Hierarchical Data Format query language (HDFql)
Reference Manual
Version 2.1.0
Copyright (C) 2016-2019

This document is part of the Hierarchical Data Format query language (HDFql). For more information about HDFql, please visit the website http://www.hdfql.com.

Disclaimer

Every effort has been made to ensure that this document is as accurate as possible. The information contained in this document is provided without any express, statutory or implied warranties. The founders of HDFql shall have neither liability nor responsibility to any person or entity with respect to any loss or damage arising from the information in this document or the usage of HDFql.
TABLE OF CONTENTS

1. INTRODUCTION ................................................................................................................. 1

2. INSTALLATION ....................................................................................................................... 3
   2.1 WINDOWS .......................................................................................................................... 4
   2.2 LINUX .............................................................................................................................. 4
   2.3 MACOS ............................................................................................................................ 5

3. USAGE .................................................................................................................................. 6
   3.1 C ........................................................................................................................................ 6
   3.2 C++ .................................................................................................................................... 10
   3.3 JAVA ................................................................................................................................ 14
   3.4 PYTHON ........................................................................................................................... 17
   3.5 C# ...................................................................................................................................... 19
   3.6 FORTRAN ......................................................................................................................... 22
   3.7 R ...................................................................................................................................... 27
   3.8 COMMAND-LINE INTERFACE ......................................................................................... 29

4. CURSOR .................................................................................................................................. 32
   4.1 DESCRIPTION ................................................................................................................... 32
   4.2 SUBCURSOR ....................................................................................................................... 36
   4.3 EXAMPLES ......................................................................................................................... 38

5. APPLICATION PROGRAMMING INTERFACE ..................................................................... 44
   5.1 CONSTANTS ...................................................................................................................... 44
   5.2 FUNCTIONS ...................................................................................................................... 52
      5.2.1 HDFQL_EXECUTE ....................................................................................................... 58
      5.2.2 HDFQL_EXECUTE_GET_STATUS ............................................................................... 59
      5.2.3 HDFQL_ERROR_GET_LINE ....................................................................................... 60
      5.2.4 HDFQL_ERROR_GET_POSITION ............................................................................... 61
<table>
<thead>
<tr>
<th>Section</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.5</td>
<td>HDFQL_ERROR_GET_MESSAGE</td>
</tr>
<tr>
<td>5.2.6</td>
<td>HDFQL_CURSOR_INITIALIZE</td>
</tr>
<tr>
<td>5.2.7</td>
<td>HDFQL_CURSOR_USE</td>
</tr>
<tr>
<td>5.2.8</td>
<td>HDFQL_CURSOR_USE_DEFAULT</td>
</tr>
<tr>
<td>5.2.9</td>
<td>HDFQL_CURSOR_CLEAR</td>
</tr>
<tr>
<td>5.2.10</td>
<td>HDFQL_CURSOR_CLONE</td>
</tr>
<tr>
<td>5.2.11</td>
<td>HDFQL_CURSOR_GET_DATA_TYPE</td>
</tr>
<tr>
<td>5.2.12</td>
<td>HDFQL_CURSOR_GET_COUNT</td>
</tr>
<tr>
<td>5.2.13</td>
<td>HDFQL_SUBCURSOR_GET_COUNT</td>
</tr>
<tr>
<td>5.2.14</td>
<td>HDFQL_CURSOR_GET_POSITION</td>
</tr>
<tr>
<td>5.2.15</td>
<td>HDFQL_SUBCURSOR_GET_POSITION</td>
</tr>
<tr>
<td>5.2.16</td>
<td>HDFQL_CURSOR_FIRST</td>
</tr>
<tr>
<td>5.2.17</td>
<td>HDFQL_SUBCURSOR_FIRST</td>
</tr>
<tr>
<td>5.2.18</td>
<td>HDFQL_CURSOR_LAST</td>
</tr>
<tr>
<td>5.2.19</td>
<td>HDFQL_SUBCURSOR_LAST</td>
</tr>
<tr>
<td>5.2.20</td>
<td>HDFQL_CURSOR_NEXT</td>
</tr>
<tr>
<td>5.2.21</td>
<td>HDFQL_SUBCURSOR_NEXT</td>
</tr>
<tr>
<td>5.2.22</td>
<td>HDFQL_CURSOR_PREVIOUS</td>
</tr>
<tr>
<td>5.2.23</td>
<td>HDFQL_SUBCURSOR_PREVIOUS</td>
</tr>
<tr>
<td>5.2.24</td>
<td>HDFQL_CURSOR_ABSOLUTE</td>
</tr>
<tr>
<td>5.2.25</td>
<td>HDFQL_SUBCURSOR_ABSOLUTE</td>
</tr>
<tr>
<td>5.2.26</td>
<td>HDFQL_CURSOR_RELATIVE</td>
</tr>
<tr>
<td>5.2.27</td>
<td>HDFQL_SUBCURSOR_RELATIVE</td>
</tr>
<tr>
<td>5.2.28</td>
<td>HDFQL_CURSOR_GET_SIZE</td>
</tr>
<tr>
<td>5.2.29</td>
<td>HDFQL_SUBCURSOR_GET_SIZE</td>
</tr>
<tr>
<td>5.2.30</td>
<td>HDFQL_CURSOR_GET</td>
</tr>
<tr>
<td>5.2.31</td>
<td>HDFQL_SUBCURSOR_GET</td>
</tr>
<tr>
<td>5.2.32</td>
<td>HDFQL_CURSOR_GET_TINYINT</td>
</tr>
</tbody>
</table>
5.2.33 HDFQL_SUBCURSOR_GET_TINYINT

5.2.34 HDFQL_CURSOR_GET_UNSIGNED_TINYINT

5.2.35 HDFQL_SUBCURSOR_GET_UNSIGNED_TINYINT

5.2.36 HDFQL_CURSOR_GET_SMALLINT

5.2.37 HDFQL_SUBCURSOR_GET_SMALLINT

5.2.38 HDFQL_CURSOR_GET_UNSIGNED_SMALLINT

5.2.39 HDFQL_SUBCURSOR_GET_UNSIGNED_SMALLINT

5.2.40 HDFQL_CURSOR_GET_INT

5.2.41 HDFQL_SUBCURSOR_GET_INT

5.2.42 HDFQL_CURSOR_GET_UNSIGNED_INT

5.2.43 HDFQL_SUBCURSOR_GET_UNSIGNED_INT

5.2.44 HDFQL_CURSOR_GET_BIGINT

5.2.45 HDFQL_SUBCURSOR_GET_BIGINT

5.2.46 HDFQL_CURSOR_GET_UNSIGNED_BIGINT

5.2.47 HDFQL_SUBCURSOR_GET_UNSIGNED_BIGINT

5.2.48 HDFQL_CURSOR_GET_FLOAT

5.2.49 HDFQL_SUBCURSOR_GET_FLOAT

5.2.50 HDFQL_CURSOR_GET_DOUBLE

5.2.51 HDFQL_SUBCURSOR_GET_DOUBLE

5.2.52 HDFQL_CURSOR_GET_CHAR

5.2.53 HDFQL_SUBCURSOR_GET_CHAR

5.2.54 HDFQL_VARIABLE_REGISTER

5.2.55 HDFQL_VARIABLE_TRANSIENT_REGISTER

5.2.56 HDFQL_VARIABLE_UNREGISTER

5.2.57 HDFQL_VARIABLE_UNREGISTER_ALL

5.2.58 HDFQL_VARIABLE_GET_NUMBER

5.2.59 HDFQL_VARIABLE_GET_DATA_TYPE

5.2.60 HDFQL_VARIABLE_GET_COUNT
5.2.61 HDFQL_VARIABLE_GET_SIZE .................................................................................. 129
5.2.62 HDFQL_VARIABLE_GET_DIMENSION_COUNT ......................................................... 130
5.2.63 HDFQL_VARIABLE_GET_DIMENSION ....................................................................... 131
5.2.64 HDFQL_MPI_GET_SIZE .............................................................................................. 132
5.2.65 HDFQL_MPI_GET_RANK .............................................................................................. 133

6. LANGUAGE .................................................................................................................. 135

6.1 DATA TYPES .............................................................................................................. 138
6.1.1 TINYINT ....................................................................................................................... 140
6.1.2 UNSIGNED TINYINT ......................................................................................................... 141
6.1.3 SMALLINT ..................................................................................................................... 142
6.1.4 UNSIGNED SMALLINT .................................................................................................... 142
6.1.5 INT ................................................................................................................................. 143
6.1.6 UNSIGNED INT ............................................................................................................... 143
6.1.7 BIGINT .......................................................................................................................... 144
6.1.8 UNSIGNED BIGINT .......................................................................................................... 145
6.1.9 FLOAT ............................................................................................................................ 145
6.1.10 DOUBLE ......................................................................................................................... 146
6.1.11 CHAR ............................................................................................................................ 147
6.1.12 VARTINYINT ............................................................................................................... 147
6.1.13 UNSIGNED VARTINYINT ............................................................................................... 148
6.1.14 VARSMALLINT .............................................................................................................. 149
6.1.15 UNSIGNED VARSMALLINT ........................................................................................... 149
6.1.16 VARINT ......................................................................................................................... 150
6.1.17 UNSIGNED VARINT ....................................................................................................... 151
6.1.18 VARBIGINT .................................................................................................................... 151
6.1.19 UNSIGNED VARBIGINT ............................................................................................... 152
6.1.20 VARFLOAT ..................................................................................................................... 153
6.1.21vardouble ...................................................................................................................... 153
6.1.22 VARCHAR .................................................................................................................. 154
6.1.23 OPAQUE .................................................................................................................. 154
6.2 POST-PROCESSING ........................................................................................................ 155
6.2.1 ORDER ..................................................................................................................... 156
6.2.2 TOP .......................................................................................................................... 158
6.2.3 BOTTOM .................................................................................................................. 160
6.2.4 FROM TO ................................................................................................................. 162
6.2.5 STEP ......................................................................................................................... 164
6.3 REDIRECTING .............................................................................................................. 166
6.3.1 FROM ....................................................................................................................... 166
6.3.2 INTO ......................................................................................................................... 172
6.4 DATA DEFINITION LANGUAGE (DDL) ..................................................................... 177
6.4.1 CREATE DIRECTORY ............................................................................................... 177
6.4.2 CREATE FILE ............................................................................................................ 179
6.4.3 CREATE GROUP ....................................................................................................... 180
6.4.4 CREATE DATASET .................................................................................................... 183
6.4.5 CREATE ATTRIBUTE ............................................................................................... 190
6.4.6 CREATE [SOFT | HARD] LINK ................................................................................ 193
6.4.7 CREATE EXTERNAL LINK ...................................................................................... 195
6.4.8 ALTERT DIMENSION ............................................................................................ 198
6.4.9 RENAME DIRECTORY ............................................................................................ 199
6.4.10 RENAME FILE ...................................................................................................... 200
6.4.12 COPY FILE ............................................................................................................. 203
6.4.13 COPY [GROUP | DATASET | ATTRIBUTE | [SOFT] LINK | EXTERNAL LINK] ............. 205
6.4.14 DROP DIRECTORY ................................................................................................. 207
6.4.15 DROP FILE ............................................................................................................. 208
6.4.16 DROP [GROUP | DATASET | ATTRIBUTE | [SOFT] LINK | EXTERNAL LINK] ............... 209
6.5 DATA MANIPULATION LANGUAGE (DML).................................................................................................................210
  6.5.1 INSERT .................................................................................................................................................................211

6.6 DATA QUERY LANGUAGE (DQL) ..............................................................................................................................222
  6.6.1 SELECT ...............................................................................................................................................................222

6.7 DATA INTROSPECTION LANGUAGE (DIL) ................................................................................................................233
  6.7.1 SHOW FILE VALIDITY ........................................................................................................................................233
  6.7.2 SHOW USE DIRECTORY ..................................................................................................................................235
  6.7.3 SHOW USE FILE ................................................................................................................................................236
  6.7.4 SHOW ALL USE FILE .........................................................................................................................................238
  6.7.5 SHOW USE GROUP ...........................................................................................................................................239
  6.7.6 SHOW [GROUP | DATASET | ATTRIBUTE | [SOFT] LINK | EXTERNAL LINK] .........................................................241
  6.7.7 SHOW TYPE ......................................................................................................................................................248
  6.7.8 SHOW DATA TYPE ..........................................................................................................................................250
  6.7.9 SHOW MASK ........................................................................................................................................................252
  6.7.10 SHOW ENDIANNESS .....................................................................................................................................253
  6.7.11 SHOW CHARSET .............................................................................................................................................255
  6.7.12 SHOW STORAGE TYPE ....................................................................................................................................257
  6.7.13 SHOW STORAGE ALLOCATION ........................................................................................................................259
  6.7.14 SHOW STORAGE DIMENSION ..........................................................................................................................261
  6.7.15 SHOW DIMENSION ............................................................................................................................................262
  6.7.16 SHOW ORDER ...................................................................................................................................................265
  6.7.17 SHOW TAG .......................................................................................................................................................267
  6.7.18 SHOW FILL TYPE ..............................................................................................................................................269
  6.7.19 SHOW FILL VALUE ..........................................................................................................................................270
  6.7.20 SHOW FILE SIZE ............................................................................................................................................272
  6.7.21 SHOW [DATASET | ATTRIBUTE] SIZE ...............................................................................................................274
  6.7.22 SHOW HDFQL VERSION ..................................................................................................................................275
  6.7.23 SHOW HDF5 VERSION ...................................................................................................................................276
Hierarchical Data Format query language (HDFql) ........................................................................................................... Reference Manual

6.7.24  SHOW PCRE VERSION ........................................................................................................................................ 277
6.7.25  SHOW ZLIB VERSION ........................................................................................................................................ 278
6.7.26  SHOW MPI VERSION ........................................................................................................................................ 279
6.7.27  SHOW DIRECTORY ........................................................................................................................................... 280
6.7.28  SHOW FILE .......................................................................................................................................................... 282
6.7.29  SHOW EXECUTE STATUS ....................................................................................................................................... 283
6.7.30  SHOW LIBRARY BOUNDS ................................................................................................................................... 284
6.7.31  SHOW CACHE ....................................................................................................................................................... 286
6.7.32  SHOW ATOMIC ...................................................................................................................................................... 288
6.7.33  SHOW EXTERNAL LINK PREFIX ............................................................................................................................. 290
6.7.34  SHOW FLUSH ......................................................................................................................................................... 291
6.7.35  SHOW DEBUG ....................................................................................................................................................... 292

6.8  MISCELLANEOUS ....................................................................................................................................................... 293

6.8.1  USE DIRECTORY ...................................................................................................................................................... 294
6.8.2  USE FILE ................................................................................................................................................................... 295
6.8.3  USE GROUP .............................................................................................................................................................. 297
6.8.4  FLUSH ......................................................................................................................................................................... 300
6.8.5  CLOSE FILE .............................................................................................................................................................. 300
6.8.6  CLOSE ALL FILE ...................................................................................................................................................... 302
6.8.7  CLOSE GROUP .......................................................................................................................................................... 303
6.8.8  SET LIBRARY BOUNDS ........................................................................................................................................... 304
6.8.9  SET CACHE ............................................................................................................................................................... 306
6.8.10  SET ATOMIC .......................................................................................................................................................... 309
6.8.11  SET EXTERNAL LINK PREFIX ................................................................................................................................. 310
6.8.12  SET FLUSH ............................................................................................................................................................. 311
6.8.13  SET DEBUG ........................................................................................................................................................... 312

GLOSSARY ........................................................................................................................................................................... 314

Application Programming Interface (API) .......................................................................................................................... 314
Hierarchical Data Format query language (HDFql) ............................................................................ Reference Manual

Attribute .............................................................................................................................................314
Cursor ..................................................................................................................................................314
Dataset ...............................................................................................................................................314
Data type .............................................................................................................................................315
Endianness ...........................................................................................................................................315
Group ..................................................................................................................................................315
Hierarchical Data Format (HDF) .......................................................................................................315
Hyperslab .............................................................................................................................................315
Message Passing Interface (MPI) ........................................................................................................316
Operation ............................................................................................................................................316
Parallel HDF5 (PHDF5) .....................................................................................................................316
Post-processing ..................................................................................................................................316
Result set ............................................................................................................................................316
Result subset ......................................................................................................................................317
Subcursor ............................................................................................................................................317
LIST OF TABLES

