HDF5 Extras

0.0.1

Generated by Doxygen 1.8.7

Tue May 31 2016 20:20:05

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

bStream . . . . ??
bstrList . . . . ??
charField . . . . ??
fcurl_data . . . . ??
genBstrList . . . . ??
GEOInfo_t . . . . ??
HDFFile . . . . ??
IFILE . . . . ??
tagbstring . . . . ??

Chapter 2

File Index

2.1 File List

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

src/bstrlib.h . . . . ??
src/**globals.h** . . . . ??
src/**gmalloc.h** . . . . ??
src/GS_CheckInternalName.c . . . . ??
src/GS_ConvertFromHDFDatatype.c . . . . ??
src/GS_ConvertToHDFDatatype.c . . . . ??
src/GS_CreateAccessPlist.c . . . . ??
src/GS_DatasetClose.c . . . . ??
src/GS_DatasetCopy.c . . . . ??
src/GS_DatasetCreate.c . . . . ??
src/GS_DatasetDelete.c . . . . ??
src/GS_DatasetGetDatatype.c . . . . ??
src/GS_DatasetGetDatatypeByID.c . . . . ??
src/GS_DatasetGetDimensions.c . . . . ??
src/GS_DatasetGetDimensionsByID.c . . . . ??
src/GS_DatasetGetParent.c . . . . ??
src/GS_DatasetGetType.c . . . . ??
src/GS_DatasetOpen.c . . . . ??
src/GS_DatasetRead.c . . . . ??
src/GS_DatasetRename.c . . . . ??
src/GS_DatasetWrite.c . . . . ??
src/GS_DatatypeAsString.c . . . . ??
src/GS_DatatypeIsComplex.c . . . . ??
src/GS_FileClose.c . . . . ??
src/GS_FileCloseAllObjects.c . . . . ??
src/GS_FileCommitDatatypes.c . . . . ??
src/GS_FileCopy.c . . . . ??
src/GS_FileCreate.c . . . . ??
src/GS_FileCreateMetadata.c . . . . ??
src/GS_FileFlush.c . . . . ??
src/GS_FileIsOpen.c . . . . ??
src/GS_FileIsWriteable.c . . . . ??
src/GS_FileOpen.c . . . . ??
src/GS_FileReport.c . . . . ??

src/GS_GetHDF5Type.c . . . . **??**
src/GS_GetStringAttribute.c . . . . **??**
src/GS_GetValidFileID.c . . . . **??**
src/GS_GroupClose.c . . . . **??**
src/GS_GroupCloseAllObjects.c . . . . **??**
src/GS_GroupCopy.c . . . . **??**
src/GS_GroupCreate.c . . . . **??**
src/GS_GroupDelete.c . . . . **??**
src/GS_GroupOpen.c . . . . **??**
src/GS_GroupRename.c . . . . **??**
src/GS_GroupSetType.c . . . . **??**
src/GS_HDFDatatypeClose.c . . . . **??**
src/GS_ObjectGetChildren.c . . . . **??**
src/GS_ObjectIsDataset.c . . . . **??**
src/GS_ObjectIsDatasetByID.c . . . . **??**
src/GS_ObjectIsFile.c . . . . **??**
src/GS_ObjectIsGroup.c . . . . **??**
src/GS_ObjectIsGroupByID.c . . . . **??**
src/GS_ObjectIsIFile.c . . . . **??**
src/GS_Pathname.c . . . . **??**
src/GS_SetCacheSize.c . . . . **??**
src/GS_SetStringAttribute.c . . . . **??**
src/GS_Time.c . . . . **??**
src/GS_ValidID.c . . . . **??**
src/H5ATTR.c . . . . **??**
src/**H5ATTR.h** . . . . **??**
src/**hdf5_extras.h** . . . . **??**
src/**hdf5mine.h** . . . . **??**
src/**hdf5vfs.h** . . . . **??**
src/**ifile.h** . . . . **??**
src/IFileOpen.c . . . . **??**
src/**string_funcs.h** . . . . **??**

Chapter 3

Data Structure Documentation

3.1 bStream Struct Reference

Data Fields

- bstring **buff**
- void ∗ **parm**
- bNread **readFnPtr**
- int **isEOF**
- int **maxBuffSz**

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

- src/bstrlib.c

3.2 bstrList Struct Reference

```
#include <bstrlib.h>
```

Data Fields

- int **qty**
- int **mlen**
- bstring ∗ **entry**

3.2.1 Detailed Description

This structure contains the data needed to implement a list of bstrings.

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

- src/bstrlib.h

3.3 charField Struct Reference

Data Fields

- LONG_TYPE **content** [CFCLEN]

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

- src/bstrlib.c

3.4 fcurl_data Struct Reference

Data Fields

- enum fcurl_type_e **type**

- char ∗ **buffer**

size_t **buffer_len**

size_t **buffer_pos**

int **still_running**

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

- src/hdf5mine.h

3.5 genBstrList Struct Reference

Data Fields

- bstring **b**
- struct bstrList ∗ **bl**

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

- src/bstrlib.c

3.6 GEOInfo_t Struct Reference

Data Fields

- int **pbx**
- int **pby**
- double **rbx** [21]

- double **rby** [21]
- double **rabx** [3]
- double **raby** [3]
- double **covx** [21][21]
- double **covy** [21][21]

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

- src/hdf5mine.h

3.7 HDFFile Struct Reference

Data Fields

- sqlite3_file **base**
- IFILE ∗ **ifilep**
- char ∗ **aBuffer**
- int **nBuffer**
- sqlite3_int64 **iBufferOfst**

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

- src/hdf5vfs.h

3.8 IFILE Struct Reference

Data Fields

- hid_t **ifile_id**
- int **readPastEOF**
- int **last_operation_status**
- int **access**
- long int **file_position**
- long int **size**
- bstring **last_operation_status_message**

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

- src/ifile.h

3.9 tagbstring Struct Reference

```
#include <bstrlib.h>
```

Data Fields

- int **mlen**
- int **slen**
- unsigned char ∗ **data**

3.9.1 Detailed Description

The tagbstring structure represents the bstring datatype that is used to have safe strings. The string is stored as a standard C string, named `data`. The length of the string is stored as the item `slen`, while the allocated size of `data` is stored in the item `mlen`.

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

- src/bstrlib.h

Chapter 4

File Documentation

4.1 src/bstrlib.h File Reference

```
#include <stdarg.h>
#include <string.h>
#include <limits.h>
#include <ctype.h>
```

Data Structures

- struct bstrList
- struct tagbstring

Macros

- #define **BSTR_ERR** (-1)
- #define **BSTR_OK** (0)
- #define **BSTR_BS_BUFF_LENGTH_GET** (0)
- #define **cstr2bstr** bfromcstr
- #define **bstrchr**(b, c) bstrchrp ((b), (c), 0)
- #define **bstrrchr**(b, c) bstrrchrp ((b), (c), blength(b)-1)
- #define **bvformata**(ret, b, fmt, lastarg)
- #define **blengthe**(b, e) (((b) == (void ∗)0 || (b)->slen < 0) ? (int)(e) : ((b)->slen))
- #define **blength**(b) (blengthe ((b), 0))
- #define **bdataofse**(b, o, e) (((b) == (void ∗)0 || (b)->data == (void∗)0) ? (char ∗)(e) : ((char ∗)(b)->data) + (o))
- #define **bdataofs**(b, o) (bdataofse ((b), (o), (void ∗)0))
- #define **bdatae**(b, e) (bdataofse (b, 0, e))
- #define **bdata**(b) (bdataofs (b, 0))
- #define **bchare**(b, p, e) ((((unsigned)(p)) < (unsigned)blength(b)) ? ((b)->data[(p)]) : (e))
- #define **bchar**(b, p) bchare ((b), (p), '\0')
- #define **bsStaticMlen**(q, m) {(m), (int) sizeof(q)-1, (unsigned char ∗) ("" q "")}
- #define **bsStatic**(q) bsStaticMlen(q,-__LINE__)
- #define **bsstatic**(q) &(struct tagbstring)bsStatic(q)
- #define **bsStaticBlkParms**(q) ((void ∗)("" q "")), ((int) sizeof(q)-1)

- #define **cstr2tbstr** btfromcstr
- #define **btfromcstr**(t, s)
- #define **blk2tbstr**(t, s, l)
- #define **btfromblk**(t, s, l) blk2tbstr(t,s,l)
- #define **bmid2tbstr**(t, b, p, l)
- #define **btfromblkltrimws**(t, s, l)
- #define **btfromblkrtrimws**(t, s, l)
- #define **btfromblktrimws**(t, s, l)
- #define **bwriteprotect**(t) { if ((t).mlen >= 0) (t).mlen = -1; }
- #define **bwriteallow**(t) { if ((t).mlen == -1) (t).mlen = (t).slen + ((t).slen == 0); }
- #define **biswriteprotected**(t) ((t).mlen <= 0)

Typedefs

- typedef struct tagbstring ∗ **bstring**
- typedef const struct tagbstring ∗ **const_bstring**
- typedef int(∗ **bNgetc**)(void ∗parm)
- typedef size_t(∗ **bNread**)(void ∗buff, size_t elsize, size_t nelem, void ∗parm)

Functions

- bstring **bfromcstr** (const char ∗str)
- bstring **bfromcstralloc** (int mlen, const char ∗str)
- bstring **blk2bstr** (const void ∗blk, int len)
- char ∗ **bstr2cstr** (const_bstring s, char z)
- int **bcstrfree** (char ∗s)
- bstring **bstrcpy** (const_bstring b1)
- int **bassign** (bstring a, const_bstring b)
- int **bassignmidstr** (bstring a, const_bstring b, int left, int len)
- int **bassigncstr** (bstring a, const char ∗str)
- int **bassignblk** (bstring a, const void ∗s, int len)
- int **bdestroy** (bstring b)
- int **balloc** (bstring s, int len)
- int **ballocmin** (bstring b, int len)
- bstring **bmidstr** (const_bstring b, int left, int len)
- int **bconcat** (bstring b0, const_bstring b1)
- int **bconchar** (bstring b0, char c)
- int **bcatcstr** (bstring b, const char ∗s)
- int **bcatblk** (bstring b, const void ∗s, int len)
- int **binsert** (bstring s1, int pos, const_bstring s2, unsigned char fill)
- int **binsertch** (bstring s1, int pos, int len, unsigned char fill)
- int **breplace** (bstring b1, int pos, int len, const_bstring b2, unsigned char fill)
- int **bdelete** (bstring s1, int pos, int len)
- int **bsetstr** (bstring b0, int pos, const_bstring b1, unsigned char fill)
- int **btrunc** (bstring b, int n)
- int **bstricmp** (const_bstring b0, const_bstring b1)
- int **bstrnicmp** (const_bstring b0, const_bstring b1, int n)
- int **biseqcaseless** (const_bstring b0, const_bstring b1)
- int **bisstemeqcaselessblk** (const_bstring b0, const void ∗blk, int len)
- int **biseq** (const_bstring b0, const_bstring b1)

- int **bisstemeqblk** (const_bstring b0, const void ∗blk, int len)
- int **biseqcstr** (const_bstring b, const char ∗s)
- int **biseqcstrcaseless** (const_bstring b, const char ∗s)
- int **bstrcmp** (const_bstring b0, const_bstring b1)
- int **bstrncmp** (const_bstring b0, const_bstring b1, int n)
- int **binstr** (const_bstring s1, int pos, const_bstring s2)
- int **binstrr** (const_bstring s1, int pos, const_bstring s2)
- int **binstrcaseless** (const_bstring s1, int pos, const_bstring s2)
- int **binstrrcaseless** (const_bstring s1, int pos, const_bstring s2)
- int **bstrchrp** (const_bstring b, int c, int pos)
- int **bstrrchrp** (const_bstring b, int c, int pos)
- int **binchr** (const_bstring b0, int pos, const_bstring b1)
- int **binchrr** (const_bstring b0, int pos, const_bstring b1)
- int **bninchr** (const_bstring b0, int pos, const_bstring b1)
- int **bninchrr** (const_bstring b0, int pos, const_bstring b1)
- int **bfindreplace** (bstring b, const_bstring find, const_bstring repl, int pos)
- int **bfindreplacecaseless** (bstring b, const_bstring find, const_bstring repl, int pos)
- struct bstrList ∗ **bstrListCreate** (void)
- int **bstrListDestroy** (struct bstrList ∗sl)
- int **bstrListAlloc** (struct bstrList ∗sl, int msz)
- int **bstrListAllocMin** (struct bstrList ∗sl, int msz)
- struct bstrList ∗ **bsplit** (const_bstring str, unsigned char splitChar)
- struct bstrList ∗ **bsplits** (const_bstring str, const_bstring splitStr)
- struct bstrList ∗ **bsplitstr** (const_bstring str, const_bstring splitStr)
- bstring **bjoin** (const struct bstrList ∗bl, const_bstring sep)
- int **bsplitcb** (const_bstring str, unsigned char splitChar, int pos, int(∗cb)(void ∗parm, int ofs, int len), void ∗parm)
- int **bsplitscb** (const_bstring str, const_bstring splitStr, int pos, int(∗cb)(void ∗parm, int ofs, int len), void ∗parm)
- int **bsplitstrcb** (const_bstring str, const_bstring splitStr, int pos, int(∗cb)(void ∗parm, int ofs, int len), void ∗parm)
- int **bpattern** (bstring b, int len)
- int **btoupper** (bstring b)
- int **btolower** (bstring b)
- int **bltrimws** (bstring b)
- int **brtrimws** (bstring b)
- int **btrimws** (bstring b)
- bstring **bformat** (const char ∗fmt,...)
- int **bformata** (bstring b, const char ∗fmt,...)
- int **bassignformat** (bstring b, const char ∗fmt,...)
- int **bvcformata** (bstring b, int count, const char ∗fmt, va_list arglist)
- bstring **bgets** (bNgetc getcPtr, void ∗parm, char terminator)
- bstring **bread** (bNread readPtr, void ∗parm)
- int **bgetsa** (bstring b, bNgetc getcPtr, void ∗parm, char terminator)
- int **bassigngets** (bstring b, bNgetc getcPtr, void ∗parm, char terminator)
- int **breada** (bstring b, bNread readPtr, void ∗parm)
- struct bStream ∗ **bsopen** (bNread readPtr, void ∗parm)
- void ∗ **bsclose** (struct bStream ∗s)
- int **bsbufflength** (struct bStream ∗s, int sz)
- int **bsreadln** (bstring b, struct bStream ∗s, char terminator)
- int **bsreadlns** (bstring r, struct bStream ∗s, const_bstring term)
- int **bsread** (bstring b, struct bStream ∗s, int n)
- int **bsreadlna** (bstring b, struct bStream ∗s, char terminator)
- int **bsreadlnsa** (bstring r, struct bStream ∗s, const_bstring term)

