
HDF5 C++ Guide

Release 1.4.1
April, 2001

Hierarchical Data Format (HDF) Group
National Center for Supercomputing Applications (NCSA)
University of Illinois at Urbana-Champaign (UIUC)

Includes:

- *HDF5 C++ Interfaces*
 - *HDF5 C++ User's Notes*
-

Copyright Notice and Statement for NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities

NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities
Copyright 1998, 1999, 2000, 2001 by the Board of Trustees of the University of Illinois
All rights reserved.

Contributors: National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign (UIUC), Lawrence Livermore National Laboratory (LLNL), Sandia National Laboratories (SNL), Los Alamos National Laboratory (LANL), Jean-loup Gailly and Mark Adler (gzip library).

Redistribution and use in source and binary forms, with or without modification, are permitted for any purpose (including commercial purposes) provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the following disclaimer in the documentation and/or materials provided with the distribution.
3. In addition, redistributions of modified forms of the source or binary code must carry prominent notices stating that the original code was changed and the date of the change.
4. All publications or advertising materials mentioning features or use of this software are asked, but not required, to acknowledge that it was developed by the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign and to credit the contributors.
5. Neither the name of the University nor the names of the Contributors may be used to endorse or promote products derived from this software without specific prior written permission from the University or the Contributors.
6. **THIS SOFTWARE IS PROVIDED BY THE UNIVERSITY AND THE CONTRIBUTORS "AS IS" WITH NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED.** In no event shall the University or the Contributors be liable for any damages suffered by the users arising out of the use of this software, even if advised of the possibility of such damage.

Portions of HDF5 were developed with support from the University of California, Lawrence Livermore National Laboratory (UC LLNL). The following statement applies to those portions of the product and must be retained in any redistribution of source code, binaries, documentation, and/or accompanying materials:

This work was partially produced at the University of California, Lawrence Livermore National Laboratory (UC LLNL) under contract no. W-7405-ENG-48 (Contract 48) between the U.S. Department of Energy (DOE) and The Regents of the University of California (University) for the operation of UC LLNL.

DISCLAIMER: This work was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately- owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

Table of Contents

| | |
|---|----|
| <i>HDF5 C++ Interfaces</i> | 1 |
| <i>HDF5 C++ User's Notes</i> | 35 |
| Overview..... | 35 |
| Classes Description..... | 39 |
| H5Library..... | 39 |
| Exception..... | 39 |
| IdComponent..... | 40 |
| RefCounter..... | 41 |
| CommonFG..... | 41 |
| H5File..... | 42 |
| H5Object..... | 43 |
| Group..... | 43 |
| AbstractDs..... | 44 |
| DataSet..... | 44 |
| Attribute..... | 45 |
| DataType..... | 45 |
| EnumType..... | 47 |
| CompType..... | 47 |
| AtomType..... | 48 |
| PredType..... | 48 |
| IntType..... | 49 |
| FloatType..... | 49 |
| StrType..... | 50 |
| DataSpace..... | 50 |
| PropList..... | 51 |
| FileCreatPropList..... | 52 |
| FileAccPropList..... | 52 |
| DSetCreatPropList..... | 53 |
| DSetMemXferPropList..... | 54 |
| Examples..... | 55 |