Table 5.1 – HDFql constants in C .................................................................48
Table 5.2 – HDFql constants in C and their corresponding definitions in C++ .................................................................49
Table 5.3 – HDFql constants in C and their corresponding definitions in Java .................................................................50
Table 5.4 – HDFql constants in C and their corresponding definitions in Python .................................................................50
Table 5.5 – HDFql constants in C and their corresponding definitions in C# .................................................................50
Table 5.6 – HDFql constants in C and their corresponding definitions in Fortran .................................................................51
Table 5.7 – HDFql constants in C and their corresponding definitions in R .................................................................51
Table 5.8 – HDFql functions in C .................................................................54
Table 5.9 – HDFql functions in C and their corresponding definitions in C++ .................................................................55
Table 5.10 – HDFql functions in C and their corresponding definitions in Java .................................................................55
Table 5.11 – HDFql functions in C and their corresponding definitions in Python .................................................................56
Table 5.12 – HDFql functions in C and their corresponding definitions in C# .................................................................56
Table 5.13 – HDFql functions in C and their corresponding definitions in Fortran .................................................................57
Table 5.14 – HDFql functions in C and their corresponding definitions in R .................................................................57
Table 6.1 – HDFql operations text formatting conventions .................................................................135
Table 6.2 – HDFql operations .................................................................138
Table 6.3 – HDFql data types and their corresponding definitions in HDF .................................................................140
Table 6.4 – HDFql post-processing options .................................................................155
Table 6.5 – HDFql redirecting options .................................................................166
LIST OF FIGURES

Figure 3.1 – Illustration of the command-line interface “HDFqlCLI” ................................................................. 31

Figure 4.1 – Linearization of a two dimensional dataset into a (one dimensional) cursor ........................................ 36

Figure 4.2 – Cursor populated with data from dataset “my_dataset0” .................................................................. 38

Figure 4.3 – Cursor populated with data from dataset “my_dataset1” ................................................................. 39

Figure 4.4 – Cursor populated with data from dataset “my_dataset2” ................................................................. 40

Figure 4.5 – Cursor and its subcursor populated with data from dataset “my_dataset3” ......................................... 41

Figure 4.6 – Cursor and its subcursors populated with data from dataset “my_dataset4” ......................................... 42

Figure 4.7 – Cursor and its subcursors populated with data from dataset “my_dataset5” ......................................... 43
1. INTRODUCTION

HDFql stands for “Hierarchical Data Format query language” and is the first tool that enables users to manage HDF5 files through a high-level language. This language was designed to be simple to use and similar to SQL thus dramatically reducing the learning effort. HDFql can be seen as an alternative to the C API (which contains more than 400 low-level functions that are far from easy to use!) and to existing wrappers for C++, Java, Python, C#, Fortran and R for manipulating HDF5 files. In addition, and whenever possible, it automatically employs parallelism to speed-up operations hiding its inherent complexity from the user.

As an example, imagine that one needs to create an HDF5 file named “myFile.h5” and, inside it, a group named “myGroup” containing a (ZLIB) compressed dataset named “myDataset” of data type float with an initial value of 12.4. In HDFql, this can easily be implemented as follows:

```
create file myFile.h5
use file myFile.h5
create dataset myGroup/myDataset as float enable zlib values(12.4)
```

In contrast, using the C API on the same example is quite cumbersome:

```
hid_t file;
hid_t group;
hid_t dataspace;
hid_t property;
hid_t dataset;
hsize_t dimension;
float value;
file = H5Fcreate("myFile.h5", H5F_ACC_EXCL, H5P_DEFAULT, H5P_DEFAULT);
group = H5Gcreate(file, "myGroup", H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
dimension = 1;
dataspace = H5Screate_simple(1, &dimension, NULL);
```

1 Hierarchical Data Format is the name of a set of file formats and libraries designed to store and organize large amounts of numerical data. It is supported by The HDF Group, whose mission is to ensure continued development of HDF technologies and the continued accessibility of data currently stored in HDF. Please refer to the website http://www.hdfgroup.org for additional information.
property = H5Pcreate(H5P_DATASET_CREATE);
H5Pset_chunk(property, 1, &dimension);
H5Pset_deflate(property, 9);
dataset = H5Dcreate(group, "myDataset", H5T_NATIVE_FLOAT, dataspase, H5P_DEFAULT, property, H5P_DEFAULT);
value = 12.4;
H5Dwrite(dataset, H5T_NATIVE_FLOAT, H5S_ALL, H5S_ALL, H5P_DEFAULT, &value);
2. INSTALLATION

The official website of the Hierarchical Data Format query language (HDFql) is http://www.hdfql.com. Here, the most recent documentation and examples that illustrate how to solve disparate use-cases using HDFql can be found. In addition, in the download area (http://www.hdfql.com/#download) all versions of HDFql ever publicly released are available. These versions are packaged as ZIP files, with each one meant for a particular platform (i.e. Windows, Linux or macOS), architecture (i.e. 32 bit or 64 bit), compiler (Microsoft Visual Studio or Gnu Compiler Collection (GCC)) and – optionally – MPI library (i.e. MPICH or Open MPI). When decompressed, such ZIP files typically have the following organization in terms of directories and files contained within:

```
HDFql-x.y.z
  | + example (directory that contains C, C++, Java, Python, C#, Fortran and R examples)
  |     | + include (directory that contains HDFql C and C++ header files)
  |     | + lib (directory that contains HDFql C release/debug static and shared libraries)
  |     | + bin (directory that contains HDFql command-line interface and a proper launcher)
  |     | + wrapper (directory that contains HDFql wrappers)
  |     |     | + cpp (directory that contains HDFql C++ wrapper)
  |     |     | + java (directory that contains HDFql Java wrapper)
  |     |     | + python (directory that contains HDFql Python wrapper)
  |     |     | + csharp (directory that contains HDFql C# wrapper)
  |     |     | + fortran (directory that contains HDFql Fortran wrapper)
  |     |     | + R (directory that contains HDFql R wrapper)
  |     | + doc (directory that contains HDFql reference manual)
  |     - LICENSE.txt (file that contains information about HDFql license)
```
The following sections provide concise instructions on how to install HDFql in the different platforms that it currently supports – namely Windows, Linux and macOS.

2.1 WINDOWS

- Download the appropriate ZIP file according to the HDFql version, architecture and compiler of interest from http://www.hdfql.com/#download. For instance, if the HDFql version of interest is 1.0.0 and it is to be used in a machine running Windows 32 bit and, eventually, be linked against C or C++ code using the Microsoft Visual Studio 2010 compiler then the file to download is “HDFql-1.0.0_Windows32_VS-2010.zip”.

- Unzip the downloaded file using Windows Explorer in-build capabilities or a free tool such as 7-Zip (http://www.7-zip.org).

2.2 LINUX

- Download the appropriate ZIP file according to the HDFql version, architecture, compiler and (optional) MPI library of interest from http://www.hdfql.com/#download. For instance, if the HDFql version of interest is 1.4.0 and it is to be used in a machine running Linux 64 bit and, eventually, be linked against C, C++, or Fortran code using the GCC 4.9.x compiler with no need to work with HDF5 files in parallel (using an MPI library) then the file to download is “HDFql-1.4.0_Linux64_GCC-4.9.zip”.

- Unzip the downloaded file using the Archive Manager or the KArchive (if in GNOME or KDE respectively), or by opening a terminal and executing “unzip <downloaded_zip_file>”. If the unzip utility is not installed, it can be done by executing from a terminal:

```
sudo yum install unzip
```
• In a Debian-based distribution:

```
sudo apt-get install unzip
```

### 2.3 MACOS

• Download the appropriate ZIP file according to the HDFql version, architecture, compiler and (optional) MPI library of interest from [http://www.hdfql.com/#download](http://www.hdfql.com/#download). For instance, if the HDFql version of interest is 2.1.0 and it is to be used in a machine running macOS 64 bit and, eventually, be linked against C, C++, or Fortran code using the GCC 4.9.x compiler with the need of working with HDF5 files in parallel using MPICH 3.2.x MPI library then the file to download is “HDFql-2.1.0_Darwin64_GCC-4.9_MPICH-3.2.zip”.

• Unzip the downloaded file using the Archive Utility or by opening a terminal and executing “`unzip <downloaded_zip_file>`”. If the unzip utility is not installed, it can be done by executing from a terminal:

```
sudo port install unzip
```
3. USAGE

After following the instructions provided in the chapter INSTALLATION, HDFql is ready for usage. It can be used programmatically in C, C++ and Fortran through static and shared libraries; in Java, Python, C# and R through wrappers; and finally, through a command-line interface named “HDFqlCLI”. Moreover, in Linux and macOS, programs written in these programming languages may manipulate HDF5 files both in serial and in parallel, as distributions of HDFql built with the serial HDF5 library and the parallel HDFS (PHDFS) library are available for these platforms. The subsequent sections provide guidance on usage and basic troubleshooting information to solve issues that may arise.

3.1 C

HDFql can be used in the C programming language through static and shared libraries. These libraries are stored in the directory “lib”. The following short program illustrates how HDFql can be used in such language.

```c
// include HDFql C header file (make sure it can be found by the C compiler)
#include "HDFql.h"

int main(int argc, char *argv[])
{
    // display HDFql version in use
    printf("HDFql version: %s\n", HDFQL_VERSION);

    // create an HDF5 file named "my_file.h5"
    hdfql_execute("CREATE FILE my_file.h5");

    // use (i.e. open) HDF5 file "my_file.h5"
    hdfql_execute("USE FILE my_file.h5");

    // create an HDF5 dataset named "my_dataset" of data type int
```

1 Through MPICH (or, alternately, one of its ABI compatible derivative libraries such as Intel MPI, Cray MPT, MVAPICH2, Parastation MPI) or Open MPI. Both MPICH and Open MPI are freely available, high performance and widely portable implementations of the Message Passing Interface (MPI), a standard for message-passing for distributed memory applications used in parallel computing. Please refer to the website https://www.mpich.org and https://www.open-mpi.org for additional information.
```c
hdfql_execute("CREATE DATASET my_dataset AS INT VALUES(10)");

// select (i.e. read) data from dataset "my_dataset" and populate cursor with it
hdfql_execute("SELECT FROM my_dataset");

// move cursor to the first position within the result set
hdfql_cursor_first(NULL);

// display content of cursor
printf("Dataset value: %d\n", *hdfql_cursor_get_int(NULL));

return EXIT_SUCCESS;
}
```

Assuming that the program is stored in a file named “example.c”, it must first be compiled before it can be launched from a terminal. To compile the program against the HDFql C static library:

- **In Windows** using Microsoft Visual Studio, by executing from a terminal:
  
  ```
  cl.exe example.c /I<hdfql_include_directory> <hdfql_lib_directory>/HDFql.lib /link /LTCG /NODEFAULTLIB:libcmt.lib
  ```

- **In Linux and macOS** using Gnu Compiler Collection (GCC), by executing from a terminal:
  
  - With an HDFql non MPI-based distribution:
    
    ```
    gcc example.c -fopenmp -I<hdfql_include_directory> <hdfql_lib_directory>/libHDFql.a -lm -ldl
    ```
  
  - With an HDFql MPI-based distribution:
    
    ```
    gcc example.c -fopenmp -I<hdfql_include_directory> <hdfql_lib_directory>/libHDFql.a -L<mpi_lib_directory> -lmpi -lm -ldl
    ```

2 When compiling a program against the HDFql C static library in Windows, the functions “hdfql_initialize” and “hdfql_finalize” must be explicitly called by the program when starting and finishing respectively (otherwise an error may occur such as a segmentation fault). Of note, these functions do not need to be called when compiling the program against the HDFql C shared library as this is automatically done by the library itself.
To compile the same program against the HDFql C shared library:

- In Windows using Microsoft Visual Studio, by executing from a terminal:

  cl.exe example.c /I<hdfql_include_directory> <hdfql_lib_directory>\HDFql_dll.lib

- In Linux and macOS using GCC, by executing from a terminal:
  
  - With an HDFql non MPI-based distribution:

    gcc example.c -I<hdfql_include_directory> -L<hdfql_lib_directory> -lHDFql -lm -ldl

  - With an HDFql MPI-based distribution:

    gcc example.c -I<hdfql_include_directory> -L<hdfql_lib_directory> -L<mpi_lib_directory> -lHDFql -lmpi -lm -ldl

  Of note, debug versions of the HDFql C static and shared libraries are also available. These are stored in the directory “debug” found under the directory “lib”. To compile C programs using debug libraries, the instructions described in the above bullet points should be followed with two modifications: (1) the directory storing the libraries should be updated (“<hdfql_lib_directory>\debug” in Microsoft Visual Studio; “<hdfql_lib_directory>/debug” in GCC); (2) the suffix “D” should be added to the name of the libraries (“HDFqlD.lib” and “HDFql_dllD.lib” in Microsoft Visual Studio; “libHDFqlID.a” and “-lHDFqlID” in GCC).

  In case the program does not compile, most likely a C compiler is not installed. If a C compiler is missing, the solution is:

  - In Windows, download and install a free version of Microsoft Visual Studio from the website https://www.visualstudio.com/downloads.

  - In Linux, install the GCC C compiler by executing from a terminal:

    - In a Red Hat-based distribution:

      sudo yum install gcc
• In a Debian-based distribution:

```
sudo apt-get install gcc
```

• In macOS, install the GCC C compiler by executing from a terminal (if xcode-select does not support the parameter “--install” (due to being outdated), download and install the Command-Line Tools package from the website http://developer.apple.com/downloads which includes GCC instead):

```
xcode-select --install
```

In case the compiled program does not launch, most likely the HDFql C shared library and/or the MPI shared library was not found (these are needed to launch the program). The solution is:

• In Windows, add the directory where the file “HDFql_dll.dll” is located to the environment variable “PATH” by executing from a terminal:

```
set PATH=<hdfql_lib_directory>%PATH%
```

• In Linux, add the directories where the files “libHDFql.so” and (optionally) “libmpi.so” are located to the environment variable “LD_LIBRARY_PATH” by executing from a terminal:

  • With an HDFql non MPI-based distribution:

```
export LD_LIBRARY_PATH=<hdfql_lib_directory>:$LD_LIBRARY_PATH
```

  • With an HDFql MPI-based distribution:

```
export LD_LIBRARY_PATH=<hdfql_lib_directory>:<mpi_lib_directory>:$LD_LIBRARY_PATH
```
• In macOS, add the directories where the files “libHDFql.dylib” and (optionally) “libmpi.dylib” are located to the environment variable “DYLD_LIBRARY_PATH” by executing from a terminal:

  - With an HDFql non MPI-based distribution:

    ```
    export DYLD_LIBRARY_PATH=<hdfql_lib_directory>:$DYLD_LIBRARY_PATH
    ```

  - With an HDFql MPI-based distribution:

    ```
    export
    DYLD_LIBRARY_PATH=<hdfql_lib_directory>:<mpi_lib_directory>:DYLD_LIBRARY_PATH
    ```

### 3.2 C++

HDFql can be used in the C++ programming language through static and shared libraries. These libraries are stored in the directory “cpp” found under the directory “wrapper”. The following short program illustrates how HDFql can be used in such language.

```c++
#include HDFql C++ header file (make sure it can be found by the C++ compiler)
#include <iostream>
#include "HDFql.hpp"

int main(int argc, char *argv[])
{
    // display HDFql version in use
    std::cout << "HDFql version: " << HDFql::Version << std::endl;

    // create an HDF5 file named "my_file.h5"
    HDFql::execute("CREATE FILE my_file.h5");

    // use (i.e. open) HDF5 file "my_file.h5"
    HDFql::execute("USE FILE my_file.h5");
}
```

---

3 Starting from version 10.11 (i.e. El Capitan), Apple introduced a security feature named System Integrity Protection (SIP) which may prevent setting the environment variable “DYLD_LIBRARY_PATH” and, ultimately, launching the program. To overcome this, SIP should be disabled (please refer to https://developer.apple.com/library/archive/documentation/Security/Conceptual/System_Integrity_Protection_Guide/ConfiguringSystemIntegrityProtection/ConfiguringSystemIntegrityProtection.html for additional information).
// create an HDF5 dataset named "my_dataset" of data type int
HDFql::execute("CREATE DATASET my_dataset AS INT VALUES(10)");

// select (i.e. read) data from dataset "my_dataset" and populate cursor with it
HDFql::execute("SELECT FROM my_dataset");

// move cursor to the first position within the result set
HDFql::cursorFirst();

// display content of cursor
std::cout << "Dataset value: " << *HDFql::cursorGetInt() << std::endl;

return EXIT_SUCCESS;
}
To compile the same program against the HDFql C++ shared library:

- **In Windows using Microsoft Visual Studio**, by executing from a terminal:

  ```bash
click example.cpp /EHsc /I<hdfql_include_directory> <hdfql_cpp_wrapper_directory>/HDFql_dll.lib
  ```

- **In Linux and macOS using GCC**, by executing from a terminal:
  
  - **With an HDFql non MPI-based distribution:**

    ```bash
g++ example.cpp -I<hdfql_include_directory> -L<hdfql_cpp_wrapper_directory> -lHDFql -ldl
  ```
  
  - **With an HDFql MPI-based distribution:**

    ```bash
g++ example.cpp -I<hdfql_include_directory> -L<hdfql_cpp_wrapper_directory> -L<mpi_lib_directory> -lHDFql -lmpi -ldl
  ```

In case the program does not compile, most likely a C++ compiler is not installed. If a C++ compiler is missing, the solution is:


- **In Linux**, install the GCC C++ compiler by executing from a terminal:

  - **In a Red Hat-based distribution:**

    ```bash
    sudo yum install gcc-c++
    ```

  - **In a Debian-based distribution:**
In macOS, install the GCC C++ compiler by executing from a terminal (if xcode-select does not support the parameter “--install” (due to being outdated), download and install the Command-Line Tools package from the website http://developer.apple.com/downloads which includes GCC instead):

```
sudo apt-get install g++
```

```
xcode-select --install
```

In case the compiled program does not launch, most likely the HDFql C++ shared library and/or the MPI shared library was not found (these are needed to launch the program). The solution is:

- In Windows, add the directory where the file “HDFql_dll.dll” is located to the environment variable “PATH” by executing from a terminal:

```
set PATH=<hdfql_cpp_wrapper_directory>;%PATH%
```

- In Linux, add the directories where the files “libHDFql.so” and (optionally) “libmpi.so” are located to the environment variable “LD_LIBRARY_PATH” by executing from a terminal:

  - With an HDFql non MPI-based distribution:

```
export LD_LIBRARY_PATH=<hdfql_cpp_wrapper_directory>:LD_LIBRARY_PATH
```

  - With an HDFql MPI-based distribution:

```
LD_LIBRARY_PATH=<hdfql_cpp_wrapper_directory>:<mpi_lib_directory>:LD_LIBRARY_PATH
```
In macOS, add the directories where the files “libHDFql.dylib” and (optionally) “libmpi.dylib” are located to the environment variable “DYLD_LIBRARY_PATH” by executing from a terminal:

- With an HDFql non MPI-based distribution:

```bash
export DYLD_LIBRARY_PATH=<hdfql_cpp_wrapper_directory>:$DYLD_LIBRARY_PATH
```

- With an HDFql MPI-based distribution:

```bash
export DYLD_LIBRARY_PATH=<hdfql_cpp_wrapper_directory>:<mpi_lib_directory>:$DYLD_LIBRARY_PATH
```

### 3.3 JAVA

HDFql can be used in the Java programming language through a wrapper named “HDFql.java”. This wrapper is stored in the directory “java” found under the directory “wrapper”. The following short program illustrates how HDFql can be used in such language.

```java
// import HDFql package (make sure it can be found by the Java compiler/JVM)
import as.hdfql.*;

public class Example {
    public static void main(String args[])
    {
        // display HDFql version in use
        System.out.println("HDFql version: "+HDFql.VERSION);

        // create an HDF5 file named "my_file.h5"
        HDFql.execute("CREATE FILE my_file.h5");

        // use (i.e. open) HDF5 file "my_file.h5"
    }
}
```

Starting from version 10.11 (i.e. El Capitan), Apple introduced a security feature named System Integrity Protection (SIP) which may prevent setting the environment variable “DYLD_LIBRARY_PATH” and, ultimately, launching the program. To overcome this, SIP should be disabled (please refer to https://developer.apple.com/library/archive/documentation/Security/Conceptual/System_Integrity_Protection_Guide/ConfiguringSystemIntegrityProtection.html for additional information).
HDFql.execute("USE FILE my_file.h5");

// create an HDF5 dataset named "my_dataset" of data type int
HDFql.execute("CREATE DATASET my_dataset AS INT VALUES(10)");

// select (i.e. read) data from dataset "my_dataset" and populate cursor with it
HDFql.execute("SELECT FROM my_dataset");

// move cursor to the first position within the result set
HDFql.cursorFirst();

// display content of cursor
System.out.println("Dataset value: "+HDFql.cursorGetInt());
}
}

Assuming that the program is stored in a file named “Example.java”, it must first be compiled before it can be launched from a terminal. The program can be compiled as follows:

```
javac -classpath <hdfql_java_wrapper_directory> Example.java
```

In case the program does not compile, most likely the Java Development Kit (JDK) is not installed. If the JDK is missing, the solution is to download and install it from the website http://www.oracle.com/technetwork/java/javase/downloads.

The compiled program may be launched as follows:

```
java Example
```

In case the compiled program does not launch, most likely the HDFql Java wrapper and/or the MPI shared library was not found (these are needed to launch the program). The solution is:

- In Windows, add the directories where the files “HDFql.java” (i.e. the wrapper) and “HDFql.dll” are located to the environment variables “CLASSPATH” and “PATH” by executing from a terminal:

```
set CLASSPATH=<hdfql_java_wrapper_directory>;.;%CLASSPATH%
set PATH=<hdfql_java_wrapper_directory>\as\hdfql;%PATH%
```
In Linux, add the directories where the files “HDFql.java”, “libHDFql.so” and (optionally) “libmpi.so” are located to the environment variables “CLASSPATH” and “LD_LIBRARY_PATH” by executing from a terminal:

- With an HDFql non MPI-based distribution:

  ```
  export CLASSPATH=<hdfql_java_wrapper_directory>:.:$CLASSPATH
  export LD_LIBRARY_PATH=<hdfql_java_wrapper_directory>/as/hdfql:$LD_LIBRARY_PATH
  ```

- With an HDFql MPI-based distribution:

  ```
  export CLASSPATH=<hdfql_java_wrapper_directory>:.:$CLASSPATH
  export LD_LIBRARY_PATH=<hdfql_java_wrapper_directory>/as/hdfql:$LD_LIBRARY_PATH
  ```

In macOS, add the directories where the files “HDFql.java”, “libHDFql.dylib” and (optionally) “libmpi.dylib” are located to the environment variables “CLASSPATH” and “DYLD_LIBRARY_PATH” by executing from a terminal:

- With an HDFql non MPI-based distribution:

  ```
  export CLASSPATH=<hdfql_java_wrapper_directory>:.:$CLASSPATH
  export DYLD_LIBRARY_PATH=<hdfql_java_wrapper_directory>/as/hdfql:$DYLD_LIBRARY_PATH
  ```

- With an HDFql MPI-based distribution:

  ```
  export CLASSPATH=<hdfql_java_wrapper_directory>:.:$CLASSPATH
  export DYLD_LIBRARY_PATH=<hdfql_java_wrapper_directory>/as/hdfql:$DYLD_LIBRARY_PATH
  ```

Starting from version 10.11 (i.e. El Capitan), Apple introduced a security feature named System Integrity Protection (SIP) which may prevent setting the environment variable “DYLD_LIBRARY_PATH” and, ultimately, launching the program. To overcome this, SIP should be disabled (please refer to https://developer.apple.com/library/archive/documentation/Security/Conceptual/System_Integrity_Protection_Guide/ConfiguringSystemIntegrityProtection.html for additional information). Alternatively, the Java library path property “java.library.path” should be set with the path where the HDFql shared library “libHDFql.dylib” is located when launching the program (e.g. `java -Djava.library.path=<hdfql_java_wrapper_directory>/as/hdfql my_program`).
3.4 PYTHON

HDFql can be used in the Python programming language through a wrapper named “HDFql.py”. This wrapper is stored in the directory “python” found under the directory “wrapper”. The following short script illustrates how HDFql can be used in such language.

```python
# import HDFql module (make sure it can be found by the Python interpreter)
import HDFql

# display HDFql version in use
print("HDFql version: %s" % HDFql.VERSION)

# create an HDF5 file named "my_file.h5"
HDFql.execute("CREATE FILE my_file.h5")

# use (i.e. open) HDF5 file "my_file.h5"
HDFql.execute("USE FILE my_file.h5")

# create an HDF5 dataset named "my_dataset" of data type int
HDFql.execute("CREATE DATASET my_dataset AS INT VALUES(10)")

# select (i.e. read) data from dataset "my_dataset" and populate cursor with it
HDFql.execute("SELECT FROM my_dataset")

# move cursor to the first position within the result set
HDFql.cursor_first()

# display content of cursor
print("Dataset value: %d" % HDFql.cursor_get_int())
```

Assuming that the script is stored in a file named “example.py” it can be launched by executing the following from a terminal:

```
python example.py
```

In case the script does not launch, most likely (1) the Python interpreter is not installed or (2) the HDFql Python wrapper and/or the MPI shared library was not found (these are needed to launch the script). To fix the former issue, download and install the Python interpreter from the website http://www.python.org/download. To fix the latter issue:
• In Windows, add the directory where the file “HDFql.py” (i.e. the wrapper) is located to the environment variable “PYTHONPATH” by executing from a terminal:

```bash
set PYTHONPATH=<hdfql_python_wrapper_directory>;%PYTHONPATH%
```

• In Linux, add the directories where the files “HDFql.py” and (optionally) “libmpi.so” are located to the environment variables “PYTHONPATH” and “LD_LIBRARY_PATH” by executing from a terminal:

  • With an HDFql non MPI-based distribution:

```bash
export PYTHONPATH=<hdfql_python_wrapper_directory>:$PYTHONPATH
```

  • With an HDFql MPI-based distribution:

```bash
export PYTHONPATH=<hdfql_python_wrapper_directory>:$PYTHONPATH
echo LD_LIBRARY_PATH=
export LD_LIBRARY_PATH=<mpi_lib_directory>:$LD_LIBRARY_PATH
```

• In macOS, add the directories where the files “HDFql.py” and (optionally) “libmpi.dylib” are located to the environment variables “PYTHONPATH” and “DYLD_LIBRARY_PATH” by executing from a terminal:

  • With an HDFql non MPI-based distribution:

```bash
export PYTHONPATH=<hdfql_python_wrapper_directory>:$PYTHONPATH
```

  • With an HDFql MPI-based distribution:

```bash
export PYTHONPATH=<hdfql_python_wrapper_directory>:$PYTHONPATH
echo DYLD_LIBRARY_PATH=
export DYLD_LIBRARY_PATH=<mpi_lib_directory>:$DYLD_LIBRARY_PATH
```

Starting from version 10.11 (i.e. El Capitan), Apple introduced a security feature named System Integrity Protection (SIP) which may prevent setting the environment variable “DYLD_LIBRARY_PATH” and, ultimately, launching the program. To overcome this, SIP should be disabled (please refer to https://developer.apple.com/library/archive/documentation/Security/Conceptual/System_Integrity_Protection_Guide/ConfiguringSystemIntegrityProtection/ConfiguringSystemIntegrityProtection.html for additional information).
Besides these steps, a scientific computing package named NumPy for Python must be installed when working with user-defined variables (please refer to the function `hdfql_variable_register` for additional information). This package can be found at [http://www.scipy.org/scipylib/download.html](http://www.scipy.org/scipylib/download.html) along with instructions on how to install and use it.

### 3.5 C#

HDFql can be used in the C# programming language through a wrapper named “HDFql.cs”. This wrapper is stored in the directory “csharp” found under the directory “wrapper”. The following short program illustrates how HDFql can be used in such language.

```csharp
// use HDFql namespace (make sure it can be found by the C# compiler)
using AS.HDFql;

public class Example
{
    public static void Main(string [] args)
    {
        // display HDFql version in use
        System.Console.WriteLine("HDFql version: {0}", HDFql.Version);

        // create an HDF5 file named "my_file.h5"
        HDFql.Execute("CREATE FILE my_file.h5");

        // use (i.e. open) HDF5 file "my_file.h5"
        HDFql.Execute("USE FILE my_file.h5");

        // create an HDF5 dataset named "my_dataset" of data type int
        HDFql.Execute("CREATE DATASET my_dataset AS INT VALUES(10)");

        // select (i.e. read) data from dataset "my_dataset" and populate cursor with it
        HDFql.Execute("SELECT FROM my_dataset");

        // move cursor to the first position within the result set
        HDFql.CursorFirst();

        // display content of cursor
        System.Console.WriteLine("Dataset value: {0}", HDFql.CursorGetInt());
    }
}
```
Assuming that the program is stored in a file named “Example.cs”, it must first be compiled before it can be launched from a terminal. In Windows, the program can be compiled as follows:

- Using Microsoft .NET Framework, by executing from a terminal:

  ```
csc.exe Example.cs <hdfql_csharp_wrapper_directory>/*.cs
  ```

- Using Mono, by executing from a terminal:

  ```
mcs.bat Example.cs <hdfql_csharp_wrapper_directory>/*.cs
  ```

In Linux and macOS, the program can be compiled using Mono by executing from a terminal (of note, Microsoft .NET Framework does not support these platforms):

```
mcs Example.cs <hdfql_csharp_wrapper_directory>/*.cs
``` 

In case the program does not compile, most likely a C# compiler is not installed. If a C# compiler is missing, the solution is:

- In Windows, download and install either Microsoft .NET Framework or Mono from the websites  
  https://www.microsoft.com/net/download/framework or http://www.mono-project.com/download, respectively.

- In Linux and macOS, download and install Mono from the website http://www.mono-project.com/download.

Depending on the platform, the compiled program may be launched as follows:

- In Windows, by executing from a terminal:

  ```
  Example.exe
  ```

- In Linux and macOS, by executing from a terminal:

  ```
  mono Example.exe
  ```
In case the compiled program does not launch, most likely the HDFql C# wrapper and/or the MPI shared library was not found (these are needed to launch the program). The solution is:

- In Windows, add the directory where the file “HDFql.cs” (i.e. the wrapper) is located to the environment variable “PATH” by executing from a terminal:

  ```
  set PATH=<hdfql_csharp_wrapper_directory>;%PATH%
  ```

- In Linux, add the directories where the files “HDFql.cs” and (optionally) “libmpi.so” are located to the environment variable “LD_LIBRARY_PATH” by executing from a terminal:

  - With an HDFql non MPI-based distribution:

    ```
    export LD_LIBRARY_PATH=<hdfql_csharp_wrapper_directory>:$LD_LIBRARY_PATH
    ```

  - With an HDFql MPI-based distribution:

    ```
    export
    LD_LIBRARY_PATH=<hdfql_csharp_wrapper_directory>:<mpi_lib_directory>:$LD_LIBRARY_PATH
    ```

- In macOS, add the directories where the files “HDFql.cs” and (optionally) “libmpi.dylib” are located to the environment variable “DYLD_LIBRARY_PATH” by executing from a terminal:

  - With an HDFql non MPI-based distribution:

    ```
    export DYLD_LIBRARY_PATH=<hdfql_csharp_wrapper_directory>:$DYLD_LIBRARY_PATH
    ```

  - With an HDFql MPI-based distribution:

    ```
    ```

---

8 Starting from version 10.11 (i.e. El Capitan), Apple introduced a security feature named System Integrity Protection (SIP) which may prevent setting the environment variable “DYLD_LIBRARY_PATH” and, ultimately, launching the program. To overcome this, SIP should be disabled (please refer to https://developer.apple.com/library/archive/documentation/Security/Conceptual/System_Integrity_Protection_Guide/ConfiguringSystemIntegrityProtection/ConfiguringSystemIntegrityProtection.html for additional information).
export
DYLD_LIBRARY_PATH=<hdfql_csharp_wrapper_directory>:<mpi_lib_directory>:$DYLD_LIBRARY_PATH

### 3.6 FORTRAN

HDFql can be used in the Fortran programming language through static and shared libraries. These libraries are stored in the directory “fortran” found under the directory “wrapper”. The following short program illustrates how HDFql can be used in such language.

```fortran
PROGRAM Example
  ! use HDFql module (make sure it can be found by the Fortran compiler)
  USE HDFql

  ! declare variable
  INTEGER :: state

  ! display HDFql version in use
  WRITE(*, *) "HDFql version: ", HDFQL_VERSION

  ! create an HDF5 file named "my_file.h5"
  state = hdfql_execute("CREATE FILE my_file.h5")

  ! use (i.e. open) HDF5 file "my_file.h5"
  state = hdfql_execute("USE FILE my_file.h5")

  ! create an HDF5 dataset named "my_dataset" of data type int
  state = hdfql_execute("CREATE DATASET my_dataset AS INT VALUES(10)")

  ! select (i.e. read) data from dataset "my_dataset" and populate cursor with it
  state = hdfql_execute("SELECT FROM my_dataset")

  ! move cursor to the first position within the result set
  state = hdfql_cursor_first()

  ! display content of cursor
  WRITE(*, *) "Dataset value: ", hdfql_cursor_get_int()
END PROGRAM
```
Assuming that the program is stored in a file named “example.f90”, it must first be compiled before it can be launched from a terminal. To compile the program against the HDFql Fortran static library:

- In Windows\(^9\) using Intel Fortran Compiler (IFORT), by executing from a terminal:

  ```
  ifort.exe example.f90 /module:<hdfql_fortran_wrapper_directory>\static
  <hdfql_fortran_wrapper_directory>\HDFql.lib /link /LTCG /NODEFAULTLIB:libcmt.lib
  ```

- In Linux using IFORT, by executing from a terminal:
  - With an HDFql non MPI-based distribution:

    ```
    ifort example.f90 -fopenmp -module <hdfql_fortran_wrapper_directory>
    <hdfql_fortran_wrapper_directory>/libHDFql.a
    ```

  - With an HDFql MPI-based distribution:

    ```
    ifort example.f90 -fopenmp -module <hdfql_fortran_wrapper_directory>
    <hdfql_fortran_wrapper_directory>/libHDFql.a -L<mpi_lib_directory> -lmpi
    ```

- In Linux and macOS using Gnu Compiler Collection (GCC)\(^10\), by executing from a terminal:
  - With an HDFql non MPI-based distribution:

    ```
    gfortran example.f90 -fopenmp -I<hdfql_fortran_wrapper_directory>
    <hdfql_fortran_wrapper_directory>/libHDFql.a -ldl
    ```

  - With an HDFql MPI-based distribution:

---

\(^9\) When compiling a program against the HDFql Fortran static library in Windows, the subroutines “hdfql_initialize” and “hdfql_finalize” must be explicitly called by the program when starting and finishing respectively (otherwise an error may occur such as a segmentation fault). Of note, these functions do not need to be called when compiling the program against the HDFql Fortran shared library as this is automatically done by the library itself.

