const std::vector<T>& Data,
  ```

```
                                          const bool forceCol = false)
```

Write a column from a vector. If the data size is 1 or all values are the same then just write a keyword unless the forceCol bool is true.

- ```
  template <class T> void SPwriteVectorCol(CCfits::Table& table,
                                           const string ColName,
                              const std::vector<std::valarray<T> >& Data,
                                           const bool forceCol = false)
  ```

Write a column from a vector of valarrays. If the data size is 1 or all values are the same then just write a keyword unless the forceCol bool is true. If all values are the same within all valarrays then write a scalar column.

- ```
  template <class T> void SPwriteVectorCol(CCfits::Table& table,
                                           const string ColName,
                               const std::vector<std::vector<T> >& Data,
                                           const bool forceCol = false)
  ```

Write a column from a vector of vectors. If the data size is 1 or all values are the same then just write a keyword unless the forceCol bool is true. If all values are the same within all valarrays then write a scalar column.

- ```
  template <class T> bool SPneedCol(const T& Data, bool& isvector)
  ```

Check whether a column is required and if it needs to be a vector column. Note that T is assumed to be a valarray or vector of a valarray of some type.

- ```
  template <class T> bool SPneedCol(const T& Data)
  ```

Check whether a column is required. For this overloaded version T is assumed to be a valarray or vector of scalar of some type.

- ```
  template <class T> bool SPneedVecCol(const T& Data)
  ```

Check whether a given column needs to be a vector. This assumes that the column is required and cannot be replaced by a keyword. T is assumed to be a valarray or a vector of a valarray of some type.

- ```
  StringVector SPreadAllPrimaryKeywords(const string& filename)
  ```

Real all keywords from the primary extension into a vector of strings each of which is of format "KeyName = KeyValue".

- ```
  StringVector SPreadAllKeywords(const string& filename,
                                 const string& hduName,
                                 const int& hduNumber)
  ```

Real all keywords from an extension into a vector of strings each of which is of format "KeyName = KeyValue".

- ```
  StringVector SPreadAllColumnNames(const string& filename,
                                    const string& hduName,
  ```

```
const int& hduNumber)
```

Real all the column names from an extension into a vector of strings.

Chapter 4

C interface

4.1 PHA files

4.1.1 PHA structure

```
struct PHA {

  long NumberChannels;                   /* Number of spectrum channels */
  long FirstChannel;                     /* First channel number */

  float* Pha;/*NumberChannels*/          /* PHA data */
  float* StatError;/*NumberChannels*/    /* Statistical error */
  float* SysError;/*NumberChannels*/     /* Statistical error */

  int*   Quality;/*NumberChannels*/      /* Data quality */
  int*   Grouping; /*NumberChannels*/    /* Data grouping */
  int*   Channel; /*NumberChannels*/     /* Channel number */

  float* AreaScaling;/*NumberChannels*//* Area scaling factor */
  float* BackScaling;/*NumberChannels*//* Background scaling factor */

  float Exposure;                        /* Exposure time */
  float CorrectionScaling;               /* Correction file scale factor */
  int   DetChans;                        /* Content of DETCHANS keyword */

  int Poisserr;                          /* If true, errors are Poisson */
  char Datatype[FLEN_KEYWORD];           /* "COUNT" for count data and */
                                         /* "RATE" for count/sec */
  char Spectrumtype[FLEN_KEYWORD];       /* "TOTAL", "NET", or "BKG" */
```

```
  char ResponseFile[FLEN_FILENAME];    /* Response filename */
  char AncillaryFile[FLEN_FILENAME];   /* Ancillary filename */
  char BackgroundFile[FLEN_FILENAME];  /* Background filename */
  char CorrectionFile[FLEN_FILENAME];  /* Correction filename */

  char ChannelType[FLEN_KEYWORD];         /* Value of CHANTYPE keyword */
  char Telescope[FLEN_KEYWORD];
  char Instrument[FLEN_KEYWORD];
  char Detector[FLEN_KEYWORD];
  char Filter[FLEN_KEYWORD];
  char Datamode[FLEN_KEYWORD];

  char StokesParameter[FLEN_VALUE]; /* For spectrapolarimetry
                                       (e.g. 'Ixx', 'xQx', 'xxU') */
  char StokesWeighting[FLEN_VALUE]; /* For spectrapolarimetry Stokes parameter
                                       weighting scheme */

