H5P_SET_SZIP

Expand all   Collapse all

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

Summary
Description
Example
JAVA
FORTRAN
C++
C

H5P_SET_SZIP

Sets up use of the SZIP compression filter

Procedure:

H5P_SET_SZIP (plist, options_mask, pixels_per_block)

Signature:

```
herr_t H5Pset_szip(hid_t plist,
                unsigned int options_mask,
                unsigned int pixels_per_block)
```

```
Fortran90 Interface: h5pset_szip_f

SUBROUTINE h5pset_szip_f(prp_id, options_mask, pixels_per_block, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id
                                   ! Dataset creation property list identifier
  INTEGER, INTENT(IN) :: options_mask
                                   ! A bit-mask conveying the desired
                                   ! SZIP options
                                   ! Current valid values in Fortran are:
                                   !    H5_SZIP_EC_OM_F
                                   !    H5_SZIP_NN_OM_F
  INTEGER, INTENT(IN) :: pixels_per_block
                                   ! The number of pixels or data elements
                                   ! in each data block
  INTEGER, INTENT(OUT)  :: hdferr  ! Error code
                                   ! 0 on success and -1 on failure
END SUBROUTINE h5pset_szip_f
```

Parameters:

hid_t `plist`	IN: Dataset creation property list identifier
unsigned int `options_mask`	IN: A bit-mask conveying the desired SZIP options Valid values are `H5_SZIP_EC_OPTION_MASK` and `H5_SZIP_NN_OPTION_MASK`.
unsigned int `pixels_per_block`	IN: The number of pixels or data elements in each data block

Description:

H5P_SET_SZIP sets an SZIP compression filter, `H5Z_FILTER_SZIP`, for a dataset. SZIP is a compression method designed for use with scientific data.

Before proceeding, all users should review the “Limitations” section below.

Users familiar with SZIP outside the HDF5 context may benefit from reviewing “Notes for Users Familiar with SZIP in Other Contexts” below.

In the text below, the term *pixel* refers to an HDF5 data element. This terminology derives from SZIP compression's use with image data, where pixel referred to an image pixel.

The SZIP `bits_per_pixel` value (see **Notes**, below) is automatically set, based on the HDF5 datatype. SZIP can be used with atomic datatypes that may have size of 8, 16, 32, or 64 bits. Specifically, a dataset with a datatype that is 8-, 16-, 32-, or 64-bit signed or unsigned integer; char; or 32- or 64-bit float can be compressed with SZIP. See **Notes**, below, for further discussion of the the SZIP `bits_per_pixel` setting.

SZIP options are passed in an options mask, `options_mask`, as follows.

Option	**Description** **(Mutually exclusive; select one.)**
`H5_SZIP_EC_OPTION_MASK`	Selects entropy coding method
`H5_SZIP_NN_OPTION_MASK`	Selects nearest neighbor coding method

The following guidelines can be used in determining which option to select:

- The entropy coding method, the EC option specified by `H5_SZIP_EC_OPTION_MASK`, is best suited for data that has been processed. The EC method works best for small numbers.
- The nearest neighbor coding method, the NN option specified by `H5_SZIP_NN_OPTION_MASK`, preprocesses the data then the applies EC method as above.

Other factors may affect results, but the above criteria provides a good starting point for optimizing data compression.

SZIP compresses data block by block, with a user-tunable block size. This block size is passed in the parameter `pixels_per_block` and must be even and not greater than 32, with typical values being `8`, `10`, `16`, or `32`. This parameter affects compression ratio; the more pixel values vary, the smaller this number should be to achieve better performance.

In HDF5, compression can be applied only to chunked datasets. If `pixels_per_block` is bigger than the total number of elements in a dataset chunk, H5P_SET_SZIP will succeed but the subsequent call to H5D_CREATE will fail; the conflict can be detected only when the property list is used.

To achieve optimal performance for SZIP compression, it is recommended that a chunk's fastest-changing dimension be equal to *N* times `pixels_per_block` where *N* is the maximum number of blocks per scan line allowed by the SZIP library. In the current version of SZIP, *N* is set to 128.

SZIP compression is an optional HDF5 filter.

