CCfits

2.6

Generated by Doxygen 1.9.1

1 Todo List

Class CCfits::PHDU

Implement functions that allow replacement of the primary image

Member CCfits::AsciiTable::AsciiTable (FITS *p, const String &hduName, int rows, const std::vector< String > &columnName=std::vector< String >(), const std::vector< String > &columnFmt=std::vector< String >(), const std::vector< String > &columnUnit=std::vector< String >(), int version=1)

Member CCfits::FITS::addImage (const String &hduName, int bpix, std::vector< long > &naxes, int version=1)

Add a function for replacing the primary image

Member CCfits::FITS::addTable (const String &hduName, int rows, const std::vector< String > &columnName=std::vector< String >(), const std::vector< String > &columnFmt=std::vector< String >(), const std::vector< String > &columnUnit=std::vector< String >(), HduType type=BinaryTbl, int version=1)

the code should one day check that the version keyword is higher than any other versions already added to the FITS object (although cfitsio doesn't do this either).

2 Module Index

2.1 Modules

Here is a list of all modules:

3 Namespace Index

3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

4 Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

5 Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

6 Module Documentation

6.1 FITS Exceptions

Classes

- class CCfits::Column::RangeError

  exception to be thrown for inputs that cause range errors in column read operations.
- class CCfits::ExtHDU::WrongExtensionType

  Exception to be thrown on unmatched extension types.
- class CCfits::FITS::CantCreate

  thrown on failure to create new file
- class CCfits::FITS::CantOpen

  thrown on failure to open existing file

- class CCfits::FITS::NoSuchHDU

  exception thrown by HDU retrieval methods.
- class CCfits::FITS::OperationNotSupported

  thrown for unsupported operations, such as attempted to select rows from an image extension.
- class CCfits::FitsError

  FitsError is the exception thrown by non-zero cfitsio status codes.
- class CCfits::FitsException

  FitsException is the base class for all exceptions thrown by this library.
- class CCfits::FitsFatal

  [potential] base class for exceptions to be thrown on internal library error.
- class CCfits::FITSUtil::UnrecognizedType

  exception thrown by MatchType if it encounters data type incompatible with cfitsio.
- class CCfits::HDU::InvalidExtensionType

  exception to be thrown if user requests extension type that can not be understood as ImageExt, AsciiTable or BinTable.
- class CCfits::HDU::InvalidImageDataType

  exception to be thrown if user requests creation of an image of type not supported by cfitsio.
- class CCfits::HDU::NoNullValue

  exception to be thrown on seek errors for keywords.
- class CCfits::HDU::NoSuchKeyword

  exception to be thrown on seek errors for keywords.
- class CCfits::Table::NoSuchColumn

  Exception to be thrown on a failure to retrieve a column specified either by name or index number.

6.1.1 Detailed Description

7 Namespace Documentation

7.1 CCfits Namespace Reference

Namespace enclosing all CCfits classes and globals definitions.

Classes

- class AsciiTable

  Class Representing Ascii Table Extensions.
- class BinTable

  Class Representing Binary Table Extensions. Contains columns with scalar or vector row entries.
- class Column

  Abstract base class for Column objects.
- class ExtHDU

  base class for all FITS extension HDUs, i.e. Image Extensions and Tables.
- class FITS

  Memory object representation of a disk FITS file.
- class FitsError

FitsError is the exception thrown by non-zero cfitsio status codes.

- class FitsException

  FitsException is the base class for all exceptions thrown by this library.
- class FitsFatal

  [potential] base class for exceptions to be thrown on internal library error.
- class GroupTable

  Class representing a hierarchical association of Header Data Units (HDUs).
- class HDU

  Base class for all HDU [Header-Data Unit] objects.
- class Keyword

  Abstract base class defining the interface for Keyword objects.
- class PHDU

  class representing the primary HDU for a FITS file.
- class Table

Typedefs

- typedef std::multimap< std::string, CCfits::Column * > ColMap

  Type definition for a table's column container.

Enumerations

- enum ValueType

  CCfits value types and their CFITSIO equivalents (in caps)

Functions

- std::ostream & operator<< (std::ostream &s, const CCfits::HDU &right)

  Output operator for HDU objects. Primarily for testing purposes.
- std::ostream & operator<< (std::ostream &s, const Column &right)

  output operator for Column objects.
- std::ostream & operator<< (std::ostream &s, const FITS &right)

  Output operator. Calls output operators for HDUs in turn.

7.1.1 Detailed Description

Namespace enclosing all CCfits classes and globals definitions.

7.1.2 Enumeration Type Documentation

7.1.2.1 ValueType `enum CCfits::ValueType`

CCfits value types and their CFITSIO equivalents (in caps)

Tnull, Tbit = TBIT, Tbyte = TBYTE, Tlogical = TLOGICAL, Tstring = TSTRING, Tushort = TUSHORT, Tshort = TSHORT, Tuint = TUINT, Tint = TINT, Tulong = TULONG, Tlong = TLONG, Tlonglong = TLONGLONG, Tfloat = TFLOAT, Tdouble = TDOUBLE, Tcomplex = TCOMPLEX, Tdblcomplex=TDBLCOMPLEX, VTbit= -TBIT, VTbyte=-TBYTE, VTlogical=-TLOGICAL, VTstring=-TSTRING, VTushort=-TUSHORT, VTshort=-TSHORT, VTuint=-TUINT, VTint=-TINT, VTulong=-TULONG, VTlong=-TLONG, VTlonglong=-TLONGLONG, VTfloat=-TFLOAT, VTdouble=-TDOUBLE, VTcomplex=-TCOMPLEX, VTdblcomplex=-TDBLCOMPLEX

7.1.3 Function Documentation

7.1.3.1 operator<<() `std::ostream & operator<< ( std::ostream & s, const FITS & right ) [inline]`

Output operator. Calls output operators for HDUs in turn.

This operator acts similarly to the ftool fdump for a fits file, except that there is no freedom to output partial information.

The current implementation of this operator for PHDU objects only outputs the array sizes, not the data, which that for tables prints the data also.

Provision of this operator is intended largely for debugging purposes.

7.2 FITSUtil Namespace Reference

FITSUtil is a namespace containing functions used internally by CCfits, but which might be of use for other applications.

7.2.1 Detailed Description

FITSUtil is a namespace containing functions used internally by CCfits, but which might be of use for other applications.

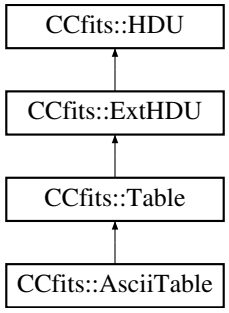
8 Class Documentation

8.1 CCfits::AsciiTable Class Reference

Class Representing Ascii Table Extensions.

`#include <AsciiTable.h>`

Inheritance diagram for CCfits::AsciiTable:



Public Member Functions

- virtual void addColumn (ValueType type, const String &columnName, long repeatWidth, const String &col↩ Unit=String(""), long decimals=0, size_t columnNumber=0)

  add a new column to an existing table HDU.
- virtual AsciiTable * clone (FITS *p) const

  virtual copy constructor
- virtual void readData (bool readFlag=false, const std::vector< String > &keys=std::vector< String >())

  read columns and keys specified in the input array.

Protected Member Functions

- AsciiTable (FITS *p, const String &hduName, int rows, const std::vector< String > &columnName=std::vector< String >(), const std::vector< String > &columnFmt=std::vector< String >(), const std::vector< String > &columnUnit=std::vector< String >(), int version=1)

  writing constructor: create new Ascii Table object with the specified columns
- AsciiTable (FITS *p, const String &hduName=String(""), bool readFlag=false, const std::vector< String > &keys=std::vector< String >(), int version=1)

  reading constructor: Construct a AsciiTable extension from an extension of an existing disk file.
- AsciiTable (FITS *p, int number)

  read AsciiTable with HDU number `number` from existing file.
- ~AsciiTable ()

  destructor.

Additional Inherited Members

8.1.1 Detailed Description

Class Representing Ascii Table Extensions.

May only contain columns with scalar row entries and a small range of data types. AsciiTable (re)implements functions prescribed in the Table abstract class. The implementations allow the calling of cfitsio specialized routines for AsciiTable header construction.

Direct instantiation of AsciiTable objects is disallowed: they are created by explicit calls to FITS::addTable(...), FITS::read(...) or internally by one of the FITS ctors on initialization. The default for FITS::addTable is to produce BinTable extensions.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 AsciiTable() [1/3]

```
CCfits::AsciiTable::AsciiTable (
        FITS * p,
        const String & hduName = String(""),
        bool readFlag = false,
        const std::vector< String > & keys = std::vector<String>(),
        int version = 1 )  [protected]
```

reading constructor: Construct a AsciiTable extension from an extension of an existing disk file.

The Table is specified by name and optional version number within the file. An array of strings representing columns or keys indicates which data are to be read. The column data are only read if readFlag is true. Reading on construction is optimized, so it is more efficient to read data at the point of instantiation. This favours a "resource acquisition is initialization" model of data management.

Parameters

p	pointer to FITS object for internal use
hduName	name of AsciiTable object to be read.
readFlag	flag to determine whether to read data on construction
keys	(optional) a list of keywords/columns to be read. The implementation will determine which are keywords. If none are specified, the constructor will simply read the header
version	(optional) version number. If not specified, will read the first extension that matches hduName.

8.1.2.2 AsciiTable() [2/3]

```
CCfits::AsciiTable::AsciiTable (
        FITS * p,
        const String & hduName,
```

```
int rows,
const std::vector< String > & columnName = std::vector<String>(),
const std::vector< String > & columnFmt = std::vector<String>(),
const std::vector< String > & columnUnit = std::vector<String>(),
int version = 1 )  [protected]
```

writing constructor: create new Ascii Table object with the specified columns

The constructor creates a valid HDU which is ready for Column::write or insertRows operations. The disk FITS file is update accordingly. The data type of each column is determined by the columnFmt argument (TFORM keywords). See cfitsio documentation for acceptable values.

Parameters

hduName	name of AsciiTable object to be written
rows	number of rows in the table (NAXIS2)
columnName	array of column names for columns to be constructed.
columnFmt	array of column formats for columns to be constructed.
columnUnit	(optional) array of units for data in columns.
version	(optional) version number for HDU.

The dimensions of columnType, columnName and columnFmt must match, although this is not enforced at present.

Todo

8.1.2.3 AsciiTable() [3/3] CCfits::AsciiTable::AsciiTable (

```
FITS * p,
int number )  [protected]
```

read AsciiTable with HDU number `number` from existing file.

This is used internally by methods that need to access HDUs for which no EXTNAME [or equivalent] keyword exists.

8.1.3 Member Function Documentation

8.1.3.1 addColumn() void CCfits::AsciiTable::addColumn (

```
ValueType type,
const String & columnName,
long repeatWidth,
const String & colUnit = String(""),
long decimals = 0,
size_t columnNumber = 0 )  [virtual]
```

add a new column to an existing table HDU.

Parameters

type	The data type of the column to be added
columnName	The name of the column to be added
repeatWidth	for a string valued, this is the width of a string. For a numeric column it supplies the vector length of the rows. It is ignored for ascii table numeric data.
colUnit	an optional field specifying the units of the data (TUNITn)
decimals	optional parameter specifying the number of decimals for an ascii numeric column
columnNumber	optional parameter specifying column number to be created. If not specified the column is added to the end. If specified, the column is inserted and the columns already read are reindexed. This parameter is provided as a convenience to support existing code rather than recommended.

Reimplemented from CCfits::ExtHDU.

8.1.3.2 readData() `void CCfits::AsciiTable::readData ( bool readFlag = false, const std::vector< String > & keys = std::vector<String>() ) [virtual]`

read columns and keys specified in the input array.

See Table class documentation for further details.

Implements CCfits::ExtHDU.

The documentation for this class was generated from the following files:

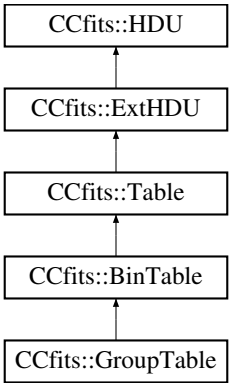
- AsciiTable.h
- AsciiTable.cxx

8.2 CCfits::BinTable Class Reference

Class Representing Binary Table Extensions. Contains columns with scalar or vector row entries.

`#include <BinTable.h>`

Inheritance diagram for CCfits::BinTable:



Public Member Functions

- virtual void addColumn (ValueType type, const String &columnName, long repeatWidth, const String &col↩ Unit=String(""), long decimals=0, size_t columnNumber=0)

  add a new column to an existing table HDU.
- virtual BinTable * clone (FITS *p) const

  virtual copy constructor
- virtual void readData (bool readFlag=false, const std::vector< String > &keys=std::vector< String >())

  read columns and keys specified in the input array.

Protected Member Functions

- BinTable (FITS *p, const String &hduName, int rows, const std::vector< String > &columnName=std::vector< String >(), const std::vector< String > &columnFmt=std::vector< String >(), const std::vector< String > &columnUnit=std::vector< String >(), int version=1)

  writing constructor
- BinTable (FITS *p, const String &hduName=String(""), bool readFlag=false, const std::vector< String > &keys=std::vector< String >(), int version=1)

  reading constructor.
- BinTable (FITS *p, int number)

  read BinTable with HDU number `number` from existing file represented by fitsfile pointer p.
- ~BinTable ()

  destructor.

Additional Inherited Members

8.2.1 Detailed Description

Class Representing Binary Table Extensions. Contains columns with scalar or vector row entries.

BinTable (re)implements functions prescribed in the Table abstract class. The implementations allow the calling of cfitsio specialized routines for BinTable header construction. functions particular to the BinTable class include those dealing with variable width columns

Direct instantiation of BinTable objects is disallowed: they are created by explicit calls to FITS::addTable(...), FITS↩ ::read(...) or internally by one of the FITS ctors on initialization. For addTable, creation of BinTables is the default.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 BinTable() [1/2]

```
CCfits::BinTable::BinTable (
        FITS * p,
        const String & hduName = String(""),
        bool readFlag = false,
        const std::vector< String > & keys = std::vector<String>(),
        int version = 1 )  [protected]
```

reading constructor.

Construct a BinTable extension from an extension of an existing disk file. The Table is specified by name and optional version number within the file. An array of strings representing columns or keys indicates which data are to be read. The column data are only read if readFlag is true. Reading on construction is optimized, so it is more efficient to read data at the point of instantiation. This favours a "resource acquisition is initialization" model of data management.

Parameters

p	Pointer to FITS class, an internal implementation detail
hduName	name of BinTable object to be read.
readFlag	flag to determine whether to read data on construction
keys	(optional) a list of keywords/columns to be read. The implementation will determine which are keywords. If none are specified, the constructor will simply read the header
version	(optional) version number. If not specified, will read the first extension that matches hduName.

8.2.2.2 BinTable() [2/2]

```
CCfits::BinTable::BinTable (
        FITS * p,
        const String & hduName,
        int rows,
        const std::vector< String > & columnName = std::vector<String>(),
        const std::vector< String > & columnFmt = std::vector<String>(),
        const std::vector< String > & columnUnit = std::vector<String>(),
        int version = 1 )  [protected]
```

writing constructor

The constructor creates a valid HDU which is ready for Column::write or insertRows operations. The disk FITS file is update accordingly. The data type of each column is determined by the columnFmt argument (TFORM keywords). See cfitsio documentation for acceptable values.

Parameters

p	Pointer to FITS class, an internal implementation detail
hduName	name of BinTable object to be written
rows	number of rows in the table (NAXIS2)
columnName	array of column names for columns to be constructed.
columnFmt	array of column formats for columns to be constructed.
columnUnit	(optional) array of units for data in columns.
version	(optional) version number for HDU.

The dimensions of columnType, columnName and columnFmt must match, but this is not enforced.

8.2.3 Member Function Documentation

8.2.3.1 addColumn()

```
void CCfits::BinTable::addColumn (
        ValueType type,
        const String & columnName,
```

```
        long repeatWidth,
        const String & colUnit = String(""),
        long decimals = 0,
        size_t columnNumber = 0 )  [virtual]
```

add a new column to an existing table HDU.

Parameters

type	The data type of the column to be added
columnName	The name of the column to be added
repeatWidth	for a string valued, this is the width of a string. For a numeric column it supplies the vector length of the rows. It is ignored for ascii table numeric data.
colUnit	an optional field specifying the units of the data (TUNITn)
decimals	optional parameter specifying the number of decimals for an ascii numeric column
columnNumber	optional parameter specifying column number to be created. If not specified the column is added to the end. If specified, the column is inserted and the columns already read are reindexed. This parameter is provided as a convenience to support existing code rather than recommended.

Reimplemented from CCfits::ExtHDU.

8.2.3.2 readData()

```
void CCfits::BinTable::readData (
        bool readFlag = false,
        const std::vector< String > & keys = std::vector<String>() )  [virtual]
```

read columns and keys specified in the input array.

See Table class documentation for further details.

Implements CCfits::ExtHDU.

The documentation for this class was generated from the following files:

- BinTable.h
- BinTable.cxx

8.3 CCfits::Column Class Reference

Abstract base class for Column objects.

```
#include <Column.h>
```

Inherited by CCfits::ColumnData< T >, and CCfits::ColumnVectorData< T >.

Classes

- class InsufficientElements

  Exception thrown if the data supplied for a write operation is less than declared.
- class InvalidDataType

  Exception thrown for invalid data type inputs.
- class InvalidNumberOfRows

  Exception thrown if user enters a non-positive number for the number of rows to write.
- class InvalidRowNumber

  Exception thrown on attempting to read a row number beyond the end of a table.
- class InvalidRowParameter

  Exception thrown on incorrect row writing request.
- class NoNullValue

  Exception thrown if a null value is specified without support from existing column header.
- class RangeError

  exception to be thrown for inputs that cause range errors in column read operations.
- class WrongColumnType

  Exception thrown on attempting to access a scalar column as vector data.

Public Member Functions

- Column (const Column &right)

  copy constructor, used in copying Columns to standard library containers.
- virtual ~Column ()

  destructor.
- template<typename T >
  void addNullValue (T nullVal)

  Set the TNULLn keyword for the column.
- const String & dimen () const

  return TDIMn keyword
- const String & display () const

  return TDISPn keyword
- const String & format () const

  return TFORMn keyword
- template<typename T >
  bool getNullValue (T *nullVal) const

  Get the value of the TNULLn keyword for the column.
- int index () const

  get the Column index (the n in TTYPEn etc).
- bool isRead () const

  flag set to true if the entire column data has been read from disk
- const String & name () const

  return name of Column (TTYPEn keyword)
- Table * parent () const

  return a pointer to the Table which owns this Column
- template<typename S >
  void read (std::valarray< S > &vals, long first, long last)

Retrieve data from a scalar column into a std::valarray.