  char *XSPECFilter[100];                 /* Filter keywords */
};
```

4.1.2   PHA routines

- `int ReadPHAtypeI(char *filename, long PHAnumber, struct PHA *phastruct)`

  Read the type I SPECTRUM extension from a FITS file - if there are multiple SPECTRUM extensions then read the one with EXTVER=PHAnumber.

- `int ReadPHAtypeII(char *filename, long PHAnumber, long NumberSpectra,`
  `                  long *SpectrumNumber, struct PHA **phastructs)`

  Read the type II SPECTRUM extension from a FITS file - if there are multiple SPECTRUM extensions then read the one with EXTVER=PHAnumber. Within the SPECTRUM extension reads the spectra listed in the SpectrumNumber vector.

- `int WritePHAtypeI(char *filename, struct PHA *phastruct)`

  Write the spectrum to a type I SPECTRUM extension in a FITS file.

- `int WritePHAtypeII(char *filename, long NumberSpectra,`
  `                   struct PHA **phastructs)`

  Write the multiple spectra to a type II SPECTRUM extension in a FITS file.

- `int ReturnPHAtype(char *filename, long PHAnumber)`

  Return the type of the SPECTRUM extension with EXTVER=PHAnumber.

- `void DisplayPHAtypeI(struct PHA *phastruct)`

  Write information about the spectrum to stdout.

- `void DisplayPHAtypeII(long NumberSpectra, struct PHA **phastructs)`
  Write information about multiple spectra to stdout.

- `int RebinPHA(struct PHA *phastruct, struct BinFactors *bin)`
  Rebin spectrum.

- `int CheckPHAcounts(char *filename, long PHAnumber)`
  Return 0 if COUNTS column exists and is integer or COUNTS column does not exist.

- `long ReturnNumberofSpectra(char *filename, long PHAnumber)`
  Return the number of spectra in the type II SPECTRUM extension which has EXTVER equal to PHAnumber.

4.2 RMF files

4.2.1 RMF structure

```
struct RMF {

  long NumberChannels;                            /*Number of spectrum channels*/
  long NumberEnergyBins;                          /*Number of response energies*/
  long NumberTotalGroups;                         /*Total number of resp groups*/
  long NumberTotalElements;                       /*Total number of resp elts*/
  long FirstChannel;                              /*First channel number*/
  long isOrder;                                   /*If true grating order*/
                                                  /*information included*/

  long* NumberGroups;/*NumberEnergyBins*/         /*Number of resp groups for*/
                                                  /*this energy bin*/
  long* FirstGroup;/*NumberEnergyBins*/           /*First resp group for this*/
                                                  /*energy bin (counts from 0)*/

  long* FirstChannelGroup;/*NumberTotalGroups*/   /*First channel number in*/
                                                  /*this group*/
  long* NumberChannelGroups;/*NumberTotalGroups*//*Num of channels in this grp*/
  long* FirstElement;/*NumberTotalGroups*/        /*First resp elt for this grp*/
                                                  /*(counts from 0)*/
  long* OrderGroup;/*NumberTotalGroups*/          /*Grating order of this grp*/

  float* LowEnergy;/*NumberEnergyBins*/           /*Start energy of bin*/
  float* HighEnergy;/*NumberEnergyBins*/          /*End energy of bin*/

  float* Matrix;/*NumberTotalElements*/           /*Matrix elements*/
```

```
  float* ChannelLowEnergy;/*NumberChannels*/       /*Start energy of channel*/
  float* ChannelHighEnergy;/*NumberChannels*/      /*End energy of channel*/

  float AreaScaling;                               /*Value of EFFAREA keyword*/
  float ResponseThreshold;                         /*Minimum value in response*/

  char EnergyUnits[FLEN_KEYWORD];                  /*Units for energies*/
  char RMFUnits[FLEN_KEYWORD];                     /*Units for RMF*/

  char ChannelType[FLEN_KEYWORD];                  /*Value of CHANTYPE keyword*/
  char Telescope[FLEN_KEYWORD];
  char Instrument[FLEN_KEYWORD];
  char Detector[FLEN_KEYWORD];
  char Filter[FLEN_KEYWORD];
  char RMFType[FLEN_KEYWORD];                      /*HDUCLAS3 keyword in MATRIX*/
  char RMFExtensionName[FLEN_VALUE];               /*EXTNAME keyword in MATRIX*/
  char EBDExtensionName[FLEN_VALUE];               /*EXTNAME keyword in EBOUNDS*/

  char StokesParameter[FLEN_VALUE]; /* For spectrapolarimetry
                                       (e.g. 'Ixx', 'xQx', 'xxU') */
  char StokesWeighting[FLEN_VALUE]; /* For spectrapolarimetry Stokes parameter
                                       weighting scheme */

};

struct RMFchan {

  long NumberChannels;                             /*Number of spectrum channels*/
  long NumberEnergyBins;                           /*Number of response energies*/
  long NumberTotalGroups;                          /*Total number of resp groups*/
  long NumberTotalElements;                        /*Total number of resp elts*/
  long FirstChannel;                               /*First channel number*/
  long isOrder;                                    /*If true grating order*/
                                                   /*information included*/

  long* NumberGroups;/*NumberChannels*/            /*Number of resp groups for*/
                                                   /*this channel bin*/
  long* FirstGroup;/*NumberChannels*/              /*First resp group for this*/
                                                   /*channel bin (counts from 0)*/

  long* FirstEnergyGroup;/*NumberTotalGroups*/   /*First energy bin in this grp*/
  long* NumberEnergyGroups;/*NumberTotalGroups*///*Number of energy bins in*/
                                                 /*this group */
  long* FirstElement;/*NumberTotalGroups*/       /*First resp elt for this grp*/
```