HDF5 C++ Interfaces

```
// HDF5 dataset and attribute have some common characteristics, so the
// term abstract dataset is used to name the element that can represent
// both objects, dataset and attribute.
//
// Class AbstractDs is an abstract base class, from which Attribute and
// DataSet inherit. It provides the services that are common to both
// Attribute and DataSet. It also inherits from H5Object and passes down
// the services that H5Object provides.
class AbstractDs : public H5Object

    // Gets the dataspace of this abstract dataset - pure virtual
    virtual DataSpace getSpace() const = 0;

    // Gets the class of the datatype that is used by this abstract
    // dataset
    H5T_class_t getTypeClass() const;

    // Gets a copy of the datatype that this abstract dataset uses.
    // Note that this datatype is a generic one and can only be accessed
    // via generic member functions, i.e., member functions belong to
    // DataType. To get specific datatype, i.e. EnumType, FloatType,
    // etc..., use the specific functions, that follow, instead.
    DataType getDataType() const;

    // Gets a copy of the specific datatype of this abstract dataset
    EnumType getEnumType() const;
    CompType getCompType() const;
    IntType getIntType() const;
    FloatType getFloatType() const;
    StrType getStrType() const;

    // Copy constructor
    AbstractDs( const AbstractDs& original );

    virtual ~AbstractDs();

// end of class AbstractDs
```

```
// Atomic datatype can be an integer, float, string, or predefined datatype.
//
// Class AtomType is a base class, from which IntType, FloatType, StrType,
// and PredType inherit. It provides the services that are common to these
// subclasses. It also inherits from DataType and passes down the
// services that are common to all the datatypes.
class AtomType : public DataType

    // Sets the total size for an atomic datatype.
    void setSize( size_t size ) const;

    // Returns the byte order of an atomic datatype.
    H5T_order_t getOrder( string& order_string ) const;

    // Sets the byte ordering of an atomic datatype.
    void setOrder( H5T_order_t order ) const;

    // Returns the precision of an atomic datatype.
    size_t getPrecision() const;

    // Sets the precision of an atomic datatype.
    void setPrecision( size_t precision ) const;

    // Gets the bit offset of the first significant bit.
    int getOffset() const;

    // Sets the bit offset of the first significant bit.
    void setOffset( size_t offset ) const;

    // Copy constructor
    AtomType( const AtomType& original );

    virtual ~AtomType();

// end of class AtomType

// An attribute is an abstract dataset because it has some characteristics
// that a dataset also has, but not all.
//
// Class Attribute inherits from AbstractDs and provides accesses to an
// attribute.
```

```
class Attribute : public AbstractDs

    // Writes data to this attribute.
    void write(const DataType& mem_type, void *buf ) const;

    // Reads data from this attribute.
    void read( const DataType& mem_type, void *buf ) const;

    // Gets a copy of the dataspace for this attribute.
    virtual DataSpace getSpace() const;

    // Gets the name of this attribute.
    string getName( size_t buf_size ) const;

    // An attribute doesn't have the ability to iterate, simply because
    // it doesn't have any attributes associated with it. Thus, the
    // implementation of this member which is inherited from H5Object
    // is overwritten to do nothing here.
    int iterateAttrs() const;

        // Creates a copy of an existing attribute using the attribute id
        Attribute( const hid_t attr_id );

    // Copy constructor
    Attribute( const Attribute& original );

    virtual ~Attribute();

// CommonFG is a protocol class. Its existence is simply to provide the
// common services that are provided by H5File and Group. The file or
// group in the context of this class is referred to as 'location'.
class CommonFG
    // Creates a new group at this location.
    Group createGroup( const string& name, size_t size_hint = 0 ) const;
    Group createGroup( const char* name, size_t size_hint = 0 ) const;

    // Opens an existing group in a location.
    Group openGroup( const string& name ) const;
    Group openGroup( const char* name ) const;

    // Creates a new dataset at this location.
```

```
DataSet createDataSet( const string& name, const DataType& data_type, const
DataSpace& data_space, const DSetCreatPropList& create_plist =
DSetCreatPropList::DEFAULT ) const;
```

```
DataSet createDataSet( const char* name, const DataType& data_type, const
DataSpace& data_space, const DSetCreatPropList& create_plist =
DSetCreatPropList::DEFAULT ) const;
```

```
// Opens an existing dataset at this location.
```

```
DataSet openDataSet( const string& name ) const;
```

```
DataSet openDataSet( const char* name ) const;
```

```
// Creates a link of the specified type from new_name to current_name;
```

```
// both names are interpreted relative to this location.
```

```
void link( H5G_link_t link_type, const string& curr_name, const string& new_name )
const;
```

```
void link( H5G_link_t link_type, const char* curr_name, const char* new_name )
const;
```

```
// Removes the specified name at this location.
```

```
void unlink( const string& name ) const;
```

```
void unlink( const char* name ) const;
```

```
// Renames an HDF5 object at this location.
```

```
void move( const string& src, const string& dst ) const;
```

```
void move( const char* src, const char* dst ) const;
```

```
// Returns information about an HDF5 object, given by its name, at this location.
```

```
void getObjinfo( const string& name, hbool_t follow_link, H5G_stat_t& statbuf )
const;
```

```
void getObjinfo( const char* name, hbool_t follow_link, H5G_stat_t& statbuf )
const;
```

```
// Returns the name of the HDF5 object that the symbolic link points to.
```

```
string getLinkval( const string& name, size_t size ) const;
```

```
string getLinkval( const char* name, size_t size ) const;
```

```
// Sets the comment for the HDF5 object specified by its name.
```

```
void setComment( const string& name, const string& comment ) const;
```

```
void setComment( const char* name, const char* comment ) const;
```

```
// Retrieves comment for the HDF5 object specified by its name.
```

```
string getComment( const string& name, size_t bufsize ) const;
```

```
string getComment( const char* name, size_t bufsize ) const;
```

```
// Mounts the file 'child' onto this location.
```

```
void mount( const string& name, H5File& child, PropList& plist ) const;
void mount( const char* name, H5File& child, PropList& plist) const;

// Unmounts the file named 'name' from this location.
void unmount( const string& name ) const;
void unmount( const char* name ) const;

// Iterates over the elements of this location - not implemented in
// C++ style yet
int iterateElems( const string& name, int *idx, H5G_iterate_t op, void *op_data );
int iterateElems( const char* name, int *idx, H5G_iterate_t op, void *op_data );

// Opens a generic named datatype at this location
DataType openDataType( const string& name ) const;
DataType openDataType( const char* name ) const;

// Opens a named enumeration datatype at this location
EnumType openEnumType( const string& name ) const;
EnumType openEnumType( const char* name ) const;

// Opens a named compound datatype at this location
CompType openCompType( const string& name ) const;
CompType openCompType( const char* name ) const;

// Opens a named integer datatype at this location
IntType openIntType( const string& name ) const;
IntType openIntType( const char* name ) const;

// Opens a named floating-point datatype at this location
FloatType openFloatType( const string& name ) const;
FloatType openFloatType( const char* name ) const;

// Opens a named string datatype at this location
StrType openStrType( const string& name ) const;
StrType openStrType( const char* name ) const;

// For H5File and Group to throw appropriate exception - pure virtual
virtual void throwException() const = 0;

// Get id of the location, either group or file - pure virtual
virtual hid_t getLocId() const = 0;
```

```
CommonFG();
virtual ~CommonFG();

// end of CommonFG declaration

// Class CompType inherits from DataType and provides accesses to a compound
// datatype.
class CompType : public DataType

    // Creates a new compound datatype, given the type's size.
    CompType( size_t size );

    // Creates a compound datatype using an existing id.
    CompType( const hid_t existing_id );

    // Gets the compound datatype of the specified dataset.
    CompType( const DataSet& dataset );

    // Returns the number of members in this compound datatype.
    int getNmembers() const;

    // Returns the name of a member of this compound datatype.
    string getMemberName( int member_num ) const;

    // Returns the offset of a member of this compound datatype.
    size_t getMemberOffset( int memb_no ) const;

    // Returns the dimensionality of the specified member of this compound datatype.
    int getMemberDims( int member_num, size_t* dims, int* perm ) const;

    // Returns the type class of the specified member of this compound
    // datatype. It provides to the user a way of knowing what type
    // to create another datatype of the same class.
    H5T_class_t getMemberClass( int member_num ) const;

    // Returns the generic datatype of the specified member in
    // this compound datatype.
    DataType getMemberDataType( int member_num ) const;

    // Returns the enumeration datatype of the specified member in
    // this compound datatype.
```

```
EnumType getMemberEnumType( int member_num ) const;

// Returns the compound datatype of the specified member in
// this compound datatype.
CompType getMemberCompType( int member_num ) const;

// Returns the integer datatype of the specified member in
// this compound datatype.
IntType getMemberIntType( int member_num ) const;

// Returns the floating-point datatype of the specified member in
// this compound datatype.
FloatType getMemberFloatType( int member_num ) const;

// Returns the string datatype of the specified member in
// this compound datatype.
StrType getMemberStrType( int member_num ) const;

// Adds a new member to this compound datatype.
void insertMember( const string name, size_t offset, const DataType& new_member )
const;

// Recursively removes padding from within this compound datatype.
void pack() const;

// Default constructor
CompType();

// Copy constructor
CompType( const CompType& original );

virtual ~CompType();

// end of class CompType

// Class DataSet inherits from AbstractDs and provides accesses to a dataset.
class DataSet : public AbstractDs

// Gets the dataspace of this dataset.
virtual DataSpace getSpace() const;

// Gets the creation property list of this dataset.
```

```
DSetCreatPropList getCreatePlist() const;

// Gets the storage size of this dataset.
hsize_t getStorageSize() const;

// Reads the data of this dataset and stores it in the provided buffer.
// The memory and file dataspace and the transferring property list
// can be defaults.
void read( void* buf, const DataType& mem_type, const DataSpace& mem_space =
DataSpace::ALL, const DataSpace& file_space = DataSpace::ALL, const
DSetMemXferPropList& xfer_plist = DSetMemXferPropList::DEFAULT ) const;

// Writes the buffered data to this dataset.
// The memory and file dataspace and the transferring property list
// can be defaults.
void write( const void* buf, const DataType& mem_type, const DataSpace& mem_space
= DataSpace::ALL, const DataSpace& file_space = DataSpace::ALL, const
DSetMemXferPropList& xfer_plist = DSetMemXferPropList::DEFAULT ) const;

// Extends the dataset with unlimited dimension.
void extend( const hsize_t* size ) const;

// Default constructor
DataSet();

// Copy constructor
DataSet( const DataSet& original );

virtual ~DataSet();

// end of class DataSet

// Class DataSpace provides accesses to the dataspace.
class DataSpace : public IdComponent

// Default DataSpace objects
static const DataSpace ALL;

// Creates a dataspace object given the space type.
DataSpace( H5S_class_t type );

// Creates a simple dataspace.
DataSpace( int rank, const hsize_t * dims, const hsize_t * maxdims = NULL);
```

```
// Makes copy of an existing dataspace.
void copy( const DataSpace& like_space );

// Determines if this dataspace is a simple one.
bool isSimple () const;

// Sets the offset of this simple dataspace.
void offsetSimple ( const hssize_t* offset ) const;

// Retrieves dataspace dimension size and maximum size.
int getSimpleExtentDims ( hsize_t *dims, hsize_t *maxdims = NULL ) const;

// Gets the dimensionality of this dataspace.
int getSimpleExtentNdims () const;

// Gets the number of elements in this dataspace.
hssize_t getSimpleExtentNpoints () const;

// Gets the current class of this dataspace.
H5S_class_t getSimpleExtentType () const;

// Copies the extent of this dataspace.
void extentCopy ( DataSpace& dest_space ) const;

// Sets or resets the size of this dataspace.
void setExtentSimple( int rank, const hsize_t *current_size, const hsize_t
*maximum_size = NULL ) const;

// Removes the extent from this dataspace.
void setExtentNone () const;

// Gets the number of elements in this dataspace selection.
hssize_t getSelectNpoints () const;

// Get number of hyperslab blocks.
hssize_t getSelectHyperNblocks () const;

// Gets the list of hyperslab blocks currently selected.
void getSelectHyperBlocklist( hsize_t startblock, hsize_t numblocks, hsize_t *buf
) const;

// Gets the number of element points in the current selection.
```

```
hssize_t getSelectElemNpoints () const;

// Retrieves the list of element points currently selected.
void getSelectElemPointlist ( hsize_t startpoint, hsize_t numpoints, hsize_t *buf
) const;

// Gets the bounding box containing the current selection.
void getSelectBounds ( hsize_t* start, hsize_t* end ) const;

// Selects array elements to be included in the selection for
// this dataspace.
void selectElements ( H5S_seloper_t op, const size_t num_elements, const hssize_t*
coord[ ] ) const;

// Selects the entire dataspace.
void selectAll () const;

// Resets the selection region to include no elements.
void selectNone () const;

// Verifies that the selection is within the extent of the dataspace.
bool selectValid () const;

// Selects a hyperslab region to add to the current selected region.
void selectHyperslab( H5S_seloper_t op, const hsize_t *count, const hssize_t
*start, const hsize_t *stride = NULL, const hsize_t *block = NULL ) const;

// Default constructor
DataSpace();

// Create a dataspace object from a dataspace identifier
DataSpace( const hid_t space_id );

// Copy constructor
DataSpace( const DataSpace& original );

virtual ~DataSpace();
// end of class DataSpace

// HDF5 datatype can be an atom datatype, a compound datatype, or an
// enumeration datatype. A datatype is itself a kind of HDF5 object.
//
```

```
// Class DataType provides accesses to a generic HDF5 datatype. It has
// characteristics which AtomType, CompType, and EnumType inherit. It also
// inherits from H5Object and passes down the services to its subclasses.
class DataType : public H5Object

    // Creates a datatype given its class and size.
    DataType( const H5T_class_t type_class, size_t size );

    // Copies an existing datatype to this datatype instance.
    void copy( const DataType& like_type );

    // Returns the datatype class identifier of this datatype.
    H5T_class_t getClass() const;

    // Commits a transient datatype to a file; this datatype becomes
    // a named datatype which can be accessed from the location.