- template<typename S >
  void read (std::valarray< S > &vals, long first, long last, S *nullValue)

  Retrieve data from a scalar column into a std::valarray, applying nullValue when relevant.
- template<typename S >
  void read (std::valarray< S > &vals, long rows)

  return a single row of a vector column into a std::valarray
- template<typename S >
  void read (std::valarray< S > &vals, long rows, S *nullValue)

  return a single row of a vector column into a std::valarray, setting undefined values
- template<typename S >
  void read (std::vector< S > &vals, long first, long last)

  Retrieve data from a scalar column into a std::vector.
- template<typename S >
  void read (std::vector< S > &vals, long first, long last, S *nullValue)

  Retrieve data from a scalar column into a std::vector>, applying nullValue when relevant.
- template<typename S >
  void read (std::vector< S > &vals, long rows)

  return a single row of a vector column into a std::vector
- template<typename S >
  void readArrays (std::vector< std::valarray< S > > &vals, long first, long last)

  return a set of rows of a vector column into a vector of valarrays
- template<typename S >
  void readArrays (std::vector< std::valarray< S > > &vals, long first, long last, S *nullValue)

  return a set of rows of a vector column into a container, setting undefined values
- virtual void readData (long firstRow, long nelements, long firstElem=1)=0

  Read (or reread) data from the disk into the Column object's internal arrays.
- size_t repeat () const

  get the repeat count for the rows
- void resetRead ()

  reset the Column's isRead flag to false
- int rows () const

  return the number of rows in the table.
- double scale () const

  get TSCALn value
- virtual void setDimen ()

  set the TDIMn keyword.
- void setDisplay ()

  set the TDISPn keyword
- ValueType type () const

  returns the data type of the column
- const String & unit () const

  get units of data in Column (TUNITn keyword)
- bool varLength () const

  boolean, set to true if Column has variable length vector rows.
- long width () const

  return column data width

- template<typename S >
  void write (const std::valarray< S > &indata, const std::vector< long > &vectorLengths, long firstRow)

  write a valarray of values into a column with specified number of entries written per row.
- template<typename S >
  void write (const std::valarray< S > &indata, long firstRow)

  write a valarray of values into a scalar column starting with firstRow
- template<typename S >
  void write (const std::valarray< S > &indata, long firstRow, S *nullValue)

  write a valarray of values into a scalar column starting with firstRow, replacing elements equal to nullValue with the FITS null value.
- template<typename S >
  void write (const std::valarray< S > &indata, long nRows, long firstRow)

  write a valarray of values into a range of rows of a vector column.
- template<typename S >
  void write (const std::valarray< S > &indata, long nRows, long firstRow, S *nullValue)

  write a valarray of values into a range of rows of a vector column.
- template<typename S >
  void write (const std::vector< S > &indata, const std::vector< long > &vectorLengths, long firstRow)

  write a vector of values into a column with specified number of entries written per row.
- template<typename S >
  void write (const std::vector< S > &indata, long firstRow)

  write a vector of values into a scalar column starting with firstRow
- template<typename S >
  void write (const std::vector< S > &indata, long firstRow, S *nullValue)

  write a vector of values into a scalar column starting with firstRow, replacing elements equal to nullValue with the FITS null value.
- template<typename S >
  void write (const std::vector< S > &indata, long nRows, long firstRow)

  write a vector of values into a range of rows of a vector column
- template<typename S >
  void write (const std::vector< S > &indata, long nRows, long firstRow, S *nullValue)

  write a vector of values into a range of rows of a vector column, processing undefined values
- template<typename S >
  void write (S *indata, long nElements, const std::vector< long > &vectorLengths, long firstRow)

  write a C-array of values of size nElements into a vector column with specified number of entries written per row.
- template<typename S >
  void write (S *indata, long nElements, long nRows, long firstRow)

  write a C array of values into a range of rows of a vector column
- template<typename S >
  void write (S *indata, long nElements, long nRows, long firstRow, S *nullValue)

  write a C array of values into a range of rows of a vector column, processing undefined values.
- template<typename S >
  void write (S *indata, long nRows, long firstRow)

  write a C array of size nRows into a scalar Column starting with row firstRow.
- template<typename S >
  void write (S *indata, long nRows, long firstRow, S *nullValue)

  write a C array into a scalar Column, processing undefined values.
- template<typename S >
  void writeArrays (const std::vector< std::valarray< S > > &indata, long firstRow)

write a vector of valarray objects to the column, starting at row firstRow >= 1

- template<typename S >
  void writeArrays (const std::vector< std::valarray< S > > &indata, long firstRow, S *nullValue)

  write a vector of valarray objects to the column, starting at row firstRow >= 1, processing undefined values
- double zero () const

  get TZEROn value

Protected Member Functions

- Column (int columnIndex, const String &columnName, ValueType type, const String &format, const String &unit, Table *p, int rpt=1, long w=1, const String &comment="")

  new column creation constructor
- Column (Table *p=0)

  Simple constructor to be called by subclass reading ctors.
- const String & comment () const

  retrieve comment for Column
- fitsfile * fitsPointer ()

  fits pointer corresponding to fits file containing column data.
- void makeHDUCurrent ()

  make HDU containing this the current HDU of the fits file.
- virtual std::ostream & put (std::ostream &s) const

  internal implementation of << operator.

8.3.1 Detailed Description

Abstract base class for Column objects.

Columns are the data containers used in FITS tables. Columns of scalar type (one entry per cell) are implemented by the template subclass ColumnData<T>. Columns of vector type (vector and variable rows) are implemented with the template subclass ColumnVectorData<T>. AsciiTables may only contain Columns of type ColumnData<T>, where T is an implemented FITS data type (see the CCfits.h header for a complete list. This requirement is enforced by ensuring that AsciiTable's addColumn method may only create an AsciiTable compatible column. The ColumnData<T> class stores its data in a std::vector<T> object.

BinTables may contain either ColumnData<T> or ColumnVectorData<T>. For ColumnVectorData, T must be a numeric type: string vectors are handled by ColumnData<T>; string arrays are not supported. The internal representation of the data is a std::vector<std::valarray<T> > object. The std::valarray class is designed for efficient numeric processing and has many vectorized numeric and transcendental functions defined on it.

Member template functions for read/write operations are provided in multiple overloads as the interface to data operations. Implicit data type conversions are supported but where they are required make the operations less efficient. Reading numeric column data as character arrays, supported by cfitsio, is not supported by CCfits.

As a base class, Column provides protected accessor/mutator inline functions to allow only its subclasses to access data members.

8.3.2 Constructor & Destructor Documentation

8.3.2.1 Column() [1/2] CCfits::Column::Column (

```
const Column & right )
```

copy constructor, used in copying Columns to standard library containers.

The copy constructor is for internal use only: it does not affect the disk fits file associated with the object.

8.3.2.2 Column() [2/2] CCfits::Column::Column (

```
int columnIndex,
const String & columnName,
ValueType type,
const String & format,
const String & unit,
Table * p,
int rpt = 1,
long w = 1,
const String & comment = "" )  [protected]
```

new column creation constructor

This constructor allows the specification of:

Parameters

columnIndex	The column number
columnName	The column name, keyword TTYPEn
type	used for determining class of T in ColumnData<T>, ColumnVectorData<T>
format	the column data format, TFORMn keyword
unit	the column data unit, TUNITn keyword
p	the Table pointer
rpt	(optional) repeat count for the row (== 1 for AsciiTables)
w	the row width
comment	comment to be added to the header.

8.3.3 Member Function Documentation

8.3.3.1 addNullValue() template<typename T >

```
void CCfits::Column::addNullValue (
T nullVal )
```

Set the TNULLn keyword for the column.

Only relevant for integer valued columns, TNULLn is the value used by cfitsio in undefined processing. All entries in the table equal to an input "null value" are set equal to the value of TNULLn. (For floating point columns a system NaN value is used).

8.3.3.2 dimen() const String & CCfits::Column::dimen ( ) const [inline]

return TDIMn keyword

represents dimensions of data arrays in vector columns. for scalar columns, returns a default value.

8.3.3.3 display() const String & CCfits::Column::display ( ) const [inline]

return TDISPn keyword

TDISPn is suggested format for output of column data.

8.3.3.4 format() const String & CCfits::Column::format ( ) const [inline]

return TFORMn keyword

TFORMn specifies data format stored in disk file.

8.3.3.5 getNullValue()

```
template<typename T >
bool CCfits::Column::getNullValue (
            T * nullVal ) const
```

Get the value of the TNULLn keyword for the column.

Only relevant for integer valued columns. If the TNULLn keyword is present, its value will be written to ∗*nullVal* and the function returns *true*. If the keyword is not found or its value is undefined, the function returns *false* and ∗*nullVal* is not modified.

Parameters

nullVal	A pointer to the variable for storing the TNULLn value.

8.3.3.6 read() [1/7]

```
template<typename S >
void CCfits::Column::read (
            std::valarray< S > & vals,
            long first,
            long last )
```

Retrieve data from a scalar column into a std::valarray.

Parameters

vals	The output container. The function will resize this as necessary
first,last	the span of row numbers to read.

8.3.3.7 read() [2/7] template<typename S >

```
void CCfits::Column::read (
            std::valarray< S > & vals,
            long first,
            long last,
            S * nullValue )
```

Retrieve data from a scalar column into a std::valarray, applying nullValue when relevant.

If both *nullValue* and ∗*nullValue* are not 0, then any undefined values in the file will be converted to ∗*nullValue* when copied into the *vals* valarray. See cfitsio documentation for further details

Parameters

vals	The output container. The function will resize this as necessary
first,last	the span of row numbers to read.
nullValue	pointer to value to be applied to undefined elements.

8.3.3.8 read() [3/7] template<typename S >

```
void CCfits::Column::read (
            std::valarray< S > & vals,
            long row )
```

return a single row of a vector column into a std::valarray

Parameters

vals	The output valarray object
row	The row number to be retrieved (starting at 1).

8.3.3.9 read() [4/7] template<typename S >

```
void CCfits::Column::read (
            std::valarray< S > & vals,
            long rows,
            S * nullValue )
```

return a single row of a vector column into a std::valarray, setting undefined values

return a single row of a vector column into a std::vector, setting undefined values

8.3.3.10 read() [5/7] template<typename S >

```
void CCfits::Column::read (
             std::vector< S > & vals,
             long first,
             long last )
```

Retrieve data from a scalar column into a std::vector.

This and the following functions perform implicit data conversions. An exception will be thrown if no conversion exists.

Parameters

vals	The output container. The function will resize this as necessary
first,last	the span of row numbers to read.

8.3.3.11 read() [6/7] template<typename S >

```
void CCfits::Column::read (
             std::vector< S > & vals,
             long first,
             long last,
             S * nullValue )
```

Retrieve data from a scalar column into a std::vector>, applying nullValue when relevant.

If both *nullValue* and ∗*nullValue* are not 0, then any undefined values in the file will be converted to ∗*nullValue* when copied into the *vals* vector. See cfitsio documentation for further details

Parameters

vals	The output container. The function will resize this as necessary
first,last	the span of row numbers to read.
nullValue	pointer to value to be applied to undefined elements.

8.3.3.12 read() [7/7] template<typename S >

```
void CCfits::Column::read (
             std::vector< S > & vals,
             long row )
```

return a single row of a vector column into a std::vector

Parameters

vals	The output vector object
row	The row number to be retrieved (starting at 1).

8.3.3.13 readArrays() [1/2] template<typename S >

```
void CCfits::Column::readArrays (
            std::vector< std::valarray< S > > & vals,
            long first,
            long last )
```

return a set of rows of a vector column into a vector of valarrays

Parameters

vals	The output container. The function will resize this as necessary
first,last	the span of row numbers to read.

8.3.3.14 readArrays() [2/2] template<typename S >

```
void CCfits::Column::readArrays (
            std::vector< std::valarray< S > > & vals,
            long first,
            long last,
            S * nullValue )
```

return a set of rows of a vector column into a container, setting undefined values

Parameters

vals	The output container. The function will resize this as necessary
first,last	the span of row numbers to read.
nullValue	pointer to integer value regarded as undefined

8.3.3.15 readData()

```
void CCfits::Column::readData (
            long firstRow = 1,
            long nelements = 1,
            long firstElem = 1 )  [pure virtual]
```

Read (or reread) data from the disk into the Column object's internal arrays.

This function normally does not need to be called. See the **resetRead** function for an alternative way of performing a reread from disk.

Parameters

firstRow	The first row to be read
nelements	The number of elements to read
firstElem	The number of the element on the first row to start at (ignored for scalar columns)

8.3.3.16 resetRead() `void CCfits::Column::resetRead ( ) [inline]`

reset the Column's isRead flag to false

This forces the data to be reread from the disk the next time a read command is called on the Column, rather than simply retrieving the data already stored in the Column object's internal arrays. This may be useful for example if trying to reread a Column using a different nullValue argument than for an earlier read.

8.3.3.17 rows() `int CCfits::Column::rows ( ) const`

return the number of rows in the table.

return number of rows in the Column

8.3.3.18 scale() `double CCfits::Column::scale ( ) const [inline]`

get TSCALn value

TSCALn is used to convert a data array represented on disk in integer format as floating. Useful for compact storage of digitized data.

8.3.3.19 write() [1/15] `template<typename S >`

```
void CCfits::Column::write (
            const std::valarray< S > & indata,
            const std::vector< long > & vectorLengths,
            long firstRow )
```

write a valarray of values into a column with specified number of entries written per row.

Data are written into vectorLengths.size() rows, with vectorLength[n] elements written to row n+firstRow -1. Although primarily intended for wrapping calls to multiple variable-width vector column rows, it may also be used to write a variable number of elements to fixed-width column rows.

When writing to fixed-width column rows, if the number of elements sent to a particular row are fewer than the column's repeat value, the remaining elements in the row will not be modified.

Since cfitsio does not support null value processing for variable width columns this function and its variants do not have version which process undefined values

Parameters

indata	The data to be written
vectorLengths	the number of elements to write to each successive row.
firstRow	the first row to be written.

8.3.3.20 write() [2/15] template<typename S >

```
void CCfits::Column::write (
            const std::valarray< S > & indata,
            long firstRow )
```

write a valarray of values into a scalar column starting with firstRow

Parameters

indata	The data to be written.
firstRow	The first row to be written

8.3.3.21 write() [3/15] template<typename S >

```
void CCfits::Column::write (
            const std::valarray< S > & indata,
            long firstRow,
            S * nullValue )
```

write a valarray of values into a scalar column starting with firstRow, replacing elements equal to nullValue with the FITS null value.

If *nullValue* is not 0, the appropriate FITS null value will be substituted for all elements of *indata* equal to $*nullValue$. For integer type columns there must be a pre-existing TNULLn keyword to define the FITS null value, otherwise a FitsError exception is thrown. For floating point columns, the FITS null is the IEEE NaN (Not-a-Number) value. See the cfitsio fits_write_colnull function documentation for more details.

Parameters

indata	The data to be written.
firstRow	The first row to be written
nullValue	Pointer to the value for which equivalent *indata* elements will be replaced in the file with the appropriate FITS null value.

8.3.3.22 write() [4/15] template<typename S >

```
void CCfits::Column::write (
            const std::valarray< S > & indata,
            long nRows,
            long firstRow )
```

write a valarray of values into a range of rows of a vector column.

The primary use of this is for fixed width columns, in which case Column's repeat attribute is used to determine how many elements are written to each row; if indata.size() is too small an exception will be thrown. If the column is variable width, the call will write indata.size()/nRows elements to each row.

Parameters

indata	The data to be written.
nRows	the number of rows to which to write the data.
firstRow	The first row to be written

8.3.3.23 write() [5/15] template<typename S >

```
void CCfits::Column::write (
            const std::valarray< S > & indata,
            long nRows,
            long firstRow,
            S * nullValue )
```

write a valarray of values into a range of rows of a vector column.

see version without undefined processing for details.

8.3.3.24 write() [6/15] template<typename S >

```
void CCfits::Column::write (
            const std::vector< S > & indata,
            const std::vector< long > & vectorLengths,
            long firstRow )
```

write a vector of values into a column with specified number of entries written per row.

Intended for writing a varying number of elements to multiple rows in a vector column, this may be used for either variable or fixed-width columns. See the indata valarray version of this function for a complete description.

8.3.3.25 write() [7/15] template<typename S >

```
void CCfits::Column::write (
            const std::vector< S > & indata,
            long firstRow )
```

write a vector of values into a scalar column starting with firstRow

Parameters

indata	The data to be written.
firstRow	The first row to be written

8.3.3.26 write() [8/15] template<typename S >

```
void CCfits::Column::write (
```

```
            const std::vector< S > & indata,
            long firstRow,
            S * nullValue )
```

write a vector of values into a scalar column starting with firstRow, replacing elements equal to nullValue with the FITS null value.

If *nullValue* is not 0, the appropriate FITS null value will be substituted for all elements of *indata* equal to ∗*nullValue*. For integer type columns there must be a pre-existing TNULLn keyword to define the FITS null value, otherwise a FitsError exception is thrown. For floating point columns, the FITS null is the IEEE NaN (Not-a-Number) value. See the cfitsio fits_write_colnull function documentation for more details.

Parameters

indata	The data to be written.
firstRow	The first row to be written
nullValue	Pointer to the value for which equivalent *indata* elements will be replaced in the file with the appropriate FITS null value.

8.3.3.27 write() [9/15] template<typename S >

```
void CCfits::Column::write (
            const std::vector< S > & indata,
            long nRows,
            long firstRow )
```

write a vector of values into a range of rows of a vector column

The primary use of this is for fixed width columns, in which case Column's repeat attribute is used to determine how many elements are written to each row; if indata.size() is too small an exception will be thrown. If the column is variable width, the call will write indata.size()/nRows elements to each row.

Parameters

indata	The data to be written.
nRows	the number of rows to which to write the data.
firstRow	The first row to be written

8.3.3.28 write() [10/15] template<typename S >

```
void CCfits::Column::write (
            const std::vector< S > & indata,
            long nRows,
            long firstRow,
            S * nullValue )
```

write a vector of values into a range of rows of a vector column, processing undefined values

see version without undefined processing for details.

8.3.3.29 write() [11/15] template<typename S >

```
void CCfits::Column::write (
            S * indata,
            long nElements,
            const std::vector< long > & vectorLengths,
            long firstRow )
```

write a C-array of values of size nElements into a vector column with specified number of entries written per row.

Intended for writing a varying number of elements to multiple rows in a vector column, this may be used for either variable or fixed-width columns. See the indata valarray version of this function for a complete description.

8.3.3.30 write() [12/15] template<typename S >

```
void CCfits::Column::write (
            S * indata,
            long nElements,
            long nRows,
            long firstRow )
```

write a C array of values into a range of rows of a vector column

Details are as for vector input; only difference is the need to supply the size of the C-array.

Parameters

indata	The data to be written.
nElements	The size of indata
nRows	the number of rows to which to write the data.
firstRow	The first row to be written

8.3.3.31 write() [13/15] template<typename S >

```
void CCfits::Column::write (
            S * indata,
            long nElements,
            long nRows,
            long firstRow,
            S * nullValue )
```

write a C array of values into a range of rows of a vector column, processing undefined values.

see version without undefined processing for details.

8.3.3.32 write() [14/15] template<typename S >

```
void CCfits::Column::write (
            S * indata,
            long nRows,
            long firstRow )
```

write a C array of size nRows into a scalar Column starting with row firstRow.

Parameters

indata	The data to be written.
nRows	The size of the data array to be written
firstRow	The first row to be written

8.3.3.33 write() [15/15] template<typename S >

```
void CCfits::Column::write (
            S * indata,
            long nRows,
            long firstRow,
            S * nullValue )
```

write a C array into a scalar Column, processing undefined values.

Parameters

indata	The data to be written.
nRows	The size of the data array to be written
firstRow	The first row to be written
nullValue	Pointer to the value in the input array to be set to undefined values

8.3.3.34 writeArrays() [1/2] template<typename S >

```
void CCfits::Column::writeArrays (
            const std::vector< std::valarray< S > > & indata,
            long firstRow )
```

write a vector of valarray objects to the column, starting at row firstRow >= 1

Intended for writing a varying number of elements to multiple rows in a vector column, this may be used for either variable or fixed-width columns. When writing to fixed-width column rows, if the number of elements sent to a particular row are fewer than the column's repeat value, the remaining elements in the row will not be modified.

