H5TB_WRITE_FIELDS_INDEX

Expand all Collapse all

- Jump to ...
- Summary
- Description
- Example
- Switch language ...
- C
- C++
- FORTRAN
- JAVA

H5TB_WRITE_FIELDS_INDEX

Overwrites fields

Procedure:

H5TB_WRITE_FIELDS_INDEX (loc_id, table_name, nfields, field_index, start, nrecords, type_size, field_offset, field_sizes, data)

Signature:

```
herr_t H5TBwrite_fields_index (
              hid_t loc_id,
              const char *table_name,
              int nfields,
              const int *field_index,
              hsize_t start,
              hsize_t nrecords,
              size_t type_size,
              const size_t *field_offset,
              const size_t* field_sizes,
              const void *data
         )
```

```
subroutine h5tbwrite_field_index_f(loc_id, dset_name, field_index, start, &
                                   nrecords, type_size, buf, errcode)
  implicit none
  integer(HID_T), intent(IN) :: loc_id           ! file or group identifier
  character(LEN=*), intent(IN) :: dset_name      ! name of the dataset
  integer, intent(IN) :: field_index             ! index
  integer(HSIZE_T), intent(IN) :: start          ! start record
  integer(HSIZE_T), intent(IN) :: nrecords       ! records
  integer(SIZE_T), intent(IN) :: type_size       ! type size
  , intent(IN), dimension(*) :: buf        ! data buffer
  integer :: errcode                             ! error code
end subroutine h5tbwrite_field_index_f
```

Parameters:

hid_t `loc_id`	IN: Identifier of the file or group where the table is located
const char `*table_name`	IN: The name of the dataset to overwrite
int `nfields`	IN: The number of fields to overwrite. This parameter is also the size of the field_index array.
const int `*field_index`	IN: The indexes of the fields to write
hsize_t `start`	IN: The zero based index record to start writing
hsize_t `nrecords`	IN: The number of records to write
size_t `type_size`	IN: The size of the structure type, as calculated by sizeof()
const size_t `*field_offset`	IN: An array containing the offsets of the fields. These offsets can be calculated with the HOFFSET macro.
const size_t `*field_sizes`	IN: An array containing the sizes of the fields
void `*data`	IN: Buffer with data

Description:

H5TB_WRITE_FIELDS_INDEX overwrites one or several fields specified by `field_index` with a buffer `data` from a dataset named `table_name` attached to the object specified by the identifier `loc_id`.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Example:

hl / examples / ex_table_05.c [36:131] 1.10/master HDFFV/hdf5

```
typedef struct Particle
 {
  char   name[16];
  int    lati;
  int    longi;
  float  pressure;
  double temperature;
 } Particle;
```

```
/* Define a subset of Particle, with latitude and longitude fields */
typedef struct Position
{
 int    lati;
 int    longi;
} Position;

/* Calculate the type_size and the offsets of our struct members */
Particle  dst_buf[NRECORDS];
size_t dst_size =  sizeof( Particle );
size_t dst_offset[NFIELDS] = { HOFFSET( Particle, name ),
                               HOFFSET( Particle, lati ),
                               HOFFSET( Particle, longi ),
                               HOFFSET( Particle, pressure ),
                               HOFFSET( Particle, temperature )};
size_t dst_sizes[NFIELDS] = { sizeof( dst_buf[0].name),
                              sizeof( dst_buf[0].lati),
                              sizeof( dst_buf[0].longi),
                              sizeof( dst_buf[0].pressure),
                              sizeof( dst_buf[0].temperature)};

size_t field_offset_pos[2] = { HOFFSET( Position, lati ),
                               HOFFSET( Position, longi )};

/* Initially no data */
Particle  *p_data = NULL;

/* Define field information */
const char *field_names[NFIELDS]  =
{ "Name","Latitude", "Longitude", "Pressure", "Temperature" };
hid_t      field_type[NFIELDS];
hid_t      string_type;
hid_t      file_id;
hsize_t    chunk_size = 10;
 Particle   fill_data[1] =
{ {"no data",-1,-1, -99.0f, -99.0} };   /* Fill value particle */
int        compress  = 0;
hsize_t    nfields;
hsize_t    start;                       /* Record to start reading/writing */
hsize_t    nrecords;                    /* Number of records to read/write */
int        i;

/* Define new values for the field "Pressure"  */
float      pressure_in  [NRECORDS_ADD] =
{ 0.0f,1.0f,2.0f};
int        field_index_pre[1]     = { 3 };
int        field_index_pos[2]     = { 1,2 };

/* Define new values for the fields "Latitude,Longitude"  */
Position   position_in[NRECORDS_ADD] = { {0,0},
{10,10},
{20,20} };

size_t field_sizes_pos[2]=
{
 sizeof(position_in[0].longi),
 sizeof(position_in[0].lati)
};
```

```
size_t field_sizes_pre[1]=
{
 sizeof(float)
};

/* Initialize the field field_type */
string_type = H5Tcopy( H5T_C_S1 );
H5Tset_size( string_type, 16 );
field_type[0] = string_type;
field_type[1] = H5T_NATIVE_INT;
field_type[2] = H5T_NATIVE_INT;
field_type[3] = H5T_NATIVE_FLOAT;
field_type[4] = H5T_NATIVE_DOUBLE;

/* Create a new file using default properties. */
file_id = H5Fcreate( "ex_table_05.h5", H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT );

/* Make the table */
H5TBmake_table( "Table Title", file_id, TABLE_NAME,NFIELDS,NRECORDS,
                        dst_size,field_names, dst_offset, field_type,
                        chunk_size, fill_data, compress, p_data  );

/* Write the pressure field starting at record 2 */
nfields  = 1;
start    = 2;
nrecords = NRECORDS_ADD;
```

```
 H5TBwrite_fields_index( file_id, TABLE_NAME, nfields, field_index_pre, start,
nrecords,
   sizeof( float ), 0, field_sizes_pre, pressure_in  );
```