    void commit( H5Object& loc, const string& name ) const;
    void commit( H5Object& loc, const char* name ) const;

    // Determines whether this datatype is a named datatype or
    // a transient datatype.
    bool committed() const;

    // Finds a conversion function that can handle the conversion
    // of this datatype to the given datatype, dest.
    H5T_conv_t find( const DataType& dest, H5T_cdata_t **pcdata ) const;

    // Converts data from this datatype into the specified datatype, dest.
    void convert( const DataType& dest, size_t nelmts, void *buf, void *background,
PropList& plist ) const;

    // Sets the overflow handler to a specified function.
    void setOverflow(H5T_overflow_t func) const;

    // Returns a pointer to the current global overflow function.
    H5T_overflow_t getOverflow(void) const;

    // Locks a datatype.
    void lock() const;

    // Returns the size of this datatype.
    size_t getSize() const;
```

```
// Returns the base datatype from which a datatype is derived.
// Not implemented yet
DataType getSuper() const;

// Registers a conversion function.
void registerFunc(H5T_pers_t pers, const string& name, const DataType& dest,
H5T_conv_t func ) const;
void registerFunc(H5T_pers_t pers, const char* name, const DataType& dest,
H5T_conv_t func ) const;

// Removes a conversion function from all conversion paths.
void unregister( H5T_pers_t pers, const string& name, const DataType& dest,
H5T_conv_t func ) const;
void unregister( H5T_pers_t pers, const char* name, const DataType& dest,
H5T_conv_t func ) const;

// Tags an opaque datatype.
void setTag( const string& tag ) const;
void setTag( const char* tag ) const;

// Gets the tag associated with an opaque datatype.
string getTag() const;

// Creates a DataType using an existing id - this datatype is
// not a predefined type
DataType( const hid_t type_id, bool preftype = false );

// Default constructor
DataType();

// Copy constructor
DataType( const DataType& original );

virtual ~DataType();

// end of class DataType

// Class DSetCreatPropList provides accesses to a dataset creation
// property list.
class DSetCreatPropList : public PropList

// Default DSetCreatPropList object
static const DSetCreatPropList DEFAULT;
```

```
// Copies a dataset creation property list using assignment statement.
DSetCreatPropList& operator=( const DSetCreatPropList& rhs );

// Sets the type of storage used to store the raw data for the
// dataset that uses this property list.
void setLayout( hid_t plist, H5D_layout_t layout ) const;

// Gets the layout of the raw data storage of the data that uses this
// property list.
H5D_layout_t getLayout() const;

// Sets the size of the chunks used to store a chunked layout dataset.
void setChunk( int ndims, const hsize_t* dim ) const;

// Retrieves the size of the chunks used to store a chunked layout dataset.
int getChunk( int max_ndims, hsize_t* dim ) const;

// Sets compression method and compression level
void setDeflate( int level ) const;

// Sets a dataset fill value.
void setFillValue( DataType& fvalue_type, const void* value ) const;

// Retrieves a dataset fill value.
void getFillValue( DataType& fvalue_type, void* value ) const;

// Adds a filter to the filter pipeline
void setFilter( H5Z_filter_t filter, unsigned int flags, size_t cd_nelmts, const
unsigned int cd_values[] ) const;

// Returns the number of filters in the pipeline.
int getNfilters() const;

// Returns information about a filter in a pipeline.
H5Z_filter_t getFilter( int filter_number, unsigned int& flags, size_t& cd_nelmts,
unsigned int* cd_values, size_t namelen, char name[] ) const;

// Adds an external file to the list of external files.
void setExternal( const char* name, off_t offset, hsize_t size ) const;

// Returns the number of external files for a dataset.
int getExternalCount() const;
```

```
// Returns information about an external file
void getExternal( int idx, size_t name_size, char* name, off_t& offset, hsize_t&
size ) const;

// Creates a copy of an existing dataset creation property list
// using the property list id
DSetCreatPropList( const hid_t plist_id );

// Default constructor
DSetCreatPropList();

// Copy constructor
DSetCreatPropList( const DSetCreatPropList& original );

virtual ~DSetCreatPropList();

// end of class DSetCreatPropList

// Class DSetMemXferPropList provides accesses to a dataset memory and
// transfer property list.
class DSetMemXferPropList : public PropList

// Default object for dataset memory and transfer property list
static const DSetMemXferPropList DEFAULT;

// Copies a dataset memory and transfer property list using
// assignment statement
DSetMemXferPropList& operator=( const DSetMemXferPropList& rhs );

// Sets type conversion and background buffers
void setBuffer( size_t size, void* tconv, void* bkg ) const;

// Reads buffer settings
size_t getBuffer( void** tconv, void** bkg ) const;

// Sets the dataset transfer property list status to TRUE or FALSE
void setPreserve( bool status ) const;

// Checks status of the dataset transfer property list
bool getPreserve() const;
```

```
// Indicates whether to cache hyperslab blocks during I/O
void setHyperCache( bool cache, unsigned limit = 0 ) const;

// Returns information regarding the caching of hyperslab blocks during I/O
void getHyperCache( bool& cache, unsigned& limit ) const;

// Sets B-tree split ratios for a dataset transfer property list
void setBtreeRatios( double left, double middle, double right ) const;

// Gets B-tree split ratios for a dataset transfer property list
void getBtreeRatios( double& left, double& middle, double& right ) const;

// Sets the memory manager for variable-length datatype
// allocation in H5Dread and H5Dvlen_reclaim
void setVlenMemManager( H5MM_allocate_t alloc, void* alloc_info,
                       H5MM_free_t free, void* free_info ) const;

// alloc and free are set to NULL, indicating that system
// malloc and free are to be used
void setVlenMemManager() const;

// Gets the memory manager for variable-length datatype
// allocation in H5Dread and H5Tvlen_reclaim
void getVlenMemManager( H5MM_allocate_t& alloc, void** alloc_info,
                       H5MM_free_t& free, void** free_info ) const;

// Sets the transfer mode - parallel mode, not currently supported
//void setXfer( H5D_transfer_t data_xfer_mode = H5D_XFER_INDEPENDENT ) const;

// Gets the transfer mode - parallel mode, not currently supported
//H5D_transfer_t getXfer() const;

// Creates a copy of an existing dataset memory and transfer
// property list using the property list id
DSetMemXferPropList (const hid_t plist_id)

// Default constructor
DSetMemXferPropList();

// Copy constructor
DSetMemXferPropList( const DSetMemXferPropList& original );
```

```
// Default destructor
virtual ~DSetMemXferPropList();

// end of class DSetMemXferPropList

class EnumType : public DataType

// Creates an empty enumeration datatype based on a native signed
// integer type, whose size is given by size.
EnumType( size_t size );

// Gets the enum datatype of the specified dataset
EnumType( const DataSet& dataset ); // H5Dget_type

// Creates a new enum datatype based on an integer datatype
EnumType( const IntType& data_type ); // H5Tenum_create

// Inserts a new member to this enumeration type.
void insert( const string& name, void *value ) const;
void insert( const char* name, void *value ) const;

// Returns the symbol name corresponding to a specified member
// of this enumeration datatype.
string nameOf( void *value, size_t size ) const;

// Returns the value corresponding to a specified member of this
// enumeration datatype.
void valueOf( const string& name, void *value ) const;
void valueOf( const char* name, void *value ) const;

// Returns the value of an enumeration datatype member
void getMemberValue( int memb_no, void *value ) const;

// Default constructor
EnumType();

// Creates an enumeration datatype using an existing id
EnumType( const hid_t existing_id );

// Copy constructor
EnumType( const EnumType& original );
```

```
    virtual ~EnumType();
// end of class EnumType

class Exception

    // Creates an exception with a detailed message
    Exception( const string& message );

    Exception( const char* message);

    // Returns the character string that describes an error specified by
    // a major error number.
    string getMajorString( H5E_major_t major_num ) const;

    // Returns the character string that describes an error specified by
    // a minor error number.
    string getMinorString( H5E_minor_t minor_num ) const;

    // Returns the detailed message set at the time the exception is thrown
    string getDetailMesg() const;

    // Turns on the automatic error printing.
    void setAutoPrint( H5E_auto_t func,
                      void* client_data ) const;

    // Turns off the automatic error printing.
    static void dontPrint();

    // Retrieves the current settings for the automatic error stack
    // traversal function and its data.
    void getAutoPrint( H5E_auto_t& func,
                      void** client_data ) const;

    // Clears the error stack for the current thread.
    void clearErrorStack() const;

    // Walks the error stack for the current thread, calling the
    // specified function.
    void walkErrorStack( H5E_direction_t direction,
                        H5E_walk_t func, void* client_data ) const;
```

```
// Default error stack traversal callback function that prints
// error messages to the specified output stream.
void walkDefErrorStack( int n, H5E_error_t& err_desc,
                      void* client_data ) const;

// Prints the error stack in a default manner.
//void printError() const;
void printError( FILE* stream = NULL ) const;

// Creates an exception with no message
Exception();

// copy constructor
Exception( const Exception& original );

// end of class Exception

// Class FileIException inherits from Exception to provide exception
// handling for H5File.
class FileIException : public Exception
    FileIException();
    FileIException( string message );
// end of class FileIException

// Class GroupIException inherits from Exception to provide exception
// handling for Group.
class GroupIException : public Exception
    GroupIException();
    GroupIException( string message );
// end of class GroupIException

// Class DataSpaceIException inherits from Exception to provide exception
// handling for DataSpace.
class DataSpaceIException : public Exception
    DataSpaceIException();
    DataSpaceIException( string message );
// end of class DataSpaceIException
```

```
// Class DataTypeIException inherits from Exception to provide exception
// handling for DataType.
class DataTypeIException : public Exception
    DataTypeIException();
    DataTypeIException( string message );
// end of class DataTypeIException

// Class PropListIException inherits from Exception to provide exception
// handling for PropList.
class PropListIException : public Exception
    PropListIException();
    PropListIException( string message );
// end of class PropListIException

// Class DataSetIException inherits from Exception to provide exception
// handling for DataSet.
class DataSetIException : public Exception
    DataSetIException();
    DataSetIException( string message );
// end of class DataSetIException

// Class AttributeIException inherits from Exception to provide exception
// handling for Attribute.
class AttributeIException : public Exception
    AttributeIException();
    AttributeIException( string message );
// end of class AttributeIException

// Class LibraryIException inherits from Exception to provide exception
// handling for H5Library.
class LibraryIException : public Exception
    LibraryIException();
    LibraryIException( string message );
// end of class LibraryIException

// Class IdComponentException inherits from Exception to provide exception
```

```
// handling for IdComponent.
class IdComponentException : public Exception
    IdComponentException();
    IdComponentException( string message );
// end of class IdComponentException

// Class FileAccPropList provides accesses to a file access property list.
class FileAccPropList : public PropList

    // Default file access property list object
    static const FileAccPropList DEFAULT;

    // Copies a file access property list using assignment statement.
    FileAccPropList& operator=( const FileAccPropList& rhs );

    // Sets alignment properties of this file access property list.
    void setAlignment( hsize_t threshold = 1, hsize_t alignment = 1 ) const;

    // Retrieves the current settings for alignment properties from
    // this file access property list.
    void getAlignment( hsize_t& threshold, hsize_t& alignment ) const;

    // Sets the meta data cache and raw data chunk cache parameters.
    void setCache( int mdc_nelmts, int rdcc_nelmts, size_t rdcc_nbytes, double rdcc_w0
) const;

    // Retrieves maximum sizes of data caches and the preemption
    // policy value.
    void getCache( int& mdc_nelmts, int& rdcc_nelmts, size_t& rdcc_nbytes, double&
rdcc_w0 ) const;

    // Sets garbage collecting references flag.
    void setGcReferences( unsigned gc_ref = 0 ) const;

    // Returns garbage collecting references setting.
    unsigned getGcReferences() const;

    // Creates a copy of an existing file access property list
    // using the property list id.
    FileAccPropList (const hid_t plist_id);

    // Default constructor
```

```
FileAccPropList();

// Copy constructor
FileAccPropList( const FileAccPropList& original );

// Default destructor
virtual ~FileAccPropList();

// end of class FileAccPropList

// Class FileCreatPropList provides accesses to a file creation property list.
class FileCreatPropList : public PropList

// Default file creation property list object
static const FileCreatPropList DEFAULT;

// Copies a file creation property list using assignment statement.
FileCreatPropList& operator=( const FileCreatPropList& rhs );

// Retrieves version information for various parts of a file.
void getVersion( int& boot, int& freelist, int& stab, int& shhdr ) const;

// Sets the userblock size field of a file creation property list.
void setUserblock( hsize_t size ) const;

// Gets the size of a user block in this file creation property list.
hsize_t getUserblock() const;

// Sets file size-of addresses and sizes.
void setSizes( size_t sizeof_addr = 4, size_t sizeof_size = 4 ) const;

// Retrieves the size-of address and size quantities stored in a
// file according to this file creation property list.
void getSizes( size_t& sizeof_addr, size_t& sizeof_size ) const;

// Sets the size of parameters used to control the symbol table nodes.
void setSymk( int int_nodes_k, int leaf_nodes_k ) const;

// Retrieves the size of the symbol table B-tree 1/2 rank and the
// symbol table leaf node 1/2 size.
void getSymk( int& int_nodes_k, int& leaf_nodes_k ) const;
```

```
// Sets the size of parameter used to control the B-trees for
// indexing chunked datasets.
void setIstorek( int ik ) const;

// Returns the 1/2 rank of an indexed storage B-tree.
int getIstorek() const;

// Creates a copy of an existing file create property list
// using the property list id.
FileCreatPropList (const hid_t plist_id);

// Default constructor
FileCreatPropList();

// Copy constructor
FileCreatPropList( const FileCreatPropList& original );

// Default destructor
virtual ~FileCreatPropList();

// end of class FileCreatPropList

// Class H5File provides accesses to an HDF5 file. It uses the services
// provided by CommonFG beside inheriting the HDF5 id management from the
// IdComponent class.
class H5File : public IdComponent, public CommonFG

    // Creates or opens an HDF5 file. The file creation and access
    // property lists can be default.