Parameters

indata	The data to be written
firstRow	the first row to be written.

8.3.3.35 writeArrays() [2/2] template<typename S >

```
void CCfits::Column::writeArrays (
```

```
const std::vector< std::valarray< S > > & indata,
long firstRow,
S * nullValue )
```

write a vector of valarray objects to the column, starting at row firstRow >= 1, processing undefined values

see version without undefined processing for details.

8.3.3.36 zero() `double CCfits::Column::zero ( ) const [inline]`

get TZEROn value

TZEROn is an integer offset used in the implementation of unsigned data

The documentation for this class was generated from the following files:

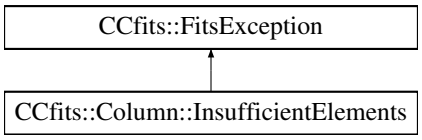
- Column.h
- Column.cxx
- ColumnT.h

8.4 CCfits::Column::InsufficientElements Class Reference

Exception thrown if the data supplied for a write operation is less than declared.

`#include <Column.h>`

Inheritance diagram for CCfits::Column::InsufficientElements:



Public Member Functions

- InsufficientElements (const String &msg, bool silent=true)

  Exception ctor, prefixes the string "FitsError: not enough elements supplied for write operation: " before the specific message.

8.4.1 Detailed Description

Exception thrown if the data supplied for a write operation is less than declared.

This circumstance generates an exception to avoid unexpected behaviour after the write operation is completed. It can be avoided by resizing the input array appropriately.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 InsufficientElements() `CCfits::Column::InsufficientElements::InsufficientElements (`

```
const String & msg,
bool silent = true )
```

Exception ctor, prefixes the string "FitsError: not enough elements supplied for write operation: " before the specific message.

Parameters

msg	A specific diagnostic message, usually the column name
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

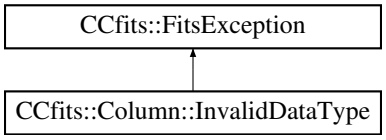
- Column.h
- Column.cxx

8.5 CCfits::Column::InvalidDataType Class Reference

Exception thrown for invalid data type inputs.

`#include <Column.h>`

Inheritance diagram for CCfits::Column::InvalidDataType:



Public Member Functions

- InvalidDataType (const String &str=string(), bool silent=true)

  Exception ctor, prefixes the string "FitsError: Incorrect data type: " before the specific message.

8.5.1 Detailed Description

Exception thrown for invalid data type inputs.

This exception is thrown if the user requests an implicit data type conversion to a datatype that is not one of the supported types (see fitsio.h for details).

8.5.2 Constructor & Destructor Documentation

8.5.2.1 InvalidDataType() `CCfits::Column::InvalidDataType::InvalidDataType (`

```
const String & str = string(),
bool silent = true )
```

Exception ctor, prefixes the string "FitsError: Incorrect data type: " before the specific message.

Parameters

str	A specific diagnostic message
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

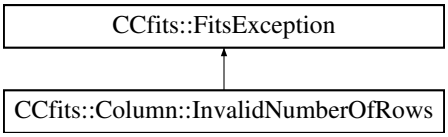
- Column.h
- Column.cxx

8.6 CCfits::Column::InvalidNumberOfRows Class Reference

Exception thrown if user enters a non-positive number for the number of rows to write.

`#include <Column.h>`

Inheritance diagram for CCfits::Column::InvalidNumberOfRows:



Public Member Functions

- InvalidNumberOfRows (int number, bool silent=true)

  Exception ctor, prefixes the string "Fits Error: number of rows to write must be positive " before the specific message.

8.6.1 Detailed Description

Exception thrown if user enters a non-positive number for the number of rows to write.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 InvalidNumberOfRows() `CCfits::Column::InvalidNumberOfRows::InvalidNumberOfRows (`

```
int number,
bool silent = true )
```

Exception ctor, prefixes the string "Fits Error: number of rows to write must be positive " before the specific message.

Parameters

number	The number of rows entered.
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

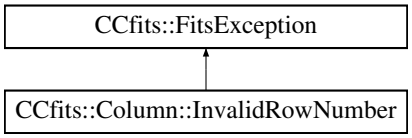
- Column.h
- Column.cxx

8.7 CCfits::Column::InvalidRowNumber Class Reference

Exception thrown on attempting to read a row number beyond the end of a table.

`#include <Column.h>`

Inheritance diagram for CCfits::Column::InvalidRowNumber:



Public Member Functions

- InvalidRowNumber (const String &diag, bool silent=true)

  Exception ctor, prefixes the string "FitsError: Invalid Row Number - Column: " before the specific message.

8.7.1 Detailed Description

Exception thrown on attempting to read a row number beyond the end of a table.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 InvalidRowNumber() `CCfits::Column::InvalidRowNumber::InvalidRowNumber (`

```
const String & diag,
bool silent = true )
```

Exception ctor, prefixes the string "FitsError: Invalid Row Number - Column: " before the specific message.

Parameters

diag	A specific diagnostic message, usually the column name.
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

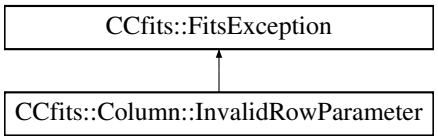
- Column.h
- Column.cxx

8.8 CCfits::Column::InvalidRowParameter Class Reference

Exception thrown on incorrect row writing request.

`#include <Column.h>`

Inheritance diagram for CCfits::Column::InvalidRowParameter:



Public Member Functions

- InvalidRowParameter (const String &diag, bool silent=true)

  Exception ctor, prefixes the string "FitsError: row offset or length incompatible with column declaration " before the specific message.

8.8.1 Detailed Description

Exception thrown on incorrect row writing request.

This exception is thrown if the user requests writing more data than a fixed width row can accommodate. An exception is thrown rather than a truncation because it is likely that the user will not otherwise realize that data loss is happening.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 InvalidRowParameter()

```
CCfits::Column::InvalidRowParameter::InvalidRowParameter (
        const String & diag,
        bool silent = true )
```

Exception ctor, prefixes the string "FitsError: row offset or length incompatible with column declaration " before the specific message.

Parameters

diag	A specific diagnostic message, usually the column name
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

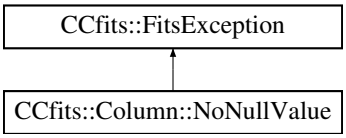
- Column.h
- Column.cxx

8.9 CCfits::Column::NoNullValue Class Reference

Exception thrown if a null value is specified without support from existing column header.

`#include <Column.h>`

Inheritance diagram for CCfits::Column::NoNullValue:



Public Member Functions

- NoNullValue (const String &diag, bool silent=true)

  Exception ctor, prefixes the string "Fits Error: No null value specified for column: " before the specific message.

8.9.1 Detailed Description

Exception thrown if a null value is specified without support from existing column header.

This exception is analogous to the fact that cfitsio returns a non-zero status code if TNULLn doesn't exist an a null value (convert all input data with the null value to the TNULLn keyword) is specified. It is only relevant for integer type data (see cfitsio manual for details).

8.9.2 Constructor & Destructor Documentation

8.9.2.1 NoNullValue() `CCfits::Column::NoNullValue::NoNullValue (`

```
const String & diag,
bool silent = true )
```

Exception ctor, prefixes the string "Fits Error: No null value specified for column: " before the specific message.

Parameters

diag	A specific diagnostic message
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

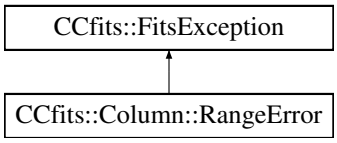
- Column.h
- Column.cxx

8.10 CCfits::Column::RangeError Class Reference

exception to be thrown for inputs that cause range errors in column read operations.

`#include <Column.h>`

Inheritance diagram for CCfits::Column::RangeError:



Public Member Functions

- RangeError (const String &msg, bool silent=true)
  Exception ctor, prefixes the string "FitsError: Range error in operation " before the specific message.

8.10.1 Detailed Description

exception to be thrown for inputs that cause range errors in column read operations.

Range errors here mean (last < first) in a request to read a range of rows.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 RangeError() `CCfits::Column::RangeError::RangeError (`

```
const String & msg,
bool silent = true )
```

Exception ctor, prefixes the string "FitsError: Range error in operation " before the specific message.

Parameters

msg	A specific diagnostic message
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

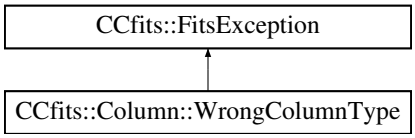
- Column.h
- Column.cxx

8.11 CCfits::Column::WrongColumnType Class Reference

Exception thrown on attempting to access a scalar column as vector data.

`#include <Column.h>`

Inheritance diagram for CCfits::Column::WrongColumnType:



Public Member Functions

- WrongColumnType (const String &diag, bool silent=true)

  Exception ctor, prefixes the string "FitsError: Attempt to return scalar data from vector column, or vice versa - Column: " before the specific message.

8.11.1 Detailed Description

Exception thrown on attempting to access a scalar column as vector data.

This exception will be thrown if the user tries to call a read/write operation with a signature appropriate for a vector column on a scalar column, or vice versa. For example in the case of write operations, the vector versions require the specification of (a) a number of rows to write over, (b) a vector of lengths to write to each row or (c) a subset specification. The scalar versions only require a number of rows if the input array is a plain C-array, otherwise the range to be written is the size of the input vector.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 WrongColumnType() `CCfits::Column::WrongColumnType::WrongColumnType (`

```
const String & diag,
bool silent = true )
```

Exception ctor, prefixes the string "FitsError: Attempt to return scalar data from vector column, or vice versa - Column: " before the specific message.

Parameters

diag	A specific diagnostic message, usually the column name.
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

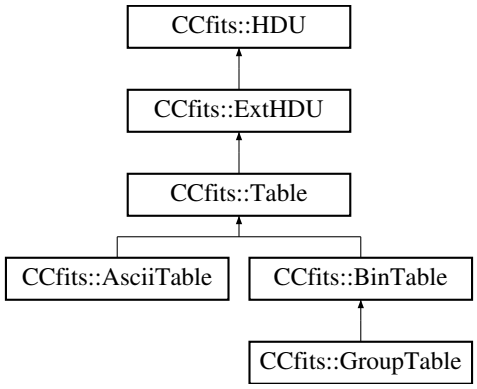
- Column.h
- Column.cxx

8.12 CCfits::ExtHDU Class Reference

base class for all FITS extension HDUs, i.e. Image Extensions and Tables.

`#include <ExtHDU.h>`

Inheritance diagram for CCfits::ExtHDU:



Classes

- class WrongExtensionType

  Exception to be thrown on unmatched extension types.

Public Member Functions

- ExtHDU (const ExtHDU &right)

  copy constructor
- virtual ~ExtHDU ()

  destructor
- virtual void addColumn (ValueType type, const String &columnName, long repeatWidth, const String &col↩ Unit=String(""), long decimals=-1, size_t columnNumber=0)

  add a new column to an existing table HDU.
- virtual HDU * clone (FITS *p) const =0

virtual copy constructor

- virtual const ColMap & column () const

  return a reference to the multimap containing the columns.
- virtual Column & column (const String &colName, bool caseSensitive=true) const

  return a reference to a Table column specified by name.
- virtual Column & column (int colIndex) const

  return a reference to a Table column specified by column index.
- virtual void copyColumn (const Column &inColumn, int colIndx, bool insertNewCol=true)

  copy a column (from different or same HDU and file) into an existing table HDU.
- virtual void deleteColumn (const String &columnName)

  delete a column in a Table extension by name.
- virtual long getRowsize () const

  return the optimal number of rows to read or write at a time
- bool isCompressed () const

  return true if image is stored using compression.
- virtual void makeThisCurrent () const

  move the fitsfile pointer to this current HDU.
- const String & name () const

  return the name of the extension.
- virtual int numCols () const

  return the number of Columns in the Table (the TFIELDS keyword).
- template<typename S >
  void read (std::valarray< S > &image)

  Read image data into container.
- template<typename S >
  void read (std::valarray< S > &image, const std::vector< long > &first, long nElements)

  read an image section starting at a location specified by an n-tuple
- template<typename S >
  void read (std::valarray< S > &image, const std::vector< long > &first, long nElements, S *nullValue)

  read part of an image array, processing null values.
- template<typename S >
  void read (std::valarray< S > &image, const std::vector< long > &firstVertex, const std::vector< long > &last↩
  Vertex, const std::vector< long > &stride)

  read an image subset
- template<typename S >
  void read (std::valarray< S > &image, const std::vector< long > &firstVertex, const std::vector< long > &last↩
  Vertex, const std::vector< long > &stride, S *nullValue)

  read an image subset into valarray image, processing null values
- template<typename S >
  void read (std::valarray< S > &image, long first, long nElements)

  read an image section starting at a specified pixel
- template<typename S >
  void read (std::valarray< S > &image, long first, long nElements, S *nullValue)

  read part of an image array, processing null values.
- virtual void readData (bool readFlag=false, const std::vector< String > &keys=std::vector< String >())=0

  read data from HDU depending on readFlag and keys.
- virtual long rows () const

  return the number of rows in the extension.

- int version () const

  return the extension version number.
- void version (int value)

  set the extension version number
- template<typename S >
  void write (const std::vector< long > &first, long nElements, const std::valarray< S > &data)

  write array starting from specified n-tuple, without undefined data processing
- template<typename S >
  void write (const std::vector< long > &first, long nElements, const std::valarray< S > &data, S *nullValue)

  Write a set of pixels to an image extension with the first pixel specified by an n-tuple, processing undefined data.
- template<typename S >
  void write (const std::vector< long > &firstVertex, const std::vector< long > &lastVertex, const std::valarray< S > &data)

  write a subset (generalize slice) of data to the image
- template<typename S >
  void write (long first, long nElements, const std::valarray< S > &data)

  write array starting from specified pixel number, without undefined data processing
- template<typename S >
  void write (long first, long nElements, const std::valarray< S > &data, S *nullValue)

  write array to image starting with a specified pixel and allowing undefined data to be processed

Static Public Member Functions

- static void readHduName (const fitsfile *fptr, int hduIndex, String &hduName, int &hduVersion)

  read extension name.

Protected Member Functions

- ExtHDU (FITS *p, HduType xtype, const String &hduName, int bitpix, int naxis, const std::vector< long > &axes, int version)

  writing constructor.
- ExtHDU (FITS *p, HduType xtype, const String &hduName, int version)

  default constructor, required as Standard Library Container content.
- ExtHDU (FITS *p, HduType xtype, int number)

  ExtHDU constructor for getting ExtHDUs by number.
- long gcount () const

  return required gcount keyword value
- void gcount (long value)

  set required gcount keyword value
- long pcount () const

  return required pcount keyword value
- void pcount (long value)

  set required pcount keyword value
- HduType xtension () const

  return the extension type
- void xtension (HduType value)

  set the extension type

8.12.1 Detailed Description

base class for all FITS extension HDUs, i.e. Image Extensions and Tables.

ExtHDU needs to have the combined public interface of Table objects and images. It achieves this by providing the same set of read and write operations as PHDU, and also providing the same operations for extracting columns from the extension as does Table [after which the column interface is accessible]. Differentiation between extension types operates by exception handling: .i.e. attempting to access image data structures on a Table object through the ExtHDU interface will or trying to return a Column reference from an Image extension will both throw an exception

8.12.2 Constructor & Destructor Documentation

8.12.2.1 ExtHDU() [1/2]

```
CCfits::ExtHDU::ExtHDU (
        FITS * p,
        HduType xtype,
        const String & hduName,
        int bitpix,
        int naxis,
        const std::vector< long > & axes,
        int version )  [protected]
```

writing constructor.

The writing constructor forces the user to supply a name for the HDU. The bitpix, naxes and naxis data required by this constructor are required FITS keywords for any HDUs.

8.12.2.2 ExtHDU() [2/2]

```
CCfits::ExtHDU::ExtHDU (
        FITS * p,
        HduType xtype,
        int number )  [protected]
```

ExtHDU constructor for getting ExtHDUs by number.

Necessary since EXTNAME is a reserved, not required, keyword. But a synthetic name is supplied by static ExtHDU::readHduName which is called by this constructor.

8.12.3 Member Function Documentation

8.12.3.1 addColumn()

```
void CCfits::ExtHDU::addColumn (
        ValueType type,
        const String & columnName,
        long repeatWidth,
        const String & colUnit = String(""),
        long decimals = -1,
        size_t columnNumber = 0 )  [virtual]
```

add a new column to an existing table HDU.

Parameters

type	The data type of the column to be added
columnName	The name of the column to be added
repeatWidth	for a string valued, this is the width of a string. For a numeric column it supplies the vector length of the rows. It is ignored for ascii table numeric data.
colUnit	an optional field specifying the units of the data (TUNITn)
decimals	optional parameter specifying the number of decimals for an ascii numeric column
columnNumber	optional parameter specifying column number to be created. If not specified the column is added to the end. If specified, the column is inserted and the columns already read are reindexed. This parameter is provided as a convenience to support existing code rather than recommended.

Reimplemented in CCfits::BinTable, and CCfits::AsciiTable.

8.12.3.2 column() [1/3] const ColMap & CCfits::ExtHDU::column ( ) const [virtual]

return a reference to the multimap containing the columns.

Exceptions

WrongExtensionType	thrown if ∗this is an image extension.

Reimplemented in CCfits::Table.

8.12.3.3 column() [2/3] Column & CCfits::ExtHDU::column (const String & *colName*, bool *caseSensitive* = *true*) const [virtual]

return a reference to a Table column specified by name.

If the *caseSensitive* parameter is set to false, the search will be case-insensitive. The overridden base class implementation ExtHDU::column throws an exception, which is thus the action to be taken if self is an image extension

Exceptions

WrongExtensionType	see above

Reimplemented in CCfits::Table.

8.12.3.4 column() [3/3]

```
Column & CCfits::ExtHDU::column (
        int colIndex ) const  [virtual]
```

return a reference to a Table column specified by column index.

This version is provided for convenience; the 'return by name' version is more efficient because columns are stored in an associative array sorted by name.

Exceptions

WrongExtensionType	thrown if *this is an image extension.

Reimplemented in CCfits::Table.

8.12.3.5 copyColumn()

```
void CCfits::ExtHDU::copyColumn (
        const Column & inColumn,
        int colIndx,
        bool insertNewCol = true )  [virtual]
```

copy a column (from different or same HDU and file) into an existing table HDU.

This is meant to provide the same functionality as CFITSIO's fits_copy_col, and therefore does not work with columns with variable length fields. Copying a column from an AsciiTable to a BinTable is prohibited. colIndx range should be from 1 to nCurrentCols+1 if inserting, or 1 to nCurrentCols if replacing.

Parameters

inColumn	The Column object which is to be copied
colIndx	The position for which the copied Column will be placed (first colIndx = 1).
insertNewCol	If 'true', new Column will be inserted in or appended to table. If 'false', Column will replace current Column at position = colIndx.

Reimplemented in CCfits::Table.

8.12.3.6 deleteColumn()

```
void CCfits::ExtHDU::deleteColumn (
        const String & columnName )  [virtual]
```

delete a column in a Table extension by name.