```
                                              /*(counts from 0)*/
  long* OrderGroup;/*NumberTotalGroups*/       /*Grating order of this group*/

  float* LowEnergy;/*NumberEnergyBins*/         /*Start energy of bin*/
  float* HighEnergy;/*NumberEnergyBins*/        /*End energy of bin*/

  float* Matrix;/*NumberTotalElements*/         /*Matrix elements*/

  float* ChannelLowEnergy;/*NumberChannels*/    /*Start energy of channel*/
  float* ChannelHighEnergy;/*NumberChannels*/   /*End energy of channel*/

  float AreaScaling;                            /*Value of EFFAREA keyword*/
  float ResponseThreshold;                      /*Minimum value in response*/

  char EnergyUnits[FLEN_KEYWORD];               /*Units for energies*/
  char RMFUnits[FLEN_KEYWORD];                  /*Units for RMF*/

  char ChannelType[FLEN_KEYWORD];              /*Value of CHANTYPE keyword*/
  char RMFVersion[FLEN_KEYWORD];               /*MATRIX format version*/
  char EBDVersion[FLEN_KEYWORD];               /*EBOUNDS format version*/
  char Telescope[FLEN_KEYWORD];
  char Instrument[FLEN_KEYWORD];
  char Detector[FLEN_KEYWORD];
  char Filter[FLEN_KEYWORD];
  char RMFType[FLEN_KEYWORD];                  /*HDUCLAS3 keyword in MATRIX*/
  char RMFExtensionName[FLEN_VALUE];           /*EXTNAME keyword in MATRIX*/
  char EBDExtensionName[FLEN_VALUE];           /*EXTNAME keyword in EBOUNDS*/
};
```

4.2.2 RMF routines

- `int ReadRMFMatrix(char *filename, long RMFnumber, struct RMF *rmf)`

  Read the RMF matrix from a FITS file - if there are multiple RMF extensions then read the one with EXTVER=RMFnumber.

- `int WriteRMFMatrix(char *filename, struct RMF *rmf)`

  Write the RMF matrix to a FITS file.

- `int ReadRMFEbounds(char *filename, long EBDnumber, struct RMF *rmf)`.

  Read the RMF ebounds from a FITS file - if there are multiple EBOUNDS extensions then read the one with EXTVER=EBDnumber.

- `int WriteRMFEbounds(char *filename, struct RMF *rmf)`

Write the RMF ebounds to a FITS file.

- `void DisplayRMF(struct RMF *rmf)`

  Write information about RMF to stdout.

- `int CheckRMF(struct RMF *rmf)`

  Check validity of RMF

- `void ReturnChannel(struct RMF *rmf, float energy, int NumberPhotons,`
  `                   long *channel)`

  Return the channel for a photon of the given input energy - draws random numbers to return NumberPhotons entries in the channel array.

- `void NormalizeRMF(struct RMF *rmf)`

  Normalize the response to unity in each energy.

- `void CompressRMF(struct RMF *rmf, float threshold)`

  Compress the response to remove all elements below the threshold value.

- `int RebinRMFChannel(struct RMF *rmf, struct BinFactors *bins)`

  Rebin the RMF in channel space.

- `int RebinRMFEnergy(struct RMF *rmf, struct BinFactors *bins)`

  Rebin the RMF in energy space.

- `void TransposeRMF(struct RMF *rmf, struct RMFchan *rmfchan)`

  Transpose the matrix.

- `float ReturnRMFElement(struct RMF *rmf, long channel, long energybin)`

  Return a single value from the matrix.

- `float ReturnRMFchanElement(struct RMFchan *rmfchan, long channel,`
  `                           long energybin)`

  Return a single value from the transposed matrix.

- `int AddRMF(struct RMF *rmf1, struct RMF *rmf2)`

  Add rmf2 onto rmf1.

4.3   ARF files

4.3.1   ARF structure

`struct ARF {`

```
  long NumberEnergyBins;                    /* Number of response energies */

  float* LowEnergy; /*NumberEnergyBins*/  /* Start energy of bin */
  float* HighEnergy; /*NumberEnergyBins*/ /* End energy of bin */

  float* EffArea;    /*NumberEnergyBins*/ /* Effective areas */

  char EnergyUnits[FLEN_KEYWORD];          /* Units for energies */
  char arfUnits[FLEN_KEYWORD];             /* Units for effective areas */

  char Telescope[FLEN_KEYWORD];
  char Instrument[FLEN_KEYWORD];
  char Detector[FLEN_KEYWORD];
  char Filter[FLEN_KEYWORD];

  char StokesParameter[FLEN_VALUE];     /* For spectrapolarimetry
                                           (e.g. 'Ixx', 'xQx', 'xxU') */
  char StokesWeighting[FLEN_VALUE];     /* For spectrapolarimetry Stokes parameter
                                           weighting scheme */

  char ARFExtensionName[FLEN_VALUE];       /* EXTNAME keyword in SPECRESP */

};
```

4.3.2   ARF routines

- `int ReadARF(char *filename, long ARFnumber, struct ARF *arf)`

  Read the effective areas from a FITS file - if there are multiple SPECRESP extensions then read the one with EXTVER=ARFFnumber.

- `int WriteARF(char *filename, struct ARF *arf)`

  Write the ARF to a FITS file.

- `void DisplayARF(struct ARF *arf)`

  Write information about ARF to stdout.

- `int AddARF(struct ARF *arf1, struct ARF *arf2)`

  Add arf2 onto arf1.

- `long MergeARFRMF(struct ARF *arf, struct RMF *rmf)`

  Multiply the ARF into the RMF.

4.4   Binning and utility

4.4.1   BinFactors structure