    H5File( const string& name, unsigned int flags, const FileCreatPropList&
create_plist = FileCreatPropList::DEFAULT, const FileAccPropList& access_plist =
FileAccPropList::DEFAULT );

    H5File( const char* name, unsigned int flags, const FileCreatPropList&
create_plist = FileCreatPropList::DEFAULT, const FileAccPropList& access_plist =
FileAccPropList::DEFAULT );

    // Throw file exception - used by CommonFG to specifically throw
    // FileIException.
    virtual void throwException() const;

    // Determines if a file, specified by its name, is in HDF5 format.
    static bool isHdf5(const string& name );
    static bool isHdf5(const char* name );
```

```
// Reopens this file.
void reopen();

// Gets the creation property list of this file.
FileCreatPropList getCreatePlist() const;

// Gets the access property list of this file.
FileAccPropList getAccessPlist() const;

// Copy constructor
H5File(const H5File& original );

virtual ~H5File();

// end of class H5File

// Class FloatType inherits from AtomType and provides accesses to a
// floating-point datatype.
class FloatType : public AtomType

    // Creates a floating-point type using a predefined type.
    FloatType( const PredType& pred_type );

    // Gets the floating-point datatype of the specified dataset.
    FloatType( const DataSet& dataset );

    // Retrieves floating point datatype bit field information.
    void getFields( size_t& spos, size_t& epos, size_t& esize, size_t& mpos, size_t&
msize ) const;

    // Sets locations and sizes of floating point bit fields.
    void setFields( size_t spos, size_t epos, size_t esize, size_t mpos, size_t msize
) const;

    // Retrieves the exponent bias of a floating-point type.
    size_t getEbias() const;

    // Sets the exponent bias of a floating-point type.
    void setEbias( size_t ebias ) const;

    // Retrieves mantissa normalization of a floating-point datatype.
```

```
H5T_norm_t getNorm( string& norm_string ) const;

// Sets the mantissa normalization of a floating-point datatype.
void setNorm( H5T_norm_t norm ) const;

// Retrieves the internal padding type for unused bits in
// floating-point datatypes.
H5T_pad_t getInpad( string& pad_string ) const;

// Fills unused internal floating point bits.
void setInpad( H5T_pad_t inpad ) const;

// Default constructor
FloatType();

// Creates a floating-point datatype using an existing id.
FloatType( const hid_t existing_id );

// Copy constructor
FloatType( const FloatType& original );

virtual ~FloatType();

// end of class FloatType

// Class Group provides accesses to an HDF5 group. As H5File, it uses the
// services provided by CommonFG. This class also inherits from H5Object.
class Group : public H5Object, public CommonFG
    public:

        // Throw group exception - used by CommonFG to specifically throw
        // GroupIException.
        virtual void throwException() const;

// Default constructor
Group();

// Copy constructor
Group( const Group& original );

virtual ~Group();
```

```
// end of class Group

// Class IdComponent provides a mechanism to handle reference counting
// for an identifier of any HDF5 object.
class IdComponent
    // Sets the identifier of this object to a new value.
    void setId( hid_t new_id );

    // Creates an object to hold an HDF5 identifier.
    IdComponent( const hid_t h5_id );

    // Gets the value of the current HDF5 object id which is held
    // by this IdComponent object.
    hid_t getId () const;

    // Increment reference counter.
    void incRefCount();

    // Decrement reference counter.
    void decRefCount();

    // Get the reference counter to this identifier.
    int getCounter();

    // Decrements the reference counter then determines if there are
    // no more reference to this object.
    bool noReference();

    // Reset this object by deleting its reference counter of the old id.
    void reset();

    // Copy constructor
    IdComponent( const IdComponent& original );

    // Destructor
    virtual ~IdComponent();

}; // end class IdComponent

// Class IntType inherits from AtomType and provides accesses to
```

```
// integer datatypes.
class IntType : public AtomType

    // Creates a integer type using a predefined type.
    IntType( const PredType& pred_type );

    // Gets the integer datatype of the specified dataset.
    IntType( const DataSet& dataset );

    // Retrieves the sign type for an integer type.
    H5T_sign_t getSign() const;

    // Sets the sign property for an integer type.
    void setSign( H5T_sign_t sign ) const;

    // Default constructor
    IntType();

    // Creates a integer datatype using an existing id.
    IntType( const hid_t existing_id );

    // Copy constructor
    IntType( const IntType& original );

    virtual ~IntType();

// end of class IntType

// Class H5Library provides accesses to the HDF5 library. All of its
// member functions are static.
class H5Library

    // Initializes the HDF5 library.
    static void open();

    // Flushes all data to disk, closes files, and cleans up memory.
    static void close();

    // Instructs library not to install atexit cleanup routine
    static void dontAtExit();
```

```
// Returns the HDF library release number.
static void getLibVersion( unsigned& majnum, unsigned& minnum, unsigned& relnum );

// Verifies that the arguments match the version numbers compiled
// into the library
static void checkVersion( unsigned majnum, unsigned minnum, unsigned relnum );

// end of class H5Library

// An HDF5 object can be a group, dataset, attribute, or named datatype.
//
// Class H5Object provides the services that are typical to an HDF5 object
// so Group, DataSet, Attribute, and DataType can use them. It also
// inherits the HDF5 id management from the class IdComponent.
class H5Object : public IdComponent

    // Flushes all buffers associated with this HDF5 object to disk.
    void flush( H5F_scope_t scope ) const;

    // Creates an attribute for a group, dataset, or named datatype.
    // PropList is currently not used, it should always be default.
    Attribute createAttribute( const char* name, const DataType& type, const
    DataSpace& space, const PropList& create_plist = PropList::DEFAULT ) const;
    Attribute createAttribute( const string& name, const DataType& type, const
    DataSpace& space, const PropList& create_plist = PropList::DEFAULT ) const;

    // Opens an attribute that belongs to this object, given the
    // attribute name.
    Attribute openAttribute( const string& name ) const;
    Attribute openAttribute( const char* name ) const;

    // Opens an attribute that belongs to this object, given the
    // attribute index.
    Attribute openAttribute( const unsigned int idx ) const;

    // Iterate user's function over the attributes of this HDF5 object
    int iterateAttrs( attr_operator_t user_op, unsigned* idx = NULL, void* op_data =
    NULL );

    // Determines the number of attributes attached to this HDF5 object.
    int getNumAttrs() const;
```

```
// Removes an attribute from this HDF5 object, given the attribute
// name.
void removeAttr( const string& name ) const;
void removeAttr( const char* name ) const;

// Copy constructor
H5Object( const H5Object& original );

virtual ~H5Object();

// end of class H5Object

// Class PredType contains all the predefined datatype objects that are
// currently available.
class PredType : public AtomType

    static const PredType STD_I8BE;
    static const PredType STD_I8LE;
    static const PredType STD_I16BE;
    static const PredType STD_I16LE;
    static const PredType STD_I32BE;
    static const PredType STD_I32LE;
    static const PredType STD_I64BE;
    static const PredType STD_I64LE;
    static const PredType STD_U8BE;
    static const PredType STD_U8LE;
    static const PredType STD_U16BE;
    static const PredType STD_U16LE;
    static const PredType STD_U32BE;
    static const PredType STD_U32LE;
    static const PredType STD_U64BE;
    static const PredType STD_U64LE;
    static const PredType STD_B8BE;
    static const PredType STD_B8LE;
    static const PredType STD_B16BE;
    static const PredType STD_B16LE;
    static const PredType STD_B32BE;
    static const PredType STD_B32LE;
    static const PredType STD_B64BE;
    static const PredType STD_B64LE;
    static const PredType STD_REF_OBJ;
```

```
static const PredType STD_REF_DSETREG;
```

```
static const PredType C_S1;
```

```
static const PredType FORTRAN_S1;
```

```
static const PredType IEEE_F32BE;
```

```
static const PredType IEEE_F32LE;
```

```
static const PredType IEEE_F64BE;
```

```
static const PredType IEEE_F64LE;
```

```
static const PredType UNIX_D32BE;
```

```
static const PredType UNIX_D32LE;
```

```
static const PredType UNIX_D64BE;
```

```
static const PredType UNIX_D64LE;
```

```
static const PredType INTEL_I8;
```

```
static const PredType INTEL_I16;
```

```
static const PredType INTEL_I32;
```

```
static const PredType INTEL_I64;
```

```
static const PredType INTEL_U8;
```

```
static const PredType INTEL_U16;
```

```
static const PredType INTEL_U32;
```

```
static const PredType INTEL_U64;
```

```
static const PredType INTEL_B8;
```

```
static const PredType INTEL_B16;
```

```
static const PredType INTEL_B32;
```

```
static const PredType INTEL_B64;
```

```
static const PredType INTEL_F32;
```

```
static const PredType INTEL_F64;
```

```
static const PredType ALPHA_I8;
```

```
static const PredType ALPHA_I16;
```

```
static const PredType ALPHA_I32;
```

```
static const PredType ALPHA_I64;
```

```
static const PredType ALPHA_U8;
```

```
static const PredType ALPHA_U16;
```

```
static const PredType ALPHA_U32;
```

```
static const PredType ALPHA_U64;
```

```
static const PredType ALPHA_B8;
```

```
static const PredType ALPHA_B16;
```

```
static const PredType ALPHA_B32;
```

```
static const PredType ALPHA_B64;
```

```
static const PredType ALPHA_F32;
static const PredType ALPHA_F64;

static const PredType MIPS_I8;
static const PredType MIPS_I16;
static const PredType MIPS_I32;
static const PredType MIPS_I64;
static const PredType MIPS_U8;
static const PredType MIPS_U16;
static const PredType MIPS_U32;
static const PredType MIPS_U64;
static const PredType MIPS_B8;
static const PredType MIPS_B16;
static const PredType MIPS_B32;
static const PredType MIPS_B64;
static const PredType MIPS_F32;
static const PredType MIPS_F64;

static const PredType NATIVE_CHAR;
static const PredType NATIVE_SCHAR;
static const PredType NATIVE_UCHAR;
static const PredType NATIVE_SHORT;
static const PredType NATIVE_USHORT;
static const PredType NATIVE_INT;
static const PredType NATIVE_UINT;
static const PredType NATIVE_LONG;
static const PredType NATIVE_ULONG;
static const PredType NATIVE_LLONG;
static const PredType NATIVE_ULLONG;
static const PredType NATIVE_FLOAT;
static const PredType NATIVE_DOUBLE;
static const PredType NATIVE_LDOUBLE;
static const PredType NATIVE_B8;
static const PredType NATIVE_B16;
static const PredType NATIVE_B32;
static const PredType NATIVE_B64;
static const PredType NATIVE_OPAQUE;
static const PredType NATIVE_HSIZE;
static const PredType NATIVE_HSSIZE;
static const PredType NATIVE_HERR;
static const PredType NATIVE_HBOOL;
```

```
static const PredType NATIVE_INT8;
static const PredType NATIVE_UINT8;
static const PredType NATIVE_INT_LEAST8;
static const PredType NATIVE_UINT_LEAST8;
static const PredType NATIVE_INT_FAST8;
static const PredType NATIVE_UINT_FAST8;

static const PredType NATIVE_INT16;
static const PredType NATIVE_UINT16;
static const PredType NATIVE_INT_LEAST16;
static const PredType NATIVE_UINT_LEAST16;
static const PredType NATIVE_INT_FAST16;
static const PredType NATIVE_UINT_FAST16;

static const PredType NATIVE_INT32;
static const PredType NATIVE_UINT32;
static const PredType NATIVE_INT_LEAST32;
static const PredType NATIVE_UINT_LEAST32;
static const PredType NATIVE_INT_FAST32;
static const PredType NATIVE_UINT_FAST32;

static const PredType NATIVE_INT64;
static const PredType NATIVE_UINT64;
static const PredType NATIVE_INT_LEAST64;
static const PredType NATIVE_UINT_LEAST64;
static const PredType NATIVE_INT_FAST64;
static const PredType NATIVE_UINT_FAST64;

// Copy constructor
PredType( const PredType& original );

// Default destructor
virtual ~PredType();

protected:
// Default constructor
PredType();

// Creates a pre-defined type using an HDF5 pre-defined constant
PredType( const hid_t predtype_id ); // used by the library only

// end of class PredType
```

```
// An HDF5 property list can be a file creation property list, a file
// access property list, a dataset creation property list, or a dataset
// memory and transfer property list.
//
// Class PropList provides accesses to an HDF5 property list. Its
// services are inherited by classes FileCreatPropList, FileAccPropList,
// DSetCreatPropList, and DSetMemXferPropList. It also inherits the HDF5
// id management from the class IdComponent.
class PropList : public IdComponent

    // Default property list object
    static const PropList DEFAULT;

    // Creates a property list given the property list type.
    PropList( H5P_class_t type );

    // Makes a copy of the given property list.
    void copy( const PropList& like_plist );

    // Gets the class of this property list, i.e. H5P_FILE_CREATE,
    // H5P_FILE_ACCESS, ...
    H5P_class_t getClass() const;

    // Default constructor
    PropList();

    // Copy constructor
    PropList( const PropList& original );

    // Creates a default property list or creates a copy of an
    // existing property list giving the property list id
    PropList( const hid_t plist_id );

    virtual ~PropList();

// end of class PropList

// Class RefCounter provides a reference counting mechanism. It is used
// mainly by IdComponent to keep track of the references to an HDF5 object
// identifier.
```

```
class RefCounter

    // Returns the value of the counter.
    int getCounter () const;

    // Increments and decrements the counter.
    void increment();
    void decrement();

    // This bool function is used to determine whether to close an
    // HDF5 object when there are no more reference to that object.
    // It decrements the counter, then returns true if there are no
    // other object references the associated identifier. When the
    // function returns true, the associated identifier can be closed
    // safely.
    bool noReference();

    // Default constructor
    RefCounter();

    ~RefCounter();

// end of class RefCounter

// Class StrType inherits from AtomType and provides accesses to a
// string datatype.
class StrType : public AtomType
    public:
    // Creates a string type using a predefined type.
    StrType( const PredType& pred_type );

        // Gets the string datatype of the specified dataset.
    StrType( const DataSet& dataset );

    // Returns the character set type of this string datatype.
    H5T_cset_t getCset() const;

    // Sets character set to be used.
    void setCset( H5T_cset_t cset ) const;

    // Returns the string padding method for this string datatype.
```

```
H5T_str_t getStrpad() const;

// Defines the storage mechanism for character strings.
void setStrpad( H5T_str_t strpad ) const;

// Default constructor
StrType();

// Copy constructor
StrType( const StrType& original );

// Creates a string datatype using an existing id.
StrType( const hid_t existing_id );

virtual ~StrType();
// end of class StrType

// This template function, resetIdComponent, is used to reset an
// IdComponent object, which includes closing the associated HDF5
// identifier if it has no other references.
// 'Type' can be of the following classes: Attribute, DataSet, DataSpace,
// DataType, H5File, Group, and PropList.
template
void resetIdComponent(
    Type* obj )// pointer to object to be reset
```