Parameters

columnName	The name of the column to be deleted.

Exceptions

WrongExtensionType	if extension is an image.

Reimplemented in CCfits::Table.

8.12.3.7 getRowsize() `long CCfits::ExtHDU::getRowsize ( ) const [virtual]`

return the optimal number of rows to read or write at a time

A wrapper for the CFITSIO function fits_get_rowsize, useful for obtaining maximum I/O efficiency. This will throw if it is not called for a Table extension.

Reimplemented in CCfits::Table.

8.12.3.8 isCompressed() `bool CCfits::ExtHDU::isCompressed ( ) const`

return true if image is stored using compression.

This is simply a wrapper around the CFITSIO fits_is_compressed_image function. It will throw if this is not an Image extension.

8.12.3.9 makeThisCurrent() `void CCfits::ExtHDU::makeThisCurrent ( ) const [virtual]`

move the fitsfile pointer to this current HDU.

This function should never need to be called by the user since it is called internally whenever required.

Reimplemented from CCfits::HDU.

8.12.3.10 numCols() `int CCfits::ExtHDU::numCols ( ) const [virtual]`

return the number of Columns in the Table (the TFIELDS keyword).

Exceptions

WrongExtensionType	thrown if ∗this is an image extension.

Reimplemented in CCfits::Table.

8.12.3.11 read() [1/4] template<typename S >

```
void CCfits::ExtHDU::read (
            std::valarray< S > & image )
```

Read image data into container.

The container image contains the entire image array after the call. This and all the other variants of read() throw a WrongExtensionType exception if called for a Table object.

8.12.3.12 read() [2/4] template<typename S >

```
void CCfits::ExtHDU::read (
            std::valarray< S > & image,
            const std::vector< long > & first,
            long nElements,
            S * nullValue )
```

read part of an image array, processing null values.

As above except for

Parameters

first	a vector<long> representing an n-tuple giving the coordinates in the image of the first pixel.

8.12.3.13 read() [3/4] template<typename S >

```
void CCfits::ExtHDU::read (
            std::valarray< S > & image,
            const std::vector< long > & firstVertex,
            const std::vector< long > & lastVertex,
            const std::vector< long > & stride,
            S * nullValue )
```

read an image subset into valarray image, processing null values

The image subset is defined by two vertices and a stride indicating the 'denseness' of the values to be picked in each dimension (a stride = (1,1,1,...) means picking every pixel in every dimension, whereas stride = (2,2,2,...) means picking every other value in each dimension.

8.12.3.14 read() [4/4] template<typename S >

```
void CCfits::ExtHDU::read (
            std::valarray< S > & image,
            long first,
            long nElements,
            S * nullValue )
```

read part of an image array, processing null values.

Implicit data conversion is supported (i.e. user does not need to know the type of the data stored. A WrongExtensionType extension is thrown if ∗this is not an image.

Parameters

image	The receiving container, a std::valarray reference
first	The first pixel from the array to read [a long value]
nElements	The number of values to read
nullValue	A pointer containing the value in the table to be considered as undefined. See cfitsio for details

8.12.3.15 readHduName()

```
static void CCfits::ExtHDU::readHduName (
        const fitsfile * fptr,
        int hduIndex,
        String & hduName,
        int & hduVersion )  [static]
```

read extension name.

Used primarily to allow extensions to be specified by HDU number and provide their name for the associative array that contains them. Alternatively, if there is no name keyword in the extension, one is synthesized from the index.

8.12.3.16 rows()

```
long CCfits::ExtHDU::rows ( ) const  [virtual]
```

return the number of rows in the extension.

Exceptions

WrongExtensionType	thrown if ∗this is an image extension.

Reimplemented in CCfits::Table.

8.12.3.17 write() [1/3]

```
template<typename S >
void CCfits::ExtHDU::write (
        const std::vector< long > & first,
        long nElements,
        const std::valarray< S > & data,
        S * nullValue )
```

Write a set of pixels to an image extension with the first pixel specified by an n-tuple, processing undefined data.

All the overloaded versions of ExtHDU::write perform operations on ∗this if it is an image and throw a WrongExtensionType exception if not. Where appropriate, alternate versions allow undefined data to be processed

Parameters

first	an n-tuple of dimension equal to the image dimension specifying the first pixel in the range to be written
nElements	number of pixels to be written
data	array of data to be written
nullValue	pointer to null value (data with this value written as undefined; needs the BLANK keyword to have been specified).

8.12.3.18 write() [2/3] template<typename S >

```
void CCfits::ExtHDU::write (
            const std::vector< long > & firstVertex,
            const std::vector< long > & lastVertex,
            const std::valarray< S > & data )
```

write a subset (generalize slice) of data to the image

A generalized slice/subset is a subset of the image (e.g. one plane of a data cube of size <= the dimension of the cube). It is specified by two opposite vertices. The equivalent cfitsio call does not support undefined data processing so there is no version that allows a null value to be specified.

Parameters

firstVertex	the coordinates specifying lower and upper vertices of the n-dimensional slice
lastVertex	
data	The data to be written

8.12.3.19 write() [3/3] template<typename S >

```
void CCfits::ExtHDU::write (
            long first,
            long nElements,
            const std::valarray< S > & data,
            S * nullValue )
```

write array to image starting with a specified pixel and allowing undefined data to be processed

parameters after the first are as for version with n-tuple specifying first element. these two version are equivalent, except that it is possible for the first pixel number to exceed the range of 32-bit integers, which is how long datatype is commonly implemented.

8.12.3.20 xtension() `HduType CCfits::ExtHDU::xtension ( ) const [inline], [protected]`

return the extension type

allowed values are ImageHDU, AsciiTbl, and BinaryTbl

The documentation for this class was generated from the following files:

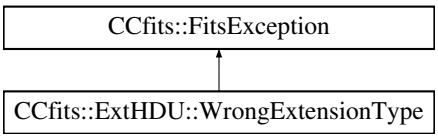
- ExtHDU.h
- ExtHDU.cxx
- ExtHDUT.h

8.13 CCfits::ExtHDU::WrongExtensionType Class Reference

Exception to be thrown on unmatched extension types.

`#include <ExtHDU.h>`

Inheritance diagram for CCfits::ExtHDU::WrongExtensionType:



Public Member Functions

- WrongExtensionType (const String &msg, bool silent=true)
  Exception ctor, prefixes the string "Fits Error: wrong extension type" before the specific message.

8.13.1 Detailed Description

Exception to be thrown on unmatched extension types.

This exception is to be thrown if the user requested a particular extension and it does not correspond to the expected type.

8.13.2 Constructor & Destructor Documentation

8.13.2.1 WrongExtensionType() `CCfits::ExtHDU::WrongExtensionType::WrongExtensionType ( const String & msg, bool silent = true )`

Exception ctor, prefixes the string "Fits Error: wrong extension type" before the specific message.

Parameters

msg	A specific diagnostic message
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

- ExtHDU.h
- ExtHDU.cxx

8.14 CCfits::FITS Class Reference

Memory object representation of a disk FITS file.

```
#include <FITS.h>
```

Classes

- class CantCreate

  thrown on failure to create new file
- class CantOpen

  thrown on failure to open existing file
- class NoSuchHDU

  exception thrown by HDU retrieval methods.
- class OperationNotSupported

  thrown for unsupported operations, such as attempted to select rows from an image extension.

Public Member Functions

- FITS (const String &fileName, const FITS &source)

  Create a new FITS object and corresponding file with a copy of the primary header of the source. If the filename corresponds to an existing file and does not start with the '!' character, the construction will fail with a CantCreate exception.
- FITS (const String &name, int bitpix, int naxis, long *naxes)

  Constructor for creating new FITS objects containing images.
- FITS (const String &name, RWmode rwmode, const std::vector< String > &hduNames, bool readDataFlag=false, const std::vector< String > &primaryKey=std::vector< String >())
- FITS (const String &name, RWmode rwmode, const std::vector< String > &hduNames, const std::vector< std::vector< String > > &hduKeys, bool readDataFlag=false, const std::vector< String > &primaryKeys=std↩ ::vector< String >(), const std::vector< int > &hduVersions=std::vector< int >())

  FITS read constructor in full generality.
- FITS (const String &name, RWmode rwmode, const std::vector< String > &searchKeys, const std::vector< String > &searchValues, bool readDataFlag=false, const std::vector< String > &hduKeys=std::vector< String >(), const std::vector< String > &primaryKey=std::vector< string >(), int version=1)

  open fits file and read HDU which contains supplied keywords with [optional] specified values (sometimes one just wants to know that the keyword is present).

- FITS (const String &name, RWmode rwmode, const string &hduName, bool readDataFlag=false, const std↩ ::vector< String > &hduKeys=std::vector< String >(), const std::vector< String > &primaryKey=std::vector< String >(), int version=1)

  Open a FITS file and read a single specified HDU.
- FITS (const string &name, RWmode rwmode, int hduIndex, bool readDataFlag=false, const std::vector< String > &hduKeys=std::vector< String >(), const std::vector< String > &primaryKey=std::vector< String >())

  read a single numbered HDU.
- FITS (const String &name, RWmode rwmode=Read, bool readDataFlag=false, const std::vector< String > &primaryKeys=std::vector< String >())

  basic constructor
- ~FITS ()

  destructor
- ExtHDU * addImage (const String &hduName, int bpix, std::vector< long > &naxes, int version=1)

  Add an image extension to an existing FITS object. (File with w or rw access).
- Table * addTable (const String &hduName, int rows, const std::vector< String > &columnName=std::vector< String >(), const std::vector< String > &columnFmt=std::vector< String >(), const std::vector< String > &columnUnit=std::vector< String >(), HduType type=BinaryTbl, int version=1)

  Add a table extension to an existing FITS object. Add extension to FITS object for file with w or rw access.
- void copy (const HDU &source)

  copy the HDU source into the FITS object.
- ExtHDU & currentExtension ()

  return a non-const reference to whichever is the current extension.
- const String & currentExtensionName () const

  return the name of the extension that the fitsfile is currently addressing.
- void deleteExtension (const String &doomed, int version=1)

  Delete extension specified by name and version number.
- void deleteExtension (int doomed)

  Delete extension specified by extension number.
- void destroy () throw ()

  Erase FITS object and close corresponding file.
- const ExtMap & extension () const

  return const reference to the extension container
- ExtHDU & extension (const String &hduName, int version=1)

  return FITS extension by name and (optionally) version number.
- const ExtHDU & extension (const String &hduName, int version=1) const

  return FITS extension by name and (optionally) version number.
- ExtHDU & extension (int i)

  return FITS extension by index number. non-const version. see const version for details.
- const ExtHDU & extension (int i) const

  return FITS extension by index number. N.B. The input index number is currently defined as enumerating extensions, so the extension(1) returns HDU number 2.
- Table & filter (const String &expression, ExtHDU &inputTable, bool overwrite=true, bool readData=false)

  Filter the rows of the inputTable with the condition expression, and return a reference to the resulting Table.
- fitsfile * fitsPointer () const

  return the CFITSIO fitsfile pointer for this FITS object
- void flush ()

  flush buffer contents to disk
- int getCompressionType () const

Get the int specifying the compression algorithm to be used when adding an image extension.

- int getNoiseBits () const

  Get the cfitsio noisebits parameter used when compressing floating-point images.
- void getTileDimensions (std::vector< long > &tileSizes) const

  Get the current settings of dimension sizes for tiles used in image compression.
- const String & name () const

  return filename of file corresponding to FITS object
- PHDU & pHDU ()

  return a reference to the primary HDU.
- const PHDU & pHDU () const

  return a const reference to the primary HDU.
- void read (const std::vector< String > &hduNames, bool readDataFlag=false)

  get data from a set of HDUs from disk file.
- void read (const std::vector< String > &hduNames, const std::vector< std::vector< String > > &keys, bool readDataFlag=false, const std::vector< int > &hduVersions=std::vector< int >())

  get data from a set of HDUs from disk file, specifying keys and version numbers.
- void read (const std::vector< String > &searchKeys, const std::vector< String > &searchValues, bool read↩ DataFlag=false, const std::vector< String > &hduKeys=std::vector< String >(), int version=1)

  read method for read header or HDU that contains specified keywords.
- void read (const String &hduName, bool readDataFlag=false, const std::vector< String > &keys=std::vector< String >(), int version=1)

  get data from single HDU from disk file.
- void read (int hduIndex, bool readDataFlag=false, const std::vector< String > &keys=std::vector< String >())

  read an HDU specified by index number.
- void resetPosition ()

  explicit call to set the fits file pointer to the primary.
- void setCompressionType (int compType)

  set the compression algorithm to be used when adding image extensions to the FITS object.
- void setNoiseBits (int noiseBits)

  Set the cfitsio noisebits parameter used when compressing floating-point images.
- void setTileDimensions (const std::vector< long > &tileSizes)

  Set the dimensions of the tiles into which the image is divided during compression.

Static Public Member Functions

- static void clearErrors ()

  clear the error stack and set status to zero.
- static void setVerboseMode (bool value)

  set verbose setting for library
- static bool verboseMode ()

  return verbose setting for library

8.14.1 Detailed Description

Memory object representation of a disk FITS file.

Constructors are provided to get FITS data from an existing file or to create new FITS data sets. Overloaded versions allow the user to:

(a) read from one or more specified extensions, specified by EXTNAME and VERSION or by HDU number. (b) read either just header information or data on construction (c) specify scalar keyword values to be read on construction (d) open and read an extension that has specified keyword values (e) create a new FITS object and corresponding file, including an empty primary header.

The memory fits object as constructed is always an image of a valid FITS object, i.e. a primary HDU is created on construction.

Calling the destructor closes the disk file, so that FITS files are automatically deleted at the end of scope unless other arrangements are made

8.14.2 Constructor & Destructor Documentation

8.14.2.1 FITS() [1/8]

```
CCfits::FITS::FITS (
        const String & name,
        RWmode rwmode = Read,
        bool readDataFlag = false,
        const std::vector< String > & primaryKeys = std::vector<String>() )
```

basic constructor

This basic constructor makes a FITS object from the given filename. The file name is the only required argument. The file name string is passed directly to the cfitsio library: thus the extended file name syntax described in the cfitsio manual should work as documented. (Though the extended file name feature which allows the opening of a particular image located in the row of a table is currently unsupported.)

If the rwmode is Read [default]: It will read all of the headers in the file, and all of the data if the readDataFlag is supplied as true. It will also read optional primary keys. Upon completion, the the last header in the file will become the current extension. (However if the file name includes extended syntax selecting a particular extension, that extension will be the current one.)

If the rwmode is Write and the file already exists: The file is opened in read-write mode, all of the headers of the file are read, and all of the data if the readDataFlag is supplied as true. It will also read optional primary keys. For backwards compatibility with older versions of CCfits (which only read the primary when in Write mode for pre-existing files), the primary will become the current extension.

If the rwmode is Write and the file does NOT exist (or is overwritten using '!' syntax): A default primary HDU will be created in the file with BITPIX=8 and NAXIS=0. However if you wish to create a new file with image data in the primary, the version of the FITS constructor that specifies the data type and number of axes should be used instead.

Parameters

name	The name of the FITS file to be read/written
rwmode	The read/write mode: must be Read or Write
readDataFlag	boolean: read data on construction if true
primaryKeys	Allows optional reading of primary header keys on construction

Exceptions

NoSuchHDU	thrown on HDU seek error either by index or {name,version}
FitsError	thrown on non-zero status code from cfitsio when not overriden by FitsException error to produce more illuminating message.

8.14.2.2 FITS() [2/8] CCfits::FITS::FITS (

```
const String & name,
RWmode rwmode,
const string & hduName,
bool readDataFlag = false,
const std::vector< String > & hduKeys = std::vector<String>(),
const std::vector< String > & primaryKey = std::vector<String>(),
int version = 1 )
```

Open a FITS file and read a single specified HDU.

This and similar constructor variants support reading table data.

Optional arguments allow the reading of primary header keys and specified data from hduName, the HDU to be read. An object representing the primary HDU is always created: if it contains an image, that image may be read by subsequent calls.

If extended file name syntax is used and selects an extension other than hduName, a FITS::OperationNotSupported exception will be thrown.

Parameters

name	The name of the FITS file to be read
rwmode	The read/write mode: takes values Read or Write
hduName	The name of the HDU to be read.
hduKeys	Optional array of keywords to be read from the HDU
version	Optional version number. If not supplied the first HDU with name *hduName* is read see above for other parameter definitions

8.14.2.3 FITS() [3/8] CCfits::FITS::FITS (

```
const String & name,
RWmode rwmode,
const std::vector< String > & hduNames,
bool readDataFlag = false,
const std::vector< String > & primaryKey = std::vector<String>() )
```

This is intended as a convenience where the file consists of single versions of HDUs and data only, not keys are to be read.

If extended file name syntax is used and selects an extension not listed in hduNames, a FITS::OperationNotSupported exception will be thrown.

Parameters

hduNames	array of HDU names to be read.

see above for other parameter definitions.

8.14.2.4 FITS() [4/8] CCfits::FITS::FITS (

```
const String & fileName,
const FITS & source )
```

Create a new FITS object and corresponding file with a copy of the primary header of the source. If the filename corresponds to an existing file and does not start with the '!' character, the construction will fail with a CantCreate exception.

Parameters

fileName	New file to be created.
source	A previously created FITS object to be copied.

8.14.2.5 FITS() [5/8] CCfits::FITS::FITS (

```
const String & name,
RWmode rwmode,
const std::vector< String > & hduNames,
const std::vector< std::vector< String > > & hduKeys,
bool readDataFlag = false,
const std::vector< String > & primaryKeys = std::vector<String>(),
const std::vector< int > & hduVersions = std::vector<int>() )
```

FITS read constructor in full generality.

Parameters

hduVersions	an optional version number for each HDU to be read
hduKeys	an array of keywords for each HDU to be read. see above for other parameter definitions.

8.14.2.6 FITS() [6/8] CCfits::FITS::FITS (

```
const String & name,
int bitpix,
int naxis,
long * naxes )
```

Constructor for creating new FITS objects containing images.

This constructor is only called for creating new files (mode is not an argument) and creates a new primary HDU with the datatype & axes specified by bitpix, naxis, and naxes. The data are added to the new fits object and file by subsequent calls to FITS::pHDU().write(<arguments>)

A file with a compressed image may be creating by appending to the end of the file name the same "[compress ...]" syntax specified in the cfitsio manual. Note however that the compressed image will be placed in the first extension and NOT in the primary HDU.

If the filename corresponds to an existing file and does not start with the '!' character the construction will fail with a CantCreate exception.

The arguments are:

Parameters

name	The file to be written to disk
bitpix	the datatype of the primary image. *bitpix* may be one of the following CFITSIO constants: BYTE_IMG, SHORT_IMG, LONG_IMG, FLOAT_IMG, DOUBLE_IMG, USHORT_IMG, ULONG_IMG, LONGLONG_IMG. Note that if you send in a *bitpix* of USHORT_IMG or ULONG_IMG, CCfits will set HDU::bitpix() to its signed equivalent (SHORT_IMG or LONG_IMG), and then set BZERO to 2^15 or 2^31.
naxis	the data dimension of the primary image
naxes	the array of axis lengths for the primary image. Ignored if naxis =0, i.e. the primary header is empty. extensions can be added arbitrarily to the file after this constructor is called. The constructors should write header information to disk:

8.14.2.7 FITS() [7/8] CCfits::FITS::FITS (

```
const string & name,
RWmode rwmode,
int hduIndex,
bool readDataFlag = false,
const std::vector< String > & hduKeys = std::vector<String>(),
const std::vector< String > & primaryKey = std::vector<String>() )
```

read a single numbered HDU.