```
struct BinFactors {

  long NumberBinFactors;

  long *StartBin;
  long *EndBin;
  long *Binning;

};
```

4.4.2   Binning and utility routines

- `int SPReadBinningFile(char *filename, struct BinFactors *binning)`

  Read an ascii file with binning factors and load the binning array.

- `int SPSetGroupArray(int inputSize, struct BinFactors *binning, int *groupArray)`

  Set up a grouping array using the BinFactors structure.

- `int SPBinArray(int inputSize, float *input, int *groupArray, int mode, float *output)`

  Bin an array using the information in the grouping array.

- `void SPsetCCfitsVerbose(int mode)`

  Set the CCfits verbose mode.

- `int SPcopyExtensions(char *infile, char *outfile)`

  Copy all HDUs which are not manipulated by this library.

- `int SPcopyKeywords(char *infile, char *outfile, char *hduname, int hdunumber)`

  Copy all non-critical keywords for the hdunumber instance of the extension hduname.

Appendix A

Building programs using heasp

To build a C++ program using the heasp library you need to use the standard heasoft tool i.e. hmake. This requires a Makefile in the following specific format.

```
HD_COMPONENT_NAME        = heacore

HD_COMPONENT_VERS        =

HD_CXXTASK               = myprogram

HD_CXXTASK_SRC_cxx       = myprogram.cxx

HD_CXXFLAGS              = ${HD_STD_CXXFLAGS} -Wno-deprecated

HD_CXXLIBS               = ${HD_LFLAGS} ${HD_STD_CXXLIBS} -l${HEASP} \
                           ${XLIBS} ${SYSLIBS}

HD_INSTALL_TASKS         =

include ${HD_STD_MAKEFILE}
```

Here myprogram.cxx is the file containing the main routine. Any other C++ source files required can be added to the same line. This will build an executable file in the current directory. The default hmake install option is disabled but if the executable is to be placed in a particular place that command can be added to the HD_INSTALL_TASKS line.

Appendix B

Ftools and Heasp

The following table lists ftools that operate on spectra or responses and the related HEASP routines. The read and write routines apply in all cases so are not included in the table. In some, relatively simple, cases the ftool equivalent could be performed by directly getting and setting class members.

Ftool	corresponding HEASP C++ routines
addarf / ftaddarf	arf::operator+=, arf::operator+
addrmf / ftaddrmf	rmf::operator+=, rmf::operator+
arf2arf1	arfII:get
ascii2pha	directly set pha class members
chkarf	arf::check, arfII::check
chkpha / ftchkpha	pha::check, phaII::check
chkrmf / ftchkrmf	rmf::check
ftcmparf	arfII::get
cmppha / ftcmppha	phaII::get
cmprmf / ftcmprmf	rmf::compress
ftdcmprmf	rmf::uncompress
dmprmf / ftdmprmf	directly access rmf class members
flx2tab / ftflx2tab	directly set table class members
flx2xsp / ftflx2xsp	directly set pha and rmf class members
gcorpha / ftgcorpha	pha::shiftChannels
gcorrmf / ftgcorrmf	rmf::shiftChannels
ftgenrsp	directly set rmf class members
ftgroupha	pha::getMinCountsGrouping
	pha::getMinSNGrouping
	pha::getMinSNOptimalGrouping
	pha::setGrouping, grouping::loadOptimal
	rmf::estimatedFWHMperChannel
grppha	grouping::load, pha::setGrouping
grppha2	phaII::get, grouping::load
	pha::setGrouping, phaII::push
marfrmf / ftmarfrmf	rmf::operator*=, rmf::operator*
mathpha	pha::operator+=, pha::operator+, pha::operator*=
mkfiltrsp / ftmkfiltrsp	directly set rmf class members
ftrbnarf	grouping::load, GroupBin
rbnpha / ftrbnpha	grouping::load, pha::setGrouping, pha::rebinChannels
rbnrmf / ftrbnrmf	grouping::load, grouping::loadOptimalEnergy
	rmf::rebinChannels, rmf::rebinEnergies
	rmf::estimatedFWHM
rsp2rmfarf / ftrsp2rmfarf	directly get and set rmf and arf class members
sdss2xsp / ftsdss2xsp	directly set pha and rmf class members
ftstripnegchan	rmf::addRow
ftVOtable2xsp	directly set pha and rmf class members

Appendix C

Error Codes

- 1 : NoSuchFile : Cannot find the file specified.

- 2 : NoData : A column read has no members.

- 3 : NoChannelData : The SPECTRUM has no Channel column and no channel data can be constructed.

- 4 : NoStatError : The SPECTRUM has no statistical error column and POISSERR=F.

- 5 : CannotCreate : Cannot create a new file.

- 6 : NoEnergLo : The ENERG_LO column in an ARF or RMF file has no data.

- 7 : NoEnergHi : The ENERG_HI column in an ARF or RMF file has no data.

- 8 : NoSpecresp : The SPECRESP column in an ARF has no data.

- 9 : NoEboundsExt : There is no EBOUNDS extension in an RMF file.

- 10 : NoEmin : The E_MIN column in an RMF file has no data.

- 11 : NoEmax : The E_MAX column in an RMF file has no data.

- 12 : NoMatrixExt : There is no MATRIX extension in an RMF file.

- 13 : NoNgrp : The N_GRP column in an RMF file has no data.

- 14 : NoFchan : The F_CHAN column in an RMF file has no data.

- 15 : NoNchan : The N_CHAN column in an RMF file has no data.

- 16 : NoMatrix : The MATRIX column in an RMF file has no data.

- 17 : CannotCreateMatrixExt : The output MATRIX extension cannot be created.

- 18 : CannotCreateEboundsExt : The output EBOUNDS extension cannot be created.

- 19 : InconsistentGrouping : The grouping information size is different from that of the array to which it is being applied.