HDF5 C++ User's Notes

This User's Note provides an overview of the structure, the availability, and the limitations of the C++ API of HDF5. It lists the classes and member functions included in the API and provides some examples of their applications. The C++ API is itself under development and does not have a complete User's Guide or Reference Manual. In addition, it is assumed that the reader has knowledge of the HDF5 file format and its components. For a complete User's Guide and Reference Manual of HDF5, please refer to the HDF home page at <http://hdf.ncsa.uiuc.edu>. At this time, to effectively utilize the C++ API, please refer to the C++ Interface, off of the Reference Manual of HDF5 page.

The User's Note includes an Overview section that gives the overall structure of the API. Following the Overview section is the Class Description section that briefly describes the classes and the functions they provide. This section also lists the limitations of the current version and describes plans for improvement/completion of some of the classes/functions. The final section, Examples, describes the examples that are provided with the source code distribution.

Overview

The HDF5 C++ API consists of the classes listed in the table below. All classes are included in a namespace called H5.

| Class | Description |
|-------------|--|
| H5Library | provides general-purpose library functions |
| IdComponent | is a base class that manage HDF5 object identifier |
| RefCounter | provides reference counting mechanism |
| CommonFG | is a base class for commonalities of H5File and Group |
| H5File | provides functions that access an HDF5 file |
| H5Object | base class for commonalties of all HDF5 objects which include groups, datasets, datatypes, and attributes |
| Group | is an H5Object; provides functions that access HDF5 groups |
| AbstractDs | base class for commonalities of DataSet and Attribute |
| DataSet | provides functions that access a dataset |
| Attribute | provides functions that access an attribute |
| DataType | is an H5Object; provides functions that access a general datatype, which can be an enumeration datatype, compound datatype, or atomic datatype |
| EnumType | is a DataType; provides functions that access an enumeration datatype |
| CompType | is a DataType; provides functions that access a compound datatype |
| AtomType | is a DataType; base class for commonalties of HDF5 predefined provides functions that access an enumeration datatype |

| | |
|---------------------|--|
| PredType | is an AtomType; provides the constant PredType objects for all the predefined datatype provided by the HDF5 library |
| IntType | is an AtomType; provides functions that access an integer datatype |
| FloatType | is an AtomType; provides functions that access an floating-point datatype |
| StrType | is an AtomType; provides functions that access an string datatype |
| DataSpace | provides functions that access the HDF5 dataspace |
| PropList | provides common accesses to the property lists |
| DSetCreatPropList | is a PropList; provides accesses to a dataset creation property list |
| DSetMemXferPropList | is a PropList; provides accesses to a dataset memory and transfer property list |
| FileAccPropList | is a PropList; provides accesses to a file access property list |
| FileCreatPropList | is a PropList; provides accesses to a file creation property list |
| Exception | provides the mechanism for handling errors returned by the C HDF5 library; it has several subclasses for specific exceptions |

Figure 1 shows the hierarchical relationship between the classes.