- int **bsreada** (bstring b, struct bStream ∗s, int n)
- int **bsunread** (struct bStream ∗s, const_bstring b)
- int **bspeek** (bstring r, const struct bStream ∗s)
- int **bssplitscb** (struct bStream ∗s, const_bstring splitStr, int(∗cb)(void ∗parm, int ofs, const_bstring entry), void ∗parm)
- int **bssplitstrcb** (struct bStream ∗s, const_bstring splitStr, int(∗cb)(void ∗parm, int ofs, const_bstring entry), void ∗parm)
- int **bseof** (const struct bStream ∗s)

4.1.1 Macro Definition Documentation

4.1.1.1 #define blk2tbstr(*t, s, l*)

Value:

```
{                \
    (t).data = (unsigned char *) (s); \
    (t).slen = l;                     \
    (t).mlen = -1;                    \
}
```

4.1.1.2 #define bmid2tbstr(*t, b, p, l*)

Value:

```
{                                                \
    const_bstring bstrtmp_s = (b);                                          \
    if (bstrtmp_s && bstrtmp_s->data && bstrtmp_s->slen >= 0) {             \
        int bstrtmp_left = (p);                                             \
        int bstrtmp_len  = (l);                                             \
        if (bstrtmp_left < 0) {                                             \
            bstrtmp_len += bstrtmp_left;                                    \
            bstrtmp_left = 0;                                               \
        }                                                                   \
        if (bstrtmp_len > bstrtmp_s->slen - bstrtmp_left)                   \
            bstrtmp_len = bstrtmp_s->slen - bstrtmp_left;                   \
        if (bstrtmp_len <= 0) {                                             \
            (t).data = (unsigned char *)"";                                 \
            (t).slen = 0;                                                   \
        } else {                                                            \
            (t).data = bstrtmp_s->data + bstrtmp_left;                      \
            (t).slen = bstrtmp_len;                                         \
        }                                                                   \
    } else {                                                                \
        (t).data = (unsigned char *)"";                                     \
        (t).slen = 0;                                                       \
    }                                                                       \
    (t).mlen = -__LINE__;                                                   \
}
```

4.1.1.3 #define btfromblkltrimws(*t, s, l*)

Value:

```
{                                             \
    int bstrtmp_idx = 0, bstrtmp_len = (l);                                 \
    unsigned char * bstrtmp_s = (s);                                        \
    if (bstrtmp_s && bstrtmp_len >= 0) {                                    \
        for (; bstrtmp_idx < bstrtmp_len; bstrtmp_idx++) {                  \
            if (!isspace (bstrtmp_s[bstrtmp_idx])) break;                   \
        }                                                                   \
    }                                                                          \
    (t).data = bstrtmp_s + bstrtmp_idx;                                        \
    (t).slen = bstrtmp_len - bstrtmp_idx;                                      \
    (t).mlen = -__LINE__;                                                      \
}
```

4.1.1.4 #define btfromblkrtrimws(*t*, *s*, *l*)

Value:

```
{                                            \
    int bstrtmp_len = (l) - 1;                                                 \
    unsigned char * bstrtmp_s = (s);                                           \
    if (bstrtmp_s && bstrtmp_len >= 0) {                                       \
        for (; bstrtmp_len >= 0; bstrtmp_len--) {                              \
            if (!isspace (bstrtmp_s[bstrtmp_len])) break;                      \
        }                                                                      \
    }                                                                          \
    (t).data = bstrtmp_s;                                                      \
    (t).slen = bstrtmp_len + 1;                                                \
    (t).mlen = -__LINE__;                                                      \
}
```

4.1.1.5 #define btfromblktrimws(*t*, *s*, *l*)

Value:

```
{                                             \
    int bstrtmp_idx = 0, bstrtmp_len = (l) - 1;                                \
    unsigned char * bstrtmp_s = (s);                                           \
    if (bstrtmp_s && bstrtmp_len >= 0) {                                       \
        for (; bstrtmp_idx <= bstrtmp_len; bstrtmp_idx++) {                    \
            if (!isspace (bstrtmp_s[bstrtmp_idx])) break;                      \
        }                                                                      \
        for (; bstrtmp_len >= bstrtmp_idx; bstrtmp_len--) {                    \
            if (!isspace (bstrtmp_s[bstrtmp_len])) break;                      \
        }                                                                      \
    }                                                                          \
    (t).data = bstrtmp_s + bstrtmp_idx;                                        \
    (t).slen = bstrtmp_len + 1 - bstrtmp_idx;                                  \
    (t).mlen = -__LINE__;                                                      \
}
```

4.1.1.6 #define btfromcstr(*t*, *s*)

Value:

```
{                                            \
    (t).data = (unsigned char *) (s);                                  \
    (t).slen = ((t).data) ? ((int) (strlen) ((char *)(t).data)) : 0; \
    (t).mlen = -1;                                                     \
}
```

4.1.1.7 #define bvformata(*ret*, *b*, *fmt*, *lastarg*)

Value:

```
{ \
bstring bstrtmp_b = (b); \
const char * bstrtmp_fmt = (fmt); \
int bstrtmp_r = BSTR_ERR, bstrtmp_sz = 16; \
    for (;;) { \
        va_list bstrtmp_arglist; \
        va_start (bstrtmp_arglist, lastarg); \
        bstrtmp_r = bvcformata (bstrtmp_b, bstrtmp_sz, bstrtmp_fmt, bstrtmp_arglist); \
        va_end (bstrtmp_arglist); \
        if (bstrtmp_r >= 0) { /* Everything went ok */ \
            bstrtmp_r = BSTR_OK; \
            break; \
        } else if (-bstrtmp_r <= bstrtmp_sz) { /* A real error? */ \
            bstrtmp_r = BSTR_ERR; \
            break; \
        } \
        bstrtmp_sz = -bstrtmp_r; /* Doubled or target size */ \
    } \
    ret = bstrtmp_r; \
}
```

4.2 src/GS_CheckInternalName.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_CheckInternalName (hid_t object_id, const_bstring thename)

    GS_CheckInternalName checks for existence of an object name in the file.

4.2.1 Function Documentation

4.2.1.1 int GS_CheckInternalName (hid_t *object_id*, const_bstring *thename*)

GS_CheckInternalName checks for existence of an object name in the file.

GS_CheckInternalName() checks if there are any objects in the GEOSCIpy database with this name.

See also

GS_CheckChannels(), GS_CheckWindow()

Parameters

in	*object_id*	Handle of an already-open GeoSciPy object.
in	*name*	Name of the object one wishes to test. Must be a relative pathname, relative to the object in "object_id". If object_id is a file, then this can be an absolute name. For example: /groupa/group1/datasetw.

Returns

TRUE if the name exists, FALSE if not.

Example:

Test if the name "Channel_3" already exists or not:

```
hid_t       file_id;
bstring name = bfromcstr("Channel_3");
if(GS_CheckInternalName(file_id,name) ){
   printf("Name: \"Channel_3\" is already in use.\n");
}// endif
bdestroy(name);
```

4.3 src/GS_ConvertFromHDFDatatype.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- int GS_ConvertFromHDFDatatype (hid_t datatype)

  GS_ConvertFromHDFDatatype returns the GeoSci datatype for the given HDF datatype.

4.3.1 Function Documentation

4.3.1.1 int GS_ConvertFromHDFDatatype (hid_t *datatype*)

GS_ConvertFromHDFDatatype returns the GeoSci datatype for the given HDF datatype.

GS_ConvertToHDFDatatype() returns the GeoSci datatype for the given HDF datatype

Parameters

in	*datatype*	A handle to an HDF5 datatype.

Returns

The GeoSci typecode is returned, and is 0 if invalid. Valid datatypes are:
GS_DATATYPE_UI1 (Unsigned Integer, 1-bit)
GS_DATATYPE_UI8 (Unsigned Integer, 8-bit)
GS_DATATYPE_SI8 (Signed Integer, 8-bit)
GS_DATATYPE_CI8 (Complex Integer, 8-bit (not supported))

GS_DATATYPE_UI16 (Unsigned Integer, 16-bit)
GS_DATATYPE_SI16 (Signed Integer, 16-bit)
GS_DATATYPE_CI16 (Complex Integer, 16-bit)
GS_DATATYPE_UI32 (Unsigned Integer, 32-bit)
GS_DATATYPE_SI32 (Signed Integer, 32-bit)
GS_DATATYPE_CI32 (Complex Integer, 32-bit)
GS_DATATYPE_CI64 (Complex Integer, 64-bit)
GS_DATATYPE_R32 (Real, 32-bit)
GS_DATATYPE_R64 (Real, 64-bit)
GS_DATATYPE_C64 (Complex, 64-bit)
GS_DATATYPE_C128 (Complex, 128-bit)
GS_DATATYPE_UI64 (Unsigned Integer, 64-bit)
GS_DATATYPE_SI64 (Signed Integer, 64-bit)

Example

Get the GeoSci equivalent of the HDF5 type H5T_NATIVE_INT8:

```
int gs_type;
gs_type = GS_ConvertFromHDFDatatype(H5T_NATIVE_INT8);
if(!gs_type) {
   printf("GS_ConvertFromHDFDatatype failure\n");
}
```

Details

Currently this checks against the HDF5 "NATIVE" datatypes. I suspect this will not work if the data is in a different byte order. Need to fix this. (LEP: Aug 9, 2014).

4.4 src/GS_ConvertToHDFDatatype.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1
- #define DEBUG

    GS_ConvertToHDFDatatype returns the HDF datatype for the given GeoSci datatype.

Functions

- hid_t **GS_ConvertToHDFDatatype** (int datatype)

4.4.1 Macro Definition Documentation

4.4.1.1 #define DEBUG

GS_ConvertToHDFDatatype returns the HDF datatype for the given GeoSci datatype.

GS_ConvertToHDFDatatype() returns the HDF5 type for the given GeoSciPy type.

Parameters

in	*datatype*	An integer representing a GeoSci datatype. Valid datatypes are: `GS_DATATYPE_UI1` (Unsigned Integer, 1-bit) `GS_DATATYPE_UI8` (Unsigned Integer, 8-bit) `GS_DATATYPE_SI8` (Signed Integer, 8-bit) `GS_DATATYPE_CI8` (Complex Integer, 8-bit (not supported)) `GS_DATATYPE_UI16` (Unsigned Integer, 16-bit) `GS_DATATYPE_SI16` (Signed Integer, 16-bit) `GS_DATATYPE_CI16` (Complex Integer, 16-bit) `GS_DATATYPE_UI32` (Unsigned Integer, 32-bit) `GS_DATATYPE_SI32` (Signed Integer, 32-bit) `GS_DATATYPE_CI32` (Complex Integer, 32-bit) `GS_DATATYPE_CI64` (Complex Integer, 64-bit) `GS_DATATYPE_R32` (Real, 32-bit) `GS_DATATYPE_R64` (Real, 64-bit) `GS_DATATYPE_C64` (Complex, 64-bit) `GS_DATATYPE_C128` (Complex, 128-bit) `GS_DATATYPE_UI64` (Unsigned Integer, 64-bit) `GS_DATATYPE_SI64` (Signed Integer, 64-bit)

Returns

An HDF5 datatype is returned, and is negative if invalid. Complex datatypes should be H5Tclose()'d when done with them.

Example

Get the HDF5 equivalent of the raster type complex-integer-16-bit.

```
hid_t hdf_type;

hdf_type = GS_ConvertToHDFDatatype(GS_DATATYPE_CI16);
if(hdf_type<0) {
   printf("GS_ConvertToHDFDatatype failure\n");
}
```

4.5 src/GS_CreateAccessPlist.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <hdf5.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **TRUE** 1
- #define **FALSE** 0
- #define **CHUNK_CACHE_NELMTS** 521
- #define **CHUNK_CACHE_PREEMPT** 0.0
- #define **CHUNK_CACHE_SIZE** (2 ∗ 1024 ∗1024)
- #define **DRIVER** "H5FD_SEC2"
- #define **DRIVER_CORE_INCREMENT** (100 ∗ 1024 ∗ 1024)
- #define **DRIVER_CORE_BACKING_STORE** TRUE

Functions

- hid_t GS_CreateAccessPlist ()

  GS_CreateAccessPlist creates and HDF5 access parameter list.

4.5.1 Function Documentation

4.5.1.1 hid_t GS_CreateAccessPlist (void)

GS_CreateAccessPlist creates and HDF5 access parameter list.

GS_CreateAccessPlist() returns an access parameter list for use when creating or opening an HDF5 file. It sets raw data chunk cache parameters, as well as a driver for file I/O.

See also

GS_FileOpen(), GS_FileCreate(), GS_FileCreateEmpty()

Returns

This routine returns a valid HDF5 handle to an access_plist, which can be used when creating or opening an HDF5 file.

4.6 src/GS_DatasetClose.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- hid_t GS_DatasetClose (hid_t id)

  GS_DatasetClose closes a dataset in a GeoSci datafile.

4.6.1 Function Documentation

4.6.1.1 hid_t GS_DatasetClose (hid_t *id*)

GS_DatasetClose closes a dataset in a GeoSci datafile.

GS_DatasetClose() closes a dataset in a GeoSci file. This is meant to work with any type of dataset.

See also

GS_DatasetOpen()

Parameters

in	*id*	A handle for the dataset. Perhaps from GS_DatasetOpen().

Returns

`TRUE` on sucess, `FALSE` on failure.

Example

Close an image raster dataset:

```
hid_t raster_id;
if(!GS_DatasetClose(raster_id)) {
  printf("Failed to close dataset.\n");
}
```

4.7 src/GS_DatasetCopy.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- hid_t GS_DatasetCopy (hid_t source, hid_t destination_group_id, const_bstring destination_dataset_name)

  GS_DatasetCopy copies a dataset to an existing Group.

4.7.1 Function Documentation

4.7.1.1 hid_t GS_DatasetCopy (hid_t *source,* hid_t *destination_group_id,* const_bstring *destination_dataset_name*)

GS_DatasetCopy copies a dataset to an existing Group.

GS_DatasetCopy() copies a dataset to an existing group. This can be in the same group, a different group, or a different file.

See also

GS_DatasetClose(), GS_DatasetOpen()

Parameters

in	*source*	A handle for a dataset.
in	*destination_↩ group_id*	The handle of the destination group.
in	*destination_↩ dataset_name*	The name of the new dataset to create.

Returns

The handle of the new dataset is returned. If it is less than zero, the copy failed.