\(^10\) An incorrect warning is raised by the GCC Fortran compiler when using the HDFql module (“Warning: Only array FINAL procedures declared for derived type 'hdfql_cursor' defined at (1), suggest also scalar one”). This warning does not interfere with the final compilation result, though, and it has been solved in the GCC Fortran compiler version 7.0.0 (please refer to https://gcc.gnu.org/bugzilla/show_bug.cgi?id=58175 for additional information).
To compile the same program against the HDFql Fortran shared library:

- In Windows using IFORT, by executing from a terminal:

  ```
  ifort.exe example.f90 /module:<hdfql_fortran_wrapper_directory> \
  <hdfql_fortran_wrapper_directory>\HDFql_dll.lib
  ```

- In Linux using IFORT, by executing from a terminal:
  - With an HDFql non MPI-based distribution:
    ```
    ifort example.f90 -module <hdfql_fortran_wrapper_directory> -L<hdfql_fortran_wrapper_directory> -lHDFql
    ```
  - With an HDFql MPI-based distribution:
    ```
    ifort example.f90 -module <hdfql_fortran_wrapper_directory> -L<hdfql_fortran_wrapper_directory> -L<mpi_lib_directory> -lHDFql -lmpi
    ```

- In Linux and macOS using GCC\(^{11}\), by executing from a terminal:
  - With an HDFql non MPI-based distribution:
    ```
    gfortran example.f90 -I<hdfql_fortran_wrapper_directory> -L<hdfql_fortran_wrapper_directory> -lHDFql -ldl
    ```
  - With an HDFql MPI-based distribution:

\(^{11}\) An incorrect warning is raised by the GCC Fortran compiler when using the HDFql module ("Warning: Only array FINAL procedures declared for derived type 'hdfql_cursor' defined at (1), suggest also scalar one"). This warning does not interfere with the final compilation result, though, and it has been solved in the GCC Fortran compiler version 7.0.0 (please refer to https://gcc.gnu.org/bugzilla/show_bug.cgi?id=58175 for additional information).
In case the program does not compile, most likely a Fortran compiler is not installed. If a Fortran compiler is missing, the solution is:


- In Linux, install the GCC Fortran compiler by executing from a terminal:
  
  - In a Red Hat-based distribution:
    
    ```
    sudo yum install gcc-gfortran
    ```

  - In a Debian-based distribution:
    
    ```
    sudo apt-get install gfortran
    ```

  - In macOS, install the GCC Fortran compiler by executing from a terminal (if `xcode-select` does not support the parameter “--install” (due to being outdated), download and install the Command-Line Tools package from the website [http://developer.apple.com/downloads](http://developer.apple.com/downloads) which includes GCC instead):
    
    ```
    xcode-select --install
    ```

In case the compiled program does not launch, most likely the HDFql Fortran shared library and/or the MPI shared library was not found (these are needed to launch the program). The solution is:

- In Windows, add the directory where the file “HDFql_dll.dll” is located to the environment variable “PATH” by executing from a terminal:
In Linux, add the directories where the files “libHDFql.so” and (optionally) “libmpi.so” are located to the environment variable “LD_LIBRARY_PATH” by executing from a terminal:

- With an HDFql non MPI-based distribution:

```
set PATH=<hdfql_fortran_wrapper_directory>;%PATH%
```

- With an HDFql MPI-based distribution:

```
export LD_LIBRARY_PATH=<hdfql_fortran_wrapper_directory>:$LD_LIBRARY_PATH
```

In macOS, add the directories where the files “libHDFql.dylib” and (optionally) “libmpi.dylib” are located to the environment variable “DYLD_LIBRARY_PATH” by executing from a terminal:

- With an HDFql non MPI-based distribution:

```
export DYLD_LIBRARY_PATH=<hdfql_fortran_wrapper_directory>:$DYLD_LIBRARY_PATH
```

- With an HDFql MPI-based distribution:

```
export
DYLD_LIBRARY_PATH=<hdfql_fortran_wrapper_directory>:<mpi_lib_directory>:$$DYLD_LIBRARY_PATH
```

Starting from version 10.11 (i.e. El Capitan), Apple introduced a security feature named System Integrity Protection (SIP) which may prevent setting the environment variable “DYLD_LIBRARY_PATH” and, ultimately, launching the program. To overcome this, SIP should be disabled (please refer to https://developer.apple.com/library/archive/documentation/Security/Conceptual/System_Integrity_Protection_Guide/ConfiguringSystemIntegrityProtection.html for additional information).
3.7 R

HDFql can be used in the R programming language through a wrapper named “HDFql.R”. This wrapper is stored in the directory “R” found under the directory “wrapper”. The following short script illustrates how HDFql can be used in such language.

```r
# load HDFql R wrapper (make sure it can be found by the R interpreter)
source("HDFql.R")

# display HDFql version in use
print(paste("HDFql version:", HDFQL_VERSION))

# create an HDF5 file named "my_file.h5"
hdfql_execute("CREATE FILE my_file.h5")

# use (i.e. open) HDF5 file "my_file.h5"
hdfql_execute("USE FILE my_file.h5")

# create an HDF5 dataset named "my_dataset" of data type int
hdfql_execute("CREATE DATASET my_dataset AS INT VALUES(10)")

# select (i.e. read) data from dataset "my_dataset" and populate cursor with it
hdfql_execute("SELECT FROM my_dataset")

# move cursor to the first position within the result set
hdfql_cursor_first()

# display content of cursor
print(paste("Dataset value:", hdfql_cursor_get_int()))
```

Assuming that the script is stored in a file named “example.R” it can be launched by executing the following from a terminal:

```
R -f example.R
```

In case the script does not launch, most likely (1) the R interpreter is not installed or (2) the HDFql R wrapper and/or the HDFql C shared library and/or the MPI shared library was not found (these are needed to launch the script). To fix the former issue, download and install the R interpreter from the website https://cloud.r-project.org. To fix the latter issue:
• In Windows, add the directories where the files “HDFql.R” (i.e. the wrapper) and “HDFql_dll.dll” are located to the environment variable “PATH” by executing from a terminal:

```
set PATH=<hdfql_r_wrapper_directory>;<hdfql_lib_directory>;%PATH%
```

• In Linux, add the directories where the files “HDFql.R”, “libHDFql.so” and (optionally) “libmpi.so” are located to the environment variable “LD_LIBRARY_PATH” by executing from a terminal:

- With an HDFql non MPI-based distribution:

```
export
LD_LIBRARY_PATH=<hdfql_r_wrapper_directory>:<hdfql_lib_directory>:LD_LIBRARY_PATH
```

- With an HDFql MPI-based distribution:

```
export
LD_LIBRARY_PATH=<hdfql_r_wrapper_directory>:<hdfql_lib_directory>:<mpi_lib_directory>:LD_LIBRARY_PATH
```

• In macOS, add the directories where the files “HDFql.R”, “libHDFql.dylib” and (optionally) “libmpi.dylib” are located to the environment variable “DYLD_LIBRARY_PATH” by executing from a terminal:

- With an HDFql non MPI-based distribution:

```
export
DYLD_LIBRARY_PATH=<hdfql_r_wrapper_directory>:<hdfql_lib_directory>:DYLD_LIBRARY_PATH
```

- With an HDFql MPI-based distribution:

---

13 Starting from version 10.11 (i.e. El Capitan), Apple introduced a security feature named System Integrity Protection (SIP) which may prevent setting the environment variable “DYLD_LIBRARY_PATH” and, ultimately, launching the program. To overcome this, SIP should be disabled (please refer to [https://developer.apple.com/library/archive/documentation/Security/Conceptual/System_Integrity_Protection_Guide/ConfiguringSystemIntegrityProtection/ConfiguringSystemIntegrityProtection.html](https://developer.apple.com/library/archive/documentation/Security/Conceptual/System_Integrity_Protection_Guide/ConfiguringSystemIntegrityProtection/ConfiguringSystemIntegrityProtection.html) for additional information).
Besides these steps, a package named bit64 for R must be installed when working with user-defined variables to store 64-bit integers as these are not natively supported by R (please refer to the function `hdfql_variable_register` for additional information). This package can be found at https://cran.r-project.org/web/packages/bit64 along with instructions on how to install and use it.

### 3.8 COMMAND-LINE INTERFACE

A command-line interface named “HDFqlCLI” is available and can be used for manipulating HDF5 files from a terminal. It is stored in the directory “bin”. To launch the command-line interface, open a terminal (“cmd” if in Windows, “xterm” if in Linux, or “Terminal” if in macOS), go to the directory “bin”, and type:

- In Windows:
  
  `HDFqlCLI.exe`

- In Linux and macOS:
  
  `./HDFqlCLI`

The list of parameters accepted by the command-line interface can be viewed by launching it with the parameter “--help”. At the time of writing, this list includes the following parameters:

- `--help` (show the list of parameters accepted by HDFqlCLI and exit)
- `--version` (show the version of HDFqlCLI and exit)
- `--debug` (show debug information when executing HDFql operations)
- `--no-path` (do not show group path currently in use in HDFqlCLI prompt)
- `--execute=X` (execute HDFql operation(s) “X” and exit)
- **execute-file=X** (execute HDFql operation(s) stored in file “X” and exit)
- **save-file=X** (save executed HDFql operation(s) to file “X”)  

In case the command-line interface does not launch, most likely the HDFql shared library (which is needed to launch the interface) was not found. Depending on the platform, the solution is:

- In Windows, to either:
  - Add the directory where the file “HDFql_dll.dll” is located to the environment variable “PATH” by executing from a terminal:
    ```
    set PATH=<hdfql_lib_directory>;%PATH%
    ```
  - Execute the batch file named “launch.bat” which properly sets up the environment variable “PATH” and launches the command-line interface from a terminal.

- In Linux, to either:
  - Add the directory where the file “libHDFql.so” is located to the environment variable “LD_LIBRARY_PATH” by executing from a terminal:
    ```
    export LD_LIBRARY_PATH=<hdfql_lib_directory>:LD_LIBRARY_PATH
    ```
  - Execute the bash script file named “launch.sh” which properly sets up the environment variable “LD_LIBRARY_PATH” and launches the command-line interface from a terminal.

- In macOS, to either:
  - Add the directory where the file “libHDFql.dylib” is located to the environment variable “DYLD_LIBRARY_PATH” by executing from a terminal:

14 Starting from version 10.11 (i.e. El Capitan), Apple introduced a security feature named System Integrity Protection (SIP) which may prevent setting the environment variable “DYLD_LIBRARY_PATH” and, ultimately, launching the program. To overcome this, SIP should be disabled (please refer to https://developer.apple.com/library/archive/documentation/Security/Conceptual/System_Integrity_Protection_Guide/ConfiguringSystemIntegrityProtection/ConfiguringSystemIntegrityProtection.html for additional information).
- Execute the bash script file named “launch.sh” which properly sets up the environment variable “DYLD_LIBRARY_PATH” and launches the command-line interface from a terminal.

![HDFqlCLI](image_url)

Figure 3.1 – Illustration of the command-line interface “HDFqlCLI”
4. CURSOR

Generally speaking, a cursor is a control structure that is used to iterate through the results returned by a query (that was previously executed). It can be seen as an effective means to abstract the programmer from low-level implementation details of accessing data stored in specific structures. This chapter provides a description of cursors and subcursors in HDFql, as well as examples and illustrations to demonstrate these two concepts in practice.

4.1 DESCRIPTION

HDFql provides cursors which offer several ways to traverse result sets according to specific needs. The following list enumerates these functionalities (please refer to their links for further information):

- First (moves cursor to the first position within the result set – `hdfql_cursor_first`)
- Last (moves cursor to the last position within the result set – `hdfql_cursor_last`)
- Next (moves cursor to the next position within the result set – `hdfql_cursor_next`)
- Previous (moves cursor to the previous position within the result set – `hdfql_cursor_previous`)
- Absolute (moves cursor to an absolute position within the result set – `hdfql_cursor_absolute`)
- Relative (moves cursor to a relative position within the result set – `hdfql_cursor_relative`)

Besides their traversal functionalities, a particular feature of cursors in HDFql is that they store result sets returned by DATA QUERY LANGUAGE (DQL) and DATA INTROSPECTION LANGUAGE (DIL) operations. To retrieve values from result sets, the functions starting with “hdfql_cursor_get” can be used. These and remaining functions offered by cursors can be found in Table 5.8 (each of these begins with the prefix “hdfql_cursor”).