Constructor analogous to the version that reads by name. This is required since HDU extensions are not required to have the EXTNAME or HDUNAME keyword by the standard. If there is no name, a dummy name based on the HDU number is created and becomes the key.

If extended file name syntax is used and selects an extension other than hduIndex, a FITS::OperationNotSupported exception will be thrown.

Parameters

hduIndex	The index of the HDU to be read. see above for other parameter definitions.

8.14.2.8 FITS() [8/8]

```
CCfits::FITS::FITS (
        const String & name,
        RWmode rwmode,
        const std::vector< String > & searchKeys,
        const std::vector< String > & searchValues,
        bool readDataFlag = false,
        const std::vector< String > & hduKeys = std::vector<String>(),
        const std::vector< String > & primaryKey = std::vector<string>(),
        int version = 1 )
```

open fits file and read HDU which contains supplied keywords with [optional] specified values (sometimes one just wants to know that the keyword is present).

Optional parameters allows the reading of specified primary HDU keys and specified columns and keywords in the HDU of interest.

Parameters

name	The name of the FITS file to be read
rwmode	The read/write mode: must be Read or Write
searchKeys	A string vector of keywords to search for in each header
searchValues	A string vector of values those keywords are required to have for success. Note that the keys must be of type string. If any value does not need to be checked the corresponding searchValue element can be empty.
readDataFlag	boolean: if true, read data if HDU is found
hduKeys	Allows optional reading of keys in the HDU that is searched for if it is successfully found
primaryKey	Allows optional reading of primary header keys on construction
version	Optional version number. If specified, checks the EXTVER keyword.

Exceptions

FitsError	thrown on non-zero status code from cfitsio when not overriden by FitsException error to produce more illuminating message.

8.14.3 Member Function Documentation

8.14.3.1 addImage()

```
void CCfits::FITS::addImage (
        const String & hduName,
        int bpix,
        std::vector< long > & naxes,
        int version = 1 )
```

Add an image extension to an existing FITS object. (File with w or rw access).

Does not make primary images, which are built in the constructor for the FITS file. The image data is not added here: it can be added by a call to one of the ExtHDU::write functions.

bpix may be one of the following CFITSIO constants: BYTE_IMG, SHORT_IMG, LONG_IMG, FLOAT_IMG, DOUBLE_IMG, USHORT_IMG, ULONG_IMG, LONGLONG_IMG. Note that if you send in a *bpix* of USHORT_IMG or ULONG_IMG, CCfits will set HDU::bitpix() to its signed equivalent (SHORT_IMG or LONG_IMG), and then set BZERO to 2^15 or 2^31.

Todo Add a function for replacing the primary image

8.14.3.2 addTable()

```
Table * CCfits::FITS::addTable (
        const String & hduName,
        int rows,
        const std::vector< String > & columnName = std::vector<String>(),
        const std::vector< String > & columnFmt = std::vector<String>(),
        const std::vector< String > & columnUnit = std::vector<String>(),
        HduType type = BinaryTbl,
        int version = 1 )
```

Add a table extension to an existing FITS object. Add extension to FITS object for file with w or rw access.

Parameters

rows	The number of rows in the table to be created.
columnName	(Optional) A vector containing the table column names
columnFmt	(Optional) A vector containing the table column formats
columnUnit	(Optional) A vector giving the units of the columns.
type	(Optional) The table type - AsciiTbl or BinaryTbl (defaults to BinaryTbl) the lists of columns are optional - one can create an empty table extension but if supplied, colType, columnName and colFmt must have equal dimensions.
version	(Optional) The EXTVER keyword for the extension version number, defaults to 1

Todo the code should one day check that the version keyword is higher than any other versions already added to the FITS object (although cfitsio doesn't do this either).

8.14.3.3 copy() `void CCfits::FITS::copy ( const HDU & source )`

copy the HDU source into the FITS object.

This function adds a copy of an HDU from another file into ∗this. It does not create a duplicate of an HDU in the file associated with ∗this.

8.14.3.4 currentExtensionName() `const String & CCfits::FITS::currentExtensionName ( ) const [inline]`

return the name of the extension that the fitsfile is currently addressing.

If the extension in question does not have an EXTNAME or HDUNAME keyword, then the function returns HDUn, where n is the sequential HDU index number (primary HDU = 0).

8.14.3.5 deleteExtension() [1/2] `void CCfits::FITS::deleteExtension ( const String & doomed, int version = 1 )`

Delete extension specified by name and version number.

Removes extension from FITS object and memory copy. The index numbers of all HDU objects which follow this in the file will be decremented by 1.

Parameters

doomed	the name of the extension to be deleted
version	an optional version number, the EXTVER keyword, defaults to 1

Exceptions

NoSuchHDU	Thrown if there is no extension with the specified version number
FitsError	Thrown if there is a non-zero status code from cfitsio, e.g. if the delete operation is applied to a FITS file opened for read only access.

8.14.3.6 deleteExtension() [2/2] `void CCfits::FITS::deleteExtension ( int doomed )`

Delete extension specified by extension number.

The index numbers of all HDU objects which follow this in the file will be decremented by 1.

8.14.3.7 destroy() `void CCfits::FITS::destroy ( ) throw ( )`

Erase FITS object and close corresponding file.

Force deallocation and erase of elements of a FITS memory object. Allows a reset of everything inside the FITS object, and closes the file. The object is inaccessible after this call.

destroy is public to allow users to reuse a symbol for a new file, but it is identical in operation to the destructor.

8.14.3.8 extension() `const ExtMap & CCfits::FITS::extension ( ) const [inline]`

return const reference to the extension container

This is useful for such operations as extension().size() etc.

8.14.3.9 filter()

```
Table & CCfits::FITS::filter (
        const String & expression,
        ExtHDU & inputTable,
        bool overwrite = true,
        bool readData = false )
```

Filter the rows of the inputTable with the condition expression, and return a reference to the resulting Table.

This function provides an object oriented version of cfitsio's fits_select_rows call. The expression string is any boolean expression involving the names of the columns in the input table (e.g., if there were a column called "density", a valid expression might be "DENSITY > 3.5": see the cfitsio documentation for further details).

[N.B. the "append" functionality described below does not work when linked with cfitsio 2.202 or prior because of a known issue with that version of the library. This causes the output to be a new extension with a correct header copy and version number but without the filtered data]. If the inputTable is an Extension HDU of this FITS object, then if overwrite is true the operation will overwrite the inputTable with the filtered version, otherwise it will append a new HDU with the same extension name but the next highest version (EXTVER) number available.

8.14.3.10 flush() `void CCfits::FITS::flush ( )`

flush buffer contents to disk

Provides manual control of disk writing operation. Image data are flushed automatically to disk after the write operation is completed, but not column data.

8.14.3.11 getTileDimensions()

```
void CCfits::FITS::getTileDimensions (
        std::vector< long > & tileSizes ) const
```

Get the current settings of dimension sizes for tiles used in image compression.

Parameters

tileSizes	A vector to be filled with cfitsio's current tile dimension settings. CCfits will resize this vector to contain the proper number of values.

8.14.3.12 read() [1/5]

```
void CCfits::FITS::read (
    const std::vector< String > & hduNames,
    bool readDataFlag = false )
```

get data from a set of HDUs from disk file.

This is provided to allow reading of HDUs after construction. see above for parameter definitions.

8.14.3.13 read() [2/5]

```
void CCfits::FITS::read (
    const std::vector< String > & hduNames,
    const std::vector< std::vector< String > > & keys,
    bool readDataFlag = false,
    const std::vector< int > & hduVersions = std::vector<int>() )
```

get data from a set of HDUs from disk file, specifying keys and version numbers.

This is provided to allow reading of HDUs after construction. see above for parameter definitions.

8.14.3.14 read() [3/5]

```
CCfits::FITS::read (
    const std::vector< String > & searchKeys,
    const std::vector< String > & searchValues,
    bool readDataFlag = false,
    const std::vector< String > & hduKeys = std::vector<String>(),
    int version = 1 )
```

read method for read header or HDU that contains specified keywords.

Parameters

searchKeys	A string vector of keywords to search for in each header
searchValues	A string vector of values those keywords are required to have for success. Note that the keys must be of type string. If any value does not need to be checked the corresponding searchValue element can be empty.
readDataFlag	boolean: if true, read data if HDU is found
hduKeys	Allows optional reading of keys in the HDU that is searched for if it is successfully found
version	Optional version number. If specified, checks the EXTVER keyword.

8.14.3.15 read() [4/5]

```
void CCfits::FITS::read (
    const String & hduName,
    bool readDataFlag = false,
    const std::vector< String > & keys = std::vector<String>(),
    int version = 1 )
```

get data from single HDU from disk file.

This is provided to allow the adding of additional HDUs to the FITS object after construction of the FITS object. After the read() functions have been called for the FITS object, subsequent read method to the Primary, ExtHDU, and Column objects will retrieve data from the FITS object in memory (those methods can be called to read data in those HDU objects that was not read when the HDU objects were constructed.

All the read functions will throw NoSuchHDU exceptions on seek errors since they involve constructing HDU objects.

The parameter definitions are as documented for the corresponding constructor.

8.14.3.16 read() [5/5] void CCfits::FITS::read (

```
int hduIndex,
bool readDataFlag = false,
const std::vector< String > & keys = std::vector<String>() )
```

read an HDU specified by index number.

This is provided to allow reading of HDUs after construction. see above for parameter definitions.

8.14.3.17 setCompressionType() void CCfits::FITS::setCompressionType (

```
int compType )
```

set the compression algorithm to be used when adding image extensions to the FITS object.

Parameters

compType	Currently 3 symbolic constants are defined in cfitsio for specifying compression algorithms: GZIP_1, RICE_1, and PLIO_1. See the cfitsio documentation for more information about these algorithms. Entering NULL for compType will turn off compression and cause normal FITS images to be written.

8.14.3.18 setNoiseBits() void CCfits::FITS::setNoiseBits (

```
int noiseBits )
```

Set the cfitsio noisebits parameter used when compressing floating-point images.

The default value is 4. Decreasing the value of noisebits will improve the overall compression efficiency at the expense of losing more information.

8.14.3.19 setTileDimensions() void CCfits::FITS::setTileDimensions (

```
const std::vector< long > & tileSizes )
```

Set the dimensions of the tiles into which the image is divided during compression.

Parameters

tileSizes	A vector of length N containing the tile dimesions. If N is less than the number of dimensions of the image it is applied to, the unspecified dimensions will be assigned a size of 1 pixel. If N is larger than the number of image dimensions, the extra dimensions will be ignored.

The default cfitsio behavior is to create tiles with dimensions NAXIS1 x 1 x 1 etc. up to the number of image dimensions.

8.14.3.20 verboseMode() `bool CCfits::FITS::verboseMode ( ) [inline], [static]`

return verbose setting for library

If true, all messages that are reported by exceptions are printed to std::cerr.

The documentation for this class was generated from the following files:

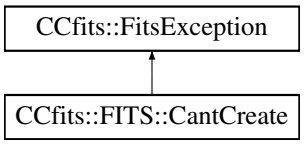
- FITS.h
- FITS.cxx

8.15 CCfits::FITS::CantCreate Class Reference

thrown on failure to create new file

`#include <FITS.h>`

Inheritance diagram for CCfits::FITS::CantCreate:



Public Member Functions

- CantCreate (const String &diag, bool silent=false)

  Exception ctor prefixes the string: "FITS Error: Cannot create file " before specific message.

8.15.1 Detailed Description

thrown on failure to create new file

8.15.2 Constructor & Destructor Documentation

8.15.2.1 CantCreate() `CCfits::FITS::CantCreate::CantCreate (`

```
const String & msg,
bool silent = false )
```

Exception ctor prefixes the string: "FITS Error: Cannot create file " before specific message.

This exception will be thrown if the user attempts to write to a protected directory or attempts to create a new file with the same name as an existing file without specifying overwrite [overwrite is specified by adding the character '!' before the filename, following the cfitsio convention].

Parameters

msg	A specific diagnostic message, the name of the file that was to be created.
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

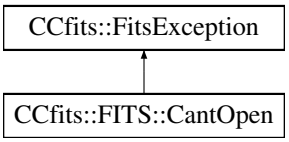
- FITS.h
- FITS.cxx

8.16 CCfits::FITS::CantOpen Class Reference

thrown on failure to open existing file

`#include <FITS.h>`

Inheritance diagram for CCfits::FITS::CantOpen:



Public Member Functions

- CantOpen (const String &diag, bool silent=true)

  Exception ctor prefixes the string: "FITS Error: Cannot create file " before specific message.

8.16.1 Detailed Description

thrown on failure to open existing file

8.16.2 Constructor & Destructor Documentation

8.16.2.1 CantOpen() `CCfits::FITS::CantOpen::CantOpen (`

```
const String & diag,
bool silent = true )
```

Exception ctor prefixes the string: "FITS Error: Cannot create file " before specific message.

This exception will be thrown if users attempt to open an existing file for write access to which they do not have permission, or of course if the file does not exist.

Parameters

diag	A specific diagnostic message, the name of the file that was to be created.
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

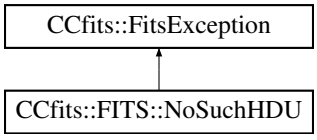
- FITS.h
- FITS.cxx

8.17 CCfits::FITS::NoSuchHDU Class Reference

exception thrown by HDU retrieval methods.

`#include <FITS.h>`

Inheritance diagram for CCfits::FITS::NoSuchHDU:



Public Member Functions

- NoSuchHDU (const String &diag, bool silent=true)

  Exception ctor, prefixes the string "FITS Error: Cannot read HDU in FITS file:" before the specific message.

8.17.1 Detailed Description

exception thrown by HDU retrieval methods.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 NoSuchHDU() `CCfits::FITS::NoSuchHDU::NoSuchHDU (`

```
const String & diag,
bool silent = true )
```

Exception ctor, prefixes the string "FITS Error: Cannot read HDU in FITS file:" before the specific message.

Parameters

diag	A specific diagnostic message, usually the name of the extension whose read was attempted.
silent	if true, print message whether FITS::verboseMode is set or not.

Exception to be thrown by failed seek operations

The documentation for this class was generated from the following files:

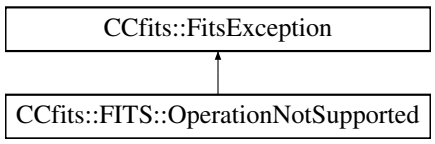
- FITS.h
- FITS.cxx

8.18 CCfits::FITS::OperationNotSupported Class Reference

thrown for unsupported operations, such as attempted to select rows from an image extension.

`#include <FITS.h>`

Inheritance diagram for CCfits::FITS::OperationNotSupported:



Public Member Functions

- OperationNotSupported (const String &msg, bool silent=true)

  Exception ctor, prefixes the string "FITS Error: Operation not supported:" before the specific message.

8.18.1 Detailed Description

thrown for unsupported operations, such as attempted to select rows from an image extension.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 OperationNotSupported()

```
CCfits::FITS::OperationNotSupported::OperationNotSupported (
        const String & msg,
        bool silent = true )
```

Exception ctor, prefixes the string "FITS Error: Operation not supported:" before the specific message.

Parameters

msg	A specific diagnostic message.
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

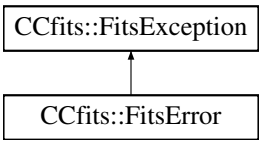
- FITS.h
- FITS.cxx

8.19 CCfits::FitsError Class Reference

FitsError is the exception thrown by non-zero cfitsio status codes.

`#include <FitsError.h>`

Inheritance diagram for CCfits::FitsError:



Public Member Functions

- FitsError (int errornum, bool silent=true)

  ctor for cfitsio exception: translates status code into cfitsio error message

8.19.1 Detailed Description

FitsError is the exception thrown by non-zero cfitsio status codes.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 FitsError() `CCfits::FitsError::FitsError (`

```
int errornum,
bool silent = true )
```

ctor for cfitsio exception: translates status code into cfitsio error message

The exception prefixes the string "Fits Error: " to the message printed by cfitsio.

Parameters

errornum	The cfitsio status code produced by the error.
silent	A boolean controlling the printing of messages

The documentation for this class was generated from the following files:

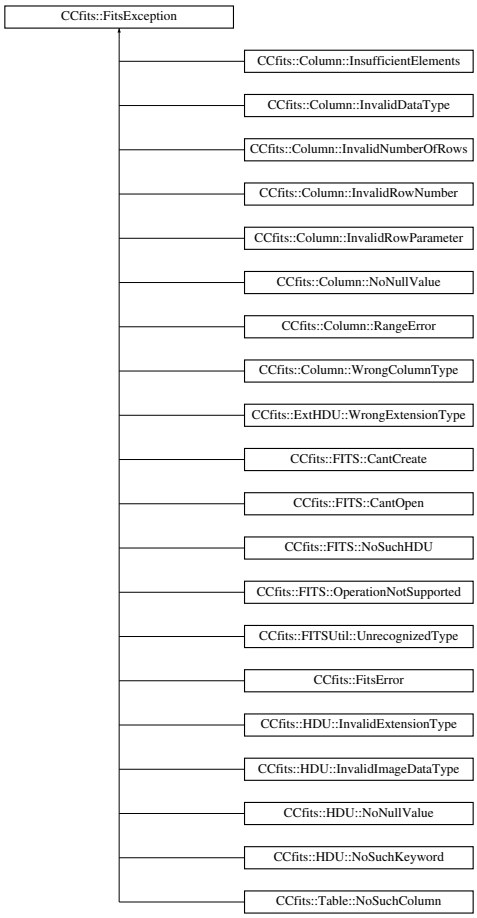
- FitsError.h
- FitsError.cxx

8.20 CCfits::FitsException Class Reference

FitsException is the base class for all exceptions thrown by this library.

`#include <FitsError.h>`

Inheritance diagram for CCfits::FitsException:



Public Member Functions

- FitsException (const string &msg, bool &silent)
- const string & message () const

  returns the error message

8.20.1 Detailed Description

FitsException is the base class for all exceptions thrown by this library.

All exceptions derived from this class can be caught by a single 'catch' clause catching FitsException by reference (which is the point of this base class design).

A static "verboseMode" parameter is provided by the FITS class to control diagnostics - if FITS::verboseMode() is true, all diagnostics are printed (for debugging purposes). If not, then a boolean *silent* determines printing of messages. Each exception derived from FitsException must define a default value for the *silent* parameter.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 FitsException() CCfits::FitsException::FitsException (const string & *diag*, bool & *silent*)

Parameters

diag	A diagnostic string to be printed optionally.
silent	A boolean controlling the printing of messages

8.20.3 Member Function Documentation

8.20.3.1 message() const string & CCfits::FitsException::message ( ) const [inline]

returns the error message

This returns the diagnostic error message associated with the exception object, and which is accessible regardless of the verboseMode and silent flag settings.

The documentation for this class was generated from the following files:

- FitsError.h
- FitsError.cxx

8.21 CCfits::FitsFatal Class Reference

[potential] base class for exceptions to be thrown on internal library error.

`#include <FitsError.h>`

Public Member Functions

- FitsFatal (const string &diag)
  *Prints a message starting "*** CCfits Fatal Error: ..." and calls terminate()*

8.21.1 Detailed Description

[potential] base class for exceptions to be thrown on internal library error.