Example

Copy an image raster dataset to a new file. We have already opened the destination group in the new file.

```
hid_t source_raster_id;
hid_t destination_group_id;
hid_t raster_id;
bstring name = bfromcstr("r3");
raster_id = GS_DatasetCopy(source_raster_id, destination_group_id, name);
bdestroy(name);
if(raster_id < 0) {
  printf("Failed to copy dataset.\n");
}
```

4.8 src/GS_DatasetCreate.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- hid_t GS_DatasetCreate (hid_t source, const_bstring name, int datatype, int ndims, int ∗size, int is_extendable, int is_compressed)

  GS_DatasetCreate creates a dataset in a GeoSci datafile.

4.8.1 Function Documentation

4.8.1.1 hid_t GS_DatasetCreate (hid_t *source*, const_bstring *name*, int *datatype*, int *ndims*, int * *size*, int *is_extendable*, int *is_compressed*)

GS_DatasetCreate creates a dataset in a GeoSci datafile.

GS_DatasetCreate() creates a dataset in a GeoSciPy file. This is meant to work with any type of dataset.

See also

GS_DatasetOpen(), GS_DatasetClose()

Parameters

`in`	*source*	A handle for the file or other container within which to create the dataset.
`in`	*name*	The name of the dataset to create.
`in`	*datatype*	The datatype of the numbers to store. Must be one of: `GS_DATATYPE_UI1` (Unsigned Integer, 1-bit) `GS_DATATYPE_UI8` (Unsigned Integer, 8-bit) `GS_DATATYPE_SI8` (Signed Integer, 8-bit) `GS_DATATYPE_CI8` (Complex Integer, 8-bit (not supported)) `GS_DATATYPE_UI16` (Unsigned Integer, 16-bit) `GS_DATATYPE_SI16` (Signed Integer, 16-bit) `GS_DATATYPE_CI16` (Complex Integer, 16-bit) `GS_DATATYPE_UI32` (Unsigned Integer, 32-bit) `GS_DATATYPE_SI32` (Signed Integer, 32-bit) `GS_DATATYPE_CI32` (Complex Integer, 32-bit) `GS_DATATYPE_CI64` (Complex Integer, 64-bit) `GS_DATATYPE_R32` (Real, 32-bit) `GS_DATATYPE_R64` (Real, 64-bit) `GS_DATATYPE_C64` (Complex, 64-bit) `GS_DATATYPE_C128` (Complex, 128-bit) `GS_DATATYPE_UI64` (Unsigned Integer, 64-bit) `GS_DATATYPE_SI64` (Signed Integer, 64-bit)

in	*ndims*	The number of dimensions for the dataset.
in	*size*	The (initial) size for each dimension
in	*is_extendable*	Set to TRUE to make the dataset size extendable, set to FALSE to make the dataset size fixed.
in	*is_compressed*	Set to TRUE to make the dataset compressed, set to FALSE to make the dataset uncompressed.

?

Returns

The handle of the newly-created and opened dataset is returned. If it is less than zero, the creation failed.

Example

Create an image raster dataset, with size 1024X512:

```
hid_t image_id, raster_id;
int   size[2];
size[0]=512;
size[1]=1024;
bstring name = bfromcstr("r1");
raster_id = GS_DatasetCreate(image_id, name,2,size,FALSE,FALSE);
bdestroy(name);
if(raster_id < 0) {
  printf("Failed to create dataset.\n");
}
```

4.9 src/GS_DatasetDelete.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- int GS_DatasetDelete (hid_t file_id, const_bstring dataset_name)

  GS_DatasetDelete deletes a a dataset in a GeoSci datafile.

4.9.1 Function Documentation

4.9.1.1 int GS_DatasetDelete (hid_t *file_id*, const_bstring *dataset_name*)

GS_DatasetDelete deletes a a dataset in a GeoSci datafile.

GS_DatasetDelete() deletes a dataset in a GeoSci datafile. This is meant to work with any type of dataset.

See also

GS_DatasetOpen(), GS_DatasetClose()

Parameters

in	*id*	A handle for the file.
in	*dataset_name*	The name of the dataset in the file. Should start with a "/".

Returns

TRUE on sucess, FALSE on failure.

Example

Delete an image raster dataset named "Image1/r1":

```
hid_t file_id;
bstring name = bfromcstr("/Image1/r1");
if(!GS_DatasetDelete(file_id, name)) {
  printf("Failed to delete dataset.\n");
}
bdelete(name);
```

4.10 src/GS_DatasetGetDatatype.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_DatasetGetDatatype (hid_t id, const_bstring name)
  GS_DatasetGetDatatype returns the GeoSci datatype for the Dataset.

4.10.1 Function Documentation

4.10.1.1 int GS_DatasetGetDatatype (hid_t *id,* const_bstring *name*)

GS_DatasetGetDatatype returns the GeoSci datatype for the Dataset.

GS_DatasetGetDatatype returns the GeoSci datatype for the Dataset.

See also

GS_DatasetOpen(), GS_DatasetCreate()

Parameters

in	*id*	File handle of selected GeoSci file, or object handle of the container for the Dataset.
in	*object_name*	Name of object to query.

Returns

The GeoSci datatype code is returned, and is 0 if there is an error. Valid datatypes are: `GS_DATATYPE_UI1` (Unsigned Integer, 1-bit)
`GS_DATATYPE_UI8` (Unsigned Integer, 8-bit)
`GS_DATATYPE_SI8` (Signed Integer, 8-bit)
`GS_DATATYPE_CI8` (Complex Integer, 8-bit (not supported))
`GS_DATATYPE_UI16` (Unsigned Integer, 16-bit)
`GS_DATATYPE_SI16` (Signed Integer, 16-bit)
`GS_DATATYPE_CI16` (Complex Integer, 16-bit)
`GS_DATATYPE_UI32` (Unsigned Integer, 32-bit)
`GS_DATATYPE_SI32` (Signed Integer, 32-bit)
`GS_DATATYPE_CI32` (Complex Integer, 32-bit)
`GS_DATATYPE_CI64` (Complex Integer, 64-bit)
`GS_DATATYPE_R32` (Real, 32-bit)
`GS_DATATYPE_R64` (Real, 64-bit)
`GS_DATATYPE_C64` (Complex, 64-bit)
`GS_DATATYPE_C128` (Complex, 128-bit)
`GS_DATATYPE_UI64` (Unsigned Integer, 64-bit)
`GS_DATATYPE_SI64` (Signed Integer, 64-bit)

Example

Get the GeoSci type code for an image raster Dataset.

```
hid_t image_id;
int gs_type;
bstring name = bfromcstr("r1");
gs_type = GS_DatasetGetDatatype(image_id,name);
bdestroy(name);
if(!gs_type) {
   printf("GS_DatasetGetDatatype failure\n");
}
```

4.11 src/GS_DatasetGetDatatypeByID.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_DatasetGetDatatypeByID (hid_t id)

  GS_DatasetGetDatatypeByID returns the GeoSci datatype for the Dataset.

4.11.1 Function Documentation

4.11.1.1 int GS_DatasetGetDatatypeByID (hid_t *id*)

GS_DatasetGetDatatypeByID returns the GeoSci datatype for the Dataset.

GS_DatasetGetDatatypeByID returns the GeoSci datatype for the Dataset.

See also

GS_DatasetOpen(), GS_DatasetCreate()

Parameters

in	*id*	Datset handle

Returns

The GeoSci datatype code is returned, and is 0 if there is an error. Valid datatypes are: `GS_DATATYPE_UI1` (Unsigned Integer, 1-bit)
`GS_DATATYPE_UI8` (Unsigned Integer, 8-bit)
`GS_DATATYPE_SI8` (Signed Integer, 8-bit)
`GS_DATATYPE_CI8` (Complex Integer, 8-bit (not supported))
`GS_DATATYPE_UI16` (Unsigned Integer, 16-bit)
`GS_DATATYPE_SI16` (Signed Integer, 16-bit)
`GS_DATATYPE_CI16` (Complex Integer, 16-bit)
`GS_DATATYPE_UI32` (Unsigned Integer, 32-bit)
`GS_DATATYPE_SI32` (Signed Integer, 32-bit)
`GS_DATATYPE_CI32` (Complex Integer, 32-bit)
`GS_DATATYPE_CI64` (Complex Integer, 64-bit)
`GS_DATATYPE_R32` (Real, 32-bit)
`GS_DATATYPE_R64` (Real, 64-bit)
`GS_DATATYPE_C64` (Complex, 64-bit)
`GS_DATATYPE_C128` (Complex, 128-bit)
`GS_DATATYPE_UI64` (Unsigned Integer, 64-bit)
`GS_DATATYPE_SI64` (Signed Integer, 64-bit)

Example

Get the GeoSci type code for an image raster Dataset.

```
hid_t dataset_id;
int gs_type;
gs_type = GS_DatasetGetDatatypeByID(dataset_id);
if(!gs_type) {
   printf("GS_DatasetGetDatatypeByID failure\n");
}
```

4.12 src/GS_DatasetGetDimensions.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_DatasetGetDimensions (hid_t id, const_bstring name, int **size)

  GS_DatasetGetDimensions returns the current dimensions for the Dataset.

4.12.1 Function Documentation

4.12.1.1 int GS_DatasetGetDimensions (hid_t *id*, const_bstring *name*, int ** *size*)

GS_DatasetGetDimensions returns the current dimensions for the Dataset.

GS_DatasetGetDimensions returns the current dimensions for the Dataset.

See also

GS_DatasetGetDatatype(), GS_DatasetOpen(), GS_DatasetCreate()

Parameters

in	*id*	File handle of selected GeoSci file, or object handle of the container for the Dataset.
in	*name*	Name of object to query.
out	*size*	A vector of integers: the size of each dimension: the number of data-objects per dimension.

Returns

The number of dimensions in the Dataset, or zero if there is an error. Also returns the length of each dimension (number of data-objects), in the integer vector `size`. This vector is allocated by this function, and should be HFree()'d when done with it.

Example

Get the size of an image raster Dataset.

```
hid_t image_id;
int ndims;
int *size;
bstring name = bfromcstr("r1");
ndims = GS_DatasetGetDimensions(image_id,name,&size);
bdestroy(name);
if(!ndims) {
   printf("GS_DatasetGetDimensions failure\n");
}
printf("the dataset has %d dimensions:\n",ndims);
for(i=0;i<ndims;i++)
  printf("dimension %d size is: %d data-objects\n",i+1,size[i]);
HFree(size);
```

Details

Note that this function returns dimensions in terms of the number of data-objects. It does NOT return the size in bytes.

4.13 src/GS_DatasetGetDimensionsByID.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_DatasetGetDimensionsByID (hid_t id, int **size)

  GS_DatasetGetDimensionsByID returns the current dimensions for the Dataset.

4.13.1 Function Documentation

4.13.1.1 int GS_DatasetGetDimensionsByID (hid_t *id,* int ** *size*)

GS_DatasetGetDimensionsByID returns the current dimensions for the Dataset.

GS_DatasetGetDimensionsByID returns the current dimensions for the Dataset-ID.

See also

GS_DatasetGetDimensions(), GS_DatasetGetDatatype(), GS_DatasetOpen(), GS_DatasetCreate()

Parameters

in	*id*	object handle of the Dataset.
out	*size*	A vector of integers: the size of each dimension: the number of data-objects per dimension.

Returns

The number of dimensions in the Dataset, or zero if there is an error. Also returns the length of each dimension (number of data-objects), in the integer vector `size`. This vector is allocated by this function, and should be HFree()'d when done with it.

Example

Get the size of an image raster Dataset.

```
hid_t raster_id;
int ndims;
int *size;
ndims = GS_DatasetGetDimensionsByID(raster_id,&size);
if(!ndims) {
   printf("GS_DatasetGetDimensionsByID failure\n");
}
printf("the dataset has %d dimensions:\n",ndims);
for(i=0;i<ndims;i++)
  printf("dimension %d size is: %d data-objects\n",i+1,size[i]);
HFree(size);
```

Details

Note that this function returns dimensions in terms of the number of data-objects. It does NOT return the size in bytes.

4.14 src/GS_DatasetGetParent.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- hid_t GS_DatasetGetParent (hid_t id)

  GS_DatasetGetParent returns the parent object-id for a Dataset.

4.14.1 Function Documentation

4.14.1.1 hid_t GS_DatasetGetParent (hid_t *id*)

GS_DatasetGetParent returns the parent object-id for a Dataset.

GS_DatasetGetParent() returns the parent object-id for a Dataset.

Parameters

in	*id*	The handle of the Dataset.

Returns

Returns the object-id of the parent object (>=0), or ERROR (-1) on failure.