When a certain operation is executed, HDFql stores the result set returned by this operation in its default cursor. This cursor is available to the programmer and is automatically created and initialized upon loading the HDFql library by a program. If additional cursors are needed, they can be created like this (in C):
// create a cursor named "my_cursor"
HDFQL_CURSOR my_cursor;

As a side note, additional cursors are created in C++, Java, Python, C#, Fortran and R as follows:

// create a cursor named "myCursor" in C++
HDFql::Cursor myCursor;

// create a cursor named "myCursor" in Java
HDFqlCursor myCursor = new HDFqlCursor();

# create a cursor named "my_cursor" in Python
my_cursor = HDFql.Cursor()

// create a cursor named "myCursor" in C#
HDFqlCursor myCursor = new HDFqlCursor();

! create a cursor named "my_cursor" in Fortran
TYPE(HDFQL_CURSOR) :: my_cursor

# create a cursor named "my_cursor" in R
my_cursor <- hdfql_cursor()

Before an additional (i.e. user-defined) cursor is used to store and eventually traverse a result set, it must be properly initialized (refer to the function hdfql_cursor_initialize for further information). The initialization of a cursor is only required in C and performed once, while in C++, Java, Python, C#, Fortran and R such initialization is redundant as it is done automatically when creating a cursor. Initializing a cursor can be done like this (in C):

// initialize a cursor named "my_cursor"
hdfql_cursor_initialize(&my_cursor);
To switch between different cursors (to be used for separate needs), the function `hdfql_cursor_use` may be employed (in C):

```c
// use a cursor named "my_cursor"
hdfql_cursor_use(&my_cursor);
```

The following C snippet illustrates usage of the HDFql default cursor and a user-defined cursor, as well as some typical operations performed on/by these.

```c
// create a cursor named "my_cursor"
HDFQL_CURSOR my_cursor;

// create an HDF5 dataset named "my_dataset0" of data type int with an initial value of 8
hdfql_execute("CREATE DATASET my_dataset0 AS INT VALUES(8)");

// create an HDF5 dataset named "my_dataset1" of data type float with initial values of 3.2, 5.3, 7.4 and 9.5
hdfql_execute("CREATE DATASET my_dataset1 AS FLOAT(4) VALUES(3.2, 5.3, 7.4, 9.5)");

// select (i.e. read) data from dataset "my_dataset0" and populate HDFql default cursor with it
hdfql_execute("SELECT FROM my_dataset0");

// initialize cursor "my_cursor"
hdfql_cursor_initialize(&my_cursor);

// use cursor "my_cursor"
hdfql_cursor_use(&my_cursor);

// select (i.e. read) data from dataset "my_dataset1" and populate cursor "my_cursor" with it
hdfql_execute("SELECT FROM my_dataset1");

// use HDFql default cursor
hdfql_cursor_use(NULL);

// display number of elements in HDFql default cursor
printf("Number of elements in HDFql default cursor is %d\n", hdfql_cursor_get_count(NULL));

// move HDFql default cursor to the next position within the result set
hdfql_cursor_next(NULL);
```
// display element of HDFql default cursor
printf("Current element of HDFql default cursor is %dn", *hdfql_cursor_get_int(NULL));

// display number of elements in cursor "my_cursor"
printf("Number of elements in cursor "my_cursor" is %dn",
    hdfql_cursor_get_count(&my_cursor));

// use cursor "my_cursor"
hdfql_cursor_use(&my_cursor);

// display elements of cursor "my_cursor"
while(hdfql_cursor_next(NULL) == HDFQL_SUCCESS)
{
    printf("Current element of cursor "my_cursor" is %fn", *hdfql_cursor_get_float(NULL));
}

The output of executing the snippet would be similar to this:

Number of elements in HDFql default cursor is 1
Current element of HDFql default cursor is 8
Number of elements in cursor "my_cursor" is 4
Current element of cursor "my_cursor" is 3.2
Current element of cursor "my_cursor" is 5.3
Current element of cursor "my_cursor" is 7.4
Current element of cursor "my_cursor" is 9.5

When populating a cursor with data from a dataset or attribute with two or more dimensions, the data is always linearized into a single dimension. The linearization process is depicted in Figure 4.1. Subsequently, if need be, it is up to the programmer to access the data (stored in the cursor) according to its original dimensions. In this case, the SHOW DIMENSION operation – which returns the original dimensions of a dataset or attribute – may be useful to help in the task of going from one dimension to the original dimensions.
4.2 SUBCURSOR

HDFql also provides subcursors – they are meant to complement (i.e. help) cursors in the task of storing data of type HDFQL_CHAR, HDFQL_VARTINYINT, HDFQL_UNSIGNED_VARTINYINT, HDFQL_VARSMALLINT, HDFQL_UNSIGNED_VARSMALLINT, HDFQL_VARINT, HDFQL_UNSIGNED_VARINT, HDFQL_VARBIGINT, HDFQL_UNSIGNED_VARBIGINT, HDFQL_VARFLOAT, HDFQL_VARDOUBLE and HDFQL_OPAQUE. In practice, when a result set is of one of these data types, only the first element of the result set is stored in the cursor (as expected), while all elements of the result set are stored in the subcursor. In other words, each position of the cursor stores the first element of the result set and also points to a subcursor that in turn stores all the elements of the result set. The values stored in a subcursor (which are also known as a result subset) can be accessed with the functions starting with “hdfql_subcursor_get” (enumerated in Table 5.8). Similar to cursors, HDFql subcursors offer several ways to traverse result subsets, namely:

- First (moves subcursor to the first position within the result subset – hdfql_subcursor_first)
- Last (moves subcursor to the last position within the result subset – hdfql_subcursor_last)
- Next (moves subcursor to the next position within the result subset – hdfql_subcursor_next)
- Previous (moves subcursor to the previous position within the result subset – hdfql_subcursor_previous)
- Absolute (moves subcursor to an absolute position within the result subset – hdfql_subcursor_absolute)
- Relative (moves subcursor to a relative position within the result subset – hdfql_subcursor_relative)
The following C snippet illustrates usage of the HDFql subcursors, as well as some typical operations performed on/by these.

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length int of one dimension
// (size 4)
hdfql_execute("CREATE DATASET my_dataset AS VARINT(4)");  

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES((7, 8, 5), (9), (6, 1, 2), (4, 0))");  

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");  

// move the cursor in use to the next position within the result set
while(hdfql_cursor_next(NULL) == HDFQL_SUCCESS)  
{  
  // display element of the cursor in use
  printf("Current element of cursor is %d\n", *hdfql_cursor_get_int(NULL));  

  // move the subcursor in use to the next position within the result subset
  while(hdfql_subcursor_next(NULL) == HDFQL_SUCCESS)  
  { 
    // display element of the subcursor in use
    printf("  Current element of subcursor is %d\n", *hdfql_subcursor_get_int(NULL));  
    
  }  
}
```

The output of executing the snippet would be similar to this:

```
Current element of cursor is 7
  Current element of subcursor is 7
  Current element of subcursor is 8
  Current element of subcursor is 5
Current element of cursor is 9
  Current element of subcursor is 9
Current element of cursor is 6
  Current element of subcursor is 6
  Current element of subcursor is 1
  Current element of subcursor is 2
Current element of cursor is 4
  Current element of subcursor is 4
  Current element of subcursor is 0
```
4.3 EXAMPLES

The following C snippets demonstrate how HDFql cursors and subcursors are populated with (variable) data stored in HDF5 datasets or attributes, along with illustrations to facilitate understanding of the populating process and its final result.

```c
// create an HDF5 dataset named "my_dataset0" of data type short
hdfql_execute("CREATE DATASET my_dataset0 AS SMALLINT");

// insert (i.e. write) a value into dataset "my_dataset0"
hdfql_execute("INSERT INTO my_dataset0 VALUES(7)");