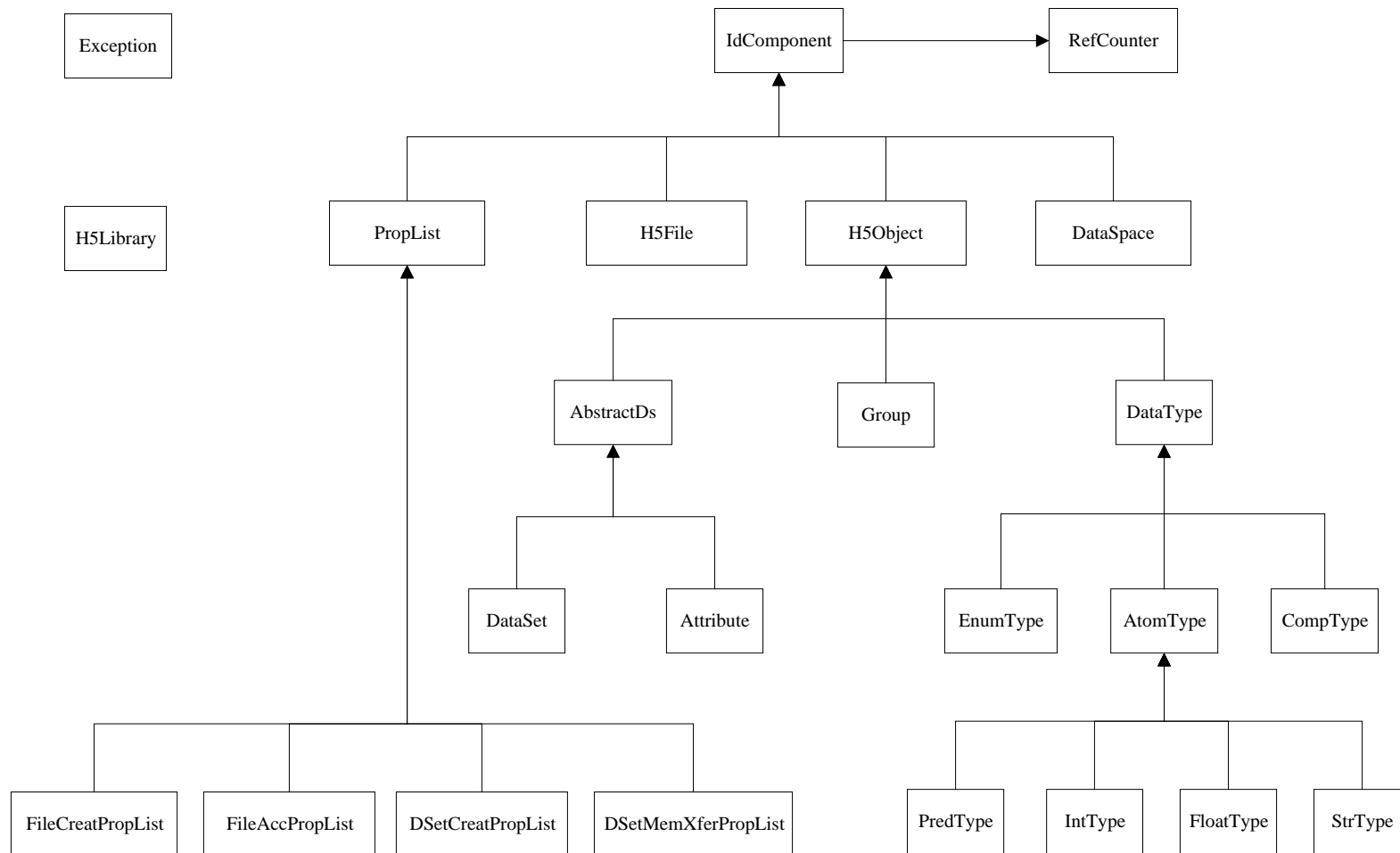


Figure 1. Class hierarchy of HDF5 C++ API

The figure shows that all the classes in the API, except H5Library and Exception, inherit from the class IdComponent. The elements that are represented by these classes are identified by an identifier that is defined and manipulated by the HDF5 library. IdComponent relieves the C++ API users from the concern about properly managing the identifiers of any HDF5 objects. The figure also illustrates the inheritance relationship between the subclasses of IdComponent.

H5Library is a stand-alone class to provide accesses to the library as a whole. Although Exception is also showed as having no inheritance relationship, in fact, it has many subclasses. These subclasses are for the specific exceptions and have no additional members. Thus, they are listed in the section about Exception class but not shown in the diagram. Another aspect that is not shown in the figure is that the classes H5File and Group also inherit from another base class, CommonFG, because of their commonality that does not exist in other subclasses of IdComponent.

The classes of the API and their members are described in the subsequent sections.

Classes Description

This section briefly describes the classes that form the C++ API of HDF5. Each subsection below gives a brief description of a class and provides a table that lists the public services that the class provides. Where necessary, the subsection also indicates the limitations of the current implementation of the described class and/or plans for its improvement or completion.

H5Library

This class provides some services that are used to access the HDF5 library. It is independent with other classes in the API. Its member functions are static so there is no need for an instance of this class to exist to use them.

| Member Function | Purpose |
|-----------------|--|
| open | Initializes the HDF5 library |
| close | Flushes all data to disk and clean up resources |
| dontAtExit | Instructs HDF5 C library not to install atexit clean up routine; this is helpful when having global objects in the application because the clean up routine might be executed before the global destructors and prematurely close any needed HDF5 components |
| getLibVersion | Retrieves the HDF library release numbers |
| checkVersion | Verifies that the arguments match the version numbers compiled into the library |

Exception

This class provides services to support user exception handling. All HDF5 C++ API calls that throw exceptions provide an instance of a class derived from Exception as a parameter. Thus the user can extract runtime information from it through the use of a corresponding catch procedure. Currently, Exception is used to derive the subclasses that support specific types of HDF5 errors that are generated by the C APIs, including H5F, H5G, H5S, H5T, H5P, H5D, and H5A. All of the functionality provided by these subclasses is inherited from Exception. These subclasses are listed below:

- FileIException for errors generated by the C API H5F
- GroupIException for errors generated by the C API H5G
- DataSpaceIException for errors generated by the C API H5S
- DataTypeIException for errors generated by the C API H5T
- PropListIException for errors generated by the C API H5P
- DataSetIException for errors generated by the C API H5D
- AttributeIException for errors generated by the C API H5A
- LibraryIException for errors generated by the C API H5

- The following table lists the services provided by Exception.

| Member Function | Purpose |
|------------------------|--|
| Exception | Default constructor |
| Exception | Constructor that stores a given detailed message |
| Exception | Copy constructor |
| GetMajorString | Returns the character string that describes an error specified by a major error number |
| GetMinorString | Returns the minor error string of the exception |
| GetFuncName | Returns the character string that describes an error specified by a minor error number |
| GetFileName | Returns the name of the file in which the error occurs |
| GetDescString | Returns the description string provided by the HDF5 library described the nature of the error |
| GetLine | Returns the line number where the error occurs |
| GetDetailMesg | Returns the user's detailed message annotating the error |
| SetAutoPrint | Turns on the automatic error printing |
| DontPrint | Turns off the automatic error printing |
| GetAutoPrint | Retrieves the current settings for the automatic error stack traversal function and its data |
| ClearErrorStack | Clears the error stack for the current thread |
| walkErrorStack | Walks the error stack for the current thread, calling the specified function |
| walkDefErrorStack | Default error stack traversal callback function that prints error messages to the specified output stream |
| printError | Displays the error information in the default manner - Note: <i>this function will be made virtual in the next release</i> |
| ~Exception | <i>missing destructor; will be virtual</i> |

IdComponent

This class provides a mean to ensure proper use of and to manage reference counting for an identifier of any HDF5 object. Hence, all HDF5 component classes benefit from this class. IdComponent uses RefCounter for its reference counting mechanism.

| Member Function | Purpose |
|------------------------|--|
| setId | Sets the identifier of this instance to a new value |
| getId | Gets the identifier of the instance, i.e. the current HDF5 object identifier |
| incRefCount | Increment reference counter |

| | |
|--------------|--|
| decRefCount | Decrement reference counter |
| getCounter | Gets the reference counter to this identifier |
| noReference | Determines whether there is no more reference to this identifier; note: the reference counter is decremented before checking |
| reset | Resets this instance by deleting its reference counter of the old identifier; this instance can then be used for another HDF5 identifier |
| IdComponent | Constructor that takes an HDF5 identifier |
| IdComponent | Copy constructor |
| ~IdComponent | Virtual destructor - <i>has a bug</i> |

RefCounter

RefCounter provides a reference counting mechanism. IdComponent uses this class to keep track of the number of copies of an HDF5 object so that the object's identifier can be properly released.

| Member Function | Purpose |
|-----------------|---|
| getCounter | Returns the value of the counter |
| noReference | Determines whether the counter is back to 0; note: the counter is decremented before checking |
| increment | Increment the counter |
| decrement | Decrement the counter |
| RefCounter | Default constructor |
| ~RefCounter | Destructor - will be virtual |

CommonFG

CommonFG means commonality between file and group. This class is a protocol class. Its existence is simply to provide the common services that are provided by H5File and Group. The file or group in the context of this class is referred to as 'location'.

| Member Function | Purpose |
|-----------------|---|
| createGroup | Creates a new group at this location |
| openGroup | Opens an existing group at this location |
| createDataSet | Creates a new dataset at this location |
| openDataSet | Opens a existing dataset at this location |
| openDataType | Opens a named generic datatype at this location |
| openEnumType | Opens a named enumeration datatype at this location |

HDF5 Release 1.4.1

| | |
|----------------|--|
| openCompType | Opens a named compound datatype at this location |
| openIntType | Opens a named integer datatype at this location |
| openFloatType | Opens a named floating-point datatype at this location |
| openStrType | Opens a named string datatype at this location |
| link | Creates a link of the specified type from a new name to the current name; both names are interpreted relative to this location. |
| unlink | Removes the specified name at this location |
| move | Renames an object within this location |
| getObjinfo | Retrieves information about an object, given its name and link, at this location |
| getLinkval | Returns the name of the HDF5 object that the given symbolic link points to |
| setComment | Sets the comment for an object, specified by its name, in this location |
| getComment | Gets the comment of an object, specified by its name, in this location |
| mount | Mounts a file, specified by its name, to this location |
| unmount | Unmounts a file, specified by its name, from this location |
| throwException | pure virtual - implemented by H5File and Group so that each class can throw the appropriate exception when an error occurs within CommonFG |
| CommonFG | Default constructor |
| ~CommonFG | Virtual default constructor |

H5File

This class uses a number of functions, which are publicly provided by class CommonFG, to access HDF5 files. In the context of these functions, a file is considered a location. Refer to Section 2.5 for the mentioned functions. In addition, H5File provides some functions that are specific to HDF5 files and not applicable to group. These functions are listed in the following table.

| Member Function | Purpose |
|------------------------|--|
| H5File | Creates or opens an HDF5 file |
| H5File | Copy constructor |
| isHdf5 | Determines if a file, specified by its name, is in HDF5 format |
| reopen | Reopens this file |
| getCreatePlist | Gets the creation property list of this file |
| getAccessPlist | Gets the access property list of this file |
| throwException | throws FileIException |
| ~H5File | Virtual destructor |

H5Object

This class provides services that are used to access an HDF5 object, which can be a group, a dataset, an attribute, or a named datatype.