Example

Get the parent of a Dataset.

```
hid_t dataset_id, object_id;
object_id = GS_DatasetGetParent(dataset_id);
if(object_id < 0){
  printf("Could not open the parent of the Dataset.\n");
}
```

4.15 src/GS_DatasetGetType.c File Reference

```
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_DatasetGetType (hid_t dataset_id)

  GS_DatasetGetType returns the type of a GeoSci Dataset.

4.15.1 Function Documentation

4.15.1.1 int GS_DatasetGetType (hid_t *dataset_id*)

GS_DatasetGetType returns the type of a GeoSci Dataset.

GS_DatasetGetType returns the type of a GeoSci Dataset.

See also

GS_DatasetOpen(), GS_DatasetCreate()

Parameters

in	*dataset_id*	A handle for the dataset.

Returns

The dataset type code is returned on success: DATASET_TYPE_INTERNAL_FILE 1 DATASET_TYPE_RASTER 2 DATASET_TYPE_METADATA 3 DATASET_TYPE_METADATA_IFILE 4

On failure, the following type code is returned: DATASET_TYPE_UNKNOWN

Example

Get the type of an image raster dataset:

```
hid_t raster_id;

if(GS_DatasetGetType(raster_id) != DATASET_TYPE_RASTER ) {
  printf("Dataset is not a raster, as expected.\n");
  return;
}
```

4.16 src/GS_DatasetOpen.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- hid_t GS_DatasetOpen (hid_t source, const_bstring name)

  GS_DatasetOpen opens a dataset in a GeoSci datafile.

4.16.1 Function Documentation

4.16.1.1 hid_t GS_DatasetOpen (hid_t *source*, const_bstring *name*)

GS_DatasetOpen opens a dataset in a GeoSci datafile.

GS_DatasetOpen() opens a dataset in a GeoSciPy file. This is meant to work with any type of dataset.

See also

GS_DatasetClose()

Parameters

in	*source*	A handle for a file or other container that has a dataset in it.
in	*name*	The name of the dataset to open.

Returns

The handle of the opened dataset is returned. If it is less than zero, the open failed.

Example

Open an image raster dataset:

```
hid_t image_id, raster_id;
bstring name = bfromcstr("r1");
raster_id = GS_DatasetOpen(image_id, name);
if(raster_id < 0) {
  printf("Failed to open dataset.\n");
}
bdestroy(name);
```

4.17 src/GS_DatasetRead.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- hid_t GS_DatasetRead (hid_t dataset_id, const int ∗offsets, const int ∗sizes, int datatype, void ∗buffer)

  GS_DatasetRead reads data from a dataset.

4.17.1 Function Documentation

4.17.1.1 hid_t GS_DatasetRead (hid_t *dataset_id,* const int ∗ *offsets,* const int ∗ *sizes,* int *datatype,* void ∗ *buffer*)

GS_DatasetRead reads data from a dataset.

GS_DatasetRead() reads data froma dataset

See also

G_DatasetWrite(), GS_DatasetOpen(), GS_DatasetCreate()

Parameters

`in`	*id*	A handle for the dataset. Perhaps from GS_DatasetOpen().
`in`	*offsets*	Must supply an offset to the first data element to read for each dimension. Zero offset is used for reading starting with the first data-element.
`in`	*sizes*	Must supply how many data elements to read for each dimension. Zero offset is used for reading starting with the first data-element. Using the offset and size for each dimension allows the specification of rectangular windows in n-dimensional space. For example, in 2-dimensions the window is: xoffset = offset[0], yoffset= offset[1] xsize = sizes[0], ysize = sizes[1]]
`in`	*datatype*	The datatype of the buffer. All the data that is read is converted to this datatype, and then returned to the caller in `buffer`. See the Details section for how this conversion is done. The valid datatypes are: `GS_DATATYPE_UI1` (Unsigned Integer, 1-bit) `GS_DATATYPE_UI8` (Unsigned Integer, 8-bit) `GS_DATATYPE_SI8` (Signed Integer, 8-bit) `GS_DATATYPE_CI8` (Complex Integer, 8-bit (not supported)) `GS_DATATYPE_UI16` (Unsigned Integer, 16-bit) `GS_DATATYPE_SI16` (Signed Integer, 16-bit) `GS_DATATYPE_CI16` (Complex Integer, 16-bit) `GS_DATATYPE_UI32` (Unsigned Integer, 32-bit) `GS_DATATYPE_SI32` (Signed Integer, 32-bit) `GS_DATATYPE_CI32` (Complex Integer, 32-bit) `GS_DATATYPE_CI64` (Complex Integer, 64-bit) `GS_DATATYPE_R32` (Real, 32-bit) `GS_DATATYPE_R64` (Real, 64-bit) `GS_DATATYPE_C64` (Complex, 64-bit) `GS_DATATYPE_C128` (Complex, 128-bit) `GS_DATATYPE_UI64` (Unsigned Integer, 64-bit) `GS_DATATYPE_SI64` (Signed Integer, 64-bit)

out	*buffer*	This is a pointer to enough memory to hold the data that is being read.

Returns

TRUE on sucess, FALSE on failure.

Example

We have a 2D dataset named "d4", Read in from data elements 0 to 500 in dimension 1, and from data elements 100 to 700 in dimension 2. The data is stored as integers, but we'd like to read them as doubles.

```
hid_t group_id;
hid_t dataset_id;
double buffer[500*600];
int offsets[10], sizes[10];
dataset_id = GS_DatasetOpen(group_id,"d4");
if(dataset_id < 0){
  printf("Failed to open dataset\n);
  return;
}
offsets[0]=0;
offsets[1]=100;
sizes[0]=500;
sizes[1]=600;
if(!GS_DatasetRead(dataset_id, offsets, sizes,
                   GS_DATATYPE_R64, (void *)buffer)) {
  printf("Failed to read from dataset.\n");
}
```

Details

YET: data type conversions.... YET: deal with interleaving spec.....

4.18 src/GS_DatasetRename.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_DatasetRename (hid_t group_id, const_bstring oldname, const_bstring newname)

  GS_DatasetRename renames a dataset in a GeoSci datafile.

4.18.1 Function Documentation

4.18.1.1 int GS_DatasetRename (hid_t *group_id*, const_bstring *oldname*, const_bstring *newname*)

GS_DatasetRename renames a dataset in a GeoSci datafile.

GS_DatasetRename() renames a dataset in a GeoSci datafile.

Parameters

in	*group_id*	A handle for the container of the dataset. This should be a group.
in	*oldname*	Current name of dataset to rename.
in	*newname*	Desired new name of the dataset.

Returns

TRUE on sucess, FALSE on failure.

Example

Rename a dataset from "r1" to "r4".

```
hid_t group_id;
bstring oldname = bfromcstr("r1");
bstring newname = bfromcstr("r2");
if(!GS_GS_DatasetRename(group_id, oldname, newname) ) {
  printf("Failed to rename the dataset.\n");
}
bdestroy(oldname);
bdestroy(newname);
```

4.19 src/GS_DatasetWrite.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- hid_t GS_DatasetWrite (hid_t dataset_id, const int ∗offsets, const int ∗sizes, int datatype, const void ∗buffer)

  GS_DatasetWrite writes data to a dataset.

4.19.1 Function Documentation

4.19.1.1 hid_t GS_DatasetWrite (hid_t *dataset_id*, const int ∗ *offsets*, const int ∗ *sizes*, int *datatype*, const void ∗ *buffer*)

GS_DatasetWrite writes data to a dataset.

GS_DatasetWrite() writes data to a dataset

See also

GS_DatasetRead(), GS_DatasetOpen(), GS_DatasetCreate()

Parameters

`in`	*id*	A handle for the dataset. Perhaps from GS_DatasetOpen().
`in`	*offsets*	Must supply an offset to the first data element to write for each dimension. Zero offset is used for writing starting with the first data-element.
`in`	*sizes*	Must supply how many data elements to write for each dimension. Zero offset is used for writing starting with the first data-element. Using the offset and size for each dimension allows the specification of rectangular windows in n-dimensional space. For example, in 2-dimensions the window is: xoffset = offset[0], yoffset= offset[1] xsize = sizes[0], ysize = sizes[1]]
`in`	*datatype*	The datatype of the buffer. All the data that is written is converted from this datatype, to the datatype used by the Dataset. See the Details section for how this conversion is done. The valid datatypes are: `GS_DATATYPE_UI1` (Unsigned Integer, 1-bit) `GS_DATATYPE_UI8` (Unsigned Integer, 8-bit) `GS_DATATYPE_SI8` (Signed Integer, 8-bit) `GS_DATATYPE_CI8` (Complex Integer, 8-bit (not supported)) `GS_DATATYPE_UI16` (Unsigned Integer, 16-bit) `GS_DATATYPE_SI16` (Signed Integer, 16-bit) `GS_DATATYPE_CI16` (Complex Integer, 16-bit) `GS_DATATYPE_UI32` (Unsigned Integer, 32-bit) `GS_DATATYPE_SI32` (Signed Integer, 32-bit) `GS_DATATYPE_CI32` (Complex Integer, 32-bit) `GS_DATATYPE_CI64` (Complex Integer, 64-bit) `GS_DATATYPE_R32` (Real, 32-bit) `GS_DATATYPE_R64` (Real, 64-bit) `GS_DATATYPE_C64` (Complex, 64-bit) `GS_DATATYPE_C128` (Complex, 128-bit) `GS_DATATYPE_UI64` (Unsigned Integer, 64-bit) `GS_DATATYPE_SI64` (Signed Integer, 64-bit)

out	*buffer*	This is a pointer to enough memory to hold the data that is being written.

Returns

TRUE on sucess, FALSE on failure.

Example

We have a 2D dataset named "d4", Write tp data elements 0 to 500 in dimension 1, and to data elements 100 to 700 in dimension 2. The data is stored in the Dataset as integers, and we are sending doubles, which therefore need to be converted.

```
hid_t group_id;
hid_t dataset_id;
double buffer[500*600];
int offsets[10], sizes[10];
dataset_id = GS_DatasetOpen(group_id,"d4");
if(dataset_id < 0){
  printf("Failed to open dataset\n);
  return;
}
offsets[0]=0;
offsets[1]=100;
sizes[0]=500;
sizes[1]=600;
if(!GS_DatasetWrite(dataset_id, offsets, sizes,
                   GS_DATATYPE_R64, (void *)buffer)) {
  printf("Failed to write to dataset.\n");
}
```

Details

YET: data type conversions.... YET: deal with interleaving spec.....

4.20 src/GS_DatatypeAsString.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- const_bstring GS_DatatypeAsString (int datatype)

  GS_DatatypeAsString return a string representing the datatype.

4.20.1 Function Documentation

4.20.1.1 const_bstring GS_DatatypeAsString (int *datatype*)

GS_DatatypeAsString return a string representing the datatype.

GS_DatatypeAsString() returns a string representing the GeoSci datatype.

See also

GS_DatatypeAsInteger(), GS_DatatypeNumbytes()

Parameters

in	*datatype*	An integer representing a GeoSciPy datatype. Valid datatypes: `GS_DATATYP←E_UI1` 1 A single bit `GS_DATATYPE_UI8` 2 Unsigned 8-bit integer `GS_DATATYPE_SI8` 3 Signed 8-bit integer `GS_DATATYPE_CI8` 4 Complex 8-bit integer `GS_DATATYPE_UI16` 5 Unsigned 16-bit integer `GS_DATATYPE_SI16` 6 Signed 16-bit integer `GS_DATATYPE_CI16` 7 Complex 16-bit integer `GS_DATATYPE_UI32` 8 Unsigned 32-bit integer `GS_DATATYPE_SI32` 9 Signed 32-bit integer `GS_DATATYPE_CI32` 10 Complex 32-bit integer `GS_DATATYPE_CI64` 11 Complex 64-bit integer `GS_DATATYPE_R32` 12 32-bit Real number `GS_DATATYPE_R64` 13 64-bit Real number `GS_DATATYPE_C64` 14 Complex 64-bit floating-point number `GS_DATATYPE_C128` 15 Complex 128-bit floating point `GS_DATATYPE_UI64` 16 Unsigned 64-bit integer `GS_DATATYPE_SI64` 17 Signed 64-bit integer

Returns

The valid bstring is returned always. It has a non-zero length on success, zero length on failure. It is a constant bstring, so need to bdestroy() it when done.

4.21 src/GS_DatatypeIsComplex.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- int GS_DatatypeIsComplex (int datatype)

  GS_DatatypeIsComplex return TRUE if a complex datatype.

4.21.1 Function Documentation

4.21.1.1 int GS_DatatypeIsComplex (int *datatype*)

GS_DatatypeIsComplex return TRUE if a complex datatype.

GS_DatatypeIsComplex() returns TRUE for complex datatypes, FALSE otherwise.

See also

GS_DatatypeAsString(), GS_DatatypeIsInteger(), GS_DatatypeAsInteger(), GS_DatatypeNumbytes()

Parameters

in	*datatype*	An integer representing a GeoSciPy datatype. Valid datatypes: `GS_DATATYPE_UI1` 1 A single bit `GS_DATATYPE_UI8` 2 Unsigned 8-bit integer `GS_DATATYPE_SI8` 3 Signed 8-bit integer `GS_DATATYPE_CI8` 4 Complex 8-bit integer `GS_DATATYPE_UI16` 5 Unsigned 16-bit integer `GS_DATATYPE_SI16` 6 Signed 16-bit integer `GS_DATATYPE_CI16` 7 Complex 16-bit integer `GS_DATATYPE_UI32` 8 Unsigned 32-bit integer `GS_DATATYPE_SI32` 9 Signed 32-bit integer `GS_DATATYPE_CI32` 10 Complex 32-bit integer `GS_DATATYPE_CI64` 11 Complex 64-bit integer `GS_DATATYPE_R32` 12 32-bit Real number `GS_DATATYPE_R64` 13 64-bit Real number `GS_DATATYPE_C64` 14 Complex 64-bit floating-point number `GS_DATATYPE_C128` 15 Complex 128-bit floating point `GS_DATATYPE_UI64` 16 Unsigned 64-bit integer `GS_DATATYPE_SI64` 17 Signed 64-bit integer

Returns

TRUE is returned for complex datatypes, FALSE for others, and ERROR otherwise.

Example

Determine if a datatype is complex:

```
int datatype;
if(GS_DatatypeIsComplex(datatype)==TRUE){
   printf("the datatype is complex.\n);
} else {
   printf("the datatype is not complex\n");
}
```

4.22 src/GS_FileClose.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_FileClose (hid_t file_id)

  GS_FileClose closes a GeoSci datafile.

4.22.1 Function Documentation

4.22.1.1 int GS_FileClose (hid_t *file_id*)

GS_FileClose closes a GeoSci datafile.

GS_FileClose() closes GEOSCIpy database files which were opened using GS_FileOpen().

See also

GS_FileOpen()

Parameters

in	*file_id*	File handle for GeoSci file to be closed.

Returns

TRUE is returned on success, while FALSE is returned on failure.

Example

This example opens and then closes a standard GEOSCIpy database file:

```
hid_t   file_id;
bstring filename = bfromcstr("testimage.hd5");
bstring access = bfromcstr("r+");
file_id = GS_FileOpen( filename, access );
bdestroy(filename);
bdestroy(access);
  ... use the file ...
GS_FileClose( file_id );
```

4.23 src/GS_FileCloseAllObjects.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_FileCloseAllObjects (hid_t id)
  GS_FileCloseAllObjects closes all open objects in a file.

4.23.1 Function Documentation

4.23.1.1 int GS_FileCloseAllObjects (hid_t *id*)

GS_FileCloseAllObjects closes all open objects in a file.

GS_FileCloseAllObjects() closes all open objects in a GeoScifile.

See also

GS_FileClose(), GS_FileOpen()

Parameters

in	*id*	The handle of the open file, or any object in the file.

Returns

TRUE is returned on success, FALSE otherwise.

Example

Close all objects in an already-open file:

```
hid_t file_id;
if(GS_FileCloseAllObjects( file_id )){
    printf("success.\n");
} else {
    printf("failure.\n");
}
```

4.24 src/GS_FileCommitDatatypes.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- int GS_FileCommitDatatypes (hid_t file_id)
  GS_FileCommitDatatypes commits the complex datatypes to a file.

4.24.1 Function Documentation

4.24.1.1 int GS_FileCommitDatatypes (hid_t *file_id*)

GS_FileCommitDatatypes commits the complex datatypes to a file.

GS_FileCommitDatatypes() commits the complex datatypes to a file.

See also

GS_FileCreate()

Parameters

in	*file_id*	The file ID

Returns

TRUE on success, FALSE on failure.

Example:

After creating a new file, commit the complex datatypes to it:

```
hid_t file_id;
if(!GS_FileCommitDatatypes(file_id)){
   printf("failed to commit the complex datatyeps to the file\n");
}
```

Details

HDF-5 requires user-created datatypes to be "committed" to file where these datatypes are used. Since all the complex datatypes used in GeoSci are "user-created", we need to make sure that they are always committed.

4.25 src/GS_FileCopy.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int copy (const_bstring oldname, const_bstring newname)
  GS_FileCopy copies a GeoSci datafile.
- int **GS_FileCopy** (bstring oldname, bstring newname)

4.25.1 Function Documentation

4.25.1.1 int copy (const_bstring *oldname,* const_bstring *newname*)

GS_FileCopy copies a GeoSci datafile.

GS_FileCopy() copies an existing GEOSCIpy database file to a new file.

See also

GS_FileCreateEmpty(), GS_FileRename()

Parameters

in	*oldname*	Name of existing file to copy.
in	*newname*	Name of new file to create and copy into.

Returns

`TRUE` is returned on success, `FALSE` on failure.

Example

Copy a file named "testimage.hd5" to "sirc_april.hd5".