// select (i.e. read) data from dataset "my_dataset0" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset0");
```

![Diagram](image)

Figure 4.2 – Cursor populated with data from dataset “my_dataset0”
// create an HDF5 dataset named "my_dataset1" of data type float of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset1 AS FLOAT(3)");

// insert (i.e. write) values into dataset "my_dataset1"
hdfql_execute("INSERT INTO my_dataset1 VALUES(5.5, 8.1, 4.9)");

// select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset1");

Figure 4.3 – Cursor populated with data from dataset “my_dataset1”
// create an HDF5 dataset named "my_dataset2" of data type double of two dimensions (size 3x2)
hdfql_execute("CREATE DATASET my_dataset2 AS DOUBLE(3, 2)");

// insert (i.e. write) values into dataset "my_dataset2"
hdfql_execute("INSERT INTO my_dataset2 VALUES((3.2, 1.3), (0, 0.2), (9.1, 6.5))");

// select (i.e. read) data from dataset "my_dataset2" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset2");

---

**Figure 4.4** – Cursor populated with data from dataset “my_dataset2”
// create an HDF5 dataset named "my_dataset3" of data type variable-length short
hdfql_execute("CREATE DATASET my_dataset3 AS VARS SMALLINT");

// insert (i.e. write) values into dataset "my_dataset3"
//hdfql_execute("INSERT INTO my_dataset3 VALUES(7, 9, 3)");

// select (i.e. read) data from dataset "my_dataset3" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset3");

Figure 4.5 – Cursor and its subcursor populated with data from dataset “my_dataset3”
// create an HDF5 dataset named "my_dataset4" of data type variable-length float of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset4 AS VARFLOAT(3)");

// insert (i.e. write) values into dataset "my_dataset4"
hdfql_execute("INSERT INTO my_dataset4 VALUES((5.5), (8.1, 2.2), (4.9, 3.4, 5.6))");

// select (i.e. read) data from dataset "my_dataset4" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset4");

![Diagram of dataset and cursor](image.png)

Figure 4.6 – Cursor and its subcursors populated with data from dataset “my_dataset4”
// create an HDF5 dataset named "my_dataset5" of data type variable-length double of two dimensions (size 3x2)
    hdfql_execute("CREATE DATASET my_dataset5 AS VARDOUBLE(3, 2)");

// insert (i.e. write) values into dataset "my_dataset5"
    hdfql_execute("INSERT INTO my_dataset5 VALUES(((3.2, 8, 6.7), (1.3, 0.2)), ((0), (0.2, 1.5)), ((9.1, 2, 4, 7), (6.5)))");

// select (i.e. read) data from dataset "my_dataset5" and populate cursor in use with it
    hdfql_execute("SELECT FROM my_dataset5");

### Dataset “my_dataset5”

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.2, 8, 6.7</td>
<td>1.3, 0.2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.2, 1.5</td>
</tr>
<tr>
<td>2</td>
<td>9.1, 2, 4, 7</td>
<td>6.5</td>
</tr>
</tbody>
</table>

### Cursor

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.2</td>
<td>1.3</td>
<td>0</td>
<td>0.2</td>
<td>9.1</td>
<td>6.5</td>
</tr>
</tbody>
</table>

Figure 4.7 – Cursor and its subcursors populated with data from dataset “my_dataset5”
5. APPLICATION PROGRAMMING INTERFACE

An application programming interface (API) specifies how software components should interact with each other. In practice, an API comes in the form of a library that includes specifications for functions, data structures, object classes, constants and variables. A good API makes it easier to develop a program by providing all the building blocks. This chapter is devoted to describing HDFql API and how to use it through practical examples in C, C++, Java, Python, C#, Fortran and R.

5.1 CONSTANTS

A constant is an identifier whose associated value cannot typically be altered by the program during its execution. Using a constant instead of specifying a value multiple times in the program not only simplifies code maintenance, but can also supply a meaningful name for it. Constants in the C programming languages follow a naming convention of writing all words in uppercase and separating each word with an underscore (_). The following table summarizes all existing HDFql constants in C.

<table>
<thead>
<tr>
<th>HDFql Constant in C</th>
<th>Description</th>
<th>Data Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFQL_VERSION</td>
<td>Represents the HDFql version in use</td>
<td>char *</td>
<td>2.1.0</td>
</tr>
<tr>
<td>HDFQL_YES</td>
<td>Represents the concept “Yes”</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>HDFQL_NO</td>
<td>Represents the concept “No”</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>HDFQL_ENABLED</td>
<td>Represents the concept “Enabled”</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>HDFQL_DISABLED</td>
<td>Represents the concept “Disabled”</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>HDFQL_UNLIMITED</td>
<td>Represents the concept “Unlimited”</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>HDFQL_UNDEFINED</td>
<td>Represents the concept “Undefined”</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>HDFQL_GLOBAL</td>
<td>Represents the concept “Global”</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>HDFQL_LOCAL</td>
<td>Represents the concept “Local”</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>HDFQL_TRACKED</td>
<td>Represents the HDF5 tracked creation order</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>Strategy</td>
<td>Description</td>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>HDFQL_INDEXED</td>
<td>Represents the HDF5 indexed creation order strategy</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>HDFQL_CONTIGUOUS</td>
<td>Represents the HDF5 contiguous storage type (layout)</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>HDFQL_COMPACT</td>
<td>Represents the HDF5 compact storage type (layout)</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>HDFQL_CHUNKED</td>
<td>Represents the HDF5 chunked storage type (layout)</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>HDFQL_EARLY</td>
<td>Represents the HDF5 early storage allocation</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>HDFQL_INCREMENTAL</td>
<td>Represents the HDF5 incremental storage allocation</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>HDFQL_LATE</td>
<td>Represents the HDF5 late storage allocation</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>HDFQL_DIRECTORY</td>
<td>Represents a directory</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>HDFQL_FILE</td>
<td>Represents a file</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>HDFQL_GROUP</td>
<td>Represents the HDF5 group object type</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>HDFQL_DATASET</td>
<td>Represents the HDF5 dataset object type</td>
<td>int</td>
<td>8</td>
</tr>
<tr>
<td>HDFQL_ATTRIBUTE</td>
<td>Represents the HDF5 attribute object type</td>
<td>int</td>
<td>16</td>
</tr>
<tr>
<td>HDFQL_SOFT_LINK</td>
<td>Represents the HDF5 soft link object type</td>
<td>int</td>
<td>32</td>
</tr>
<tr>
<td>HDFQL_EXTERNAL_LINK</td>
<td>Represents the HDF5 external link object type</td>
<td>int</td>
<td>64</td>
</tr>
<tr>
<td>HDFQL_TINYINT</td>
<td>Represents the tiny integer data type (TINYINT)</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_TINYINT</td>
<td>Represents the unsigned tiny integer data type (UNSIGNED TINYINT)</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>HDFQL_SMALLINT</td>
<td>Represents the small integer data type (SMALLINT)</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_SMALLINT</td>
<td>Represents the unsigned small integer data type (UNSIGNED SMALLINT)</td>
<td>int</td>
<td>8</td>
</tr>
<tr>
<td>HDFQL_INT</td>
<td>Represents the integer data type (INT)</td>
<td>int</td>
<td>16</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_INT</td>
<td>Represents the unsigned integer data type (UNSIGNED INT)</td>
<td>int</td>
<td>32</td>
</tr>
<tr>
<td>HDFQL_BIGINT</td>
<td>Represents the big integer data type (BIGINT)</td>
<td>int</td>
<td>64</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------</td>
<td>-----</td>
<td>----</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_BIGINT</td>
<td>Represents the unsigned big integer data type (UNSIGNED BIGINT)</td>
<td>int</td>
<td>128</td>
</tr>
<tr>
<td>HDFQL_FLOAT</td>
<td>Represents the float data type (FLOAT)</td>
<td>int</td>
<td>256</td>
</tr>
<tr>
<td>HDFQL_DOUBLE</td>
<td>Represents the double data type (DOUBLE)</td>
<td>int</td>
<td>512</td>
</tr>
<tr>
<td>HDFQL_CHAR</td>
<td>Represents the char data type (CHAR)</td>
<td>int</td>
<td>1024</td>
</tr>
<tr>
<td>HDFQL_VARTINYINT</td>
<td>Represents the variable-length tiny integer data type (VARTINYINT)</td>
<td>int</td>
<td>2048</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_VARTINYINT</td>
<td>Represents the unsigned variable-length tiny integer data type (UNSIGNED VARTINYINT)</td>
<td>int</td>
<td>4096</td>
</tr>
<tr>
<td>HDFQL_VARSMALLINT</td>
<td>Represents the variable-length small integer data type (VARSMALLINT)</td>
<td>int</td>
<td>8192</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_VARSMALLINT</td>
<td>Represents the unsigned variable-length small integer data type (UNSIGNED VARSMALLINT)</td>
<td>int</td>
<td>16384</td>
</tr>
<tr>
<td>HDFQL_VARINT</td>
<td>Represents the variable-length integer data type (VARINT)</td>
<td>int</td>
<td>32768</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_VARINT</td>
<td>Represents the unsigned variable-length integer data type (UNSIGNED VARINT)</td>
<td>int</td>
<td>65536</td>
</tr>
<tr>
<td>HDFQL_VARBIGINT</td>
<td>Represents the variable-length big integer data type (VARBIGINT)</td>
<td>int</td>
<td>131072</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_VARBIGINT</td>
<td>Represents the unsigned variable-length big integer data type (UNSIGNED VARBIGINT)</td>
<td>int</td>
<td>262144</td>
</tr>
<tr>
<td>HDFQL_VARFLOAT</td>
<td>Represents the variable-length float data type (VARFLOAT)</td>
<td>int</td>
<td>524288</td>
</tr>
<tr>
<td>HDFQL_VARDOUBLE</td>
<td>Represents the variable-length double data type (VARDDOUBLE)</td>
<td>int</td>
<td>1048576</td>
</tr>
<tr>
<td>HDFQL_VARCHAR</td>
<td>Represents the variable-length char data type (VARCHAR)</td>
<td>int</td>
<td>2097152</td>
</tr>
<tr>
<td>HDFQL_OPAQUE</td>
<td>Represents the opaque data type (OPAQUE)</td>
<td>int</td>
<td>4194304</td>
</tr>
<tr>
<td>HDFQL_BITFIELD</td>
<td>Represents the bitfield data type</td>
<td>int</td>
<td>8388608</td>
</tr>
<tr>
<td>HDFQL_ENUMERATION</td>
<td>Represents the enumeration data type</td>
<td>int</td>
<td>16777216</td>
</tr>
<tr>
<td>HDFQL_COMPOUND</td>
<td>Represents the compound data type</td>
<td>int</td>
<td>33554432</td>
</tr>
<tr>
<td>HDFQL_LITTLE_ENDIAN</td>
<td>Represents the little endian byte ordering</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------</td>
<td>-----</td>
<td>---</td>
</tr>
<tr>
<td>HDFQL_BIG_ENDIAN</td>
<td>Represents the big endian byte ordering</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>HDFQL_ASCII</td>
<td>Represents the ASCII character encoding</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>HDFQL_UTF8</td>
<td>Represents the UTF8 character encoding</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>HDFQL_FILL_DEFAULT</td>
<td>Represents the default fill type</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>HDFQL_FILL_USER_DEFINED</td>
<td>Represents the user defined fill type</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>HDFQL_FILL_UNDEFINED</td>
<td>Represents the undefined fill type</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>HDFQL_EARLIEST</td>
<td>Represents the HDF5 library bound earliest</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>HDFQL_LATEST</td>
<td>Represents the HDF5 library bound latest</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>HDFQL_VERSION_18</td>
<td>Represents the HDF5 library bound version 18</td>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>HDFQL_SUCCESS</td>
<td>Represents an operation that succeeded</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>HDFQL_ERROR_PARSE</td>
<td>Represents an operation that failed due to a parsing error</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>HDFQL_ERROR_NOT_FOUND</td>
<td>Represents an operation that failed due to an object (e.g. directory, file, group, dataset) not being found</td>
<td>int</td>
<td>-2</td>
</tr>
<tr>
<td>HDFQL_ERROR_NO_ACCESS</td>
<td>Represents an operation that failed due to an object (e.g. directory, file, group, dataset) not being accessible</td>
<td>int</td>
<td>-3</td>
</tr>
<tr>
<td>HDFQL_ERROR_NOT_OPEN</td>
<td>Represents an operation that failed due to an object (e.g. file) not being opened</td>
<td>int</td>
<td>-4</td>
</tr>
<tr>
<td>HDFQL_ERROR_INVALID_FILE</td>
<td>Represents an operation that failed due to a file being invalid (i.e. not a valid HDF5 file)</td>
<td>int</td>
<td>-5</td>
</tr>
<tr>
<td>HDFQL_ERROR_NOT_SUPPORTED</td>
<td>Represents an operation that failed due to not being supported</td>
<td>int</td>
<td>-6</td>
</tr>
<tr>
<td>HDFQL_ERROR_NOT_ENOUGH_SPACE</td>
<td>Represents an operation that failed due to the machine not having enough (storage) space</td>
<td>int</td>
<td>-7</td>
</tr>
<tr>
<td>HDFQL_ERROR_NOT_ENOUGH_MEMORY</td>
<td>Represents an operation that failed due to the machine not having enough (RAM) memory</td>
<td>int</td>
<td>-8</td>
</tr>
<tr>
<td>HDFQL_ERROR_ALREADY_EXISTS</td>
<td>Represents an operation that failed due to an object (e.g. directory, file, group, dataset) already existing</td>
<td>int</td>
<td>-9</td>
</tr>
<tr>
<td>HDFQL_ERROR_EMPTY</td>
<td>Represents an operation that failed due to its internal structure being empty</td>
<td>int</td>
<td>-10</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------</td>
<td>-----</td>
</tr>
<tr>
<td>HDFQL_ERROR_FULL</td>
<td>Represents an operation that failed due to its internal structure being full</td>
<td>int</td>
<td>-11</td>
</tr>
<tr>
<td>HDFQL_ERROR_BEFORE_FIRST</td>
<td>Represents an operation that failed due to trying to position/access an element before the first one</td>
<td>int</td>
<td>-12</td>
</tr>
<tr>
<td>HDFQL_ERROR_AFTER_LAST</td>
<td>Represents an operation that failed due to trying to position/access an element after the last one</td>
<td>int</td>
<td>-13</td>
</tr>
<tr>
<td>HDFQL_ERROR_OUTSIDE_LIMIT</td>
<td>Represents an operation that failed due to being outside the limit</td>
<td>int</td>
<td>-14</td>
</tr>
<tr>
<td>HDFQL_ERROR_NO_ADDRESS</td>
<td>Represents an operation that failed due to a user-defined variable having no address (i.e. is NULL)</td>
<td>int</td>
<td>-15</td>
</tr>
<tr>
<td>HDFQL_ERROR_UNEXPECTED_TYPE</td>
<td>Represents an operation that failed due to an object (e.g. group, dataset) being of an unexpected type</td>
<td>int</td>
<td>-16</td>
</tr>
<tr>
<td>HDFQL_ERROR_UNEXPECTED_DATA_TYPE</td>
<td>Represents an operation that failed due to a user-defined variable being of an unexpected data type</td>
<td>int</td>
<td>-17</td>
</tr>
<tr>
<td>HDFQL_ERROR_UNEXPECTED_STORAGE_TYPE</td>
<td>Represents an operation that failed due to a dataset being of an unexpected storage type (layout)</td>
<td>int</td>
<td>-18</td>
</tr>
<tr>
<td>HDFQL_ERROR_DANGLING_LINK</td>
<td>Represents an operation that failed due to an object being a dangling (soft or external) link</td>
<td>int</td>
<td>-19</td>
</tr>
<tr>
<td>HDFQL_ERROR_NOT_REGISTERED</td>
<td>Represents an operation that failed due to a user-defined variable not being registered</td>
<td>int</td>
<td>-20</td>
</tr>
<tr>
<td>HDFQL_ERROR_INVALID_REGULAR_EXPRESSION</td>
<td>Represents an operation that failed due to a regular expression being invalid</td>
<td>int</td>
<td>-21</td>
</tr>
<tr>
<td>HDFQL_ERROR_UNKNOWN</td>
<td>Represents an operation that failed due to an unknown/unexpected error</td>
<td>int</td>
<td>-99</td>
</tr>
</tbody>
</table>

Table 5.1 – HDFql constants in C
HDFql also supports other programming languages namely C++, Java, Python, C#, Fortran and R through wrappers. The below tables provide examples on how HDFql constants are defined in these programming languages.

In C++, the prefix “HDFQL_” of the name of constants (defined in C) is replaced by the namespace “HDFql” and its underscores (_) are discarded. The remainder of the name of constants follows the upper camel-case convention. The following table lists a subset of HDFql constants as defined in C and details how these are defined/can be used in C++.

<table>
<thead>
<tr>
<th>HDFql Constant in C</th>
<th>Corresponding Definition in C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFQL_VERSION</td>
<td>HDFql::Version</td>
</tr>
<tr>
<td>HDFQL_SUCCESS</td>
<td>HDFql::Success</td>
</tr>
<tr>
<td>HDFQL_ERROR_PARSE</td>
<td>HDFql::ErrorParse</td>
</tr>
<tr>
<td>HDFQL_TINYINT</td>
<td>HDFql::TinyInt</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_BIGINT</td>
<td>HDFql::UnsignedBigInt</td>
</tr>
<tr>
<td>HDFQL_UTF8</td>
<td>HDFql::Utf8</td>
</tr>
</tbody>
</table>

Table 5.2 – HDFql constants in C and their corresponding definitions in C++

In Java, the prefix “HDFQL_” of the name of constants (defined in C) is replaced by the class “HDFql”. The remainder of the name of constants remains exactly the same. The following table lists a subset of HDFql constants as defined in C and details how these are defined/can be used in Java.

<table>
<thead>
<tr>
<th>HDFql Constant in C</th>
<th>Corresponding Definition in Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFQL_VERSION</td>
<td>HDFql.VERSION</td>
</tr>
<tr>
<td>HDFQL_SUCCESS</td>
<td>HDFql.SUCCESS</td>
</tr>
<tr>
<td>HDFQL_ERROR_PARSE</td>
<td>HDFql.ERROR_PARSE</td>
</tr>
<tr>
<td>HDFQL_TINYINT</td>
<td>HDFql.TINYINT</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_BIGINT</td>
<td>HDFql.UNSIGNED_BIGINT</td>
</tr>
<tr>
<td>HDFQL_UTF8</td>
<td>HDFql.UTF8</td>
</tr>
</tbody>
</table>

In C++, the prefix “HDFQL_” of the name of constants (defined in C) is replaced by the namespace “HDFql” and its underscores (_) are discarded. The remainder of the name of constants follows the upper camel-case convention. The following table lists a subset of HDFql constants as defined in C and details how these are defined/can be used in C++.

<table>
<thead>
<tr>
<th>HDFql Constant in C</th>
<th>Corresponding Definition in C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFQL_VERSION</td>
<td>HDFql::Version</td>
</tr>
<tr>
<td>HDFQL_SUCCESS</td>
<td>HDFql::Success</td>
</tr>
<tr>
<td>HDFQL_ERROR_PARSE</td>
<td>HDFql::ErrorParse</td>
</tr>
<tr>
<td>HDFQL_TINYINT</td>
<td>HDFql::TinyInt</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_BIGINT</td>
<td>HDFql::UnsignedBigInt</td>
</tr>
<tr>
<td>HDFQL_UTF8</td>
<td>HDFql::Utf8</td>
</tr>
</tbody>
</table>

Table 5.2 – HDFql constants in C and their corresponding definitions in C++

In Java, the prefix “HDFQL_” of the name of constants (defined in C) is replaced by the class “HDFql”. The remainder of the name of constants remains exactly the same. The following table lists a subset of HDFql constants as defined in C and details how these are defined/can be used in Java.