As of this version there are no subclasses. This error requests that the user reports this circumstance to HEASARC.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 FitsFatal() CCfits::FitsFatal::FitsFatal (const string & *diag*)

Prints a message starting "*** CCfits Fatal Error: ..." and calls *terminate()*

Parameters

diag	A diagnostic string to be printed identifying the context of the error.

The documentation for this class was generated from the following files:

- FitsError.h
- FitsError.cxx

8.22 CCfits::FITSUtil::auto_array_ptr< X > Class Template Reference

A class that mimics the std:: library auto_ptr class, but works with arrays.

`#include <FITSUtil.h>`

Public Member Functions

- auto_array_ptr (auto_array_ptr< X > &right) throw ()
  copy constructor
- auto_array_ptr (X *p=0) throw ()
  constructor. allows creation of pointer to null, can be modified by reset()
- ~auto_array_ptr ()
  destructor.
- X * get () const
  *return a token for the underlying content of *this*
- X & operator* () throw ()
  deference operator
- void operator= (auto_array_ptr< X > &right)
  assignment operator: transfer of ownership semantics
- X & operator[ ] (size_t i) throw ()
  return a reference to the ith element of the array
- X operator[ ] (size_t i) const throw ()
  return a copy of the ith element of the array
- X * release () throw ()
  *return underlying content of *this, transferring memory ownership*
- X * reset (X *p) throw ()
  change the content of the auto_array_ptr to p

Static Public Member Functions

- static void remove (X *&x)
  utility function to delete the memory owned by x and set it to null.

8.22.1 Detailed Description

template<typename X>
class CCfits::FITSUtil::auto_array_ptr< X >

A class that mimics the std:: library auto_ptr class, but works with arrays.

This code was written by Jack Reeves and first appeared C++ Report, March 1996 edition. Although some authors think one shouldn't need such a contrivance, there seems to be a need for it when wrapping C code.

Usage: replace

float* *f* = new float[200];

with

FITSUtil::auto_array_ptr<float> *f(new float[200])*;

Then the memory will be managed correctly in the presence of exceptions, and delete will be called automatically for *f* when leaving scope.

The documentation for this class was generated from the following file:

- FITSUtil.h

8.23 CCfits::FITSUtil::CAarray< T > Class Template Reference

function object returning C array from a valarray. see CVarray for details

`#include <FITSUtil.h>`

Public Member Functions

- T * operator() (const std::valarray< T > &inArray)
  operator returning C array for use with image data.

8.23.1 Detailed Description

template<typename T>
class CCfits::FITSUtil::CAarray< T >

function object returning C array from a valarray. see CVarray for details

The documentation for this class was generated from the following file:

- FITSUtil.h

8.24 CCfits::FITSUtil::CVAarray< T > Class Template Reference

function object returning C array from a vector of valarrays. see CVarray for details

`#include <FITSUtil.h>`

Public Member Functions

- T * operator() (const std::vector< std::valarray< T > > &inArray)
  operator returning C array for use with vector column data.

8.24.1 Detailed Description

template<typename T>
class CCfits::FITSUtil::CVAarray< T >

function object returning C array from a vector of valarrays. see CVarray for details

The documentation for this class was generated from the following file:

- FITSUtil.h

8.25 CCfits::FITSUtil::CVarray< T > Class Template Reference

Function object class for returning C arrays from standard library objects used in the FITS library implementation.

`#include <FITSUtil.h>`

Public Member Functions

- T * operator() (const std::vector< T > &inArray)
  operator returning C array for use with scalar column data.

8.25.1 Detailed Description

template<typename T>
class CCfits::FITSUtil::CVarray< T >

Function object class for returning C arrays from standard library objects used in the FITS library implementation.

There are 3 versions which convert std::vector<T>, std::valarray<T>, and std::vector<std::valarray<T> > objects to pointers to T, called CVarray, CAarray, and CVAarray.

An alternative function, CharArray, is provided to deal with the special case of vector string arrays.

The documentation for this class was generated from the following file:

- FITSUtil.h

8.26 CCfits::FITSUtil::MatchName< T > Class Template Reference

predicate for classes that have a name attribute; match input string with instance name.

`#include <FITSUtil.h>`

8.26.1 Detailed Description

template<class T>
class CCfits::FITSUtil::MatchName< T >

predicate for classes that have a name attribute; match input string with instance name.

Usage: MatchName<NamedClass> Ex;

list<NamedClass> ListObject;

... ...

find_if(ListObject.begin(),ListObject().end(),bind2nd(Ex,"needle"));

Since most of the classes within CCfits are not implemented with lists, these functions are now of little direct use.

The documentation for this class was generated from the following file:

- FITSUtil.h

8.27 CCfits::FITSUtil::MatchNum< T > Class Template Reference

predicate for classes that have an index attribute; match input index with instance value.

`#include <FITSUtil.h>`

8.27.1 Detailed Description

template<class T>
class CCfits::FITSUtil::MatchNum< T >

predicate for classes that have an index attribute; match input index with instance value.

Usage: MatchName<IndexedClass> Ex;

list<NamedClass> ListObject;

... ...

find_if(ListObject.begin(),ListObject().end(),bind2nd(Ex,5));

Since most of the classes within CCfits are implemented with std::maps rather than lists, these functions are now of little direct use.

The documentation for this class was generated from the following file:

- FITSUtil.h

8.28 CCfits::FITSUtil::MatchPtrName< T > Class Template Reference

as for MatchName, only with the input class a pointer.

`#include <FITSUtil.h>`

8.28.1 Detailed Description

template<class T>
class CCfits::FITSUtil::MatchPtrName< T >

as for MatchName, only with the input class a pointer.

The documentation for this class was generated from the following file:

- FITSUtil.h

8.29 CCfits::FITSUtil::MatchPtrNum< T > Class Template Reference

as for MatchNum, only with the input class a pointer.

`#include <FITSUtil.h>`

8.29.1 Detailed Description

template<class T>
class CCfits::FITSUtil::MatchPtrNum< T >

as for MatchNum, only with the input class a pointer.

The documentation for this class was generated from the following file:

- FITSUtil.h

8.30 CCfits::FITSUtil::MatchType< T > Class Template Reference

function object that returns the FITS ValueType corresponding to an input intrinsic type

`#include <FITSUtil.h>`

8.30.1 Detailed Description

template<typename T>
class CCfits::FITSUtil::MatchType< T >

function object that returns the FITS ValueType corresponding to an input intrinsic type

This is particularly useful inside templated class instances where calls to cfitsio need to supply a value type. With this function one can extract the value type from the class type.

usage:

MatchType<T> type;

ValueType dataType = type();

Uses run-time type information (RTTI) methods.

The documentation for this class was generated from the following file:

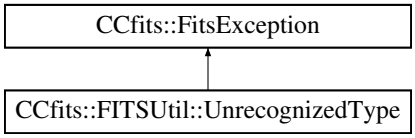
- FITSUtil.h

8.31 CCfits::FITSUtil::UnrecognizedType Class Reference

exception thrown by MatchType if it encounters data type incompatible with cfitsio.

`#include <FITSUtil.h>`

Inheritance diagram for CCfits::FITSUtil::UnrecognizedType:



Additional Inherited Members

8.31.1 Detailed Description

exception thrown by MatchType if it encounters data type incompatible with cfitsio.

The documentation for this class was generated from the following files:

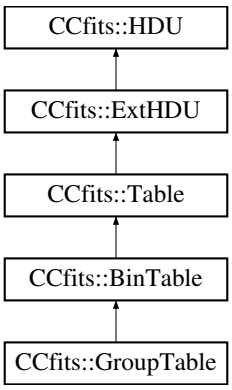
- FITSUtil.h
- FITSUtil.cxx

8.32 CCfits::GroupTable Class Reference

Class representing a hierarchical association of Header Data Units (HDUs).

`#include <GroupTable.h>`

Inheritance diagram for CCfits::GroupTable:



Public Member Functions

- HDU * addMember (HDU &newMember)

  Add a new member to the group table. Adds GRPIDn/GRPLCn keywords to the member HDU.
- HDU * addMember (int memberPosition)

  Add a new member to the group table. Adds GRPIDn/GRPLCn keywords to the member HDU. The member must be in the same file as the group table.
- void listMembers () const

  List group members.

Protected Member Functions

- GroupTable (FITS *p, int groupID, const String &groupName)

  ctor for creating a new group table

Additional Inherited Members

8.32.1 Detailed Description

Class representing a hierarchical association of Header Data Units (HDUs).

Groups of HDUs allow for the hierarchical association of HDUs. Offices may want to group together HDUs in order to organize data files. The associated HDUs need not be in the same FITS file. Group Composites are the holding structure for the group members. Composites may also be members of a group.

The specification for grouping is defined in "A Hierarchical Grouping Convention for FITS" by Jennings, Pence, Folk and Schlesinger at https://fits.gsfc.nasa.gov/registry/grouping/grouping.pdf

8.32.2 Constructor & Destructor Documentation

8.32.2.1 GroupTable() `CCfits::GroupTable::GroupTable (`

```
FITS * p,
int groupID,
const String & groupName )  [protected]
```

ctor for creating a new group table

Parameters

p	The FITS file in which to place the new HDU
groupID	ID of new group table
groupName	Name of new group table

8.32.3 Member Function Documentation

8.32.3.1 addMember() `[1/2]` `HDU * CCfits::GroupTable::addMember (`

```
HDU & newMember )
```

Add a new member to the group table. Adds GRPIDn/GRPLCn keywords to the member HDU.

Parameters

newMember	Member HDU to be added

8.32.3.2 addMember() `[2/2]` `HDU * CCfits::GroupTable::addMember (`

```
int memberPosition )
```

Add a new member to the group table. Adds GRPIDn/GRPLCn keywords to the member HDU. The member must be in the same file as the group table.

Parameters

memberPosition	Position of HDU to add (Primary array == 1)

The documentation for this class was generated from the following files:

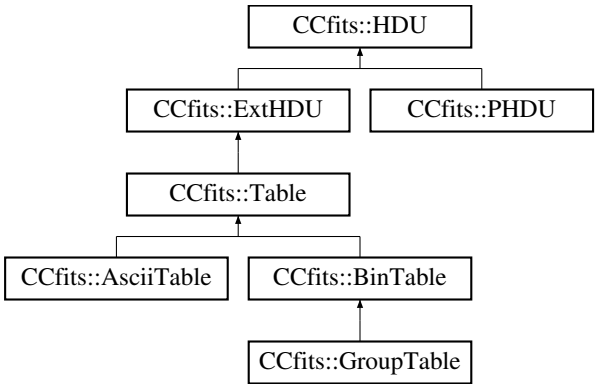
- GroupTable.h
- GroupTable.cxx

8.33 CCfits::HDU Class Reference

Base class for all HDU [Header-Data Unit] objects.

`#include <HDU.h>`

Inheritance diagram for CCfits::HDU:



Classes

- class InvalidExtensionType

  exception to be thrown if user requests extension type that can not be understood as ImageExt, AsciiTable or BinTable.
- class InvalidImageDataType

  exception to be thrown if user requests creation of an image of type not supported by cfitsio.
- class NoNullValue

  exception to be thrown on seek errors for keywords.
- class NoSuchKeyword

  exception to be thrown on seek errors for keywords.

Public Member Functions

- HDU (const HDU &right)

  copy constructor
- Keyword * addKey (const Keyword *inKeyword)

  create a copy of an existing Keyword and add to HDU
- template<typename T >
  Keyword & addKey (const String &name, T val, const String &comment, bool isLongStr=false)

  create a new keyword in the HDU with specified value and comment fields
- long axes () const

  return the number of axes in the HDU data section (always 2 for tables).
- long axis (size_t index) const

return the size of axis numbered index [zero based].

- long bitpix () const

  return the data type keyword.

- virtual HDU * clone (FITS *p) const =0

  virtual copy constructor, to be implemented in subclasses.

- const string & comment () const

  return the comment string previously read by getComment()

- void copyAllKeys (const HDU *inHdu, const std::vector< int > &keyCategories=std::vector< int >())

  copy all keys from another header

- void deleteKey (const String &doomed)

  delete a keyword from the header

- fitsfile * fitsPointer () const

  return the fitsfile pointer for the FITS object containing the HDU

- std::pair< unsigned long, unsigned long > getChecksum () const

  compute and return the checksum values for the HDU without creating or modifying the CHECKSUM/DATASUM keywords.

- const String & getComments ()

  read the comments from the HDU and add it to the FITS object.

- const String & getHistory ()

  read the history information from the HDU and add it to the FITS object.

- const string & history () const

  return the history string previously read by getHistory()

- int index () const

  return the HDU number

- void index (int value)

  set the HDU number

- std::map< String, Keyword * > & keyWord ()

  return the associative array containing the HDU keywords so far read.

- const std::map< string, Keyword * > & keyWord () const

  return the associative array containing the HDU Keywords that have been read so far.

- Keyword & keyWord (const String &keyName)

  return a (previously read) keyword from the HDU object.

- const Keyword & keyWord (const string &keyname) const

  return a (previously read) keyword from the HDU object. const version

- virtual void makeThisCurrent () const

  move the fitsfile pointer to this current HDU.

- bool operator!= (const HDU &right) const

  inequality operator

- bool operator== (const HDU &right) const

  equality operator

- FITS * parent () const

  return reference to the pointer representing the FITS object containing the HDU

- void readAllKeys (const std::vector< int > &keyCategories=std::vector< int >())

  read all of the keys in the header

- template<typename T >
  void readKey (const String &keyName, T &val)

  read a keyword of specified type from the header of a disk FITS file and return its value.

- template<typename T >
  void readKeys (std::vector< String > &keyNames, std::vector< T > &vals)

  read a set of specified keywords of the same data type from the header of a disk FITS file and return their values
- Keyword & readNextKey (const std::vector< String > &incList, const std::vector< String > &excList, bool searchFromBeginning=false)

  Read the next key in the HDU which matches a string in incList, and does not match string in excList.
- virtual void resetImageRead ()

  force next image reading operation to read from file instead of object cache.
- virtual double scale () const

  return the BSCALE keyword value
- virtual void scale (double value)

  set the BSCALE keyword value for images (see warning for images of int type)
- virtual void suppressScaling (bool toggle=true)

  turn off image scaling regardless of the BSCALE and BZERO keyword values
- void updateChecksum ()

  update the CHECKSUM keyword value, assuming DATASUM exists and is correct
- std::pair< int, int > verifyChecksum () const

  verify the HDU by computing the checksums and comparing them with the CHECKSUM/DATASUM keywords
- void writeChecksum ()

  compute and write the DATASUM and CHECKSUM keyword values
- void writeComment (const String &comment="Generic Comment")

  write a comment string.
- void writeDate ()

  write a date string to ∗this.
- void writeHistory (const String &history="Generic History String")

  write a history string.
- virtual double zero () const

  return the BZERO keyword value
- virtual void zero (double value)

  set the BZERO keyword value for images (see warning for images of int type)

Static Public Member Functions

- static std::vector< int > keywordCategories ()

  Return the default enumerated keyword categories used by copyAllKeys()

Protected Member Functions

- HDU (FITS ∗p, int bitpix, int naxis, const std::vector< long > &axes)

  constructor for creating new HDU objects, called by HDU subclasses writing to FITS files.
- HDU (FITS ∗p=0)

  default constructor, called by HDU subclasses that read from FITS files.
- virtual ∼HDU ()

  destructor
- std::vector< long > & naxes ()

  return the HDU data axis array.

8.33.1 Detailed Description

Base class for all HDU [Header-Data Unit] objects.

HDU objects in CCfits are either PHDU (Primary HDU objects) or ExtHDU (Extension HDU) objects. ExtHDUs are further subclassed into ImageExt or Table objects, which are finally AsciiTable or BinTable objects.

HDU's public interface gives access to properties that are common to all HDUs, largely required keywords, and functions that are common to all HDUs, principally the manipulation of keywords and their values.

HDUs must be constructed by HDUCreator objects which are called by FITS methods. Each HDU has an embedded pointer to its parent FITS object.

8.33.2 Member Function Documentation

8.33.2.1 addKey() [1/2]

```
Keyword * CCfits::HDU::addKey (
        const Keyword * inKeyword )
```

create a copy of an existing Keyword and add to HDU

This is particularly useful for copying Keywords from one HDU to another. For example the inKeyword pointer might come from a different HDU's std::map<string,Keyword*>. If a keyword with this name already exists, it will be overwritten. The return value is a pointer to the newly created Keyword inserted into this HDU. Also see copyAllKeys().

8.33.2.2 addKey() [2/2]

```
template<typename T >
Keyword & CCfits::HDU::addKey (
        const String & name,
        T value,
        const String & comment,
        bool isLongStr = false )
```

create a new keyword in the HDU with specified value and comment fields

The function returns a reference to keyword object just created. If a keyword with this name already exists, it will be overwritten. Note that this is mostly intended for adding user-defined keywords. It should not be used to add keywords for which there are already specific HDU functions, such as scaling or checksum. Nor should it be used for image or column structural keywords, such as BITPIX, NAXIS, TFORMn, etc. As a general rule, it is best to use this for keywords belonging to the same categories listed in the keywordCategories() function.

Parameters

name	(String) The keyword name
value	(Recommended T = String, double, std::complex<float>, int, or bool) The keyword value
comment	(String) The keyword comment
isLongStr	(Bool, default=false) Is the keyword long (greater than 68 characters)

It is possible to create a keyword with a value of any of the allowed data types in fitsio (see the cfitsio manual section 4.3). However one should be aware that if this keyword value is read in from the file at a later time, it will be stored in a templated Keyword subclass (KeyData<T>) where T will be one of the recommended types listed above. Also see Keyword::value (T& val) for more details.

If the keyword is long (isLong=true), the keyword is written using the Long String Keyword convention, as described in the "Local FITS Conventions" section of the cfitsio documentation.

8.33.2.3 axis()

```
long CCfits::HDU::axis (
        size_t index ) const  [inline]
```

return the size of axis numbered index [zero based].

return the length of HDU data axis i.

8.33.2.4 bitpix()

```
long CCfits::HDU::bitpix ( ) const  [inline]
```

return the data type keyword.

Takes values denoting the image data type for images, and takes the fixed value 8 for tables.

8.33.2.5 copyAllKeys()

```
void CCfits::HDU::copyAllKeys (
        const HDU * inHdu,
        const std::vector< int > & keyCategories = std::vector<int>() )
```

copy all keys from another header

Parameters

inHdu	An existing HDU whose keys will be copied into the current object.
keyCategories	(optional) A user-defined list of keyword categories to copy into current object

This will copy all keys that exist in the keyWord map of *inHDU*, and which belong to one of the desired keyword classes: either the default classes returned by the keywordCategories() function, or those given by the user-supplied vector of categories. The keywordMap can be populated with *readAllKeys()*. Only those keys in the map that match the desired category are copied.

Keywords are written in alphabetical order.

History and comment keys are copied to the current HDU from the input HDU, using the *getComments()*/*getHistory()* and *writeComment()*/*writeHistory()* methods.

All of the input comments and history are merged into one block, not spaced as they were in the input file.

It is highly recommended to use the default keyword categories, and not to pass in a user-defined list of categories, unless the user is very careful. Using the wrong categories can lead to incorrect keywords in files. For example, if the user passed in the TYP_CKSUM_KEY category to *copyAllKeys()*, then the CHECKSUM keyword would be copied from the old file to the new file. Similar care must be taken with metadata keywords for columns. This is more important for *copyAllKeys()* than for *readAllKeys()* because *copyAllKeys()* is the method that is actually inserting the keywords into the new file, while *readAllKeys()* is only reading in the source file's keywords.