Please see Tables for Fortran programming hints.

hl / fortran / test / tsttable.F90 [66:340] 1.10/master HDFFV/hdf5

```
INTEGER(HSIZE_T), PARAMETER :: nrecords = 5            ! nrecords
  CHARACTER(LEN=9),DIMENSION(1:nfields) :: field_names  ! field names
  INTEGER(SIZE_T),  DIMENSION(1:nfields) :: field_offset ! field offset
  INTEGER(HID_T),   DIMENSION(1:nfields) :: field_types  ! field types
  INTEGER(HSIZE_T), PARAMETER  :: chunk_size = 5         ! chunk size
  INTEGER, PARAMETER :: compress = 0                     ! compress
  INTEGER            :: errcode = 0                      ! Error flag
  INTEGER            :: i                                ! general purpose integer
  INTEGER(SIZE_T)    :: type_size                        ! Size of the datatype
  INTEGER(SIZE_T)    :: type_sizec                       ! Size of the character
datatype
  INTEGER(SIZE_T)    :: type_sizei                       ! Size of the integer
datatype
  INTEGER(SIZE_T)    :: type_sized                       ! Size of the double
precision datatype
  INTEGER(SIZE_T)    :: type_sizer                       ! Size of the real datatype
  INTEGER(HID_T)     :: type_id_c                        ! Memory datatype identifier
(for character field)
  INTEGER(SIZE_T)    :: offset                           ! Member's offset
  INTEGER(HSIZE_T)   :: start = 0                        ! start record
  INTEGER, DIMENSION(nrecords) :: bufi                   ! Data buffer
  INTEGER, DIMENSION(nrecords) :: bufir                  ! Data buffer
  REAL, DIMENSION(nrecords) :: bufr                      ! Data buffer
  REAL, DIMENSION(nrecords) :: bufrr                     ! Data buffer
  DOUBLE PRECISION, DIMENSION(nrecords) :: bufd          ! Data buffer
  DOUBLE PRECISION, DIMENSION(nrecords) :: bufdr         ! Data buffer
  CHARACTER(LEN=2), DIMENSION(nrecords), PARAMETER :: bufs =
(/"AB","CD","EF","GH","IJ"/) ! Data buffer
  CHARACTER(LEN=2), DIMENSION(nrecords) :: bufsr         ! Data buffer
  INTEGER(HSIZE_T) :: nfieldsr                           ! nfields
  INTEGER(HSIZE_T) :: nrecordsr                          ! nrecords
  CHARACTER(LEN=9), DIMENSION(1:nfields) :: field_namesr  ! field names
  INTEGER(SIZE_T),  DIMENSION(1:nfields) :: field_offsetr ! field offset
  INTEGER(SIZE_T),  DIMENSION(1:nfields) :: field_sizesr  ! field sizes
  INTEGER(SIZE_T)  :: type_sizeout = 0                    ! size of the datatype
  INTEGER(SIZE_T)  :: maxlen = 0                          ! max character length of a
field name
  INTEGER :: Cs_sizeof_double = H5_SIZEOF_DOUBLE          ! C's sizeof double
  INTEGER :: SIZEOF_X
  LOGICAL :: Exclude_double
  CHARACTER(LEN=62) :: test_txt

  ! Find size of DOUBLE PRECISION
#ifdef H5_FORTRAN_HAVE_STORAGE_SIZE
  SIZEOF_X = storage_size(bufd(1))/storage_size(c_char_'a')
#else
  SIZEOF_X = SIZEOF(bufd(1))
#endif
```

```
! If Fortran DOUBLE PRECISION and C DOUBLE sizeofs don't match then disable
! creating a DOUBLE RECISION field, and instead create a REAL field. This
! is needed to handle when DOUBLE PRECISION is promoted via a compiler flag.
Exclude_double = .FALSE.