| Member Function | Purpose |
|-----------------|--|
| H5Object | Copy constructor |
| flush | Flushes all buffers associated with this object, which belongs to a file, to disk |
| createAttribute | Creates an attribute for this object, which can be a group, a dataset, or a named datatype |
| openAttribute | Opens an attribute for this object given the attribute's name or index; the object can be either a group, a dataset, or a named datatype |
| iterateAttrs | Iterate user's function over the attributes of this object |
| getNumAttrs | Determines the number of attributes attached to this object |
| removeAttr | Removes the named attribute from this object |
| ~H5Object | Virtual destructor |

Group

Group represents the HDF5 group. As with H5File, this class inherits from CommonFG those functions that access an HDF5 group, which is called location in that context. It also inherits from another base class, H5Object, those characteristics of an HDF5 object, namely, the functions that access HDF5 attributes.

| Member Function | Purpose |
|-----------------|---|
| Group | Default constructor |
| Group | Copy constructor |
| Group | Constructor that takes an HDF5 identifier |
| iterateElems | Iterates over the elements of this group - <i>C++ style version not yet implemented</i> |
| throwException | throw GroupIException |
| | Default constructor and copy constructor |
| ~Group | Virtual destructor |

AbstractDs

AbstractDs is from the term abstract dataset. Because an HDF5 attribute is similar to a dataset, this abstract dataset class is introduced to provide their common functionality. AbstractDs is an abstract base class, from which the classes Attribute and DataSet are derived. This class also publicly inherits from H5Object and passes down the services that H5Object provides.

| Member Function | Purpose |
|------------------------|--|
| AbstractDs | Copy constructor |
| getSpace | Gets the dataspace of this dataset - pure virtual |
| getTypeClass | Gets the class of the datatype that is used by this dataset |
| getDataType | Gets the generic datatype of a dataset or an attribute. |
| getEnumType | Gets the enumeration datatype of a dataset or an attribute. |
| getCompType | Gets the compound datatype of a dataset or an attribute. |
| getIntType | Gets the integer datatype of a dataset or an attribute. |
| getFloatType | Gets the floating-point datatype of a dataset or an attribute. |
| getStrType | Gets the string datatype of a dataset or an attribute. |
| ~AbstractDs | Virtual destructor |

Notes on implementation:

getSpace is a pure virtual function. DataSet and Attribute provide their own implementation.

In the next version of the C++ API, the DataSet and Attribute member functions read and write might be overloaded and moved up into this base class.

DataSet

This class provides the services that are used to access an HDF5 dataset.

| Member Function | Purpose |
|------------------------|--|
| DataSet | Default constructor |
| DataSet | Copy constructor |
| getCreatePlist | Gets the creation property list of this dataset |
| getStorageSize | Gets the storage size of this dataset |
| read | Reads the data of this dataset and stores it in the provided buffer. The memory and file dataspace and the transferring property list can be defaults. |
| write | Writes the buffered data to this dataset. The memory and file dataspace and the transferring property list can be defaults |

| | |
|--------------|--|
| iterateElems | Iterates over all selected elements in a dataspace - C++ style version not yet implemented |
| extend | Extends the dataset with unlimited dimension |
| ~DataSet | Virtual destructor |

Notes on implementation:

read and write may be implemented using operators >> and << in the next version of the C++ API.

iterateElems is not yet implemented. It may be moved to class DataSet since it is iterating over elements that are in a dataspace.

Attribute

This class provides the services that are used to access an HDF5 object's attribute.

| Member Function | Purpose |
|-----------------|---|
| Attribute | Constructor that takes an HDF5 identifier |
| Attribute | Copy constructor |
| read | Reads data from this attribute |
| write | Writes data to this attribute |
| getName | Gets the name of this attribute |
| ~Attribute | Virtual destructor |

Notes on implementation:

read and write may be implemented using operators >> and << in the next version of the C++ API.

DataType

This class provides the services that are used to access an HDF5 generic datatype. Several subclasses are derived from DataType.

| Member Function | Purpose |
|-----------------|--|
| DataType | Default constructor |
| DataType | Copy constructor |
| DataType | Constructor that takes an existing identifier |
| DataType | Constructor that takes a datatype's class and size |

| | |
|--------------|---|
| copy | Copies an existing datatype to this datatype instance |
| getClass | Returns the datatype class identifier |
| commit | Commits a transient datatype to a file, creating a new named datatype |
| committed | Determines whether a datatype is a named type or a transient type |
| find | Finds a conversion function that can handle the conversion this datatype to the given datatype, dest. |
| convert | Converts data from between specified datatypes |
| setOverflow | Sets the overflow handler to a specified function |
| getOverflow | Returns a pointer to the current global overflow function |
| lock | Locks a datatype |
| getSize | Returns the size of a datatype |
| getSuper | Returns the base datatype from which a datatype is derived - <i>not completely implemented yet</i> |
| registerFunc | Registers a conversion function |
| unregister | Removes a conversion function from all conversion paths |
| setTag | Tags an opaque datatype |
| getTag | Gets the tag associated with an opaque datatype |
| ~DataType | Virtual destructor |

Notes on implementation:

Following is the structure of the subclasses of DataType. Those that are in italic face are not yet implemented.

DataType

CompType: is a compound datatype.

EnumType: is an enumeration datatype.

AtomType: is an atomic datatype and has the following subclasses.

PredType: is a predefined datatype for integer, float, and string. Note that this class may need inherit directly from DataType; more study is necessary.

Reference: is predefined datatype for object and region references and is not yet implemented. Note that once this class is implemented an intermediate base class might be introduced for PredType and Reference.

IntType: is a user-defined integer datatype.

FloatType: is a user-defined floating-point datatype.

StrType: is a user-defined string datatype.

BitFieldType: is a user-defined bitfield datatype and is not yet implemented.

OpaqueType: is a user-defined opaque datatype and is not yet implemented.

The function getSuper currently only returns the generic datatype. To get the specific datatype, the user must cast it. In future versions, its implementation will be improved.

EnumType

This class provides the services that are used to access an enumeration datatype. It is derived from DataType.

| Member Function | Purpose |
|-----------------|---|
| EnumType | Default constructor |
| EnumType | Copy constructor |
| EnumType | Creates a new enumeration datatype based on a native signed integer type, whose size is given by size |
| EnumType | Gets the enumeration datatype of the specified dataset |
| EnumType | Creates a new enum datatype based on an integer datatype |
| insert | Inserts a new member to this enumeration type |
| nameOf | Returns the symbol name corresponding to a specified member of this enumeration datatype |
| valueOf | Returns the value corresponding to a specified member of this enumeration datatype |
| GetMemberValue | Returns the value of an enumeration datatype member |
| ~EnumType | Virtual destructor |

CompType

This class provides the services that are used to access a compound datatype. It is derived from DataType.

| Member Function | Purpose |
|-------------------|---|
| CompType | Default constructor |
| CompType | Copy constructor |
| CompType | Creates a new compound datatype given its size |
| CompType | Gets the compound datatype of the specified dataset |
| getNmembers | Gets the number of members in this compound datatype |
| getMemberName | Gets the name of a member of this compound datatype |
| getMemberOffset | Gets the offset of a member of this compound datatype |
| getMemberDims | Gets the dimensionality of the specified member |
| getMemberClass | Gets the type class identifier of the specified member |
| getMemberDataType | Gets the generic datatype of the specified member in this compound datatype; the subsequent functions are for the specific sup-types. |
| getMemberEnumType | Gets the enumeration datatype of a dataset or an attribute. |

| | |
|--------------------|---|
| getMemberCompType | Gets the compound datatype of a dataset or an attribute. |
| getMemberIntType | Gets the integer datatype of a dataset or an attribute. |
| getMemberFloatType | Gets the floating-point datatype of a dataset or an attribute. |
| getMemberStrType | Gets the string datatype of a dataset or an attribute. |
| insertMember | Adds a new member to this compound datatype; <i>the ability to insert an array is removed from this member function</i> |
| pack | Recursively removes padding from within this compound datatype |

AtomType

This class is derived from DataType and, in addition, provides common services to access predefined types, integer type, floating-point type, and string type.

| Member Function | Purpose |
|-----------------|---|
| AtomType | Copy constructor |
| setSize | Sets the total size for an atomic datatype |
| getOrder | Returns the byte order of an atomic datatype |
| setOrder | Sets the byte ordering of an atomic datatype |
| getPrecision | Returns the precision of an atomic datatype |
| setPrecision | Sets the precision of an atomic datatype |
| getOffset | Retrieves the bit offset of the first significant bit |
| setOffset | Sets the bit offset of the first significant bit |
| getPad | <i>temporarily removed from this class</i> |
| setPad | <i>temporarily removed from this class</i> |
| ~AtomType | Virtual destructor |

PredType

This class contains the definition of objects that correspond to the predefined datatypes defined in the HDF5 library. Refer to the header file PredType.h for specific names.