- 20 : InconsistentEnergies : The energy information differs between the objects which are being compared.

- 21 : InconsistentChannels : The channel information differs between the objects which are being compared.

- 22 : InconsistentUnits : The units information differs between the objects which are being compared.

- 23 : UnknownXUnits : The wavelength or energy units are not supported.

- 24 : UnknownYUnits : The flux units are not supported.

- 25 : InconsistentNumelt : The RMF MATRIX extension NUMELT keyword is inconsistent with the actual number of response elements.

- 26 : InconsistentNumgrp : The RMF MATRIX extension NUMGRP keyword is inconsistent with the actual number of response groups.

- 27 : InconsistentNumTableParams : The wrong number of parameters were given to table::getValues.

- 28 : TableParamValueOutsideRange : A parameter value was given to table::getValues which is outside the tabulated range.

- 29 : InconsistentKeywordValues : Two keywords which should have the same value do not.

- 30 : CannotCopyColumn : A column cannot be copied from one HDU to another.

- 31 : CannotWriteMatrix : Failed to write rmf MATRIX data to a file.

- 32 : InconsistentTableFilter : Number of filters input is not the same as in the file.

- 33 : NoChannels : Failed to read the CHANNELS column.

- 34 : InconsistentChannelMin

- 35 : InconsistentFChan

- 36 : InconsistentNChan

Appendix D

Change log

D.1 v2.8

D.1.1 General

- In SPutils added routines to copy a single HDU from one file to another and to split an extension suffix from a filename.

D.1.2 arf

- Added remapEnergies method analogous to that in the rmf class.

D.1.3 pha

- Fixed setGrouping to make sure that any information about bad quality channels in the spectrum is combined with any setting of channels as bad quality in the grouping process.

- Fixed coding error in getMinSNOptimalGrouping which could cause overrunning the end of a vector.

- Fixed read method so that it uses the extension suffix on an input filename. Also added the option to pass an extension version number into the write method.

D.1.4 phaII

- Fixed read method so that it uses the extension suffix on an input filename. Also added the option to pass an extension version number into the write method.

D.1.5 grouping

- Improved diagnostics in loadOptimal if FWHM and Counts have different sizes.

D.1.6   rmf

- Fixed remapEnergies so that it divides by the total weighting if operating on a REDIST matrix. This makes it consistent with rebinEnergies.

- Corrected use of J or I when writing depending on the number of channels.

- Added append method to explicitly write rmf to an existing file with an option to write only the MATRIX or only the EBOUNDS.

D.2   v2.7

D.2.1   General

- In SPutils improved SPreadStrings so that it works correctly even if the line does not end with a newline.

- In SPio added a handy routine, SPreadAllColumnNames, to return all the column names in an extension.

D.2.2   grouping

- Added a method, isGroupingOrQuality(), to test whether the grouping flags are not identically 1 and the quality flags are not identically 0.

D.2.3   pha

- Fixed test for negative exposure time in pha::check method - it should not return an error if the exposure is zero.

- Fixed writing grouping keyword when the grouping vector is identically 1. It was writing the keyword value as 1 but the standard requires this to be 0.

D.2.4   rmf

- Fix for an occasional crash in rmf::rebinChannels and some code simplification.

- Added a check that F_CHAN + N_CHAN do not exceed the number of channels.

- Added a test during the rmf::read method for the MATRIX and EBOUNDS extensions having the same first channel.

- Fixed error in rmf::rebinEnergies if the grouping vector has zero values indicating the energy should be ignored. This only makes sense if energy bins are ignored at the beginning or end.

- Made the MATRIX extension read slightly cleverer if there is no TLMIN keyword for the F_CHAN column. In this case assume the first channel is 1 unless the minimum in the F_CHAN column is 0.

- Fixed case of empty FCHAN in the first energy bins.

D.3   v2.6

D.3.1   arf

- Added ARFVersionNumbers routine to return version numbers of all SPECRESP extensions in a file.

- Renamed ARFspecrespExtensions as ARFExtensionNumbers

D.3.2   pha

- Added isSpectrumValid routine to test whether an extension is a valid spectrum. At present this just checks the HDUCLAS1 and checks for a COUNTS or RATE column.

- Renamed SpectrumExtensions as SpectrumExtensionNumbers.

D.3.3   rmf

- Renamed RMFeboundsExtensions as RMFeboundsExtensionNumbers and RMFmatrixExtensions as RMFmatrixExtensionNumbers.

- Added RMFeboundsVersionNumbers and RMFmatrixVersionNumbers to return version numbers of all EBOUNDS and MATRIX extensions, respectively.

D.4   v2.5

D.4.1   General

- The directives "using namespace std" and "using namespace CCfits" have been removed from heasp.h. If required they can be added to the start of any program linking to heasp.

- All of heasp is now enclosed in a namespace called heasp.