```
bstring filename1 = bfromcstr("testimage.hd5");
bstring filename2 = bfromcstr(sirc_april.hdf5");
if(!GS_FileCopy( filename1, filename2 )){
    printf("GS_FileCopy failure.\n");
}
bdestroy(filename1);
bdestroy(filename2);
```

Details

GS_FileCopy() will return an error if the file is open, or if a file with the same destination name already exists.

4.26 src/GS_FileCreate.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1
- #define **USER_BLOCK_SIZE** 0
- #define **METADATA_CACHE_SIZE** (1 ∗ 1024∗ 1024)
- #define **PRINTIT** FALSE

Functions

- hid_t GS_FileCreate (const_bstring filename)

  GS_FileCreate creates an empty GeoSci datafile.

4.26.1 Function Documentation

4.26.1.1 hid_t GS_FileCreate (**const_bstring** *filename*)

GS_FileCreate creates an empty GeoSci datafile.

GS_FileCreate() creates a new, empty, GEOSCIpy database image file and opens it. The basic file metadata is created, but nothing else.

GS_FileCreate() assumes the file does NOT exist before creating it, and will return an error if a file with the same name already exists.

See also

GS_ImageGeorefIO(), GS_ImagePixelSize().

Parameters

`in`	*filename*	Name of database file to be created. If there is no ".hd5" extension, this routine will create a file with the ".hdf5" extension.

Returns

The routine returns a file handle on success, which is negative if creation fails.

Example

Create an empty file named "testimage.hd5".

```
hid_t       file_id;
bstring     database=bfromcstr("testimage.hd5");
file_id = GS_FileCreate( database );
    ... use the file ...
GS_FileClose( file_id );
```

Details

For every GeoSci file we create we also "commit" every complex datatype into it. That way, no matter what a user does, the datatypes are there for them to use. This simplifies the other codes, who now need not be concerned about dealing with this.

4.27 src/GS_FileCreateMetadata.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_FileCreateMetadata (hid_t file_id, const_bstring descriptor)
  GS_FileCreateMetadata() Creates metadata in a GeoSci datafile.

4.27.1 Function Documentation

4.27.1.1 int GS_FileCreateMetadata (hid_t *file_id*, const_bstring *descriptor*)

GS_FileCreateMetadata() Creates metadata in a GeoSci datafile.

GS_FileCreateMetadata() creates a new metadata group to store per-file metadata in, it then fills them in with default values, and the file descriptor string that is passed in.

See also

FileCreate()

Parameters

in	*file_id*	The file ID
in	*descriptor*	A character string describing the file contents.

Returns

TRUE on success, FALSE on failure.

Example:

Create a new file named "av1.h5" and set it's file metadata, in particular make the file descriptor string be "Aviris over Boston, MA".

```
hid_t file_id;
bstring filename = bfromcstr("av1.h5");
file_id = GS_FileCreateEmpty(filename);
bdestroy(filename);
if(file_id > 0) {
   bstring description = bfromcstr("Aviris over Boston, MA");
   if(GS_FileCreateMetadata(file_id,description){
      printf("success\n");
   }
   bdestroy(description);
}
```

Details:

The file metadata is stored in a group named "/_Header". The file metadata items are: `grouptype` (= "←
Metadata"), `filetype` (= "GEOSCI"), `software_version` (= "V0.0.1"), `creation_datetime`, `last_update_datetime`, `history`, and `descriptor`, the last one being set by a call to this function.
Note that until the file metadata is set up correctly the file is not recognized as a valid GeoSci file, and so most functions in this library will not operate on it successfully. Because of this, the file creation functions, such as GS_FileCreate and GS_FileCreateEmpty call this function as part of the file creation process.

4.28 src/GS_FileFlush.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_FileFlush (hid_t object_id)

  GS_FileFlush forces datafile to be updated.

4.28.1 Function Documentation

4.28.1.1 int GS_FileFlush (hid_t *object_id*)

GS_FileFlush forces datafile to be updated.

GS_FileFlush() forces all changes to a file to be flushed to the hard drive, forcing them to occur rather than waiting in a "buffered" state.

See also

GS_FileOpen(), GS_FileClose()

Parameters

in	*object_id*	ID of a file, an image, or any other valid object in a GeoSci datafile.

Returns

Returns TRUE on success, FALSE otherwise.

Example

After writing to an image, make sure all changes are written to disk.

```
hid_t image_id;
bstring image_name=bfromcstr("Image3");
image_id=GS_ImageOpen(file_id,image_name);
bdestroy(image_name);
    ... modify the image ....
GS_FileFlush(image_id);
```

Note that we could have used file_id as well.

4.29 src/GS_FileIsOpen.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_FileIsOpen (const_bstring filename)

  GS_FileIsOpen determines if a GeoSci datafile is already open.

4.29.1 Function Documentation

4.29.1.1 int GS_FileIsOpen (const_bstring *filename*)

GS_FileIsOpen determines if a GeoSci datafile is already open.

GS_FileIsOpen() checks to see if the current process has already opened a GeoSci file.

See also

GS_FileCreateEmpty(), GS_FileRename()

Parameters

in	*filename*	Name of existing file to query.

Returns

TRUE is returned if the file is open, FALSE otherwise.

Example

See if a file named "testimage.hd5" is open.

```
bstring filename = bfromcstr("testimage.hd5");
if(GS_FileIsOpen( filename)){
    printf("testimage.hd5 is already open.\n");
} else {
    printf("testimage.hd5 is NOT open.\n");
}
bdestroy(filename);
```

Details

GS_FileIsOpen() is not able to say if another process has opened the file.

4.30 src/GS_FileIsWriteable.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- int GS_FileIsWriteable (hid_t id)

    GS_FileIsWriteable returns if file is writeable or not.

4.30.1 Function Documentation

4.30.1.1 int GS_FileIsWriteable (hid_t *id*)

GS_FileIsWriteable returns if file is writeable or not.

GS_FileIsWriteable() returns TRUE if the file associated with the given handle is writeable, or FALSE otherwise.

Parameters

in	*id*	The handle of the file, or any object in the file.

Returns

Returns `TRUE` if the file associated with the given handle is writeable, or `FALSE` if not. Returns `ERROR` (-1) if cannot determine the status.

Example

```
int status;
         ...
status = GS_FileIsWriteable(file_id);
if(status < 0){
  printf("Could not determine read/write status of file.\n");
  return;
}
if(status==FALSE) return;
...here we can write to the file....
```

Details

A file is writeable if it was opened in read/write mode using GS_FileOpen.

4.31 src/GS_FileOpen.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- hid_t GS_FileOpen (const_bstring dbname, const_bstring access)

  GS_FileOpen opens a GeoSci datafile.

4.31.1 Function Documentation

4.31.1.1 hid_t GS_FileOpen (const_bstring *dbname,* const_bstring *access*)

GS_FileOpen opens a GeoSci datafile.

GS_FileOpen() opens an existing GeoSci datafile.

See also

GS_FileCreate(), GS_FileClose()

Parameters

in	*dbname*	The name of the GeoSci datafile to be opened.
in	*access*	Read/Write flag: • "r" : Open file for Read only • "r+": Open file for Read/Write

Returns

Returns the `hid_t` file handle for the file opened. If the GeoSci file does not exist, a negative file handle is returned, and the global `error_string` is set.

Example

Open a file named "testimage.hd5" for read/write. Note that the default extension ".hd5" is tried by GS_FileOpen() if the file "testimage" does not exist.

```
hid_t      file_id;
bstring filename= bfromcstr("testimage");
bstring access = bfromcstr("r+");
file_id = GS_FileOpen ( filename, access);
bdestroy(filename);
bdestroy(access);
  ... use the file ...
GS_FileClose (file_id);
```

4.32 src/GS_FileReport.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- herr_t GS_SummarizeGroup (hid_t g_id, const char ∗name, const H5L_info_t ∗info, void ∗op_data)

    GS_SummarizeGroup writes a quick summary of a group.
- int GS_FileReport (const_bstring filename, bstring report_string)

    GS_FileReport summarizes the contents of a GeoSci datafile.

4.32.1 Function Documentation

4.32.1.1 int GS_FileReport (const_bstring *filename*, bstring *report_string*)

GS_FileReport summarizes the contents of a GeoSci datafile.

GS_FileReport() generates a report of the contents of a GEOSCIpy database file.

Parameters

`in`	*filename*	Name of datafile to be queried.
`in`	*report_string*	The returned report, as a multi-line string. On entry, must be a valid bstring.

Returns

Returns `TRUE` on success, `FALSE` on failure.

Example

Obtain report on a file named "testimage.hd5" with 8 channels of 8-bit data and 2 channels of 32-bit real data.

```
bstring report_string=bfromcstr("");
bstring filename = bfromcstr("testimage.hd5");
if(GS_FileReport(filename,report_string)){
   printf("%s",report_string);
} else {
   printf("%s\n",error_string);
}
bdestroy(filename);
bdestroy(report_string);
```

4.32.1.2 herr_t GS_SummarizeGroup (hid_t *g_id,* const char ∗ *name,* const H5L_info_t ∗ *info,* void ∗ *op_data*)

GS_SummarizeGroup writes a quick summary of a group.

Given a name, which is part of a given group, writes a quick summary (if the name is a group) to the global g_report_↩ string.

Parameters

`in`	*g_id*	Group within which 'name' is defined.
`in`	*name*	Name of object in that group.
`in`	*info*	not used.
`in`	*op_data*	Must be a valid bstring that has been casted to a (void ∗).

Returns

Returns FALSE (0) on success, ERROR (-1) on failure.

Details

This is meant to be called from within H5Literate(), so that we can create a report for all objects in a group. Currently we do not report on the file header group, nor on datatypes, but only on other groups, and only if those groups have the metadata variables: grouptype, and descriptor.

4.33 src/GS_GetHDF5Type.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- hid_t GS_GetHDF5Type (int datatype)

  GetHDF5Type returns the HDF type for the given GeoSciPy Type.

4.33.1 Function Documentation

4.33.1.1 hid_t GS_GetHDF5Type (int *datatype*)

GetHDF5Type returns the HDF type for the given GeoSciPy Type.

GetHDF5Type returns the HDF type for the given GeoSciPy Type

Parameters

in	*datatype*	An integer representing a GeoSciPy datatype. Valid datatypes are: GS_DATATYPE_UI1 1 A single bit GS_DATATYPE_UI8 2 Unsigned 8-bit integer GS_DATATYPE_SI8 3 Signed 8-bit integer GS_DATATYPE_CI8 4 Complex 8-bit integer GS_DATATYPE_UI16 5 Unsigned 16-bit integer GS_DATATYPE_SI16 6 Signed 16-bit integer GS_DATATYPE_CI16 7 Complex 16-bit integer GS_DATATYPE_UI32 8 Unsigned 32-bit integer GS_DATATYPE_SI32 9 Signed 32-bit integer GS_DATATYPE_CI32 10 Complex 32-bit integer GS_DATATYPE_CI64 11 Complex 64-bit integer GS_DATATYPE_R32 12 32-bit Real number GS_DATATYPE_R64 13 64-bit Real number GS_DATATYPE_C64 14 Complex 64-bit floating-point number GS_DATATYPE_C128 15 Complex 128-bit floating point number GS_DATATYPE_SI64 16 Signed 64-bit integer

Returns

The type is returned, and is negative if invalid.

Example

Get the HDF5 equivalent of the raster type complex-integer-8-bit.

```
hid_t hdf_type;

hdf_type = GetHDF5Type(DATATYPE_CI8);
if(hdf_type<0) {
   printf("GetHDF5Type failure\n");
   exit(-1);
}
```

4.34 src/GS_GetStringAttribute.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <hdf5.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1
- #define **SUCCESS** 0

Functions

- int GS_GetStringAttribute (hid_t object_id, const_bstring name, bstring value)
  GS_GetStringAttribute reads a string attribute from an object and returns its value.

4.34.1 Function Documentation

4.34.1.1 int GS_GetStringAttribute (hid_t *object_id*, const_bstring *name*, bstring *value*)

GS_GetStringAttribute reads a string attribute from an object and returns its value.

GS_GetStringAttribute() returns the value of a string attribute with the given name, from the named object.

Parameters

in	*object_id*	The opened object where the attribute is stored.
in	*name*	The name of the attribute variable.
in	*value*	The returned value of the variable. On input, this must be a valid bstring.

Returns

TRUE on success, FALSE on failure

Example:

Read the group type attribute:

```
bstring group_type=bfromcstr("");
bstring grouptype =bfromcstr("grouptype"):
hid_t object_id;
if(!GS_GetStringAttribute(object_id,grouptype,group_type)){
   bassigncstr(group_type,"...unknown...");
}// endif
bdestroy(grouptype);
...use group_type...
bdestroy(group_type);
```

4.35 src/GS_GetValidFileID.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- hid_t GS_GetValidFileID (hid_t id)

  GS_GetValidFileID returns a valid FileID that contains the object.

4.35.1 Function Documentation

4.35.1.1 hid_t GS_GetValidFileID (hid_t *id*)

GS_GetValidFileID returns a valid FileID that contains the object.

GS_GetValidFileID returns a valid FileID that contains the object-id.

Parameters

in	*id*	An id for an object in a GeoSciPY file.

Returns

A valid FileID is returned or a value less than 0 on error.

Example

Assume one has opened a file and an image, passes the image_id into a function, and needs the file_id asociated with the image_id.