IntType

This class provides the services used to access the user-defined integer datatype. It is derived from AtomType.

| Member Function | Purpose |
|-----------------|--|
| IntType | Default constructor |
| IntType | Copy constructor |
| IntType | Creates an IntType using a predefined integer type |
| IntType | Gets the integer datatype of the specified dataset |
| getSign | Returns the sign type for an integer type |
| setSign | Sets the sign property for an integer type |
| ~IntType | Virtual destructor |

FloatType

This class provides the services used to access the user-defined floating-point datatype. It is derived from AtomType.

| Member Function | Purpose |
|-----------------|---|
| FloatType | Default constructor |
| FloatType | Copy constructor |
| FloatType | Creates a new FloatType using a predefined floating-point type |
| FloatType | Gets the floating-point datatype of the specified dataset |
| getFields | Retrieves floating point datatype bit field information |
| setFields | Sets locations and sizes of floating point bit fields |
| getEbias | Retrieves the exponent bias of a floating-point type |
| SetEbias | Sets the exponent bias of a floating-point type |
| GetNorm | Returns the mantissa normalization of a floating-point datatype |
| SetNorm | Sets the mantissa normalization of a floating-point datatype |
| GetInpad | Retrieves the internal padding type for unused bits in floating-point datatypes |
| SetInpad | Fills unused internal floating point bits |
| ~FloatType | Virtual destructor |

StrType

This class provides the services used to access the user-defined string datatype. It is derived from AtomType.

| Member Function | Purpose |
|------------------------|---|
| StrType | Default constructor |
| StrType | Copy constructor |
| StrType | Creates a new StrType datatype using a predefined string type |
| StrType | Gets the string datatype of the specified dataset |
| GetCset | Returns the character set type of this string datatype |
| SetCset | Sets character set to be used |
| GetStrpad | Retrieves the string padding method for this string datatype |
| SetStrpad | Defines the storage mechanism for character strings |
| ~StrType | Virtual destructor |

DataSpace

This class provides services that are used to access an HDF5 dataspace. It inherits the HDF5 object identifier management from the base class IdComponent.

| Member Function | Purpose |
|------------------------|---|
| DataSpace | Default constructor |
| DataSpace | Copy constructor |
| DataSpace | Creates a dataspace object given the space type |
| DataSpace | Creates a simple dataspace |
| Copy | Makes copy of an existing dataspace instance |
| IsSimple | Determines if this dataspace is a simple one |
| OffsetSimple | Sets the offset of this simple dataspace |
| GetSimpleExtentDims | Retrieves dataspace dimension size and maximum size |
| GetSimpleExtentNdims | Gets the dimensionality of this dataspace |
| GetSimpleExtentNpoints | Gets the number of elements in this dataspace |
| GetSimpleExtentType | Gets the current class of this dataspace |
| ExtentCopy | Copies the extent of this dataspace |
| SetExtentSimple | Sets or resets the size of this dataspace |
| SetExtentNone | Removes the extent from this dataspace |

| | |
|-------------------------|---|
| GetSelectNpoints | Gets the number of elements in this dataspace selection |
| GetSelectHyperNblocks | Get number of hyperslab blocks |
| GetSelectHyperBlocklist | Gets the list of hyperslab blocks currently selected |
| GetSelectElemNpoints | Gets the number of element points in the current selection |
| GetSelectElemPointlist | Retrieves the list of element points currently selected |
| GetSelectBounds | Gets the bounding box containing the current selection |
| SelectElements | Selects array elements to be included in the selection |
| SelectAll | Selects the entire dataspace |
| SelectNone | Resets the selection region to include no elements |
| SelectValid | Verifies that the selection is within the extent of the dataspace |
| SelectHyperslab | Selects a hyperslab region to add to the current selected region |

Notes on implementation:

In the next version of the C++ API, this class may be broken into a class hierarchy that reflects the nature of the dataspace.

PropList

This class provides the services used to access the HDF5 file and data set property lists. It inherits the HDF5 object identifier management from the base class IdComponent.

| Member Function | Purpose |
|-----------------|---|
| PropList | Default constructor |
| PropList | Copy constructor |
| PropList | Creates a property list given its type |
| copy | Makes a copy of the given property list |
| getClass | Gets the type of the property list, i.e, H5P_FILE_CREATE, H5P_FILE_ACCESS, etc... |
| ~PropList | Virtual destructor |

FileCreatPropList

This class is derived from class PropList. It also provides the services specifically used to access the HDF5 file creation property lists.

| Member Function | Purpose |
|------------------------|---|
| FileCreatPropList | Default constructor |
| FileCreatPropList | Copy constructor |
| FileCreatPropList | Creates a file creation property list |
| getVersion | Retrieves version information for various parts of a file. |
| setUserblock | Sets the userblock size field of a file creation property list. |
| getUserblock | Gets the size of a user block in this file creation property list. |
| setSize | Sets file size-of addresses and sizes. |
| getSize | Retrieves the size-of address and size quantities stored in a file according to this file creation property list. |
| setSymk | Sets the size of parameters used to control the symbol table nodes. |
| setIstorek | Sets the size of parameter used to control the B-trees for indexing chunked datasets. |
| getIstorek | Returns the 1/2 rank of an indexed storage B-tree. |
| ~FileCreatPropList | Virtual destructor |

FileAccPropList

This class is derived from class PropList. It also provides the services specifically used to access the HDF5 file access property lists.

| Member Function | Purpose |
|------------------------|--|
| FileAccPropList | Default constructor |
| FileAccPropList | Copy constructor |
| FileAccPropList | Creates a file access property list |
| setCache | Sets the meta data cache and raw data chunk cache parameters. |
| getCache | Retrieves maximum sizes of data caches and the preemption policy value. |
| setAlignment | Sets alignment properties of this file access property list. |
| getAlignment | Retrieves the current settings for alignment properties from this file access property list. |
| setGcReferences | Sets garbage collecting references flag |

| | |
|------------------|--|
| getGcReferences | Returns garbage collecting references setting. |
| ~FileAccPropList | Virtual destructor |
| setStdio | <i>The following member functions were removed since parallel mode is not supported by C++ API</i> |
| getStdio | <i>removed</i> |
| getDriver | <i>removed</i> |
| setSec2 | <i>removed</i> |
| getSec2 | <i>removed</i> |
| setCore | <i>removed</i> |
| getCore | <i>removed</i> |
| setFamily | <i>removed</i> |
| getFamily | <i>removed</i> |
| setSplit | <i>removed</i> |
| getSplit | <i>removed</i> |

DSetCreatPropList

This class is derived from class PropList. It also provides the services specifically used to access the HDF5 dataset creation property lists.

| Member Function | Purpose |
|-------------------|--|
| DsetCreatPropList | Default constructor |
| DsetCreatPropList | Copy constructor |
| DsetCreatPropList | Creates a dataset creation property list |
| setLayout | Sets the type of storage used to store the raw data for the dataset that uses this property list |
| getLayout | Gets the layout of the raw data storage of the data that uses this property list |
| setChunk | Sets the size of the chunks used to store a chunked layout dataset. |
| getChunk | Retrieves the size of the chunks used to store a chunked layout dataset. |
| setDeflate | Sets compression method and compression level |
| setFillValue | Sets a dataset fill value |
| getFillValue | Retrieves a dataset fill value |
| setFilter | Adds a filter to the filter pipeline |
| getNfilters | Returns the number of filters in the pipeline |
| getFilter | Returns information about a filter in a pipeline |
| setExternal | Adds an external file to the list of external files |
| getExternalCount | Returns the number of external files for a dataset |

| | |
|--------------------|--|
| getExternal | Returns information about an external file |
| ~DsetCreatPropList | Virtual destructor |

DSetMemXferPropList

This class is derived from class PropList. It also provides the services specifically used to access the HDF5 data set memory and transfer property lists.

| Member Function | Purpose |
|----------------------|--|
| DsetMemXferPropList | Default constructor |
| DsetMemXferPropList | Copy constructor |
| DsetMemXferPropList | Creates a dataset memory and transfer property list |
| setBuffer | Sets type conversion and background buffers |
| getBuffer | Reads buffer settings |
| setPreserve | Sets the dataset transfer property list status to TRUE or FALSE |
| getPreserve | Checks status of the dataset transfer property list |
| setHyperCache | Indicates whether to cache hyperslab blocks during I/O |
| getHyperCache | Returns information regarding the caching of hyperslab blocks during I/O |
| setBtreeRatios | Sets B-tree split ratios for a dataset transfer property list |
| getBtreeRatios | Gets B-tree split ratios for a dataset transfer property list |
| setVlenMemManager | Sets the memory manager for variable-length datatype allocation in H5Dread and H5Dvlen_reclaim |
| getVlenMemManager | Gets the memory manager for variable-length datatype allocation in H5Dread and H5Tvlen_reclaim |
| ~DSetMemXferPropList | Virtual destructor |

Examples

The following examples show the application of some of the available functionality:

`create.cpp`: writes a dataset to an HDF5 file. Specific functions used include:

- constructors of `H5File`, `DataSet`, and `IntType`
- `H5File::createDataSet` to create a new dataset in this file
- `DataSet::write`
- `DataType::setOrder`
- exception handlings

`readdata.cpp`: obtains dataset information from an HDF5 file and reads selected data from the file. Specific functions, that are not in the previous example, include:

- `H5File::openDataSet` to open an existing dataset that belongs to this file
- `DataSet::getTypeClass` to get the type class identifier of the datatype of this dataset to determine what type is to be expected, in this case, it is `H5T_INTEGER`, i.e. the datatype is an integer
- constructor an empty `IntType` instance after knowing what the expecting type is; this `IntType` object is passed into the subsequent function
- `DataSet::getType` to retrieve the dataset's datatype which is an integer
- `IntType::getOrder`
- `IntType::getSize`
- `DataSet::getSpace`
- `DataSet::getSimpleExtentNdims`
- `DataSet::getSimpleExtentDims`
- `DataSet::selectHyperslab`
- `DataSet::read`

`writedata.cpp`: writes selected data to an HDF5 file. Specific functions that are not in the previous examples include:

- `DataSet::selectNone`
- `DataSet::selectElements`

`compound.cpp`: creates a compound datatype, write an array, which has the compound datatype to the file, and read back fields' subsets. Specific functions that are not in the previous examples include:

- constructor `CompType`
- `CompType::insertMember` to insert some members into the compound datatype

- `CompType::getMemberClass` to get the type class identifier to determine what type is to be expected, in this case, it is `H5T_FLOAT`, i.e. the member datatype is floating-point
- constructor an empty `FloatType` instance after knowing what the expecting type is; this `FloatType` object is passed into the subsequent function.
- `CompType::getMemberType` to retrieve the specific datatype, `FloatType`
- `FloatType::getNorm` to get the mantissa normalization of the floating-point datatype

`extend_ds.cpp`: works with extendible dataset. Specific functions that are not in the previous examples include:

- constructor of `PropList`
- `DsetCreatPropList::setChunk`
- `DataSet::extend`

`chunks.cpp`: reads data from a chunked dataset. Specific functions that are not in the previous examples include:

- `DataSet::getSpace` to get the dataspace in the file of this dataset
- `DataSet::getCreatePlist` to get the dataset creation property list
- `DsetCreatPropList::getLayout`
- `DsetCreatPropList::getChunk`

`h5group.cpp`: creates and . Specific functions that are not in the previous examples include:

- `H5File::createGroup` to get a group in the file
- constructor of `DsetCreatPropList`
- `DsetCreatPropList::setDeflate`
- `H5File::openGroup` to open a group in the file
- `Group::openDataSet` to open a dataset in the group
- `H5File::link` to create a hard link to a group
- `H5File::unlink` to remove the hard link