<table>
<thead>
<tr>
<th>HDFql Constant in C</th>
<th>Corresponding Definition in Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFQL_VERSION</td>
<td>HDFql.VERSION</td>
</tr>
<tr>
<td>HDFQL_SUCCESS</td>
<td>HDFql.SUCCESS</td>
</tr>
<tr>
<td>HDFQL_ERROR_PARSE</td>
<td>HDFql.ERROR_PARSE</td>
</tr>
<tr>
<td>HDFQL_TINYINT</td>
<td>HDFql.TINYINT</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_BIGINT</td>
<td>HDFql.UNSIGNED_BIGINT</td>
</tr>
<tr>
<td>HDFQL_UTF8</td>
<td>HDFql.UTF8</td>
</tr>
</tbody>
</table>
Table 5.3 – HDFql constants in C and their corresponding definitions in Java

In Python, the prefix “HDFQL_” of the name of constants (defined in C) is replaced by the class “HDFql”. The remainder of the name of constants remains exactly the same. The following table lists a subset of HDFql constants as defined in C and details how these are defined/can be used in Python.

<table>
<thead>
<tr>
<th>HDFql Constant in C</th>
<th>Corresponding Definition in Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFQL_VERSION</td>
<td>HDFql.VERSION</td>
</tr>
<tr>
<td>HDFQL_SUCCESS</td>
<td>HDFql.SUCCESS</td>
</tr>
<tr>
<td>HDFQL_ERROR_PARSE</td>
<td>HDFql.ERROR_PARSE</td>
</tr>
<tr>
<td>HDFQL_TINYINT</td>
<td>HDFql.TINYINT</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_BIGINT</td>
<td>HDFql.UNSIGNED_BIGINT</td>
</tr>
<tr>
<td>HDFQL_UTF8</td>
<td>HDFql.UTF8</td>
</tr>
</tbody>
</table>

Table 5.4 – HDFql constants in C and their corresponding definitions in Python

In C#, the prefix “HDFQL_” of the name of constants (defined in C) is replaced by the class “HDFql” and its underscores (_) are discarded. The remainder of the name of constants follows the upper camel-case convention. The following table lists a subset of HDFql constants as defined in C and details how these are defined/can be used in C#.

<table>
<thead>
<tr>
<th>HDFql Constant in C</th>
<th>Corresponding Definition in C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFQL_VERSION</td>
<td>HDFql.Version</td>
</tr>
<tr>
<td>HDFQL_SUCCESS</td>
<td>HDFql_Success</td>
</tr>
<tr>
<td>HDFQL_ERROR_PARSE</td>
<td>HDFql_ErrorParse</td>
</tr>
<tr>
<td>HDFQL_TINYINT</td>
<td>HDFql.TinyInt</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_BIGINT</td>
<td>HDFql.UnsignedBigInt</td>
</tr>
<tr>
<td>HDFQL_UTF8</td>
<td>HDFql.Utf8</td>
</tr>
</tbody>
</table>

Table 5.5 – HDFql constants in C and their corresponding definitions in C#
In Fortran, the name of constants is the same as in C and can be written in any case. The following table lists a subset of HDFql constants as defined in C and details how these are defined/can be used in Fortran.

<table>
<thead>
<tr>
<th>HDFql Constant in C</th>
<th>Corresponding Definition in Fortran</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFQL_VERSION</td>
<td>HDFQL_VERSION</td>
</tr>
<tr>
<td>HDFQL_SUCCESS</td>
<td>HDFQL_SUCCESS</td>
</tr>
<tr>
<td>HDFQL_ERROR_PARSE</td>
<td>HDFQL_ERROR_PARSE</td>
</tr>
<tr>
<td>HDFQL_TINYINT</td>
<td>HDFQL_TINYINT</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_BIGINT</td>
<td>HDFQL_UNSIGNED_BIGINT</td>
</tr>
<tr>
<td>HDFQL_UTF8</td>
<td>HDFQL_UTF8</td>
</tr>
</tbody>
</table>

Table 5.6 – HDFql constants in C and their corresponding definitions in Fortran

In R, the name of constants is the same as in C. The following table lists a subset of HDFql constants as defined in C and details how these are defined/can be used in R.

<table>
<thead>
<tr>
<th>HDFql Constant in C</th>
<th>Corresponding Definition in R</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFQL_VERSION</td>
<td>HDFQL_VERSION</td>
</tr>
<tr>
<td>HDFQL_SUCCESS</td>
<td>HDFQL_SUCCESS</td>
</tr>
<tr>
<td>HDFQL_ERRORPARSE</td>
<td>HDFQL_ERRORPARSE</td>
</tr>
<tr>
<td>HDFQL_TINYINT</td>
<td>HDFQL_TINYINT</td>
</tr>
<tr>
<td>HDFQL_UNSIGNED_BIGINT</td>
<td>HDFQL_UNSIGNED_BIGINT</td>
</tr>
<tr>
<td>HDFQL_UTF8</td>
<td>HDFQL_UTF8</td>
</tr>
</tbody>
</table>

Table 5.7 – HDFql constants in C and their corresponding definitions in R
5.2 FUNCTIONS

A function is a group of instructions that together perform a specific task, requiring direction back to the caller on completion of the task. Any given function might be called at any point during a program’s execution, including by other functions or itself. It provides better modularity of a program and a high degree of code reusing. The following table summarizes all existing HDFql functions in C.

<table>
<thead>
<tr>
<th>HDFql Function in C</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdfql_execute</td>
<td>Execute a script (composed of one or more operations)</td>
</tr>
<tr>
<td>hdfql_execute_get_status</td>
<td>Get status of the last executed operation</td>
</tr>
<tr>
<td>hdfql_error_get_line</td>
<td>Get error line of the last executed operation</td>
</tr>
<tr>
<td>hdfql_error_get_position</td>
<td>Get error position of the last executed operation</td>
</tr>
<tr>
<td>hdfql_error_get_message</td>
<td>Get error message of the last executed operation</td>
</tr>
<tr>
<td>hdfql_cursor_initialize</td>
<td>Initialize a cursor for subsequent use</td>
</tr>
<tr>
<td>hdfql_cursor_use</td>
<td>Set the cursor to be used for storing the result of operations</td>
</tr>
<tr>
<td>hdfql_cursor_use_default</td>
<td>Set HDFql default cursor as the one to be used for storing the result of operations</td>
</tr>
<tr>
<td>hdfql_cursor_clear</td>
<td>Clear (i.e. empty) the cursor in use</td>
</tr>
<tr>
<td>hdfql_cursor_clone</td>
<td>Clone (i.e. duplicate) a cursor into another one</td>
</tr>
<tr>
<td>hdfql_cursor_get_data_type</td>
<td>Get data type of the cursor in use</td>
</tr>
<tr>
<td>hdfql_cursor_get_count</td>
<td>Get number of elements (i.e. result set size) stored in the cursor in use</td>
</tr>
<tr>
<td>hdfql_subcursor_get_count</td>
<td>Get number of elements (i.e. result subset size) stored in the subcursor in use</td>
</tr>
<tr>
<td>hdfql_cursor_get_position</td>
<td>Get current position of cursor in use within result set</td>
</tr>
<tr>
<td>hdfql_subcursor_get_position</td>
<td>Get current position of subcursor in use within result subset</td>
</tr>
<tr>
<td>hdfql_cursor_first</td>
<td>Move the cursor in use to the first position within result set</td>
</tr>
<tr>
<td>hdfql_subcursor_first</td>
<td>Move the subcursor in use to the first position within result subset</td>
</tr>
<tr>
<td>hdfql_cursor_last</td>
<td>Move the cursor in use to the last position within result set</td>
</tr>
<tr>
<td>hdfql_subcursor_last</td>
<td>Move the subcursor in use to the last position within result subset</td>
</tr>
<tr>
<td>HDFql Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>hdfql_cursor_next</td>
<td>Move the cursor in use one position forward from its current position</td>
</tr>
<tr>
<td>hdfql_subcursor_next</td>
<td>Move the subcursor in use one position forward from its current position</td>
</tr>
<tr>
<td>hdfql_cursor_previous</td>
<td>Move the cursor in use one position backward from its current position</td>
</tr>
<tr>
<td>hdfql_subcursor_previous</td>
<td>Move the subcursor in use one position backward from its current position</td>
</tr>
<tr>
<td>hdfql_cursor_absolute</td>
<td>Move the cursor in use to an absolute position within the result set</td>
</tr>
<tr>
<td>hdfql_subcursor_absolute</td>
<td>Move the subcursor in use to an absolute position within the result subset</td>
</tr>
<tr>
<td>hdfql_cursor_relative</td>
<td>Move the cursor in use to a relative position within result set</td>
</tr>
<tr>
<td>hdfql_subcursor_relative</td>
<td>Move the subcursor in use to a relative position within result subset</td>
</tr>
<tr>
<td>hdfql_cursor_get_size</td>
<td>Get current element size (in bytes) of the cursor in use</td>
</tr>
<tr>
<td>hdfql_subcursor_get_size</td>
<td>Get current element size (in bytes) of the subcursor in use</td>
</tr>
<tr>
<td>hdfql_cursor_get</td>
<td>Get current element of the cursor in use as a generic (typeless) pointer</td>
</tr>
<tr>
<td>hdfql_subcursor_get</td>
<td>Get current element of the subcursor in use as a generic (typeless) pointer</td>
</tr>
<tr>
<td>hdfql_cursor_get_tinyint</td>
<td>Get current element of the cursor in use as a TINYINT</td>
</tr>
<tr>
<td>hdfql_subcursor_get_tinyint</td>
<td>Get current element of the subcursor in use as a TINYINT</td>
</tr>
<tr>
<td>hdfql_cursor_get_unsigned_tinyint</td>
<td>Get current element of the cursor in use as an UNSIGNED TINYINT</td>
</tr>
<tr>
<td>hdfql_subcursor_get_unsigned_tinyint</td>
<td>Get current element of the subcursor in use as an UNSIGNED TINYINT</td>
</tr>
<tr>
<td>hdfql_cursor_get_smallint</td>
<td>Get current element of the cursor in use as a SMALLINT</td>
</tr>
<tr>
<td>hdfql_subcursor_get_smallint</td>
<td>Get current element of the subcursor in use as a SMALLINT</td>
</tr>
<tr>
<td>hdfql_cursor_get_unsigned_smallint</td>
<td>Get current element of the cursor in use as an UNSIGNED SMALLINT</td>
</tr>
<tr>
<td>hdfql_subcursor_get_unsigned_smallint</td>
<td>Get current element of the subcursor in use as an UNSIGNED SMALLINT</td>
</tr>
<tr>
<td>hdfql_cursor_get_int</td>
<td>Get current element of the cursor in use as an INT</td>
</tr>
<tr>
<td>hdfql_subcursor_get_int</td>
<td>Get current element of the subcursor in use as an INT</td>
</tr>
<tr>
<td>hdfql_cursor_get_unsigned_int</td>
<td>Get current element of the cursor in use as an UNSIGNED INT</td>
</tr>
<tr>
<td>hdfql_subcursor_get_unsigned_int</td>
<td>Get current element of the subcursor in use as an UNSIGNED INT</td>
</tr>
<tr>
<td>hdfql_cursor_get_bigint</td>
<td>Get current element of the cursor in use as a BIGINT</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>hdfql_subcursor_get_bigint</td>
<td>Get current element of the subcursor in use as a BIGINT</td>
</tr>
<tr>
<td>hdfql_cursor_get_unsigned_bigint</td>
<td>Get current element of the cursor in use as an UNSIGNED BIGINT</td>
</tr>
<tr>
<td>hdfql_subcursor_get_unsigned_bigint</td>
<td>Get current element of the subcursor in use as an UNSIGNED BIGINT</td>
</tr>
<tr>
<td>hdfql_cursor_get_float</td>
<td>Get current element of the cursor in use as a FLOAT</td>
</tr>
<tr>
<td>hdfql_subcursor_get_float</td>
<td>Get current element of the subcursor in use as a FLOAT</td>
</tr>
<tr>
<td>hdfql_cursor_get_double</td>
<td>Get current element of the cursor in use as a DOUBLE</td>
</tr>
<tr>
<td>hdfql_subcursor_get_double</td>
<td>Get current element of the subcursor in use as a DOUBLE</td>
</tr>
<tr>
<td>hdfql_cursor_get_char</td>
<td>Get current element of the cursor in use as a CHAR</td>
</tr>
<tr>
<td>hdfql_subcursor_get_char</td>
<td>Get current element of the subcursor in use as a CHAR</td>
</tr>
<tr>
<td>hdfql_variable_register</td>
<td>Register a variable for subsequent use</td>
</tr>
<tr>
<td>hdfql_variable_transient_register</td>
<td>Register a variable in a transient way for subsequent use</td>
</tr>
<tr>
<td>hdfql_variable_unregister</td>
<td>Unregister a variable</td>
</tr>
<tr>
<td>hdfql_variable_unregister_all</td>
<td>Unregister all variables</td>
</tr>
<tr>
<td>hdfql_variable_get_number</td>
<td>Get number of a variable</td>
</tr>
<tr>
<td>hdfql_variable_get_data_type</td>
<td>Get data type of a variable</td>
</tr>
<tr>
<td>hdfql_variable_get_count</td>
<td>Get number of elements (i.e. result set size) stored in a variable</td>
</tr>
<tr>
<td>hdfql_variable_get_size</td>
<td>Get size (in bytes) of a variable</td>
</tr>
<tr>
<td>hdfql_variable_get_dimension_count</td>
<td>Get number of dimensions of a variable</td>
</tr>
<tr>
<td>hdfql_variable_get_dimension</td>
<td>Get size of a certain dimension of a variable</td>
</tr>
<tr>
<td>hdfql_mpi_get_size</td>
<td>Get number (i.e. size) of processes associated to the MPI communicator</td>
</tr>
<tr>
<td>hdfql_mpi_get_rank</td>
<td>Get number (i.e. rank) of the calling process associated to the MPI communicator</td>
</tr>
</tbody>
</table>

Table 5.8 – HDFql functions in C
In C++, the prefix “hdfql_” of the name of functions (defined in C) is replaced by the namespace “HDFql” and its underscores (_) are discarded. The remainder of the name of functions follows the lower camel-case convention. The following table lists a subset of HDFql functions as defined in C and details how these are defined/can be used in C++.

<table>
<thead>
<tr>
<th>HDFql Function in C</th>
<th>Corresponding Definition in C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdfql_execute</td>
<td>HDFql::execute</td>
</tr>
<tr>
<td>hdfql_cursor_next</td>
<td>HDFql::cursorNext</td>
</tr>
<tr>
<td>hdfql_cursor_get_tinyint</td>
<td>HDFql::cursorGetTinyInt</td>
</tr>
<tr>
<td>hdfql_cursor_get_unsigned_int</td>
<td>HDFql::cursorGetUnsignedInt</td>
</tr>
<tr>
<td>hdfql_subcursor_get_big_int</td>
<td>HDFql::subcursorGetBigInt</td>
</tr>
<tr>
<td>hdfql_variable_get_number</td>
<td>HDFql::variableGetNumber</td>
</tr>
</tbody>
</table>

Table 5.9 – HDFql functions in C and their corresponding definitions in C++

In Java, the prefix “hdfql_” of the name of functions (defined in C) is replaced by the class “HDFql” and its underscores (_) are discarded. The remainder of the name of functions follows the lower camel-case convention. The following table lists a subset of HDFql functions as defined in C and details how these are defined/can be used in Java.

<table>
<thead>
<tr>
<th>HDFql Function in C</th>
<th>Corresponding Definition in Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdfql_execute</td>
<td>HDFql.execute</td>
</tr>
<tr>
<td>hdfql_cursor_next</td>
<td>HDFql.cursorNext</td>
</tr>
<tr>
<td>hdfql_cursor_get_tinyint</td>
<td>HDFql.cursorGetTinyInt</td>
</tr>
<tr>
<td>hdfql_cursor_get_unsigned_int</td>
<td>HDFql.cursorGetUnsignedInt</td>
</tr>
<tr>
<td>hdfql_subcursor_get_big_int</td>
<td>HDFql.subcursorGetBigInt</td>
</tr>
<tr>
<td>hdfql_variable_get_number</td>
<td>HDFql.variableGetNumber</td>
</tr>
</tbody>
</table>

Table 5.10 – HDFql functions in C and their corresponding definitions in Java
In Python, the prefix “hdfql_” of the name of functions (defined in C) is replaced by the class “HDFql”. The remainder of the name of functions remains exactly the same. The following table lists a subset of HDFql functions as defined in C and details how these are defined/can be used in Python.

<table>
<thead>
<tr>
<th>HDFql Function in C</th>
<th>Corresponding Definition in Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdfql_execute</td>
<td>HDFql.execute</td>
</tr>
<tr>
<td>hdfql_cursor_next</td>
<td>HDFql.cursor_next</td>
</tr>
<tr>
<td>hdfql_cursor_get_tinyint</td>
<td>HDFql.cursor_get_tinyint</td>
</tr>
<tr>
<td>hdfql_cursor_get_unsigned_int</td>
<td>HDFql.cursor_get_unsigned_int</td>
</tr>
<tr>
<td>hdfql_subcursor_get_big_int</td>
<td>HDFql.subcursor_get_big_int</td>
</tr>
<tr>
<td>hdfql_variable_get_number</td>
<td>HDFql.variable_get_number</td>
</tr>
</tbody>
</table>

Table 5.11 – HDFql functions in C and their corresponding definitions in Python

In C#, the prefix “hdfql_” of the name of functions (defined in C) is replaced by the class “HDFql” and its underscores (_) are discarded. The remainder of the name of functions follows the upper camel-case convention. The following table lists a subset of HDFql functions as defined in C and details how these are defined/can be used in C#.

<table>
<thead>
<tr>
<th>HDFql Function in C</th>
<th>Corresponding Definition in C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdfql_execute</td>
<td>HDFql.Execute</td>
</tr>
<tr>
<td>hdfql_cursor_next</td>
<td>HDFql.CursorNext</td>
</tr>
<tr>
<td>hdfql_cursor_get_tinyint</td>
<td>HDFql.CursorGetTinyInt</td>
</tr>
<tr>
<td>hdfql_cursor_get_unsigned_int</td>
<td>HDFql.CursorGetUnsignedInt</td>
</tr>
<tr>
<td>hdfql_subcursor_get_big_int</td>
<td>HDFql.SubcursorGetBigInt</td>
</tr>
<tr>
<td>hdfql_variable_get_number</td>
<td>HDFql.VariableGetNumber</td>
</tr>
</tbody>
</table>

Table 5.12 – HDFql functions in C and their corresponding definitions in C#
In Fortran, the name of functions is the same as in C and can be written using any case. The following table lists a subset of HDFql functions as defined in C and details how these are defined/can be used in Fortran.

<table>
<thead>
<tr>
<th>HDFql Function in C</th>
<th>Corresponding Definition in Fortran</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdfql_execute</td>
<td>hdfql_execute</td>
</tr>
<tr>
<td>hdfql_cursor_next</td>
<td>hdfql_cursor_next</td>
</tr>
<tr>
<td>hdfql_cursor_get_tinyint</td>
<td>hdfql_cursor_get_tinyint</td>
</tr>
<tr>
<td>hdfql_cursor_get_unsigned_int</td>
<td>hdfql_cursor_get_unsigned_int</td>
</tr>
<tr>
<td>hdfql_subcursor_get_big_int</td>
<td>hdfql_subcursor_get_big_int</td>
</tr>
<tr>
<td>hdfql_variable_get_number</td>
<td>hdfql_variable_get_number</td>
</tr>
</tbody>
</table>

Table 5.13 – HDFql functions in C and their corresponding definitions in Fortran

In R, the name of functions is the same as in C. The following table lists a subset of HDFql functions as defined in C and details how these are defined/can be used in R.

<table>
<thead>
<tr>
<th>HDFql Function in C</th>
<th>Corresponding Definition in R</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdfql_execute</td>
<td>hdfql_execute</td>
</tr>
<tr>
<td>hdfql_cursor_next</td>
<td>hdfql_cursor_next</td>
</tr>
<tr>
<td>hdfql_cursor_get_tinyint</td>
<td>hdfql_cursor_get_tinyint</td>
</tr>
<tr>
<td>hdfql_cursor_get_unsigned_int</td>
<td>hdfql_cursor_get_unsigned_int</td>
</tr>
<tr>
<td>hdfql_subcursor_get_big_int</td>
<td>hdfql_subcursor_get_big_int</td>
</tr>
<tr>
<td>hdfql_variable_get_number</td>
<td>hdfql_variable_get_number</td>
</tr>
</tbody>
</table>

Table 5.14 – HDFql functions in C and their corresponding definitions in R
5.2.1 HDFQL_EXECUTE

Syntax

int hdfql_execute(const char *script)

Description

Execute a script named script. A script can be composed of one or more operations – in case of multiple operations these can either be separated with a semicolon (;) or an end of line (EOL) terminator. In HDFql, operations are case insensitive meaning that, for example, operation “SHOW DATASET” is equivalent to “show dataset” or any other case variation. If a certain operation raises an error, any subsequent operations within script are not executed. Please refer to Table 6.2 for a complete enumeration of HDFql operations.

Parameter(s)

script – string containing one or more operations to execute. Multiple operations are either separated with a semicolon (;) or an end of line (EOL) terminator.

Return

int – depending on the success in executing script, it can either be HDFQL_SUCCESS, HDFQL_ERROR_PARSE, HDFQL_ERROR_NOT_FOUND, HDFQL_ERROR_NO_ACCESS, HDFQL_ERROR_NOT_OPEN, HDFQL_ERROR_INVALID_FILE, HDFQL_ERROR_NOT_SUPPORTED, HDFQL_ERROR_NOT_ENOUGH_SPACE, HDFQL_ERROR_NOT_ENOUGH_MEMORY, HDFQL_ERROR_ALREADY_EXISTS, HDFQL_ERROR_EMPTY, HDFQL_ERROR_FULL, HDFQL_ERROR_BEFORE_FIRST, HDFQL_ERROR_AFTER_LAST, HDFQL_ERROR_OUTSIDE_LIMIT, HDFQL_ERROR_NO_ADDRESS, HDFQL_ERROR_UNEXPECTED_TYPE, HDFQL_ERROR_UNEXPECTED_DATA_TYPE, HDFQL_ERROR_UNEXPECTED_STORAGE_TYPE, HDFQL_ERROR_DANGLING_LINK, HDFQL_ERROR_NOT_REGISTERED, HDFQL_ERROR_INVALID_REGULAR_EXPRESSION or HDFQL_ERROR_UNKNOWN.

Example(s)