```
hid_t image_id, file_id;
file_id = GS_GetValidFileID(image_id);
if(file_id < 0) {
   printf("Could not get valid file_id\n");
}
```

Details

This function either returns an `id` that is the same as the passed-in `id`, but with an incremented reference-count, or it returns a different `id`, also with an incremented reference-count. The user should always GS_FileClose() this `id` when done with it. The associated file will only be closed when the reference count reaches zero.
A typical usage of this is to make sure a passed-in `id` is a `file_id`. In such a case, this will increment the reference count by 1, and the user should then GS_FileClose() it when done, which will NOT close the file that the passed-in `file_id` refers to.
In another scenario, where the code needs to return a newly-opened object in the passed-in `file_id`, one needs to be more careful. Do not use this routine. Instead call GS_ObjectIsFile(id) to make sure the `id` refers to a file, and return an error if it doesn't.

4.36 src/GS_GroupClose.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- int GS_GroupClose (hid_t id)

  GS_GroupClose closes a group in a GeoSci Datafile.

4.36.1 Function Documentation

4.36.1.1 int GS_GroupClose (hid_t *id*)

GS_GroupClose closes a group in a GeoSci Datafile.

Parameters

in	*id*	The handle for the already-open group.

Returns

TRUE is returned if the group is succesfully closed, FALSE otherwise.

Example

Close the file metadata group after opening it:

```
hid_t file_id, meta_id;
bstring headername = bfromcstr("/_Header");
meta_id = GroupOpen(file_id, headername);
bdestroy(headername);
if(meta_id < 0) {
  printf("Failed to open group.\n");
  return;
}
GroupClose(meta_id);
```

4.37 src/GS_GroupCloseAllObjects.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_GroupCloseAllObjects (hid_t id)

  GS_GroupCloseAllObjects closes all open objects in a group.

4.37.1 Function Documentation

4.37.1.1 int GS_GroupCloseAllObjects (hid_t *id*)

GS_GroupCloseAllObjects closes all open objects in a group.

GS_GroupCloseAllObjects() closes all open objects in a group in a GeoSci file.

See also

GS_GroupClose(), GS_FileOpen()

Parameters

in	*id*	The handle of the open group.

Returns

`TRUE` is returned on success, `FALSE` otherwise.

Example

Close all objects in an already-open group:

```
hid_t group_id;
if(GS_GroupCloseAllObjects( group_id )){
    printf("success.\n");
} else {
    printf("failure.\n");
}
```

4.38 src/GS_GroupCopy.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- hid_t GS_GroupCopy (hid_t source, hid_t destination_object_id, const_bstring destination_group_name)

  GS_GroupCopy copies a group to another location.

4.38.1 Function Documentation

4.38.1.1 hid_t GS_GroupCopy (hid_t *source,* hid_t *destination_object_id,* const_bstring *destination_group_name*)

GS_GroupCopy copies a group to another location.

GS_GroupCopy() copies a group and all it contains to another location. This can be a different group, or a different file.

See also

GS_GroupClose(), GS_GroupOpen()

Parameters

in	*source*	A handle for a group.
in	*destination_↵ object_id*	The handle of the destination group or file.
in	*destination_↵ group_name*	The name of the new dataset to create.

Returns

The handle of the new group is returned. If it is less than zero, the copy failed.

Example

Copy an image to a new file, call it "Boston day3". We have already opened the destination file.

```
hid_t source_image_id;
hid_t destination_file_id;
hid_t group_id;
bstring newname = bfromcstr("Boston day3");
group_id = GS_GroupCopy(source_image_id, destination_file_id, newname);
if(group_id < 0) {
  printf("Failed to copy group.\n");
}
bdestroy(newname);
```

4.39 src/GS_GroupCreate.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- hid_t GS_GroupCreate (hid_t file_id, const_bstring groupname)
  GS_GroupCreate creates a new group in a GeoSci Datafile.

4.39.1 Function Documentation

4.39.1.1 hid_t GS_GroupCreate (hid_t *file_id*, const_bstring *groupname*)

GS_GroupCreate creates a new group in a GeoSci Datafile.

GS_GroupCreate creates a new group in a GeoSci Datafile

GS_GroupCreate() creates a new group in an existing GeoSci datafile.

See also

FileOpen(), GroupDelete()

Parameters

in	*file_id*	A handle for an already-open GeoSci datafile.
in	*groupname*	The name of the group in an existing GeoSci datafile to be created. Use Unix/↩ Web filenaming conventions, giving the full pathname, starting with "/". All but the last component of this name must already exist.

Returns

A valid handle to the new group is returned on success. A negative value is returned on failure.

Example

Create a group named "/group3" in an already-open GeoSciPy file.

```
hid_t file_id, group_id;
bstring groupname = bfromcstr("/group3");
group_id = GS_GroupCreate(file_id,groupname);
bdestroy(groupname);
if(group_id < 0 ){
  printf("GS_GroupCreate failure\n");
  return;
}
.... use it here ....
GS_GroupClose(group_id);
```

4.40 src/GS_GroupDelete.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- int GS_GroupDelete (hid_t file_id, const_bstring group_name)

  GS_GroupDelete deletes a group in a GeoSci datafile.

4.40.1 Function Documentation

4.40.1.1 int GS_GroupDelete (hid_t *file_id,* const_bstring *group_name*)

GS_GroupDelete deletes a group in a GeoSci datafile.

GS_GroupDelete() delete a group and all objects it contains within a GeoSci datafile.

Parameters

in	*file_id*	A handle for the already-open GeoSciPy file.
in	*group_name*	The name of the group, starting with a '/' and giving the full path to the group.

Returns

`TRUE` is returned if the group and all objects it contains are successfully deleted, `FALSE` otherwise.

Example

Delete an image group named Image1 (should normally call GS_ImageDelete()):

```
hid_t file_id;
bstring imagename=bfromcstr("/Image1");
if(!GS_GroupDelete(file_id, imagename)){
   printf("Failed to delete group\n");
}
bdestroy(imagename);
```

4.41 src/GS_GroupOpen.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- hid_t GS_GroupOpen (hid_t source, const_bstring name)

  GS_GroupOpen opens an existing group in a GeoSci datafile.

4.41.1 Function Documentation

4.41.1.1 hid_t GS_GroupOpen (hid_t *source*, const_bstring *name*)

GS_GroupOpen opens an existing group in a GeoSci datafile.

GS_GroupOpen() opens an existing group in a GeoSci datafile.

Parameters

in	*source*	A handle for a file or other container that has a group in it.
in	*name*	The name of the group to open.

Returns

The handle of the opened group is returned. If less than zero, the open failed.

Example

Open the file metadata group:

```
hid_t file_id, meta_id;
meta_id = GS_GroupOpen(file_id, bssttic("_Header"));
if(meta_id < 0) {
  printf("Failed to open group.\n");
}
```

4.42 src/GS_GroupRename.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_GroupRename (hid_t id, const_bstring oldname, const_bstring newname)

  GS_GroupRename renames a group in a GeoSci datafile.

4.42.1 Function Documentation

4.42.1.1 int GS_GroupRename (hid_t *id*, const_bstring *oldname*, const_bstring *newname*)

GS_GroupRename renames a group in a GeoSci datafile.

GS_GroupRename() renames a group in a GeoSci datafile.

Parameters

in	*id*	A handle for the file, or other container of the group.
in	*oldname*	Current name of group to rename.
in	*newname*	Desired new name of the group.

Returns

TRUE on sucess, FALSE on failure.

Example

Rename an image from "/SIRC-1" to "/SIRC-1a".

```
hid_t file_id;
bstring oldname = bfromcstr("/SIRC-1");
bstring newname = bfromcstr("/SIRC-1a");
if(!GS_GS_GroupRename(file_id, oldname, newname) ) {
  printf("Failed to rename the group.\n");
}
bdestroy(oldname);
bdstroy(newname);
```

4.43 src/GS_GroupSetType.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_GroupSetType (hid_t group_id, int typecode)
  GS_GroupSetType sets the type of a group in a GeoSci datafile.

4.43.1 Function Documentation

4.43.1.1 int GS_GroupSetType (hid_t *group_id,* int *typecode*)

GS_GroupSetType sets the type of a group in a GeoSci datafile.

GS_GroupSetType() sets the type of a group in a GeoSci datafile.

See also

GS_GroupOpen()

Parameters

`in`	*group_id*	A handle for the group.
`in`	*typecode*	An integer type code specifying the type of dataset. One of: `GS_OBJECT_TYPE_METADATA_GROUP` 1 `GS_OBJECT_TYPE_IMAGE` 2 `GS_OBJECT_TYPE_VECTOR` 3 `GS_OBJECT_TYPE_VECTOR2D` 4 `GS_OBJECT_TYPE_VECTOR3D` 5 `GS_OBJECT_TYPE_TIN` 6 `GS_OBJECT_TYPE_MESH2D` 7 `GS_OBJECT_TYPE_MESH3D` 8 `GS_OBJECT_TYPE_RASTER` 9 `GS_OBJECT_TYPE_IFILE` 10 `GS_OBJECT_TYPE_METADATA_DATASET` 12 `GS_OBJECT_TYPE_METADATA_IFILE` 13

Returns

`TRUE` on sucess, `FALSE` on failure.

Example

Set the type of an image: hid_t image_id; if(!GS_GroupSetType(image_id, GS_OBJECT_TYPE_IMAGE)) { printf("Failed to set the type of the image.\n"); }

4.44 src/GS_HDFDatatypeClose.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_HDFDatatypeClose (hid_t id)

  GS_HDFDatatypeClose closes an HDF datatype.

4.44.1 Function Documentation

4.44.1.1 int GS_HDFDatatypeClose (hid_t *id*)

GS_HDFDatatypeClose closes an HDF datatype.

GS_HDFDatatypeClose closes an HDF datatype.

See also

GS_ConvertToHDFDatatype()

Parameters

in	*id*	The id of the datatype.

Returns

`TRUE` on success, ERROR (-1) on failure.

Example

After opening a datatype using GS_ConvertToHDFDatatype(), close it:

```
hid_t hdf_type;
hdf_type = GS_ConvertToHDFDatatype(GS_DATATYPE_R64);
if(hdf_type<0){
   printf("GS_ConvertToHDFDatatype failed\n");
   return;
}
if(!GS_HDFDatatypeClose(hdf_type)){
  printf("failed to close the datatype\n");
}
```

Details

This function is not needed for most types, as they are not really opened to begin with. However, the complex datatypes are opened, and so need to be closed.

4.45 src/GS_ObjectGetChildren.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- herr_t getcount_it (hid_t o_id, const char ∗name, const H5O_info_t ∗object_info, void ∗op_data)

  GS_ObjectGetChildren returns list of all children objects.
- herr_t **getnames_it** (hid_t o_id, const char ∗name, const H5O_info_t ∗object_info, void ∗op_data)
- int **GS_ObjectGetChildren** (hid_t object_id, int ∗nobjs, bstring ∗∗objnames, int ∗∗objtypes)

Variables

- int **gcount**
- int ∗ **hdftypes**

4.45.1 Function Documentation

4.45.1.1 herr_t getcount_it (hid_t *o_id,* const char ∗ *name,* const H5O_info_t ∗ *object_info,* void ∗ *op_data*)

GS_ObjectGetChildren returns list of all children objects.

GS_ObjectGetChildren() returns a list of all children objects of the given object in the GeoSci datafile.

Parameters

in	*object_id*	Handle of an open object.
out	*nobjs*	The number of objects (children).
out	*objnames*	The returned list of object names. bdestroy() each element, and HFree the list when done, but only if nobjs>0.
out	*objtypes*	The returned list of object types. HFree it when done, but only if nobjs>0. Valid values are: H5O_TYPE_GROUP H5O_TYPE_DATASET H5O_TYPE_NAME←D_DATATYPE

Returns

Returns `TRUE` on success, `FALSE` on failure.

Example

Obtain list of objects in a file named "testimage.hd5".

```
int objhdftypes;
int nobjs;
int i;

bstring filename = bfromcstr("testiamge.hd5");
bstring access = bfromcstr("r+");
file_id = GS_FileOpen(filename,access);
if(file_id<0){
   printf("GS_FileOpen failed\n");
   printf("%s\n",error_string);
   bdestroy(filename);
   bdestroy(access);
   exit(-1);
}//endif
bdestroy(filename);
bdestroy(access);

bstring *objnames;
if(!GS_ObjectGetChildren(file_id, &nobjs, &objnames,&objhdftypes)){
   printf("GS_ObjectGetChildren failed\n");
   printf("%s\n",error_string);
   bdestroy(objnames);
   exit(-1);
}

// print results:
for(i=0;i<nobjs;i++){
   printf("obj# %d: %s (%s)\n",i,bdata(objnames[i]),
           H5ObjectTypeAsString(objhdftypes[i]));
}

// clean up:

for(i=0;i<nobjs;i++){
   bdestroy(objnames[i]);
}
if(nobjs>0){
  GFree(objnames);
  GFree(objhdftypes);
}
```

4.46 src/GS_ObjectIsDataset.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_ObjectIsDataset (hid_t id, const_bstring name)

  GS_ObjectIsDataset determines if an object is a dataset.

4.46.1 Function Documentation

4.46.1.1 int GS_ObjectIsDataset (hid_t *id*, const_bstring *name*)

GS_ObjectIsDataset determines if an object is a dataset.

GS_ObjectIsDataset() determines if a name in a GEOSCIPy file is a dataset or not.

See also

GS_FileOpen(), GS_FileCreate()

Parameters

in	*id*	File handle or Image handle of selected GeoSci file.
in	*object_name;*	Name of object to query.

Returns

TRUE is returned if the object is a dataset, FALSE otherwise.

Example:

Query about a raster named "r1" in an image named "Channel_1", in a file named "test123.h5".