8.33.2.6 deleteKey() `void CCfits::HDU::deleteKey ( const String & doomed )`

delete a keyword from the header

removes *doomed* from the FITS file and from the FITS object

8.33.2.7 getChecksum() `std::pair< unsigned long, unsigned long > CCfits::HDU::getChecksum ( ) const`

compute and return the checksum values for the HDU without creating or modifying the CHECKSUM/DATASUM keywords.

Wrapper for the CFITSIO function fits_get_chksum: This returns a std::pair<unsigned long, unsigned long> where the pair's first data member holds the datasum value and second holds the hdusum value.

8.33.2.8 getComments() `const String & CCfits::HDU::getComments ( )`

read the comments from the HDU and add it to the FITS object.

The comment string found in the header is concatenated and returned to the calling function

8.33.2.9 getHistory() `const String & CCfits::HDU::getHistory ( )`

read the history information from the HDU and add it to the FITS object.

The history string found in the header is concatenated and returned to the calling function

8.33.2.10 keywordCategories() `static std::vector< int > CCfits::HDU::keywordCategories ( ) [static]`

Return the default enumerated keyword categories used by copyAllKeys()

This returns a vector of integers indicating which categories of keywords are the default for the copyAllKeys functions. The list of categories currently hardcoded is: TYP_REFSYS_KEY (120) and TYP_USER_KEY (150).

For the list of all possible keyword categories, see the CFITSIO documentation for the fits_get_keyclass function.

8.33.2.11 makeThisCurrent() `void CCfits::HDU::makeThisCurrent ( ) const [virtual]`

move the fitsfile pointer to this current HDU.

This function should never need to be called by the user since it is called internally whenever required.

Reimplemented in CCfits::ExtHDU.

8.33.2.12 readAllKeys() `void CCfits::HDU::readAllKeys ( const std::vector< int > & keyCategories = std::vector<int>() )`

read all of the keys in the header

Parameters

keyCategories	(optional) A user-defined list of keyword categories to read

This member function reads and stores keys from the current object that are one of the desired keyword categories : either the default categories or those given by the user-supplied vector of categories. The default categories are: TYP_CMPRS_KEY (20), TYP_CKSUM_KEY (100), TYP_WCS_KEY (110), TYP_REFSYS_KEY (120), and TYP_USER_KEY (150).

History and comment keys are also read from the current HDU.

Note that readAllKeys can only construct keys of type string, double, complex<float>, integer, and bool because the FITS header records do not encode exact type information.

8.33.2.13 readKey() template<typename T >

```
void CCfits::HDU::readKey (
            const String & keyName,
            T & val )
```

read a keyword of specified type from the header of a disk FITS file and return its value.

T is one of the types String, double, float, int, std::complex<float>, and bool. If a Keyword object with the name *keyName* already exists in this HDU due to a previous read call, then this will re-read from the file and create a new Keyword object to replace the existing one.

8.33.2.14 readKeys() template<typename T >

```
void CCfits::HDU::readKeys (
            std::vector< String > & keyNames,
            std::vector< T > & vals )
```

read a set of specified keywords of the same data type from the header of a disk FITS file and return their values

T is one of the types String, double, float, int, std::complex<float>, and bool.

8.33.2.15 readNextKey()

```
Keyword & CCfits::HDU::readNextKey (
            const std::vector< String > & incList,
            const std::vector< String > & excList,
            bool searchFromBeginning = false )
```

Read the next key in the HDU which matches a string in incList, and does not match string in excList.

Parameters

incList	Vector of strings specifying keyword names to search.
excList	Vector of strings specifying names to exclude from search. This may be left empty.
searchFromBeginning	If 'true', search will be conducted from the start of the HDU. Otherwise it starts from the current position.

This is a wrapper around the CFITSIO fits_find_nextkey function. It reads in and returns the next keyword whose name matches matches one of the strings in incList, which may contain wild card characters (∗,?, and #). It will exclude keywords whose name matches a string in excList. If no keyword is found, a FitError is thrown.

By default the search is conducted from the current keyword position in the HDU. If searchFromBeginning is set to 'true', search will start from the beginning of the HDU. If HDU is not the currently open extension, this will make it so and start the keyword search from the beginning.

8.33.2.16 resetImageRead() `void CCfits::HDU::resetImageRead ( ) [inline], [virtual]`

force next image reading operation to read from file instead of object cache.

[Note: It is not necessary to call this function for normal image reading operations.] For primary HDUs and image extensions, this forces the next read operation to retrieve data from the file regardless of whether the data has already been read and stored in the HDU's internal arrays. This does nothing if the HDU does not contain an image.

8.33.2.17 scale() `void CCfits::HDU::scale ( double value ) [inline], [virtual]`

set the BSCALE keyword value for images (see warning for images of int type)

For primary HDUs and image extensions, this will add (or update) the BSCALE keyword in the header. The new setting will affect future image array read/writes as described in section 4.7 Data Scaling of the CFITSIO manual. For table extensions this function does nothing.

WARNING: If the image contains **integer-type data** (as indicated by the bitpix() return value), the new scale and zero value combination must not be such that the scaled data would require a floating-point type (this uses the CFITSIO function fits_get_img_equivtype to make the determination). If this situation occurs, the function will throw a FitsException.

Reimplemented in CCfits::PHDU.

8.33.2.18 suppressScaling() `void CCfits::HDU::suppressScaling ( bool toggle = true ) [virtual]`

turn off image scaling regardless of the BSCALE and BZERO keyword values

For *toggle* = true, this turns off image scaling for future read/writes by resetting the scale and zero to 1.0 and 0.↩ 0 respectively. It does NOT modify the BSCALE and BZERO keywords. If *toggle* = false, the scale and zero values will be restored to the keyword values.

8.33.2.19 updateChecksum() `void CCfits::HDU::updateChecksum ( )`

update the CHECKSUM keyword value, assuming DATASUM exists and is correct

Wrapper for the CFITSIO function fits_update_chksum: This recomputes and writes the CHECKSUM value with the assumption that the DATASUM value is correct. If the DATASUM keyword doesn't yet exist or is not up-to-date, use the HDU::writeChecksum function instead. This will throw a FitsError exception if called when there is no DATASUM keyword in the header.

8.33.2.20 verifyChecksum() `std::pair< int, int > CCfits::HDU::verifyChecksum ( ) const`

verify the HDU by computing the checksums and comparing them with the CHECKSUM/DATASUM keywords

Wrapper for the CFITSIO function fits_verify_chksum: The data unit is verified correctly if the computed checksum equals the DATASUM keyword value, and the HDU is verified if the entire checksum equals zero (see the CFITSIO manual for further details).

This returns a std::pair<int,int> where the pair's first data member = DATAOK and second = HDUOK. DATAOK and HDUOK values will be = 1 if verified correctly, 0 if the keyword is missing, and -1 if the computed checksum is not correct.

8.33.2.21 writeChecksum() `void CCfits::HDU::writeChecksum ( )`

compute and write the DATASUM and CHECKSUM keyword values

Wrapper for the CFITSIO function fits_write_chksum: This performs the datasum and checksum calculations for this HDU, as described in the CFITSIO manual. If either the DATASUM or CHECKSUM keywords already exist, their values will be updated.

8.33.2.22 writeComment() `void CCfits::HDU::writeComment ( const String & comment = "Generic Comment" )`

write a comment string.

A default value for the string is given ("Generic Comment String") so users can put a placeholder call to this function in their code.

8.33.2.23 writeHistory() `void CCfits::HDU::writeHistory ( const String & history = "Generic History String" )`

write a history string.

A default value for the string is given ("Generic History String") so users can put a placeholder call to this function in their code.

8.33.2.24 zero() `void CCfits::HDU::zero ( double value ) [inline], [virtual]`

set the BZERO keyword value for images (see warning for images of int type)

For primary HDUs and image extensions, this will add (or update) the BZERO keyword in the header. The new setting will affect future image array read/writes as described in section 4.7 Data Scaling of the CFITSIO manual. For table extensions this function does nothing.

WARNING: If the image contains **integer-type data** (as indicated by the bitpix() return value), the new scale and zero value combination must not be such that the scaled data would require a floating-point type (this uses the CFITSIO function fits_get_img_equivtype to make the determination). If this situation occurs, the function will throw a FitsException.

Reimplemented in CCfits::PHDU.

The documentation for this class was generated from the following files:

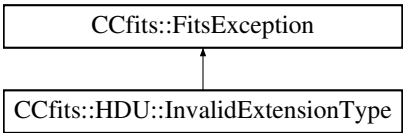
- HDU.h
- HDU.cxx

8.34 CCfits::HDU::InvalidExtensionType Class Reference

exception to be thrown if user requests extension type that can not be understood as ImageExt, AsciiTable or BinTable.

`#include <HDU.h>`

Inheritance diagram for CCfits::HDU::InvalidExtensionType:



Public Member Functions

- InvalidExtensionType (const string &diag, bool silent=true)

  Exception ctor, prefixes the string "Fits Error: Extension Type: " before the specific message.

8.34.1 Detailed Description

exception to be thrown if user requests extension type that can not be understood as ImageExt, AsciiTable or BinTable.

8.34.2 Constructor & Destructor Documentation

8.34.2.1 InvalidExtensionType() `CCfits::HDU::InvalidExtensionType::InvalidExtensionType (`

```
const string & diag,
bool silent = true )
```

Exception ctor, prefixes the string "Fits Error: Extension Type: " before the specific message.

Parameters

diag	A specific diagnostic message
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

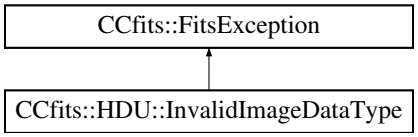
- HDU.h
- HDU.cxx

8.35 CCfits::HDU::InvalidImageDataType Class Reference

exception to be thrown if user requests creation of an image of type not supported by cfitsio.

`#include <HDU.h>`

Inheritance diagram for CCfits::HDU::InvalidImageDataType:



Public Member Functions

- InvalidImageDataType (const string &diag, bool silent=true)

  Exception ctor, prefixes the string "Fits Error: Invalid Data Type for Image " before the specific message.

8.35.1 Detailed Description

exception to be thrown if user requests creation of an image of type not supported by cfitsio.

8.35.2 Constructor & Destructor Documentation

8.35.2.1 InvalidImageDataType() `CCfits::HDU::InvalidImageDataType::InvalidImageDataType (`

```
const string & diag,
bool silent = true )
```

Exception ctor, prefixes the string "Fits Error: Invalid Data Type for Image " before the specific message.

Parameters

diag	A specific diagnostic message
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

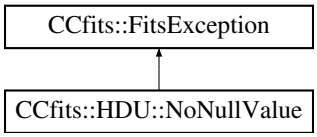
- HDU.h
- HDU.cxx

8.36 CCfits::HDU::NoNullValue Class Reference

exception to be thrown on seek errors for keywords.

`#include <HDU.h>`

Inheritance diagram for CCfits::HDU::NoNullValue:



Public Member Functions

- NoNullValue (const string &diag, bool silent=true)

  Exception ctor, prefixes the string "Fits Error: No Null Pixel Value specified for Image " before the specific message.

8.36.1 Detailed Description

exception to be thrown on seek errors for keywords.

8.36.2 Constructor & Destructor Documentation

8.36.2.1 NoNullValue() `CCfits::HDU::NoNullValue::NoNullValue (`

```
const string & diag,
bool silent = true )
```

Exception ctor, prefixes the string "Fits Error: No Null Pixel Value specified for Image " before the specific message.

Parameters

diag	A specific diagnostic message, the name of the HDU if not the primary.
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

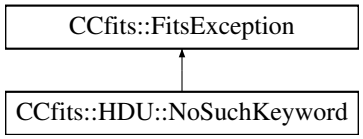
- HDU.h
- HDU.cxx

8.37 CCfits::HDU::NoSuchKeyword Class Reference

exception to be thrown on seek errors for keywords.

`#include <HDU.h>`

Inheritance diagram for CCfits::HDU::NoSuchKeyword:



Public Member Functions

- NoSuchKeyword (const string &diag, bool silent=true)

  Exception ctor, prefixes the string "Fits Error: Keyword not found: " before the specific message.

8.37.1 Detailed Description

exception to be thrown on seek errors for keywords.

8.37.2 Constructor & Destructor Documentation

8.37.2.1 NoSuchKeyword() `CCfits::HDU::NoSuchKeyword::NoSuchKeyword (`

```
const string & diag,
bool silent = true )
```

Exception ctor, prefixes the string "Fits Error: Keyword not found: " before the specific message.

Parameters

diag	A specific diagnostic message, usually the name of the keyword requested.
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

- HDU.h
- HDU.cxx

8.38 CCfits::Keyword Class Reference

Abstract base class defining the interface for Keyword objects.

`#include <Keyword.h>`

Inherited by CCfits::KeyData< T >.

Public Member Functions

- virtual ~Keyword ()
  virtual destructor
- virtual Keyword * clone () const =0
  virtual copy constructor
- const String & comment () const
  return the comment field of the keyword
- fitsfile * fitsPointer () const
  return a pointer to the FITS file containing the parent HDU.
- ValueType keytype () const
  return the type of a keyword
- const String & name () const
  return the name of a keyword
- bool operator!= (const Keyword &right) const
  inequality operator
- Keyword & operator= (const Keyword &right)
  assignment operator
- bool operator== (const Keyword &right) const
  equality operator
- template<typename T >
  void setValue (const T &newValue)
  modify the value of an existing Keyword and write it to the file
- template<typename T >
  T & value (T &val) const
  get the keyword value
- virtual void write ()
  left in for historical reasons, this seldom needs to be called by users

Protected Member Functions

- Keyword (const Keyword &right)
  copy constructor
- Keyword (const String &keyname, ValueType keytype, HDU *p, const String &comment="", bool isLongStr=false)
  Keyword constructor.
- void keytype (ValueType value)
  set keyword type.
- const HDU * parent () const
  return a pointer to parent HDU.

8.38.1 Detailed Description

Abstract base class defining the interface for Keyword objects.

Keyword object creation is normally performed inside FITS constructors or FITS::read, HDU::readKey, and HDU::addKey functions. Output is performed in HDU::addKey functions and Keyword::setValue.

Keywords consists of a name, a value and a comment field. Concrete templated subclasses, KeyData<T>, have a data member that holds the value of keyword.

Typically, the mandatory keywords for a given HDU type are not stored as object of type Keyword, but as intrinsic data types. The Keyword hierarchy is used to store user-supplied information.

8.38.2 Constructor & Destructor Documentation

8.38.2.1 Keyword()

```
CCfits::Keyword::Keyword (
        const String & keyname,
        ValueType keytype,
        HDU * p,
        const String & comment = "",
        bool isLongStr = false )  [protected]
```

Keyword constructor.

This is the common behavior of Keywords of any type. Constructor is protected as the class is abstract.

8.38.3 Member Function Documentation

8.38.3.1 setValue()

```
template<typename T >
void CCfits::Keyword::setValue (
        const T & newValue )
```

modify the value of an existing Keyword and write it to the file

Parameters

newValue	(T) New value for the Keyword

Allowed T types: This must copy *newValue* to a data member of type U in the Keyword subclass **KeyData<U>** (see description for Keyword::value (T& val) for more details). To avoid compilation errors, it is generally best to provide a *newValue* of type T = type U, though the following type conversions will also be handled:

T (from newValue)	**U** (to Keyword obj)
float	double, float
double	double, float (will lose precision)
int	double, float, int, integer string

8.38.3.2 value() `template<typename T >`

```
T & CCfits::Keyword::value (
              T & val ) const
```

get the keyword value

Parameters

val	(T) Will be filled with the keyword value, and is also the function return value.

Allowed T types: CCfits stores keyword values of type U in a templated subclass of Keyword, **KeyData**<**U**>. Normally U is set when reading the Keyword in from the file, and is limited to types int, double, string, bool, and complex<float>. (The exception is when the user has created and added a new Keyword using an HDU::addKey function, in which case they might have specified other types for U.) To avoid compilation errors, the user should generally try to provide a *val* of type T = type U, though there is some flexibility here as the following conversions are handled:

T (to val)	**U** (from Keyword obj)
float	double (will lose precision), float, int, integer string
double	double, float, int, integer string
int	int, integer string
string	double, float, int, string

More conversions may be added in the future as the need arises.

8.38.3.3 write() `void CCfits::Keyword::write ( ) [virtual]`

left in for historical reasons, this seldom needs to be called by users

This writes the Keyword to the file, and is called internally during HDU::addKey operations or the Keyword::setValue function. It shouldn't normally need to be called explicitly.

The documentation for this class was generated from the following files:

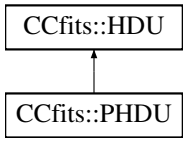
- Keyword.h
- Keyword.cxx
- KeywordT.h

8.39 CCfits::PHDU Class Reference

class representing the primary HDU for a FITS file.

`#include <PHDU.h>`

Inheritance diagram for CCfits::PHDU:



Public Member Functions

- virtual ~PHDU ()

  destructor
- virtual PHDU * clone (FITS *p) const =0

  virtual copy constructor for Primary HDUs.
- bool extend () const

  Returns the value of the Primary's EXTEND keyword.
- template<typename S >
  void read (std::valarray< S > &image, const std::vector< long > &first, long nElements)

  read an image section starting at a location specified by an n-tuple
- template<typename S >
  void read (std::valarray< S > &image, const std::vector< long > &first, long nElements, S *nullValue)

  read part of an image array, processing null values.
- template<typename S >
  void read (std::valarray< S > &image, const std::vector< long > &firstVertex, const std::vector< long > &last↩
  Vertex, const std::vector< long > &stride)

  read an image subset
- template<typename S >
  void read (std::valarray< S > &image, const std::vector< long > &firstVertex, const std::vector< long > &last↩
  Vertex, const std::vector< long > &stride, S *nullValue)

  read an image subset into valarray image, processing null values
- template<typename S >
  void read (std::valarray< S > &image, long first, long nElements)

  read an image section starting at a specified pixel
- template<typename S >
  void read (std::valarray< S > &image, long first, long nElements, S *nullValue)

  read part of an image array, processing null values.
- virtual void readData (bool readFlag=false, const std::vector< String > &keys=std::vector< String >())=0

  read primary HDU data
- virtual double scale () const

  return the BSCALE keyword value
- virtual void scale (double value)

  set the BSCALE keyword value for images (see warning for images of int type)

- bool simple () const

  Returns the value of the Primary's SIMPLE keyword.
- template<typename S >
  void write (const std::vector< long > &first, long nElements, const std::valarray< S > &data)

  write array starting from specified n-tuple, without undefined data processing
- template<typename S >
  void write (const std::vector< long > &first, long nElements, const std::valarray< S > &data, S *nullValue)

  Write a set of pixels to an image extension with the first pixel specified by an n-tuple, processing undefined data.
- template<typename S >
  void write (const std::vector< long > &firstVertex, const std::vector< long > &lastVertex, const std::vector< long > &stride, const std::valarray< S > &data)

  write a subset (generalize slice) of data to the image
- template<typename S >
  void write (long first, long nElements, const std::valarray< S > &data)

  write array starting from specified pixel number, without undefined data processing
- template<typename S >
  void write (long first, long nElements, const std::valarray< S > &data, S *nullValue)

  write array to image starting with a specified pixel and allowing undefined data to be processed
- virtual double zero () const

  return the BZERO keyword value
- virtual void zero (double value)

  set the BZERO keyword value for images (see warning for images of int type)

Protected Member Functions

- PHDU (const PHDU &right)

  copy constructor
- PHDU (FITS *p, int bpix, int naxis, const std::vector< long > &axes)

  Writing Primary HDU constructor, called by PrimaryHDU<T> class.
- PHDU (FITS *p=0)

  Reading Primary HDU constructor.
- virtual void initRead ()

Additional Inherited Members

8.39.1 Detailed Description

class representing the primary HDU for a FITS file.

