Datatypes

Creating and manipulating datatypes which describe elements of a dataset (H5T)

General Datatype Operations

- H5T_CLOSE
- *H5T_COMMIT*
    - H5T_COMMIT1 *
    - H5T_COMMIT2
- H5T_COMMIT_ANON
- H5T_COMMITTED
- H5T_COPY
- H5T_CREATE
- H5T_DETECT_CLASS
- H5T_EQUAL
- H5T_FLUSH
- H5T_GET_CLASS
- H5T_GET_CREATE_PLIST
- H5T_GET_NATIVE_TYPE
- H5T_GET_SIZE
- H5T_GET_SUPER
- H5T_LOCK
- *H5T_OPEN*
    - H5T_OPEN1 *
    - H5T_OPEN2
- H5T_REFRESH
- H5T_SET_SIZE

Conversion Functions

- H5T_COMPILER_CONV
- H5T_CONVERT
- H5T_DECODE
- H5T_ENCODE
- H5T_FIND
- H5T_REGISTER
- H5T_UNREGISTER

Atomic Datatype Properties

- H5T_GET_CSET
- H5T_GET_EBIAS
- H5T_GET_FIELDS
- H5T_GET_INPAD
- H5T_GET_NORM
- H5T_GET_OFFSET
- H5T_GET_ORDER
- H5T_GET_PAD
- H5T_GET_PRECISION
- H5T_GET_SIGN
- H5T_GET_STRPAD
- H5T_IS_VARIABLE_STR
- H5T_SET_CSET
- H5T_SET_EBIAS
- H5T_SET_FIELDS
- H5T_SET_INPAD
- H5T_SET_NORM
- H5T_SET_OFFSET
- H5T_SET_ORDER
- H5T_SET_PAD
- H5T_SET_PRECISION
- H5T_SET_SIGN
- H5T_SET_STRPAD

Array Datatypes

- *H5T_ARRAY_CREATE*
    - H5T_ARRAY_CREATE1 *
    - H5T_ARRAY_CREATE2
- *H5T_GET_ARRAY_DIMS*
    - H5T_GET_ARRAY_DIMS1 *
    - H5T_GET_ARRAY_DIMS2
- H5T_GET_ARRAY_NDIMS

Compound Datatype Properties

- H5T_GET_MEMBER_CLASS
- H5T_GET_MEMBER_INDEX
- H5T_GET_MEMBER_NAME
- H5T_GET_MEMBER_OFFSET
- H5T_GET_MEMBER_TYPE
- H5T_GET_NMEMBERS
- H5T_INSERT
- H5T_PACK

Variable-length Array Datatypes

- H5T_VLEN_CREATE

Opaque Datatypes

- H5T_GET_TAG
- H5T_SET_TAG

Enumeration Datatypes

- H5T_ENUM_CREATE
- H5T_ENUM_INSERT
- H5T_ENUM_NAMEOF
- H5T_ENUM_VALUEOF
- H5T_GET_MEMBER_INDEX
- H5T_GET_MEMBER_NAME
- H5T_GET_MEMBER_VALUE
- H5T_GET_NMEMBERS

** Use of these functions is deprecated in Release 1.8.0.*

The Datatype interface, H5T, provides a mechanism to describe the storage format of individual data points of a data set and is hopefully designed in such a way as to allow new features to be easily added without disrupting applications that use the data type interface. A dataset (the H5D interface) is composed of a collection or raw data points of homogeneous type organized according to the data space (the H5S interface).

A datatype is a collection of datatype properties, all of which can be stored on disk, and which when taken as a whole, provide complete information for data conversion to or from that datatype. The interface provides functions to set and query properties of a datatype.

A *data point* is an instance of a *datatype*, which is an instance of a *type class*. We have defined a set of type classes and properties which can be extended at a later time. The atomic type classes are those which describe types which cannot be decomposed at the datatype interface level; all other classes are compound.

See HDF5 Datatypes in the —*HDF5 User's Guide* for more information.

List of pre-defined datatypes in HDF5

- Predefined Datatypes

Creating variable-length string datatypes

As the term implies, variable-length strings are strings of varying lengths; they can be arbitrarily long, anywhere from 1 character to thousands of characters.

HDF5 provides the ability to create a variable-length string datatype. Like all string datatypes, this type is based on the atomic string datatype: `H5T_C_S1` in C or `H5T_FORTRAN_S1` in Fortran. While these datatypes default to one character in size, they can be resized to specific fixed lengths or to variable length.

Variable-length strings will transparently accommodate ASCII strings or UTF-8 strings. This characteristic is set with H5T_SET_CSET in the process of creating the datatype.

The following HDF5 calls create a C-style variable-length string datatype, `vls_type_c_id`:

```
vls_type_c_id = H5Tcopy(H5T_C_S1)
    status        = H5Tset_size(vls_type_c_id, H5T_VARIABLE)
```

In a C environment, variable-length strings will always be NULL-terminated, so the buffer to hold such a string must be one byte larger than the string itself to accommodate the NULL terminator.

In Fortran, strings are normally of fixed length. Variable-length strings come into play only when data is shared with a C application that uses them. For such situations, the datatype class `H5T_STRING` is predefined by the HDF5 library to accommodate variable-length strings. The first HDF5 call below creates a Fortran string, `vls_type_f_id`, that will handle variable-length string data. The second call sets the string padding value to space padding:

```
h5tcopy_f(H5T_STRING, vls_type_f_id, hdferr)
    h5tset_strpad_f(vls_type_f_id, H5T_STR_SPACEPAD_F, hdferr)
```

While Fortran-style strings are generally space-padded, they may be NULL-terminated in cases where the data is also used in a C environment.

Note: Under the covers, variable-length strings are stored in a heap, potentially impacting efficiency in the following ways:

- Heap storage requires more space than regular raw data storage.
- Heap access generally reduces I/O efficiency because it requires individual read or write operations for each data element rather than one read or write per dataset or per data selection.
- Chunking and filters, including compression, are not available for heaps.