```
hid_t    file_id, image_id;
bstring filename = bfromcstr("test123.h5");
bstring access = bfromcstr("r+");
file_id = GS_FileOpen(filename,access);
if(file_id < 0) {
   printf("Could not open file.\n");
   bdestroy(filename);
   bdestroy(access);
   return;
}
bdestroy(filename);
bdestroy(access);
bstring channel=bfromcstr("Channel_1");
image_id = GS_ImageOpen(file_id,channel);
if(image_id < 0) {
   printf("Could not open image.\n");
   bdestroy(channel);
   return;
}
```

```
bdestroy(channel);
bstring dataset_name = bfromcstr("r1");
if(GS_ObjectIsDataset(image_id,dataset_name)){
   ... open the object as a dataset and do stuff ....
} else {
   printf("Object is not a dataset.\n");
} // endif
bdestroy(dataset_name);
```

4.47 src/GS_ObjectIsDatasetByID.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_ObjectIsDatasetByID (hid_t id)

  GS_ObjectIsDatasetByID determines if an object-id refers to a dataset.

4.47.1 Function Documentation

4.47.1.1 int GS_ObjectIsDatasetByID (hid_t *id*)

GS_ObjectIsDatasetByID determines if an object-id refers to a dataset.

GS_ObjectIsDatasetByID() determines if an id of an object in a GEOSCIPy file is a dataset or not.

See also

GS_FileOpen(), GS_FileCreate()

Parameters

in	*id*	object handle in selected GEOSCIPY file.

Returns

TRUE is returned if the object is a dataset, FALSE otherwise.

Example:

Query about a raster that was already opened:

```
hid_t    raster_id;
if(GS_ObjectIsDatasetByID(raster_id)){
   ... the object is a dataset: use it...
} else {
   printf("Object is not a dataset.\n");
} // endif
```

4.48 src/GS_ObjectIsFile.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_ObjectIsFile (hid_t id)
  GS_ObjectIsFile determines if an object-id refers to a file.

4.48.1 Function Documentation

4.48.1.1 int GS_ObjectIsFile (hid_t *id*)

GS_ObjectIsFile determines if an object-id refers to a file.

GS_ObjectIsFile() determines if a object-id refers to a file or not.

See also

GS_FileOpen(), GS_FileCreate()

Parameters

in	*id*	Object handle of selected GEOSCIPY file.

Returns

TRUE is returned if the object is a file, FALSE otherwise.

Example:

Query about an image that was already opened:

```
C       hid_t    image_id;
C       if(GS_GS_ObjectIsFile(image_id)){
C          ... the object is a file: use it...
C       } else {
C          printf("Object is not a file.\n");
C       } // endif
```

4.49 src/GS_ObjectIsGroup.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- int GS_ObjectIsGroup (hid_t id, const_bstring name)

  GS_ObjectIsGroup determines if a named object is a group.

4.49.1 Function Documentation

4.49.1.1 int GS_ObjectIsGroup (hid_t *id*, const_bstring *name*)

GS_ObjectIsGroup determines if a named object is a group.

GS_ObjectIsGroup() determines if a name in a GeoSci datafile is a group or not.

See also

GS_FileOpen(), GS_FileCreate()

Parameters

in	*id*	object handle of a file or another group in the selected GeoSci datafile.
in	*object_name*	Name of object to query. This name is relative to the given object.

Returns

TRUE is returned if the object is a group, FALSE otherwise.

Example:

Query about an image named "Channel_1" in a file named "test123.h5".

```
hid_t    file_id;
bstring filename = bfromcstr("test123.h5");
bstring access = bfromcstr("r+");
file_id = GS_FileOpen(filename,access);
if(file_id < 0) {
   printf("Could not open file.\n");
   bdestroy(filename);
   bdestroy(access);
   return;
}
bstring channel = bfromcstr("Channel_1");
if(GS_ObjectIsGroup(file_id,channel)){
   ... open the object as a group and use it ....
} else {
   printf("Object is not a group.\n");
} // endif
bdestroy(channel);
```

4.50 src/GS_ObjectIsGroupByID.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1
- #define **SUCCESS** 0

Functions

- int GS_ObjectIsGroupByID (hid_t id)

  GS_ObjectIsGroupByID determines if an object-d refers to a group.

4.50.1 Function Documentation

4.50.1.1 int GS_ObjectIsGroupByID (hid_t *id*)

GS_ObjectIsGroupByID determines if an object-d refers to a group.

GS_ObjectIsGroupByID determines if an object-d refers to a group.

See also

GS_FileOpen(), GS_FileCreate()

Parameters

in	*object_id*	object handle in a GEOSCIPY database.

Returns

TRUE is returned if the object is a group, FALSE otherwise.

Example:

Query about an image named "Channel_1" in a file named "test123.h5".

```
hid_t    file_id, image_id;
bstring filename = bfromcstr("test123.h5");
bstring access = bfromcstr("r+");
file_id = GS_FileOpen(filename,access);
if(file_id < 0) {
   printf("Could not open file.\n");
   bdestroy(filename);
   bdestroy(access);
   return;
}
bdestroy(filename);
bdestroy(access);
bstring channel = bfromcstr("Channel_1");
image_id = GS_ImageOpen(file_id,channel);
if(image_id < 0) {
   printf("Could not open image.\n");
   bdestroy(channel);
   return;
}
bdestroy(channel);
if(GS_ObjectIsGroupByID(image_id)){
   ... open the object as a group and use it ....
} else {
   printf("Object is not a group.\n");
} // endif
```

4.51 src/GS_ObjectIsIFile.c File Reference

```
#include "ifile.h"
#include "bstrlib.h"
```

Functions

- int GS_ObjectIsIFile (hid_t id, const_bstring name)

  GS_ObjectIsIFile determines if the object is an internal file.
- int **GS_ObjectIsIFileByID** (hid_t dataset_id)

4.51.1 Function Documentation

4.51.1.1 int GS_ObjectIsIFile (hid_t *id,* const_bstring *name*)

GS_ObjectIsIFile determines if the object is an internal file.

GS_ ObjectIsIFile() determines if the object is an internal file in a geoscipy database file.

Parameters

in	*id*	The handle for the already-open group.
in	*name*	The name of the ifile within the group.

Returns

`TRUE` is returned if the object is an IFile, `FALSE` otherwise.

Example

Open an internal file and query if it's valid IFile handle:

```
hid_t file_id;
hid_t group_id;

group_id = GS_GroupOpen(file_id,"/somename");
if(group_id < 0) {
   printf("GS_GroupOpen failed on /somename\n");
}
bstring internal_file_name=bfromcstr("internal_file_name");
if(!GS_ObjectIsIFile(group_id, internal_file_name )){
  printf("Object is not an internal file.\n");
}
bdestroy(internal_file_name);
```

GS_ObjectIsIFileByID determines if the objectID is an internal file

GS_ ObjectIsIFile() determines if the objectID is an internal file in a geoscipy database file.

Parameters

in	*id*	The handle of the open internal file.

Returns

TRUE is returned if the object is an IFile, FALSE otherwise.

Example

Open an internal file and query if it's valid IFile handle:

```
hid_t file_id;
hid_t internal_file_id;

internal_file_id = GS_IFileOpen(file_id,"/somename","w");
if(internal_file_id < 0) {
   printf("GS_IFileOpen failed on /somename\n");
}

if(!GS_ObjectIsIFileByID(internal_file_id)){
  printf("Object is not an internal file.\n");
}
```

4.52 src/GS_Pathname.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1

Functions

- bstring GS_PathnameNodir (const_bstring name)

    GS_PathnameStripDir strips the directory name from the pathname.
- bstring **GS_PathnameGetDir** (const_bstring name)
- bstring **GS_PathnameGetHDFDir** (const_bstring name)
- bstring **GS_PathnameJoin** (const_bstring front, const_bstring back)

4.52.1 Function Documentation

4.52.1.1 bstring GS_PathnameNodir (const_bstring *name*)

GS_PathnameStripDir strips the directory name from the pathname.

GS_PathnameStripDir() strips the directory name from the pathname.

See also

GS_PathnameGetDir(), GS_PathnameJoin()

Parameters

in	*name*	The pathname to process.

Returns

Returns a bstring of the stripped name, which has zero length if there is any kind of error. bdestroy() it when done.

Example

This example starts with a path of "/some/path/name/here/blah" and the returned value from PathnameStripDir is: "blah".

```
bstring longname=bfromcstr("/some/path/name/here/blah");
bstring the_name;
the_name = PathnameStripDir(longname);
if(bstrlen(the_name)==0){
  printf("Could not determine the name\n");
}
printf("The stripped name is: %s\n",bdata(the_name));
bdestroy(the_name);
bdestroy(long_name);
```

Details

For filenames that end with a "/" ('\' in windows), a zero-length string is returned. It's not null. It still needs to be bdestroy()'d when you are done with it.PathnameGetDir keeps the directory name of the pathname

PathnameGetDir() keeps the directory name of the pathname, stripping any trailing filename portion.

See also

GS_PathnameStripDir(), GS_PathnameJoin()

Parameters

in	*name*	The pathname to process.

Returns

Returns a bstring of the stripped name, which has zero length if there is any kind of error. bdestroy() it when done.

Example

This example starts with a path of "/some/path/name/here/blah" and the returned value from PathnameGetDir is: "/some/path/name/here".

```
bstring longname=bfromcstr("/some/path/name/here/blah");
bstring the_name;
the_name = PathnameGetDir(longname);
if(bstrlen(the_name)==0){
  printf("Could not determine the name\n");
}
printf("The directory name is: %s\n",bdata(the_name));
bdestroy(the_name);
bdestroy(long_name);
```

Details

For filenames that have no "/" characters ('\' in windows), a zero-length string is returned. It's not null. It still needs to be bdestroy()'d when you are done with it.PathnameGetHDFDir keeps the directory name of the pathname

PathnameGetHDFDir() keeps the directory name of the pathname, stripping any trailing filename portion. Uses the '/' character for separating directory names, as is done in the HDF5 standard.

See also

GS_PathnameStripDir(), GS_PathnameJoin()

Parameters

in	*name*	The pathname to process.

Returns

Returns a bstring of the stripped name, which has zero length if there is any kind of error. bdestroy() it when done.

Example

This example starts with a path of "/some/path/name/here/blah" and the returned value from PathnameGetHDFDir is: "/some/path/name/here".

```
bstring longname=bfromcstr("/some/path/name/here/blah");
bstring the_name;
the_name = PathnameGetHDFDir(longname);
if(bstrlen(the_name)==0){
  printf("Could not determine the name\n");
}
printf("The directory name is: %s\n",bdata(the_name));
bdestroy(the_name);
bdestroy(long_name);
```

Details

For filenames that have no "/" characters, a zero-length string is returned. It's not null. It still needs to be bdestroy()'d when you are done with it.GS_PathnameJoin joins 2 pieces of a pathname

PathnameJoin() joins 2 pieces of a pathname. It uses the correct joining character for the operating system: LINUX, MACOS: '/' Windows: '\'

See also

GS_PathnameGetDir(), GS_PathnameStripDir()

Parameters

in	*front*	The front part of the pathname. Should not end with a '/' or a '\'.
in	*back*	The back part of the pathname. Should not start or end end with a '/' or a '\'.

Returns

Returns a bstring of the joined names, which has zero length if there is any kind of error. bdestroy() it when done.

Example

This example starts with a path of "/some/path/name/here/blah" and a relative filename of "trash.dat". The returned value from PathnameJoin (on unix) is: "/some/path/name/here/blah/trash.dat".

```
bstring longname=bfromcstr("/some/path/name/here/blah");
bstring ending=bfromcstr("trash.dat");
bstring the_name;
the_name = PathnameJoin(longname,ending);
if(bstrlen(the_name)==0){
  printf("Could not join the names\n");
  return;
}
printf("The joined name is: %s\n",bdata(the_name));
bdestroy(the_name);
bdestroy(longname);
bdestroy(ending);
```

4.53 src/GS_SetCacheSize.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1
- #define **SUCCESS** 0

Functions

- herr_t GS_SetCacheSize (hid_t file_id, size_t cache_size)

  GS_SetCacheSize sets the metadata cache size for a GeoSci datafile.

4.53.1 Function Documentation

4.53.1.1 herr_t GS_SetCacheSize (hid_t *file_id,* size_t *cache_size*)

GS_SetCacheSize sets the metadata cache size for a GeoSci datafile.

GS_SetCacheSize() sets the metadata cache size for a GeoSci datafile. Used when creating a file.

Parameters

in	*file_id*	File handle for GeoSci datafile
in	*cache_size*	Size in bytes for the metadata cache. 1048576 (1 MByte) is a good default.

Returns

A non-negative integer is returned on success, while a negative integer is returned on failure.

Example

Set the cache size to be 1MByte:

```
hid_t file_id;
if(GS_SetCacheSize(file_id,1048576) < 0){
   printf("Error setting cache size.\n");
}
```

4.54 src/GS_SetStringAttribute.c File Reference

```
#include <string.h>
```

```
#include <stdlib.h>
#include <stdio.h>
#include <hdf5.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1
- #define **SUCCESS** 0
- #define DEBUG

  GS_SetStringAttribute sets the value of a string attribute for a given object.

Functions

- int **GS_SetStringAttribute** (hid_t object_id, const_bstring name, const_bstring value)

4.54.1 Macro Definition Documentation

4.54.1.1 #define DEBUG

GS_SetStringAttribute sets the value of a string attribute for a given object.

GS_SetStringAttribute() sets the value of a string attribute in the named object given the attribute name.

See also

GS_GetStringAttribute()

Parameters

in	*group_id*	The opened object where the attribute is stored.
in	*name*	The name of the attribute variable.
in	*value*	The value of the variable to set.

Returns

TRUE on success, FALSE on failure.