A PHDU object is automatically instantiated and added to a FITS object when a FITS file is accessed in any way. If a new file is created without specifying the data type for the header, CCfits assumes that the file is to be used for table extensions and creates a dummy header. PHDU instances are *only* created by FITS ctors. In the first release of CCfits, the Primary cannot be changed once declared.

PHDU and ExtHDU provide the same interface to writing images: multiple overloads of the templated PHDU::read and PHDU::write operations provide for (a) writing image data specified in a number of ways [C-array, std::vector, std↩ ::valarray] and with input location specified by initial pixel, by n-tuple, and by rectangular subset [generalized slice]; (b) reading image data specified similarly to the write options into a std::valarray.

Todo Implement functions that allow replacement of the primary image

8.39.2 Constructor & Destructor Documentation

8.39.2.1 ~PHDU() CCfits::PHDU::~PHDU ( ) [virtual]

destructor

Destructor

8.39.2.2 PHDU() [1/3] CCfits::PHDU::PHDU (const PHDU & *right*) [protected]

copy constructor

required for cloning primary HDUs when copying FITS files.

8.39.2.3 PHDU() [2/3] CCfits::PHDU::PHDU (FITS * *p*, int *bpix*, int *naxis*, const std::vector< long > & *axes*) [protected]

Writing Primary HDU constructor, called by PrimaryHDU<T> class.

Constructor used for creating new PHDU (i.e. for writing data to FITS). also doubles as default constructor since all arguments have default values, which are passed to the HDU constructor

8.39.2.4 PHDU() [3/3] CCfits::PHDU::PHDU (FITS * *p* = *0*) [protected]

Reading Primary HDU constructor.

Constructor used when reading the primary HDU from an existing file. Does nothing except initialize, with the real work done by the subclass PrimaryHDU<T>.

8.39.3 Member Function Documentation

8.39.3.1 clone() CCfits::PHDU::clone (FITS * *p*) const [pure virtual]

virtual copy constructor for Primary HDUs.

The operation is used when creating a copy of a FITS object.

Implements CCfits::HDU.

8.39.3.2 initRead() `void CCfits::PHDU::initRead ( ) [protected], [virtual]`

Read image header and update fits pointer accordingly.

Private: called by ctor.

Implements CCfits::HDU.

8.39.3.3 read() [1/3] `template<typename S >`

```
void CCfits::PHDU::read (
            std::valarray< S > & image,
            const std::vector< long > & first,
            long nElements,
            S * nullValue )
```

read part of an image array, processing null values.

As above except for

Parameters

first	a vector<long> representing an n-tuple giving the coordinates in the image of the first pixel.

8.39.3.4 read() [2/3] `template<typename S >`

```
void CCfits::PHDU::read (
            std::valarray< S > & image,
            const std::vector< long > & firstVertex,
            const std::vector< long > & lastVertex,
            const std::vector< long > & stride,
            S * nullValue )
```

read an image subset into valarray image, processing null values

The image subset is defined by two vertices and a stride indicating the 'denseness' of the values to be picked in each dimension (a stride = (1,1,1,...) means picking every pixel in every dimension, whereas stride = (2,2,2,...) means picking every other value in each dimension.

8.39.3.5 read() [3/3] `template<typename S >`

```
void CCfits::PHDU::read (
            std::valarray< S > & image,
            long first,
            long nElements,
            S * nullValue )
```

read part of an image array, processing null values.

Implicit data conversion is supported (i.e. user does not need to know the type of the data stored. A WrongExtension↩ Type extension is thrown if ∗this is not an image.

Parameters

image	The receiving container, a std::valarray reference
first	The first pixel from the array to read [a long value]
nElements	The number of values to read
nullValue	A pointer containing the value in the table to be considered as undefined. See cfitsio for details

8.39.3.6 readData()

```
void CCfits::PHDU::readData (
        bool readFlag = false,
        const std::vector< String > & keys = std::vector<String>() )  [pure virtual]
```

read primary HDU data

Called by FITS ctor, not intended for general use. parameters control how much gets read on initialization. An abstract function, implemented in the subclasses.

Parameters

readFlag	read the image data if true
key	a vector of strings of keyword names to be read from the primary HDU

8.39.3.7 scale()

```
void CCfits::PHDU::scale (
        double value )  [virtual]
```

set the BSCALE keyword value for images (see warning for images of int type)

For primary HDUs and image extensions, this will add (or update) the BSCALE keyword in the header. The new setting will affect future image array read/writes as described in section 4.7 Data Scaling of the CFITSIO manual. For table extensions this function does nothing.

WARNING: If the image contains **integer-type data** (as indicated by the bitpix() return value), the new scale and zero value combination must not be such that the scaled data would require a floating-point type (this uses the CFITSIO function fits_get_img_equivtype to make the determination). If this situation occurs, the function will throw a FitsException.

Reimplemented from CCfits::HDU.

8.39.3.8 write() [1/3] template<typename S >

```
void CCfits::PHDU::write (
            const std::vector< long > & first,
            long nElements,
            const std::valarray< S > & data,
            S * nullValue )
```

Write a set of pixels to an image extension with the first pixel specified by an n-tuple, processing undefined data.

All the overloaded versions of PHDU::write perform operations on ∗this if it is an image and throw a WrongExtensionType exception if not. Where appropriate, alternate versions allow undefined data to be processed

Parameters

first	an n-tuple of dimension equal to the image dimension specifying the first pixel in the range to be written
nElements	number of pixels to be written
data	array of data to be written
nullValue	pointer to null value (data with this value written as undefined; needs the BLANK keyword to have been specified).

8.39.3.9 write() [2/3] template<typename S >

```
void CCfits::PHDU::write (
            const std::vector< long > & firstVertex,
            const std::vector< long > & lastVertex,
            const std::vector< long > & stride,
            const std::valarray< S > & data )
```

write a subset (generalize slice) of data to the image

A generalized slice/subset is a subset of the image (e.g. one plane of a data cube of size <= the dimension of the cube). It is specified by two opposite vertices. The equivalent cfitsio call does not support undefined data processing so there is no version that allows a null value to be specified.

Parameters

firstVertex	The coordinates specifying lower and upper vertices of the n-dimensional slice
lastVertex	
stride	Pixels to skip in each to dimension, e.g. stride = (1,1,1,...) means picking every pixel in every dimension, whearas stride = (2,2,2,...) means picking every other value in each dimension.
data	The data to be written

8.39.3.10 write() [3/3] template<typename S >

```
void CCfits::PHDU::write (
            long first,
            long nElements,
            const std::valarray< S > & data,
            S * nullValue )
```

write array to image starting with a specified pixel and allowing undefined data to be processed

parameters after the first are as for version with n-tuple specifying first element. these two version are equivalent, except that it is possible for the first pixel number to exceed the range of 32-bit integers, which is how long datatype is commonly implemented.

8.39.3.11 zero() `void CCfits::PHDU::zero ( double value ) [virtual]`

set the BZERO keyword value for images (see warning for images of int type)

For primary HDUs and image extensions, this will add (or update) the BZERO keyword in the header. The new setting will affect future image array read/writes as described in section 4.7 Data Scaling of the CFITSIO manual. For table extensions this function does nothing.

WARNING: If the image contains **integer-type data** (as indicated by the bitpix() return value), the new scale and zero value combination must not be such that the scaled data would require a floating-point type (this uses the CFITSIO function fits_get_img_equivtype to make the determination). If this situation occurs, the function will throw a FitsException.

Reimplemented from CCfits::HDU.

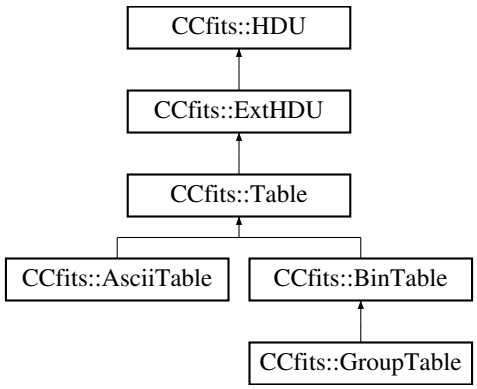
The documentation for this class was generated from the following files:

- PHDU.h
- PHDU.cxx
- PHDUT.h

8.40 CCfits::Table Class Reference

`#include <Table.h>`

Inheritance diagram for CCfits::Table:



Classes

- class NoSuchColumn

  Exception to be thrown on a failure to retrieve a column specified either by name or index number.

Public Member Functions

- Table (const Table &right)

  copy constructor
- virtual ~Table ()

  destructor
- virtual ColMap & column ()

  return a reference to the multimap containing the columns.
- virtual const ColMap & column () const

  return a reference to the multimap containing the columns.
- virtual Column & column (const String &colName, bool caseSensitive=true) const

  return a reference to a Table column specified by name.
- virtual Column & column (int colIndex) const

  return a reference to the column identified by colIndex
- virtual void copyColumn (const Column &inColumn, int colIndx, bool insertNewCol=true)

  copy a column (from different or same HDU and file) into an existing table HDU.
- virtual void deleteColumn (const String &columnName)

  delete a column in a Table extension by name.
- void deleteRows (const std::vector< long > &rowList)

  delete a set of rows in the table specified by an input array.
- void deleteRows (long first, long number=1)

  delete a range of rows in a table.
- virtual long getRowsize () const

  return the optimal number of rows to read or write at a time
- void insertRows (long first, long number=1)

  insert empty rows into the table
- virtual int numCols () const

  return the number of Columns in the Table (the TFIELDS keyword).
- virtual long rows () const

  return the number of rows in the table (NAXIS2).
- void rows (long numRows)

  set the number of rows in the Table.
- void updateRows ()

  update the number of rows in the table

Protected Member Functions

- Table (FITS *p, HduType xtype, const String &hduName, int rows, const std::vector< String > &columnName, const std::vector< String > &columnFmt, const std::vector< String > &columnUnit=std::vector< String >(), int version=1)

  Constructor to be used for creating new HDUs.
- Table (FITS *p, HduType xtype, const String &hduName=String(""), int version=1)

  Constructor to be called by operations that read Table specified by hduName and version.
- Table (FITS *p, HduType xtype, int number)

  Table constructor for getting Tables by number.
- Table (FITS *p, int version=1, const String &groupName=String(""))

  Constructor to be called when creating a grouping table.
- void init (bool readFlag=false, const std::vector< String > &keys=std::vector< String >())
- void numCols (int value)

  set the number of Columns in the Table

Additional Inherited Members

8.40.1 Detailed Description

Table is the abstract common interface to Binary and Ascii Table HDUs.

Table is a subclass of ExtHDU that contains an associative array of Column objects. It implements methods for reading and writing columns

8.40.2 Constructor & Destructor Documentation

8.40.2.1 Table() [1/3] CCfits::Table::Table (

```
FITS * p,
HduType xtype,
const String & hduName,
int rows,
const std::vector< String > & columnName,
const std::vector< String > & columnFmt,
const std::vector< String > & columnUnit = std::vector<String>(),
int version = 1 )  [protected]
```

Constructor to be used for creating new HDUs.

Parameters

p	The FITS file in which to place the new HDU
xtype	An HduType enumerator defined in CCfits.h for type of table (AsciiTbl or BinaryTbl)
hduName	The name of this HDU extension
rows	The number of rows in the new HDU (the value of the NAXIS2 keyword).
columnName	a vector of names for the columns.
columnFmt	the format strings for the columns
columnUnit	the units for the columns.
version	a version number

8.40.2.2 Table() [2/3] CCfits::Table::Table (

```
FITS * p,
int version = 1,
const String & groupName = String("") )  [protected]
```

Constructor to be called when creating a grouping table.

Parameters

p	The FITS file in which to place the new HDU
version	Version number
groupName	The name of the grouping table

8.40.2.3 Table() [3/3] CCfits::Table::Table (FITS * p, HduType xtype, int number) [protected]

Table constructor for getting Tables by number.

Necessary since EXTNAME is a reserved not required keyword, and users may thus read FITS files without an extension name. Since an HDU is completely specified by extension number, this is part of the public interface.

8.40.3 Member Function Documentation

8.40.3.1 column() [1/4] ColMap & CCfits::Table::column ( ) [inline], [virtual]

return a reference to the multimap containing the columns.

To be used in the implementation of subclasses.

8.40.3.2 column() [2/4] const ColMap & CCfits::Table::column ( ) const [inline], [virtual]

return a reference to the multimap containing the columns.

This public version might be used to query the size of the column container in a routine that manipulates column table data.

Reimplemented from CCfits::ExtHDU.

8.40.3.3 column() [3/4] Column & CCfits::Table::column (const String & colName, bool caseSensitive = true) const [virtual]

return a reference to a Table column specified by name.

If the *caseSensitive* parameter is set to false, the search will be case-insensitive. The overridden base class implementation ExtHDU::column throws an exception, which is thus the action to be taken if self is an image extension

Exceptions

WrongExtensionType	see above

Reimplemented from CCfits::ExtHDU.

8.40.3.4 column() [4/4]

```
Column & CCfits::Table::column (
        int colIndex ) const  [virtual]
```

return a reference to the column identified by colIndex

Throws NoSuchColumn if the index is out of range -index must satisfy (1 <= index <= numCols()).

N.B. the column number is assigned as 1-based, as in FORTRAN rather than 0-based as in C.

Exceptions

Table::NoSuchColumn	passes colIndex to the diagnostic message printed when the exception is thrown

Reimplemented from CCfits::ExtHDU.

8.40.3.5 copyColumn()

```
void CCfits::Table::copyColumn (
        const Column & inColumn,
        int colIndx,
        bool insertNewCol = true )  [virtual]
```

copy a column (from different or same HDU and file) into an existing table HDU.

This is meant to provide the same functionality as CFITSIO's fits_copy_col, and therefore does not work with columns with variable length fields. Copying a column from an AsciiTable to a BinTable is prohibited. colIndx range should be from 1 to nCurrentCols+1 if inserting, or 1 to nCurrentCols if replacing.

Parameters

inColumn	The Column object which is to be copied
colIndx	The position for which the copied Column will be placed (first colIndx = 1).
insertNewCol	If 'true', new Column will be inserted in or appended to table. If 'false', Column will replace current Column at position = colIndx.

Reimplemented from CCfits::ExtHDU.

8.40.3.6 deleteColumn() `void CCfits::Table::deleteColumn ( const String & columnName ) [virtual]`

delete a column in a Table extension by name.

Parameters

columnName	The name of the column to be deleted.

Exceptions

WrongExtensionType	if extension is an image.

Reimplemented from CCfits::ExtHDU.

8.40.3.7 deleteRows() [1/2] `void CCfits::Table::deleteRows ( const std::vector< long > & rowlist )`

delete a set of rows in the table specified by an input array.

Parameters

rowlist	The vector of row numbers to be deleted.

Exceptions

FitsError	thrown if the underlying cfitsio call fails to return without error.

8.40.3.8 deleteRows() [2/2] `void CCfits::Table::deleteRows ( long first, long number = 1 )`

delete a range of rows in a table.

In both this and the overloaded version which allows a selection of rows to be deleted, the cfitsio library is called first to perform the operation on the disk file, and then the FITS object is updated.

Parameters

first	the start row of the range
number	the number of rows to delete; defaults to 1.

Exceptions

FitsError	thrown if the cfitsio call fails to return without error.

8.40.3.9 getRowsize() `long CCfits::Table::getRowsize ( ) const [virtual]`

return the optimal number of rows to read or write at a time

A wrapper for the CFITSIO function fits_get_rowsize, useful for obtaining maximum I/O efficiency. This will throw if it is not called for a Table extension.

Reimplemented from CCfits::ExtHDU.

8.40.3.10 init()

```
void CCfits::Table::init (
    bool readFlag = false,
    const std::vector< String > & keys = std::vector<String>() ) [protected]
```

"Late Constructor." wrap-up of calls needed to construct a table. Reads header information and sets up the array of column objects in the table.

Protected function, provided to allow the implementation of extensions of the library.

8.40.3.11 insertRows()

```
void CCfits::Table::insertRows (
    long first,
    long number = 1 )
```

insert empty rows into the table

Parameters

first	the start row of the range
number	the number of rows to insert.

Exceptions

FitsError	thrown if the underlying cfitsio call fails to return without error.

8.40.3.12 updateRows() `void CCfits::Table::updateRows ( )`

update the number of rows in the table

Called to force the Table to reset its internal "rows" attribute. public, but is called when needed internally.

The documentation for this class was generated from the following files:

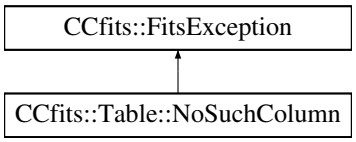
- Table.h
- Table.cxx

8.41 CCfits::Table::NoSuchColumn Class Reference

Exception to be thrown on a failure to retrieve a column specified either by name or index number.

`#include <Table.h>`

Inheritance diagram for CCfits::Table::NoSuchColumn:



Public Member Functions

- NoSuchColumn (const String &name, bool silent=true)

  Exception ctor for exception thrown if the requested column (specified by name) is not present.
- NoSuchColumn (int index, bool silent=true)

  Exception ctor for exception thrown if the requested column (specified by name) is not present.

8.41.1 Detailed Description

Exception to be thrown on a failure to retrieve a column specified either by name or index number.

When a Table object is created, the header is read and a column object created for each column defined. Thus id this exception is thrown the column requested does not exist in the HDU (note that the column can easily exist and not contain any data since the user controls whether the column will be read when the FITS object is instantiated).

It is expected that the index number calls will be primarily internal. The underlying implementation makes lookup by name more efficient.

The exception has two variants, which take either an integer or a string as parameter. These are used according to the accessor that threw them, either by name or index.

8.41.2 Constructor & Destructor Documentation

8.41.2.1 NoSuchColumn() [1/2]

```
CCfits::Table::NoSuchColumn::NoSuchColumn (
        const String & name,
        bool silent = true )
```

Exception ctor for exception thrown if the requested column (specified by name) is not present.

Message: Fits Error: cannot find Column named: *name* is printed.

Parameters

name	the requested column name
silent	if true, print message whether FITS::verboseMode is set or not.

8.41.2.2 NoSuchColumn() [2/2]

```
CCfits::Table::NoSuchColumn::NoSuchColumn (
        int index,
        bool silent = true )
```

Exception ctor for exception thrown if the requested column (specified by name) is not present.

Message: Fits Error: column not present - Column number *index* is printed.

Parameters

index	the requested column number
silent	if true, print message whether FITS::verboseMode is set or not.

The documentation for this class was generated from the following files:

- Table.h
- Table.cxx

8.42 ImageExt< T > Class Reference

```
#include <ImageExt.h>
```

8.42.1 Detailed Description

ImageExt<T> is a subclass of ExtHDU that contains image data of type T.

The documentation for this class was generated from the following file:

- ImageExt.h

Index