- RealVector, IntegerVector, and StringVector have been added as typedefs.

- The SPutils routines isValidXUnits, isValidYUnits, calcXfactor, and calcYfactor have been renamed to SPisValidXUnits, SPisValidYUnits, SPcalcXfactor, and SPcalcYfactor for consistency.

D.4.2   arf

- m_StokesParameter and m_StokesWeighting have been added to store the contents of the STOKESPR and STOKESWT keywords used for spectropolarimetric data.

D.4.3   pha

- m_StokesParameter and m_StokesWeighting have been added to store the contents of the STOKESPR and STOKESWT keywords used for spectropolarimetric data.

D.4.4   rmf

- m_StokesParameter and m_StokesWeighting have been added to store the contents of the STOKESPR and STOKESWT keywords used for spectropolarimetric data.

D.5   v2.4

D.5.1   General

- Added routine to SPutils to calculate the Kaastra & Bleeker optimal binsizes so it can be shared between classes.

D.5.2   grouping

- added inBin method to return whether an input element is in any output bin.

- trapped a possible segfault if the first channel of a group is set to bad quality but the other channels are good.

D.5.3   pha

- fixed rebinChannels for seg fault if the grouping starts by not including channels.

- added getMinSNOptimalGrouping to do optimal binning with an additional S/N criterion.

D.5.4   rmf

- improved handling of the case of the FCHAN and/or NCHAN arrays being longer than the number of groups specified in N_GRP.

D.5.5 table

- fixed getValues error in calculating spectrum numbers for interpolation when there are XFLT keys set (bug 2.3a).

- fixed copy to add copying the filter expression map.

D.6 v2.3

D.6.1 General

- Added SPtrimString function to remove leading and trailing whitespace. Used this in check-Compatibility method for arf, pha, and rmf to avoid incorrect failure flagging due to leading or trailing blank spaces. (bug 2.2c)

- Added version of SPwriteKey to write to the primary header and modified SPwriteKey for string values so it automatically uses the CONTINUE convention if the string is more than 68 characters.

D.6.2 grouping

- Added optimal binning with minimum counts.

- Fixed loadMin, loadOptimal, and loadOptimalEnergy so that any channels not included in a bin due to an incomplete bin at the end of the channels to be grouped are given GROUP-ING=1 and QUALITY=2. This is consistent with the behaviour of grppha.

D.6.3 pha

- Fixed error in rebinChannels in the way that the new quality vector is calculated if the grouping included elements with bad quality.

D.6.4 rmf

- Fix to rebinChannels. For EBOUNDS arrays that are grouped AND in descending order, must switch the FirstEltMode/LastEltMode flags when calling GroupBin for m_channelLowEnergy and m_channelHighEnergy. (bug 2.2a)

- Fix to write the EBOUNDS HDUVERS as 1.2.0

- Fixed bug in rebinEnergies: in the case where the rmf is REDIST the response should be averaged over the binned energies, not summed. (bug 2.2b)

- Modified uncompress so it does not change the value of the LO_THRES keyword. Also changed so that uncompressed data are written as fixed-length vectors.

- Fixed checkCompatibility over-strictness and improved diagnostic messages. (bug 2.2d)

D.6.5   table

- Added an m_Units member to the tableParameter class.

- Added m_FiltExps to the table class to allow multiple model spectra per parameter grid point. The model spectrum used is selected by comparing the table filter expression with that in the data set.

- In getValues function, commented out the section that selects a subset of the tableEnergies array based on overlap with the input energies. This is incompatible with external usages, such as Xspec's OGIPTable::energyWeights() function, which assume they are using the full tableEnergies array.

- Replaced addKey calls with SPwriteKey.

D.7   v2.2

D.7.1   General

- Changed read column methods so that they check for a keyword if the column is not found rather than the other way round. This avoids a possible problem if the extension has both.

D.7.2   arf

- In read check for an EBOUNDS extension to read ENERG_LO and ENERG_HI which might occur in a type II file.

D.7.3   grouping

- Added options for start and end channel to loadOptimal.

- Fixed a bug in grouping::GroupBin which was not taking into account of the quality flag.

D.7.4   pha

- Fixed a bug which could cause pha files to have both a keyword and column with the same name.

D.7.5   rmf

- Fixed a bug incheck which incorrectly flagged an error if the F_CHAN column had the maximum channel value and the first channel was non-zero.

- Modified the rmf and rmft classes to change from vectors to vectors of vectors for the first channel in group, number of channels in group, order of group, and matrix elements. This leads to a simplification in the code.

- Added a new version of addRow which takes the channel group information. Added a substituteRow method which takes a compressed format row as input.

- Changed normalize to return the vector of normalization factors.

- Added an changeFirstChannel method to change the first channel value used (usually switching between 0 and 1.

D.7.6   table

