# H5DO_READ_CHUNK



# H5DO_READ_CHUNK

Reads a raw data chunk directly from a dataset in a file into a buffer (DEPRECATED)

> *This function was deprecated in favor of the function H5D_READ_CHUNK as of HDF5-1.10.3.*

**Procedure:**

H5DO_READ_CHUNK (dset_id, dxpl_id, filter_mask, offset, buf)

**Signature:**

```
herr_t H5DOread_chunk( hid_t dset_id, hid_t dxpl_id, const hsize_t *offset, uint32_t *filter_mask, void
*buf )
```

**Parameters:**

| | |
|---|---|
| *hid_t* `dset_id` | IN: Identifier for the dataset to be read |
| *hid_t* `dxpl_id` | IN: Transfer property list identifier for this I/O operation |
| *uint32_t* `* filter_mask` | IN/OUT: Mask for identifying the filters used with the chunk |
| *const hsize_t* `*offset` | IN: Logical position of the chunk's first element in the dataspace |
| *void* `*buf` | IN: Buffer containing the chunk read from the dataset |

### Description:

H5DO_READ_CHUNK reads a raw data chunk as specified by its logical `offset` in a chunked dataset `dset_id` from the dataset in the file into the application memory buffer `buf`. The data in `buf` is read directly from the file bypassing the library's internal data transfer pipeline, including filters.

`dxpl_id` is a data transfer property list identifier.

The mask `filter_mask` indicates which filters are used with the chunk when written. A zero value indicates that all enabled filters are applied on the chunk. A filter is skipped if the bit corresponding to the filter's position in the pipeline (`0 position < 32`) is turned on.

`offset` is an array specifying the logical position of the first element of the chunk in the dataset's dataspace. The length of the offset array must equal the number of dimensions, or rank, of the dataspace. The values in `offset` must not exceed the dimension limits and must specify a point that falls on a dataset chunk boundary.

`buf` is the memory buffer containing the chunk read from the dataset in the file.

> In HDF5 1.10.3, the functionality of H5DO_READ_CHUNK was moved to H5D_READ_CHUNK. For compatibility, this API call has been left as a stub which simply calls H5D_READ_CHUNK. New code should use H5D_READ_CHUNK.

### Returns:

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

### Example:

The following code illustrates the use of H5DO_READ_CHUNK to read a chunk from a dataset:

```
#include <zlib.h>
#include <math.h>
#define DEFLATE_SIZE_ADJUST(s) (ceil(((double)(s))*1.001)+12)
               :
               :
size_t       buf_size = CHUNK_NX*CHUNK_NY*sizeof(int);
const Bytef *z_src = (const Bytef*)(direct_buf);
Bytef       *z_dst;             /* Destination buffer            */
uLongf       z_dst_nbytes = (uLongf)DEFLATE_SIZE_ADJUST(buf_size);
uLong        z_src_nbytes = (uLong)buf_size;
int          aggression = 9;    /* Compression aggression setting */
uint32_t     filter_mask = 0;
size_t       buf_size = CHUNK_NX*CHUNK_NY*sizeof(int);
/* For H5DOread_chunk() */
void        *readbuf = NULL;                   /* Buffer for reading data */
const Bytef *pt_readbuf;                       /* Point to the buffer for data read */
hsize_t     read_chunk_nbytes;                 /* Size of chunk on disk */
int         read_dst_buf[CHUNK_NX][CHUNK_NY];  /* Buffer to hold un-compressed data */

/* Create the data space */
if((dataspace = H5Screate_simple(RANK, dims, maxdims)) < 0)
    goto error;

/* Create a new file */
if((file = H5Fcreate(FILE_NAME5, H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT)) < 0)
   goto error;

/* Modify dataset creation properties, i.e. enable chunking and compression */
if((cparms = H5Pcreate(H5P_DATASET_CREATE)) < 0)
   goto error;

if((status = H5Pset_chunk( cparms, RANK, chunk_dims)) < 0)
   goto error;

if((status = H5Pset_deflate( cparms, aggression)) < 0)
   goto error;

/* Create a new dataset within the file using cparms creation properties */
if((dset_id = H5Dcreate2(file, DATASETNAME, H5T_NATIVE_INT, dataspace, H5P_DEFAULT,
                        cparms, H5P_DEFAULT)) < 0)
```

```
   goto error;

/* Initialize data for one chunk */
for(i = n = 0; i < CHUNK_NX; i++)
    for(j = 0; j < CHUNK_NY; j++)
        direct_buf[i][j] = n++;

/* Allocate output (compressed) buffer */
outbuf = malloc(z_dst_nbytes);
z_dst = (Bytef *)outbuf;

/* Perform compression from the source to the destination buffer */
ret = compress2(z_dst, &z_dst_nbytes, z_src, z_src_nbytes, aggression);

/* Check for various zlib errors */
if(Z_BUF_ERROR == ret) {
    fprintf(stderr, "overflow");
    goto error;
} else if(Z_MEM_ERROR == ret) {
    fprintf(stderr, "deflate memory error");
    goto error;
} else if(Z_OK != ret) {
    fprintf(stderr, "other deflate error");
    goto error;
}

/* Write the compressed chunk data repeatedly to cover all the
 * chunks in the dataset, using the direct write function.     */
for(i=0; i<NX/CHUNK_NX; i++) {
    for(j=0; j<NY/CHUNK_NY; j++) {
        status = H5DOwrite_chunk(dset_id, H5P_DEFAULT, filter_mask,
                                 offset, z_dst_nbytes, outbuf);
        offset[1] += CHUNK_NY;
    }
    offset[0] += CHUNK_NX;
    offset[1] = 0;
}

if(H5Fflush(dataset, H5F_SCOPE_LOCAL) < 0)
    goto error;

if(H5Dclose(dataset) < 0)
    goto error;

if((dataset = H5Dopen2(file, DATASETNAME1, H5P_DEFAULT)) < 0)
    goto error;

offset[0] = CHUNK_NX;
offset[1] = CHUNK_NY;

/* Get the size of the compressed chunk */
ret = H5Dget_chunk_storage_size(dataset, offset, &read_chunk_nbytes);

readbuf = HDmalloc(read_chunk_nbytes);
pt_readbuf = (const Bytef *)readbuf;

/* Use H5DOread_chunk() to read the chunk back */
if((status = H5DOread_chunk(dataset, H5P_DEFAULT, offset, &read_filter_mask, readbuf)) < 0)
    goto error;

ret = uncompress((Bytef *)read_dst_buf, (uLongf *)&buf_size, pt_readbuf, (uLong)read_chunk_nbytes);

/* Check for various zlib errors */
if(Z_BUF_ERROR == ret) {
    fprintf(stderr, "error: not enough room in output buffer");
    goto error;
} else if(Z_MEM_ERROR == ret) {
    fprintf(stderr, "error: not enough memory");
    goto error;
} else if(Z_OK != ret) {
    fprintf(stderr, "error: corrupted input data");
    goto error;
```

```
}

/* Data verification here */
            :
            :
```

**History:**

| Release | Change |
| --- | --- |
| 1.10.3 | Function deprecated in favor of `H5Dread_chunk`. |
| 1.10.2, 1.8.19 | C function was introduced |

--- Last Modified: May 22, 2019 | 08:12 AM