Example:

Set the group type attribute:

```
bstring group_type=bfromcstr("");
hid_t object_id;
bassigncstr(group_type,"unknown type");
if(!GS_SetStringAttribute(object_id,bfromcstr("grouptype"),group_type)){
   printf("Unable to set group-type string\n");
} else {
   printf("Group-type successfully set.\n");
}// endif
... use group_type ...
bdestroy(group_type);
```

4.55 src/GS_Time.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/time.h>
#include "bstrlib.h"
#include "gmalloc.h"
#include "H5ATTR.h"
#include "globals.h"
```

Functions

- void **GS_getdate** (bstring string)
- void **GS_GetMS** (bstring string)
- void GS_Time (bstring string, int format)

  GS_Time get the local time and date.

4.55.1 Function Documentation

4.55.1.1 void GS_Time (bstring *string*, int *format*)

GS_Time get the local time and date.

GS_Time() returns a formatted character TIME and/or DATE string obtained from the operating system.

The time string is based on a 24 hour clock (e.g., ten past 2 p.m. would appear as 14:10). The overall format is controlled by the format argument as any of 7 different time, date, or time and date formats. Use number 7 for compatibility with the rest of the GeoSci C library.

Parameters

out	*bstring*	string; String that will be updated with the current date/time. It must be a valid safe string when this routine is called.
in	*format;*	Integer selector for which format of date and/or time is desired. One of: • Integer: selected string • 0 : "HH:MM:SS " • 1 : "DD-MMM-YY " • 2 : "DD-MM-YY " • 3 : "MM-DD-YY " • 4 : "HH:MM DD-MMM-YY " • 5 : "HH:MM:SS:sss " • 6 : "SSSSSSSSS.sss " • 7 : "HH:MM:SS DD-MMM-YYYY"

Example

Get the curent data and time in GeoSci format:

```
bstring thedatetime=bfromcstr("");
GS_Time (thedatetime, 7);
...use thedatetime...
bdestroy(thedatetime);
```

Details

This code is derived from source provided by PCI, Inc and so may need to be rewritten to avoid lawsuits.

4.56 src/GS_ValidID.c File Reference

```
#include <hdf5.h>
```

Macros

- #define **TRUE** 1
- #define **FALSE** 0

Functions

- int GS_ValidID (hid_t id)

  GS_ValidID returns whether the given object-id is a valid identifier.

4.56.1 Function Documentation

4.56.1.1 int GS_ValidID (hid_t *id*)

GS_ValidID returns whether the given object-id is a valid identifier.

GS_ValidID() determines if the identifier is valid, this means that it is a handle to a resource that still works.

See also

GS_FileCreate()

Parameters

in	*id*	An identifier that may have been returned from one of the other routines in the library, possibly referring to a file, an image, or some other object of interest in the file.

Returns

TRUE is returned if the identifier is valid, FALSE otherwise.

Exmaple:

We have obtained an identifier for an image object, and we wish to see if it is still valid:

```
hid_t id;
if(GS_ValidID(id)){
  printf("The id is valid\n");
} else {
  printf("The id is invalid\n");
}
```

4.57 src/H5ATTR.c File Reference

```
#include "H5ATTR.h"
#include "globals.h"
```

Macros

- #define **ERROR** -1
- #define **SUCCESS** 0

Functions

- herr_t **H5ATTRset_attribute** (hid_t obj_id, const_bstring attr_name, hid_t type_id, size_t rank, hsize_t ∗dims, const_bstring attr_data)
- herr_t H5ATTRset_attribute_string (hid_t obj_id, const_bstring attr_name, const_bstring attr_data, hsize_t attr↩_size, int cset)

  H5ATTRset_attribute_string sets the value of a string attribute.
- herr_t **H5ATTRget_attribute** (hid_t obj_id, const_bstring attr_name, hid_t type_id, void ∗data)
- hsize_t H5ATTRget_attribute_string (hid_t obj_id, const_bstring attr_name, bstring data, int ∗cset)

  H5ATTRget_attribute_string gets the value of a string attribute.
- herr_t **H5ATTRfind_attribute** (hid_t loc_id, const_bstring attr_name)
- herr_t **H5ATTRget_type_ndims** (hid_t obj_id, const_bstring attr_name, hid_t ∗type_id, H5T_class_t ∗class_id, size_t ∗type_size, int ∗rank)
- herr_t **H5ATTRget_dims** (hid_t obj_id, const_bstring attr_name, hsize_t ∗dims)

4.57.1 Function Documentation

4.57.1.1 hsize_t H5ATTRget_attribute_string (hid_t *obj_id,* const_bstring *attr_name,* bstring *attr_data,* int ∗ *cset*)

H5ATTRget_attribute_string gets the value of a string attribute.

H5ATTRget_attribute_string gets the value of string attribute

See also

H5ATTRset_attribute_string()

Parameters

`in`	*obj_id*	The id of the object to which the attribute should be attached. This is usually a group.

in	*attr_name*	The name of the attribute to be set.
out	*attr_data*	The value for the attribute, as a safe string.
in	*cset*	The character set for the string. Valid values are: H5T_CSET_ASCII and H5T_↩ CSET_UTF8.

Returns

On success it returns the length of the string, on failure it returns ERROR (-1).

Example

Read the value of the `descriptor` metadata item.

```
hid_t object_id;
bstring desc_value=bfromcstr("");
bstring descriptor = bfromcstr(descriptor);
if( H5ATTRget_attribute_string( object_id, descriptor,desc_value,
                                H5T_CSET_ASCII) == ERROR){
   printf("ERROR getting the attribute value\n");
   bdestroy(descriptor);
}
bdestroy(descriptor);
...use the desc_value ....
bdestroy(desc_value);
```

Details

This function is normally called by GS_GetStringAttribute, which is the preferred function for getting values of string attributes.

4.57.1.2 herr_t H5ATTRset_attribute_string (hid_t *obj_id*, const_bstring *attr_name*, const_bstring *attr_data*, hsize_t *attr_size*, int *cset*)

H5ATTRset_attribute_string sets the value of a string attribute.

H5ATTRset_attribute_string sets the value of string attribute

See also

H5ATTRget_attribute_string()

Parameters

in	*obj_id*	The id of the object to which the attribute should be attached. This is usually a group.
in	*attr_name*	The name of the attribute to be set.
in	*attr_data*	The value for the attribute, as a safe string.
in	*attr_size*	The number of characters in `attr_data`.
in	*cset*	The character set for the string. Valid values are: H5T_CSET_ASCII and H5T_↩ CSET_UTF8.

Returns

On success it returns SUCCESS (0), on failure it returns ERROR (-1).

Example

Set the value of `descriptor` to "Arizona".

```
hid_t object_id;
bstring descriptor=bfromcstr("descriptor");
bstring arizona = bfromcstr("Arizona");
if( H5ATTRset_attribute_string( object_id, descriptor,
                                arizona,7,H5T_CSET_ASCII)
                                 == ERROR){
   printf("ERROR setting the attribute value\n");
   bdestroy(descriptor);
   bdestroy(arizona);
}
```

Details

This function is normally called by GS_SetStringAttribute, which is the preferred function for setting values of string attributes. }

4.58 src/IFileOpen.c File Reference

`#include "ifile.h"`

Functions

- IFILE ∗ **ifileopen_setup_read** (hid_t ifile_id, const char ∗ifilename, int access)
- IFILE ∗ **ifileopen_setup_write** (hid_t ifile_id, const char ∗ifilename, int access)
- IFILE ∗ **ifileopen_setup_append** (hid_t ifile_id, const char ∗ifilename, int access)
- IFILE ∗ **IFileCreate** (hid_t ifile_id, const char ∗ifilename)
- IFILE ∗ IFileOpen (hid_t file_id, const char ∗ifilename, const char ∗access)

    IFileOpen is used to open an internal file in a GEOSCI Database File}.

4.58.1 Function Documentation

4.58.1.1 IFILE∗ IFileOpen (hid_t *file_id,* const char ∗ *ifilename,* const char ∗ *access*)

IFileOpen is used to open an internal file in a GEOSCI Database File}.

Parameters

in	*file_id*	The handle of the open GeoSciPy file.
in	*ifilename*	The name of the internal file in an existing GEOSCI database to be opened. Use Unix filenaming conventions, giving the full pathname, starting with "/". All but the last component of this name must already exist.
in	*access*	Access mode for the IFile, similar to that used for files:

r

read: Open internal file for input operations. Must already exist.

w

write: Create an empty file for output operations. If an internal file with the same name already exists, its contents are discarded and the file is treated as a new empty internal file.

a

append: Open internal file for output at the end of a the file. Output operations always write data at the end of the file, expanding it. Repositioning operations (IFileSeek, IFileSetpos, IFileRewind) are silently ignored. The internal file is created if it does not exist.

r+

read/update: Open an internal file for update (both for input and output). The file must already exist.

w+

write/update: Create an empty internal file and open it for update (both for input and output). If an internal file with the same name already exists its contents are discarded and the internal file is treated as a new empty file.

a+

append/update: Open an internal file for update (both for input and output) with all output operations writing data at the end of the file. Repositioning operations (IFileSeek, IFileSetpos, IFileRewind) affect the next input operations, but output operations move the position back to the end of internal file. The internal file is created if it does not exist.

The letter "x" can be appended to any "w" specifier (to form "wx" or "w+x"). This subspecifier forces the function to fail if the internal file exists, instead of overwriting it.

Sometimes a "b" is appended to indicate binary mode for reading and writing, but that is not necessary, so it is ignored.

Returns

If the internal file is successfully opened, the function returns a pointer to a structure, otherwise a NULL pointer is returned.

Example 1

Let's assume that one already has an GeoSci file, and one wants to add an internal file, named "somename", off of the root:

```
hid_t file_id;
IFILE *ifilep;

ifilep = IFileOpen(file_id,"/somename","w");
if(!ifilep) {
   printf("IFileOpen failed on /somename\n");
}
```

Example 2

Let's assume that one already has an GeoSci file, and one wants to add an internal file inside another group:

```
hid_t file_id, group_id;
IFILE *ifilep;

group_id = GroupCreate(file_id,"/group1");
if(group_id < 0) {
   printf("GroupCreate failed on /group1\n");
   exit(-1);
}
ifilep = IFileOpen(file_id,"/group1/somename","w");
if(!ifilep) {
   printf("IFileOpen failed on /group1/somename\n");
   exit(-1);
}
```

Details

This function is meant to emulate as close as possible the standard C fopen() function.
For internal files open for update (those which include a "+" sign), on which both input and output operations are allowed, the file should be flushed (IFileFlush) or repositioned (IFileSeek, IFileSetpos, IFileRewind) between either a writing operation followed by a reading operation or a reading operation which did not reach the end-of-file followed by a writing operation.
When appropriate, this routine creates a single-dimensional unsigned-8-bit dataset with the given name. The components of the pathname above the filename must already exist, and be HDF5 groups in order for this to work.
Note that even if you intend to only read from an IFile, you need to open the GeoSciPy file for reading AND writing ("r+"). This is because the state of the internal file is written to metadata in the GeoSciPy file, even if one is only reading from the IFile.

Implementation

For developers, the details of the implementation are presented here.

The complete list of all 34 IFile-related functions is given below: GS_ObjectIsIFile.c (also contains ObjectIsIFileBy↩ID) GS_ObjectIsMetadataIFile.c DatasetGetType.c DatasetSetType.c IFileAllocate.c IFileClearError.c IFileClose.↩c IFileEOF.c IFileError.c IFileFlush.c IFileGetc.c IFileGets.c IFileGetWrite.c IFileOpen.c IFilePerror.c IFilePrintf.↩c IFilePutc.c IFilePuts.c IFileReadAccess.c IFileRead.c IFileReadStatus.c IFileRewind.c IFileScanf.c IFileSeek.c IFileSetEOF.c IFileSetWriteability.c IFileSetWrite.c IFileSize.c IFileTell.c IFileTruncate.c IFileWriteAccess.c IFile↩Write.c IFileWriteStatus.c IFileWriteStatusMessage.c

Some of these are named based on the standard C library functions that they emulate. Others are implementation-specific functions.

The overall idea of a IFile is that of a HDF5 dataset. This dataset is 1-dimensional, with a datatype of unsigned-8-bit. It is infinitely extendable. It has three string attributes:

writeable: "TRUE" or "FALSE". The user can set this so that don't accidentally over-write it or delete it. All IFiles are created with writeable="TRUE"

access: an integer betwen 0 and 6, converted to a string. Corresponds to one of the following #define'd parameters:

- IFILE_CLOSED 0
- IFILE_R 1
- IFILE_W 2
- IFILE_A 3
- IFILE_RP 4
- IFILE_WP 5
- IFILE_AP 6

These are named based on the access string used during the IFileOpen() function-call, the "P" meaning "+". This keeps track of how the file was opened so only appropriate operations are allowed. Note that a file that is open cannot be opened by another command: the file must be CLOSED in order to be opened. This implements a very simple locking mechanism

dataset_type: for an IFile this must be "1", which corresponds to the C #define GS_DATASET_TYPE_INTERNA↩L_FILE This is needed because image rasters are stored as datasets as well, and we need to distinguish between these.

The returned data structure is defined as:

```
typedef struct {
  hid_t ifile_id;
  long int file_position;
  int readPastEOF;
  int last_operation_status;
  char last_operation_status_message[512];
  int access;
  long int size;
} IFILE;
```

```
   The parameters in this structure are defined as follows:

   ifile_id:       the HDF5 object-id for this dataset.

   file_position: This is the next byte to read-from or write-to,
                  where file_position=0 means to read/write the first
                  byte in the file.

   readPastEOF:    "TRUE"=1 or "FALSE"=0. Whether the previous operation
                   tried to read past the last byte in the file.

   last_operation_status: "SUCCESS"=1 or "FAILURE"=0
                   Each operation on the IFile sets this.

   last_operation_status_message: a string written by each of the
                   IFile functions, whether they succeed or fail.
```

```
access:        A copy of the dataset-metadata entry.

size:          The current file size in bytes.
               This is obtained when opened using HDF5 function calls.
               Thereafter it is updated by the the functions as
               needed.


Functions that deal with these metadata items are given below.
Generally the user should never call any of these, except for:
   IFileSetWriteability() for setting the writeability of an IFile
                   when it is closed.
   IFileError()    for determining error status
   IFilePerror()   for printing the latest error
   IFileEOF()      for determining the EOF status of last read
   IFileSeek()     for moving the file position
   IFileTell()     for determining the file position
   IFileRewind()   for setting the file position to 0

GS_ObjectIsIFile -- must be a dataset with dataset_type="1"
GS_DatasetGetType-- returns the dataset_type
GS_DatasetSetType-- sets the dataset_type

IFileClearError  -- sets last_operation_status="SUCCESS" and
                    sets last_operation_status_message
IFileError       -- returns TRUE if last_operation_status="FAILURE"
IFilePerror      -- prints last_operation_status_message
IFileReadStatus  -- reads last_operation_status, and
                    last_operation_status_message
IFileWriteStatus -- writes last_operation_status, and
                    last_operation_status_message
IFileWriteStatusMessage -- writes last_operation_status_message

IFileEOF         -- returns TRUE if readPastEOF="TRUE"
IFileSetEOF      -- writes to readPastEOF

IFileGetWrite    -- returns TRUE If writeable="TRUE"
IFileSetWrite    -- sets writeability (developer function)
IFileSetWriteability  -- sets writeability (user function)

IFileReadAccess  -- reads access
IFileWriteAccess -- writes access

IFileSeek        -- sets file_position
IFileTell        -- reads file_position
IFileRewind      -- sets file_position to "0"



Functions to deal with the file size directly:

IFileTruncate    -- sets size of file (can shrink or grow)
IFileAllocate    -- expands file size if needed
IFileSize        -- gets the current file size from the struct
IFileSizeHDF     -- gets the current file size from HDF



Other functions that the user can use include:

IFileOpen        -- open an IFile
IFileClose       -- close an IFile
IFileFlush       -- flush any changes to the IFile to disk

IFileGetc        -- read a single character
IFileGets        -- read a newline-terminated string
IFileRead        -- read a set number of bytes
```

```
IFileScanf       -- formatted-read from next "line"

IFilePutc        -- write a single character
IFilePuts        -- write a newline-terminated string
IFileWrite       -- write a set number of bytes
IFilePrintf      -- write a formatted string



Note that there is only ONE COPY of the file state, so there can
only be one process that is using it at one time.
This is on purpose so that there are no parallel-read-write issues.
```