IF(Cs_sizeof_double.NE.SIZEOF_X)THEN
   Exclude_double = .TRUE.
ENDIF

!
! Initialize the data arrays.
!
DO i = 1, nrecords
   bufi(i) = i
   bufr(i) = i
   bufd(i) = i
END DO

!
! Create a new file using default properties.
!
CALL h5fcreate_f(filename, H5F_ACC_TRUNC_F, file_id, errcode)


!-------------------------------------------------------------------------
! make table
! initialize the table parameters
!-------------------------------------------------------------------------

field_names(1) = "field1"
field_names(2) = "field2a"
field_names(3) = "field3ab"
field_names(4) = "field4abc"

!
! calculate total size by calculating sizes of each member
!
CALL h5tcopy_f(H5T_NATIVE_CHARACTER, type_id_c, errcode)
type_size = 2
CALL h5tset_size_f(type_id_c, type_size, errcode)
CALL h5tget_size_f(type_id_c, type_sizec, errcode)
CALL h5tget_size_f(H5T_NATIVE_INTEGER, type_sizei, errcode)
IF(exclude_double)THEN
   CALL h5tget_size_f(H5T_NATIVE_REAL, type_sized, errcode)
ELSE
   CALL h5tget_size_f(H5T_NATIVE_DOUBLE, type_sized, errcode)
ENDIF
CALL h5tget_size_f(H5T_NATIVE_REAL, type_sizer, errcode)
type_size = type_sizec + type_sizei + type_sized + type_sizer

!
! type ID's
!
field_types(1) = type_id_c
field_types(2) = H5T_NATIVE_INTEGER
IF(exclude_double)THEN
   field_types(3) = H5T_NATIVE_REAL
ELSE
   field_types(3) = H5T_NATIVE_DOUBLE
ENDIF
```

```
  field_types(4) = H5T_NATIVE_REAL

  !
  ! offsets
  !
  offset = 0
  field_offset(1) = offset
  offset = offset + type_sizec ! Offset of the second memeber is 2
  field_offset(2) = offset
  offset = offset + type_sizei ! Offset of the second memeber is 6
  field_offset(3) = offset
  offset = offset + type_sized ! Offset of the second memeber is 14
  field_offset(4) = offset

  !-------------------------------------------------------------------------
  ! make table
  !-------------------------------------------------------------------------

  test_txt = "Make table"
  CALL test_begin(test_txt)

  CALL h5tbmake_table_f(dsetname1,&
       file_id,&
       dsetname1,&
       nfields,&
       nrecords,&
       type_size,&
       field_names,&
       field_offset,&
       field_types,&
       chunk_size,&
       compress,&
       errcode )

  CALL passed()


  !-------------------------------------------------------------------------
  ! write field
  !-------------------------------------------------------------------------

  test_txt = "Read/Write field by name"
  CALL test_begin(test_txt)

  CALL
h5tbwrite_field_name_f(file_id,dsetname1,field_names(1),start,nrecords,type_sizec,&
       bufs,errcode)

  CALL
h5tbwrite_field_name_f(file_id,dsetname1,field_names(2),start,nrecords,type_sizei,&
       bufi,errcode)
  IF(exclude_double)THEN
     CALL
h5tbwrite_field_name_f(file_id,dsetname1,field_names(3),start,nrecords,type_sized,&
          bufr,errcode)
  ELSE
     CALL
h5tbwrite_field_name_f(file_id,dsetname1,field_names(3),start,nrecords,type_sized,&
          bufd,errcode)
```

```
  ENDIF

  CALL
h5tbwrite_field_name_f(file_id,dsetname1,field_names(4),start,nrecords,type_sizer,&
       bufr,errcode)