- Trapped case in getValues of there being no overlap between the input energy range and the tabulated energy range.

D.8   v2.1

D.8.1   General

- Made arf, pha, and rmf classes a bit more tolerant by allowing reading of files where extensions are identified only by HDUCLAS* keywords.

- Replaced deprecated auto_ptr by unique_ptr.

- Added SPneedVecCol routine to check whether a column should be vector or scalar.

- Changed SPcopyKeys so it only copies keywords that do not already exist in the target extension. This prevents overwriting keyword values that have been set by the tool calling the routine.

D.8.2   arf

- Added handy routines to return the extension numbers of SPECRESP extensions in a file and the type of an extension.

D.8.3   arfII

- Made recognition of arf extensions more flexible in NumberofARFs.

D.8.4   grouping

- Changed the grouping class to include separate grouping and quality vectors instead of combining them in a single flag vector. Made corresponding changes to methods.

- Fixed bug which was incorrectly rejecting binning factors of -1 in a binning text file.

D.8.5   pha

- Fixed a bug in convertUnits() which prevented the conversion working in the case of input fluxes in $ph/cm^2/s/keV$

- In rebinChannels set the grouping identically to 0 rather than 1 for consistency with the standard since a GROUPING keyword will be written.

- Changed setting of output quality to be bad if either any input channel making the bin is bad or if phafile grouping is being used any channel in the grouping pha is bad.

D.8.6   rmf

- When writing output files change N_GRP and CHANNEL to J format if required.

- Fixed out-of-bounds writes in cases when the F_CHAN and N_CHAN vector columns are longer than N_GRP.

- Modified rmf::read so that the EBOUNDS data is first read into a separate rmf object so it can test for consistency between MATRIX and EBOUNDS keywords such as CHANTYPE and INSTRUME. If inconsistencies are found then messages are written to SPreportError so the calling program can check for them. The keywords from MATRIX take priority over those from EBOUNDS.

- Add handy functions to return vectors of extension numbers of EBOUNDS and MATRIX extensions.

- Changed rmf compression criterion to only include elements with values > LO_THRES instead of >= LO_THRES.

- Write EBOUNDS and MATRIX energy columns as D instead of E to avoid precision issues with high-resolution optical/UV spectra.

D.8.7   table

- Changed ENERGIES columns from single to double precision.

- Fixed a bug in convertUnits() which prevented the conversion working in the case of input fluxes in $ph/cm^2/s/keV$

D.9   v2.00

D.9.1   General

- Major rewrite for all classes to make data private. Internal data members now start with m_. There are get and set operations to manipulate their values. So, for instance getfoo() returns m_foo and setfoo(bar) sets m_foo to bar. If m_foo is a vector then individual elements can be accessed using getfooElement(i) and setfooElement(i,bar).

- Used const wherever possible on methods and arguments for methods.

- Where possible used default values of arguments instead of overloading.

- Removed copying of other keywords and extensions from write methods since these should not be considered operations of the class.

- Removed storing OGIP version numbers in classes since this doesn't make sense - just write the latest version number in write methods.

- Added copy, load, and, where appropriate, read constructors.

- Added SWIG typemaps to map C++ vectors to numpy arrays in Python.

D.9.2   rmf

- Added estimatedFWHMperChannel method to estimate the FWHM in each channel in contrast to estimatedFWHM which estimates it in each energy bin.

D.9.3   grouping

- Added loadOptimalEnergy method to set the grouping array using the Kaastra & Bleeker optimal binning in response energy.

D.10   v1.20

D.10.1   General

- Fixed some cases where vectors were being passed to methods by value, not by address.

- Modified Makefile to remove dependencies on heautils, heaio and ape libraries, which are no longer required.

D.10.2   pha, phaII

- Add getGrouping method to return the content of the grouping array to a grouping object.

D.10.3   rmf

- Modified RandomChannels so that vectors of random numbers are passed in rather than generated internally.

D.10.4   table

- Fixed an error when reading which caused the last two entries in the Energies vector to have the same value.

D.10.5   grouping

- Added loadOptimal method for Kaastra & Bleeker optimal binning scheme.

- Added loadFromVector method to make a grouping object from quality and grouping vectors.

D.11   v1.10

D.11.1   General

- Changed definition of Real from float to double.

- Added SPstring2double and SPstring2float to convert from string to double and float.

- Added SPfind and SPbisect routines to search for the position of a target value within an array.

- Added SPfindExtensions to return the numbers of extensions with a given value of some keyword.

- Added chapter on building programs using the heasp library.

D.11.2   pha, phaII

- Added selectChannels to select a subset of the pha channels.