Limitations:

- SZIP compression cannot be applied to compound, array, variable-length, enumeration, or any other user-defined datatypes.
  If an SZIP filter is set in a dataset creation property list used to create a dataset containing a non-allowed datatype, the call to H5D_CREATE will fail; the conflict can be detected only when the property list is used.

- Users should be aware that there are factors that affect one's rights and ability to use SZIP compression. See the documents at SZIP Compression in HDF5 for *important information regarding terms of use and the SZIP copyright notice*, for further discussion of SZIP compression in HDF5, and for a list of SZIP-related references.

Notes for Users Familiar with SZIP in Other Contexts:

The following notes are of interest primarily to those who have used SZIP compression outside of the HDF5 context.

In non-HDF5 applications, SZIP typically requires that the user application supply additional parameters:

- `pixels_in_object`, the number of pixels in the object to be compressed
- `bits_per_pixel`, the number of bits per pixel
- `pixels_per_scanline`, the number of pixels per scan line

These values need not be independently supplied in the HDF5 environment as they are derived from the datatype and dataspace, which are already known. In particular, HDF5 sets `pixels_in_object` to the number of elements in a chunk and `bits_per_pixel` to the size of the element or pixel datatype. The following algorithm is used to set `pixels_per_scanline`:

- If the size of a chunk's fastest-changing dimension, *size*, is greater than 4K, set `pixels_per_scanline` to 128 times `pixels_per_block`.
- If *size* is less than 4K but greater than `pixels_per_block`, set `pixels_per_scanline` to the minimum of *size* and 128 times `pixels_per_block`.
- If *size* is less than `pixels_per_block` but greater than the number elements in the chunk, set `pixels_per_scanline` to the minimum of the number elements in the chunk and 128 times `pixels_per_block`.

The HDF5 datatype may have precision that is less than the full size of the data element, e.g., an 11-bit integer can be defined using H5T_SET_PRECISION. To a certain extent, SZIP can take advantage of the precision of the datatype to improve compression:

- If the HDF5 datatype size is 24-bit or less and the offset of the bits in the HDF5 datatype is zero (see H5T_SET_OFFSET or H5T_GET_OFFSET), the data is the in lowest N bits of the data element. In this case, the SZIP `bits_per_pixel` is set to the precision of the HDF5 datatype.
- If the offset is not zero, the SZIP `bits_per_pixel` will be set to the number of bits in the full size of the data element.
- If the HDF5 datatype precision is 25-bit to 32-bit, the SZIP `bits_per_pixel` will be set to 32.
- If the HDF5 datatype precision is 33-bit to 64-bit, the SZIP `bits_per_pixel` will be set to 64.

HDF5 always modifies the options mask provided by the user to set up usage of `RAW_OPTION_MASK`, `ALLOW_K13_OPTION_MASK`, and one of `LSB_OPTION_MASK` or `MSB_OPTION_MASK`, depending on endianness of the datatype.

Returns:

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

Example:

1_10 / C / H5D / h5ex_d_szip.c [85:93] master HDFFV/hdf5-examples

```
dcpl = H5Pcreate (H5P_DATASET_CREATE);
    status = H5Pset_szip (dcpl, H5_SZIP_NN_OPTION_MASK, 8);
    status = H5Pset_chunk (dcpl, 2, chunk);

    /*
     * Create the dataset.
     */
    dset = H5Dcreate (file, DATASET, H5T_STD_I32LE, space, H5P_DEFAULT, dcpl,
                H5P_DEFAULT);
```

1_12 / FORTRAN / H5D / h5ex_d_szip.f90 [85:91] master HDFFV/hdf5-examples

```
CALL h5pcreate_f(H5P_DATASET_CREATE_F, dcpl, hdferr)
  CALL h5pset_szip_f(dcpl, H5_SZIP_NN_OM_F, 8, hdferr)
  CALL h5pset_chunk_f(dcpl, 2, chunk, hdferr)
  !
  ! Create the dataset.
  !
  CALL h5dcreate_f(file, dataset, H5T_STD_I32LE, space, dset, hdferr, dcpl)
```

History:

Release	Change
1.6.0	Function introduced in this release.

--- Last Modified: August 07, 2019 | 02:59 PM