  !-----------------------------------------------------------------------
  ! read field
  !-----------------------------------------------------------------------

  ! Read an invalid field, should fail
  CALL
h5tbread_field_name_f(file_id,dsetname1,'DoesNotExist',start,nrecords,type_sizec,&
       bufsr,errcode)

  IF(errcode.GE.0)THEN
     PRINT *, 'error in h5tbread_field_name_f'
     CALL h5fclose_f(file_id, errcode)
     CALL h5close_f(errcode)
     STOP
  ENDIF

  ! Read a valid field, should pass
  CALL
h5tbread_field_name_f(file_id,dsetname1,field_names(1),start,nrecords,type_sizec,&
       bufsr,errcode)
  IF(errcode.LT.0)THEN
     PRINT *, 'error in h5tbread_field_name_f'
     CALL h5fclose_f(file_id, errcode)
     CALL h5close_f(errcode)
     STOP
  ENDIF

  !
  ! compare read and write buffers.
  !
  DO i = 1, nrecords
     IF ( bufsr(i) .NE. bufs(i) ) THEN
        PRINT *, 'read buffer differs from write buffer'
        PRINT *,  bufsr(i), ' and ',   bufs(i)
        STOP
     ENDIF
  END DO

  CALL
h5tbread_field_name_f(file_id,dsetname1,field_names(2),start,nrecords,type_sizei,&
       bufir,errcode)

  !
  ! compare read and write buffers.
  !
  DO i = 1, nrecords
     IF ( bufir(i) .NE. bufi(i) ) THEN
        PRINT *, 'read buffer differs from write buffer'
        PRINT *,  bufir(i), ' and ',   bufi(i)
        STOP
     ENDIF
  END DO
```

```
  IF(exclude_double)THEN

     CALL
h5tbread_field_name_f(file_id,dsetname1,field_names(3),start,nrecords,type_sized,&
          bufrr,errcode)

  !
  ! compare read and write buffers.
  !
     DO i = 1, nrecords
       CALL VERIFY("h5tbread_field_name_f", bufrr(i), bufr(i), errcode)
       IF (errcode .NE.0 ) THEN
           PRINT *, 'read buffer differs from write buffer'
           PRINT *,  bufrr(i), ' and ',   bufr(i)
           STOP
        ENDIF
     END DO

  ELSE
     CALL
h5tbread_field_name_f(file_id,dsetname1,field_names(3),start,nrecords,type_sized,&
          bufdr,errcode)

  !
  ! compare read and write buffers.
  !
     DO i = 1, nrecords
        CALL VERIFY("h5tbread_field_name_f", bufdr(i), bufd(i), errcode)
        IF (errcode .NE.0 ) THEN
           PRINT *, 'read buffer differs from write buffer'
           PRINT *,  bufdr(i), ' and ',   bufd(i)
           STOP
        ENDIF
     END DO
  ENDIF



  CALL
h5tbread_field_name_f(file_id,dsetname1,field_names(4),start,nrecords,type_sizer,&
       bufrr,errcode)

  !
  ! compare read and write buffers.
  !
  DO i = 1, nrecords
     CALL VERIFY("h5tbread_field_name_f", bufrr(i), bufr(i), errcode)
     IF (errcode .NE.0 ) THEN
        PRINT *, 'read buffer differs from write buffer'
        PRINT *,  bufrr(i), ' and ',   bufr(i)
        STOP
     ENDIF
  END DO

  CALL passed()


  !-------------------------------------------------------------------------
  ! write field
```

```
!-------------------------------------------------------------------------

test_txt = "Read/Write field by index"
CALL test_begin(test_txt)
```

```
CALL h5tbwrite_field_index_f(file_id,dsetname1,1,start,nrecords,type_sizec,&
     bufs,errcode)
```

--- Last Modified: November 20, 2019 | 02:02 PM