- Added getMinCountsGrouping to get the grouping using a minimum number of counts per bin

- Added getMinSNGrouping to get the grouping using a minimum S/N ratio per bin. Optionally also uses a background pha file.

D.11.3 rmf

- Added multiplyByModel to matrix multiply the response by a vector to output a vector of pha values.

- Added estimatedFWHM to estimate the FWHM in channels for each energy range.

- Added binarySearch to return the index in the energy array for an input energy

D.11.4 arf and arfII

- Added NumberofARFs to return the number of ARFs in the type II SPECRESP extension.

D.11.5 table

- Added FluxError and addFluxError to store errors on tabulated data when available.

- Added LowEnergyLimit and HighEnergyLimit to store values to be used when input energies are below and above, respectively, the tabulated energies.

- Added Filename to store name of file from which the table was read.

- Added option not to read all the Spectra but just set up the objects. Added a new method readSpectra which then loads the requested Spectra.

- Added getValues to return the interpolated spectrum based on the input parameter values.

D.11.6 grouping

- Corrected flag to be +1 for start of a bin, -1 for continuation, and 0 for ignore.

- Added loadMin which sets the grouping flag based requiring a minimum value in each grouped bin for the input array of values.

D.12 v1.03

D.12.1 General

- Added SPstring2Real and SPstring2Integer functions to convert vectors of strings into vectors of Reals and Integers, respectively.

D.12.2 rmf

- Added interpolateAndMultiply which multiplies the rmf by a vector which may not be on the same energy binning as the rmf.

D.13   v1.02

D.13.1   General

- Added SPwriteCreator routine to write the CREATOR keyword.

- Added SPreadStrings routine to read a text file into a vector of strings.

- Added SPtokenize routine to divide a string into substrings.

- Added SPmatchString routine to partial match a string from a vector of strings.

- Added SPstring2Real routine to convert a string to a Real.

- Added SPstring2Integer routine to convert a string to an Integer.

- Added SPcalcShift routine to calculate factors used for shifting an array.

D.13.2   pha, phaII

- Added shiftChannels options to include a stretch by specified factor, to deal with multiple shifts at the same time, and to allow the shift to be specified in energy instead of channels.

D.13.3   rmf

- Added RandomChannels options to use multiple input energies, each with a different number of output photons.

- Added an uncompress method to turn a compressed matrix into the full rectangular form.

- Added shiftChannels options to include a stretch by specified factor, to deal with multiple shifts at the same time, and to allow the shift to be specified in energy instead of channels.

- Added a *= and * operators to multiply by a constant.

- Added substituteRow method to replace a given row in the rmf. Note that this is inefficient so should not be used repeatedly to construct a new rmf.

D.13.4   arf

- Added a rebin method.

- Added *= and * operators to multiply by a constant.

D.14   v1.01

D.14.1   General

- Added unit conversions through calcXfactor and calcYfactor routines.

- Added a global SPerrorStack to improve error reporting. Entries are added to the stack using SPreportError and can be retrieved using SPgetErrorStack which returns a string. SPclearErrorStack resets the stack.

- Added modifications to all classes to handle case where user writes an extension of some type into a file which already contains an extension of the same type. Now writes the new extension with an EXTVER keyword set to one more than the highest EXTVER of other extensions of the same type in the file.

- Made checkCompatibility routines consistent across classes and added a check for units.

D.14.2   pha, phaII

- Overloaded +, += for pha objects to add them and *= to multiply by a constant.

- Added version of pha::rebinChannels which allows control of the method for calculating the new StatError.

- Added pha::shiftChannels to shift channels.

- Added missing write of the DETCHANS keyword/column for type II PHA files.

- Fixed problem in C wrappers for spectra with arrays containing identical values.

D.14.3   rmf

- Replaced the compressLine function with the rmf::addRow method. Added the reverseRows method to use in cases where the original energy bins are not in increasing order. Added calcGaussResp function to calculate the gaussian response for one row of a response matrix.

- Added rmf::clearMatrix method to wipe only the response matrix part of the rmf object.

- Added rmf::shiftChannels to shift channels.

- Added support for grating orders to the rmf class.

- Corrected error in rmf when reading a file with MATRIX as a fixed length vector column.

- Fixed a counting from 0 or 1 bug in rmf::RowValues.

D.14.4   table

- Added table::read method.

- Added table::reverseRows method to reverse the order of the Energy, Flux and addFlux vectors. This may be necessary if input is in wavelengths and convertUnits has been used.