```c
// declare variable
int status;

// execute script (composed of only one operation - i.e. SHOW USE FILE)
status = hdfql_execute("SHOW USE FILE");

// display message about the status of executed script (i.e. successful or not)
```
if (status == HDFQL_SUCCESS)
    printf("Execution was successful\n");
else
    printf("Execution was not successful and returned status is %d\n", status);

// execute script (composed of two operations - i.e. USE FILE my_file.h5 and SHOW)
hdfql_execute("USE FILE my_file.h5 ; SHOW");

5.2.2 HDFQL_EXECUTE_GET_STATUS

Syntax

int hdfql_execute_get_status(void)

Description

Get status of the last executed operation. In other words, this function returns the status of the last call of hdfql_execute.

Parameter(s)

None

Return

int – depending on the success of the last executed operation, it can either be HDFQL_SUCCESS, HDFQL_ERROR_PARSE, HDFQL_ERROR_NOT_FOUND, HDFQL_ERROR_NO_ACCESS, HDFQL_ERROR_NOT_OPEN, HDFQL_ERROR_INVALID_FILE, HDFQL_ERROR_NOT_SUPPORTED, HDFQL_ERROR_NOT_ENOUGH_SPACE, HDFQL_ERROR_NOT_ENOUGH_MEMORY, HDFQL_ERROR_ALREADY_EXISTS, HDFQL_ERROR_EMPTY, HDFQL_ERROR_FULL, HDFQL_ERROR_BEFORE_FIRST, HDFQL_ERROR_AFTER_LAST, HDFQL_ERROR_OUTSIDE_LIMIT, HDFQL_ERROR_NO_ADDRESS, HDFQL_ERROR_UNEXPECTED_TYPE, HDFQL_ERROR_UNEXPECTED_DATA_TYPE, HDFQL_ERROR_UNEXPECTED_STORAGE_TYPE, HDFQL_ERROR_DANGLING_LINK, HDFQL_ERROR_NOT_REGISTERED, HDFQL_ERROR_INVALID_REGULAR_EXPRESSION or HDFQL_ERROR_UNKNOWN.

Example(s)

// declare variable
int status;
// execute script (composed of only one operation - i.e. SHOW USE DIRECTORY)
hdfql_execute("SHOW USE DIRECTORY");

// get status of last executed script (i.e. SHOW USE DIRECTORY)
status = hdfql_execute_get_status();

// display message about the status of last executed script (i.e. successful or not)
if (status == HDFQL_SUCCESS)
    printf("Execution was successful\n");
else
    printf("Execution was not successful and returned status is %d\n", status);

### 5.2.3 HDFQL_ERROR_GET_LINE

**Syntax**

int hdfql_error_get_line(void)

**Description**

Get error line of the last executed operation. In other words, this function returns the number of the line (in the script) where an error was raised during the last call of `hdfql_execute`. The first line in the script is designated as number one (1).

**Parameter(s)**

None

**Return**

int – number of the line (in the script) where an error has occurred during the last executed operation. If the last executed operation was successful, the number of the line will be HDFQL_UNDEFINED.

**Example(s)**

```
// execute script (composed of only one operation - i.e. CREATE FILE my_file.h5 - which is syntactically correct)
hdfql_execute("CREATE FILE my_file.h5");

// display number of the line where an error occurred during the last executed operation
```
(should be -1 - i.e. HDFQL_UNDEFINED)
printf("Error line number is %d\n", hdfql_error_get_line());

// execute script (composed of only one operation - i.e. CREATE FILE my_file.h5 - which is
// syntactically incorrect due to a typo in "FILEX")
hdfql_execute("CREATE FILE my_file.h5");

// display number of the line where an error occurred during the last executed operation
// (should be 1)
printf("Error line number is %d\n", hdfql_error_get_line());

5.2.4 HDFQL_ERROR_GET_POSITION

Syntax

int hdfql_error_get_position(void)

Description

Get error position of the last executed operation. In other words, this function returns the position in the line where an error was raised during the last call of hdfql_execute. The first position in the line is designated as number one (1).

Parameter(s)

None

Return

int – position in the line where an error has occurred during the last executed operation. If the last executed operation was successful, the position in the line will be HDFQL_UNDEFINED.

Example(s)

// execute script (composed of only one operation - i.e. CREATE FILE my_file.h5 - which is
// syntactically correct)
hdfql_execute("CREATE FILE my_file.h5");

// display position in the line where an error occurred during the last executed operation
// (should be -1 - i.e. HDFQL_UNDEFINED)
printf("Error position is %d\n", hdfql_error_get_position());
5.2.5 **HDFQL_ERROR_GET_MESSAGE**

**Syntax**

```c
char *hdfql_error_get_message(void)
```

**Description**

Get error message of the last executed operation. In other words, this function returns the message of the error that was raised during the last call of `hdfql_execute`.

**Parameter(s)**

None

**Return**

char * – pointer to the message of an error that has occurred during the last executed operation. If the last executed operation was successful, the pointer will be NULL.

**Example(s)**

```c
// execute script (composed of only one operation - i.e. CREATE FILE my_file.h5 - which is syntactically correct)
hdfql_execute("CREATE FILE my_file.h5");

// display message of an error that occurred during the last executed operation (should be "NULL")
printf("%s\n", hdfql_error_get_message());
```

syntactically incorrect due to a typo in "FILEX")

hdfql_execute("CREATE FILEX my_file.h5");

// display message of an error that occurred during the last executed operation (should be "Unknown token "FILEX")
printf("%s\n", hdfql_error_get_message());

5.2.6 HDFQL_CURSOR_INITIALIZE

Syntax

int hdfql_cursor_initialize(HDFQL_CURSOR *cursor)

Description

Initialize a cursor named cursor for subsequent use. Before a new cursor is used for the first time, it should always be initialized (otherwise unexpected errors may arise such as a segmentation fault). The initialization of a cursor sets its data type attribute to undefined (HDFQL_UNDEFINED), its current element to NULL, and resets its count and position attributes to zero and minus one respectively, making it ready for usage. Of note, the process of initializing a cursor is only required in C and performed once, while in other programming languages supported by HDFql – namely C++, Java, Python, C#, Fortran and R – such initialization is redundant (in other words, it is not needed) as it is done automatically when creating a cursor.

Parameter(s)

cursor – pointer to a cursor (previously declared) to initialize with default values. If the pointer is NULL (in C), the cursor in use is initialized instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the cursor in use is initialized instead).

Return

int – depending on the success in initializing cursor, it can either be HDFQL_SUCCESS or HDFQL_ERROR_UNKNOWN.

Example(s)

// create a cursor named "my_cursor"
HDFQL_CURSOR my_cursor;
// initialize cursor "my_cursor"
hdfql_cursor_initialize(&my_cursor);

// use cursor "my_cursor"
hdfql_cursor_use(&my_cursor);

// display number of elements in cursor "my_cursor" (should be 0)
printf("Number of elements in cursor is %d\n", hdfql_cursor_get_count(NULL));

5.2.7 HDFQL_CURSOR_USE

Syntax

int hdfql_cursor_use(const HDFQL_CURSOR *cursor)

Description

Set the cursor named cursor as the one to be used for storing results of operations.

Parameter(s)

cursor – pointer to a cursor to use for storing the result of operations. If the pointer is NULL (in C), the HDFql default cursor is used instead (i.e. equivalent of calling the function hdfql_cursor_use_default). The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively.

Return

int – depending on the success in using cursor, it can either be HDFQL_SUCCESS or HDFQL_ERROR_NOT REGISTERED.

Example(s)

// create a cursor named "my_cursor"
HDFQL_CURSOR my_cursor;

// use cursor "my_cursor"
hdfql_cursor_use(&my_cursor);

// initialize cursor "my_cursor"
hdfql_cursor_initialize(NULL);
// display data type of cursor "my_cursor" (should be -1 - i.e. HDFQL_UNDEFINED)
printf("Data type of cursor is %d\n", hdfql_cursor_get_data_type(NULL));

// show (i.e. get) current working directory
hdfql_execute("SHOW USE DIRECTORY");

// display (again) data type of cursor "my_cursor" (should be 2097152 - i.e. HDFQLVARCHAR)
printf("Data type of cursor is %d\n", hdfql_cursor_get_data_type(NULL));

// use HDFql default cursor
hdfql_cursor_use(NULL);

// display data type of HDFql default cursor (should be -1 - i.e. HDFQL_UNDEFINED)
printf("Data type of cursor is %d\n", hdfql_cursor_get_data_type(NULL));

5.2.8 HDFQL_CURSOR_USE_DEFAULT

Syntax

int hdfql_cursor_use_default(void)

Description

Set HDFql default cursor as the one to be used for storing results of operations.

Parameter(s)

None

Return

int – depending on the success in using HDFql default cursor, it can either be HDFQL_SUCCESS or HDFQL_ERROR_UNKNOWN.

Example(s)

// create a cursor named "my_cursor"
HDFQL_CURSOR my_cursor;

// initialize cursor "my_cursor"
hdfql_cursor_initialize(&my_cursor);

// use cursor "my_cursor"
  hdfql_cursor_use(&my_cursor);

// display data type of cursor "my_cursor" (should be -1 - i.e. HDFQL_UNDEFINED)
  printf("Data type of cursor is %d\n", hdfql_cursor_get_data_type(NULL));

// show (i.e. get) current working directory
  hdfql_execute("SHOW USE DIRECTORY");

// display (again) data type of cursor "my_cursor" (should be 2097152 - i.e. HDFQL_VARCHAR)
  printf("Data type of cursor is %d\n", hdfql_cursor_get_data_type(NULL));

// use HDFql default cursor
  hdfql_cursor_use_default();

// display data type of HDFql default cursor (should be -1 - i.e. HDFQL_UNDEFINED)
  printf("Data type of cursor is %d\n", hdfql_cursor_get_data_type(NULL));

### 5.2.9 HDFQL_CURSOR_CLEAR

**Syntax**

```c
int hdfql_cursor_clear(HDFQL_CURSOR *cursor)
```

**Description**

Clear (i.e. empty) a cursor named `cursor`. Specifically, this function removes all elements (i.e. result set) stored in the cursor, specifies its data type attribute to undefined (`HDFQL_UNDEFINED`), changes its current element to NULL, and resets its count and position attributes to zero and minus one respectively.

**Parameter(s)**

- `cursor` – pointer to a cursor to clear (i.e. empty). If the pointer is NULL (in C), the cursor in use is cleared instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C `cursor` is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the cursor in use is cleared instead).
5.2.10 HDFQL_CURSOR_CLONE

Syntax

int hdfql_cursor_clone(const HDFQL_CURSOR *cursor_original, HDFQL_CURSOR *cursor_clone)

Description

Clone (i.e. duplicate) a cursor named cursor_original into another one named cursor_clone. In other words, cursor_clone will be an exact copy of cursor_original, meaning that it will have the same data type, count and position values, store the same result set, and have the same current element as the original cursor.

Parameter(s)

cursor_original – pointer to a cursor to clone. If the pointer is NULL (in C), the cursor in use is the one to be cloned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the cursor in use is the one to be cloned instead).

cursor_clone – pointer to the cursor that will be a clone (i.e. duplicate) of the original cursor.
Return

int – depending on the success in cloning `cursor_original` into `cursor_clone`, it can either be HDFQL_SUCCESS, HDFQL_ERROR_NOT_ENOUGH_MEMORY or HDFQL_ERROR_NOT_REGISTERED.

Example(s)

```c
// create a cursor named "my_cursor"
HDFQL_CURSOR my_cursor;

// initialize cursor "my_cursor"
hdfql_cursor_initialize(&my_cursor);

// show (i.e. get) current working directory and populate cursor in use (i.e. HDFql default cursor) with it
hdfql_execute("SHOW USE DIRECTORY");

// clone the cursor in use (i.e. HDFql default cursor) into the cursor "my_cursor"
hdfql_cursor_clone(NULL, &my_cursor);

// use cursor "my_cursor"
hdfql_cursor_use(&my_cursor);

// display number of elements in the cursor in use (should be 1)
printf("Number of elements in cursor is %d\n", hdfql_cursor_get_count(NULL));
```

5.2.11 HDFQL_CURSOR_GET_DATA_TYPE

Syntax

```c
int hdfql_cursor_get_data_type(const HDFQL_CURSOR *cursor)
```

Description

Get the data type of a cursor named `cursor`. If the cursor has never been populated or has been initialized or cleared, the returned data type is undefined (HDFQL_UNDEFINED). Please refer to Table 6.3 for a complete enumeration of HDFql data types.
Parameter(s)

cursor – pointer to a cursor to get its data type. If the pointer is NULL (in C), the data type of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the data type of the cursor in use is returned instead).

Return

int – depending on the data type of the cursor or its state (i.e. whether it has never been populated or has been initialized or cleared), it can either be HDFQL_TINYINT, HDFQL_UNSIGNED_TINYINT, HDFQL_SMALLINT, HDFQL_UNSIGNED_SMALLINT, HDFQL_INT, HDFQL_UNSIGNED_INT, HDFQL_BIGINT, HDFQL_UNSIGNED_BIGINT, HDFQL_FLOAT, HDFQL_DOUBLE, HDFQL_CHAR, HDFQL_VARTINYINT, HDFQL_UNSIGNED_VARTINYINT, HDFQL_VARSMALLINT, HDFQL_UNSIGNED_VARSMALLINT, HDFQL_VARINT, HDFQL_UNSIGNED_VARINT, HDFQL_VARBIGINT, HDFQL_UNSIGNED_VARBIGINT, HDFQL_VARFLOAT, HDFQL_VARDDOUBLE, HDFQL_VARCHAR, HDFQL_OPAQUE or HDFQL_UNDEFINED.

Example(s)

```c
// show (i.e. get) current working directory
hdfql_execute("SHOW USE DIRECTORY");

// display data type of the cursor in use (should be 2097152 – i.e. HDFQL_VARCHAR)
printf("Data type of cursor is %d\n", hdfql_cursor_get_data_type(NULL));

// clear the cursor in use
hdfql_cursor_clear(NULL);

// display (again) data type of the cursor in use (should be -1 – i.e. HDFQL_UNDEFINED)
printf("Data type of cursor is %d\n", hdfql_cursor_get_data_type(NULL));
```

5.2.12 HDFQL_CURSOR_GET_COUNT

Syntax

```c
int hdfql_cursor_get_count(const HDFQL_CURSOR *cursor)
```
Description

Get the number of elements (i.e. result set size) stored in a cursor named `cursor`. If the result set stores data from a dataset or attribute that does not have a dimension (i.e. if it is scalar), the returned number of elements is one. Otherwise, if the result set stores data from a dataset or attribute that has dimensions, the returned number of elements equals the multiplication of all its dimensions’ sizes (e.g. if a cursor stores a result set of two dimensions of size 10x3, the number of elements is 30). If the cursor has never been populated or has been initialized or cleared, the returned number of elements is zero.

Parameter(s)

cursor – pointer to a cursor to get its number of elements (i.e. result set size). If the pointer is NULL (in C), the number of elements of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C `cursor` is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the number of elements of the cursor in use is returned instead).

Return

int – number of elements (i.e. result set size) stored in the cursor.

Example(s)

```c
// show (i.e. get) current working directory
hdfql_execute("SHOW USE DIRECTORY");

// display number of elements in the cursor in use (should be 1)
printf("Number of elements in cursor is %d\n", hdfql_cursor_get_count(NULL));

// clear the cursor in use
hdfql_cursor_clear(NULL);

// display (again) number of elements in the cursor in use (should be 0)
printf("Number of elements in cursor is %d\n", hdfql_cursor_get_count(NULL));
```

5.2.13 HDFQL_SUBCURSOR_GET_COUNT

Syntax

```c
int hdfql_subcursor_get_count(const HDFQL_CURSOR *cursor)
```
**Description**

Get the number of elements (i.e. result subset size) stored in the subcursor in use. If the cursor that the subcursor belongs to has never been populated or has been initialized or cleared, the returned number of elements is zero.

**Parameter(s)**

`cursor` – pointer to a cursor to get the number of elements (i.e. result subset size) stored in the subcursor in use. If the pointer is NULL (in C), the number of elements of the subcursor of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C `cursor` is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the number of elements of the subcursor of the cursor in use is returned instead).

**Return**

`int` – number of elements (i.e. result subset size) stored in the subcursor.

**Example(s)**

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length int of two dimensions (size 2x2)
hdfql_execute("CREATE DATASET my_dataset AS VARINT(2, 2)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES((7, 8, 5), (9)), ((6, 1, 2, 3), (4, 0))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// display number of elements in the cursor in use (should be 4 - i.e. 2x2)
printf("Number of elements in cursor is %d\n", hdfql_cursor_get_count(NULL));

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display number of elements in the subcursor in use (should be 3)
printf("Number of elements in subcursor is %d\n", hdfql_subcursor_get_count(NULL));

// move the cursor in use to next position within the result set (i.e. second position)
hdfql_cursor_next(NULL);

// display number of elements in the subcursor in use (should be 1)
```
5.2.14 HDFQL_CURSOR_GET_POSITION

Syntax

```c
int hdfql_cursor_get_position(const HDFQL_CURSOR *cursor)
```

Description

Get current position of a cursor named `cursor` within the result set. The first element of the result set is at position zero, while the last element is located at the position returned by `hdfql_cursor_get_count` - 1. If the cursor has never been populated or has been initialized or cleared, or in case the result set is empty, the returned current position is minus one. If the cursor was moved before the first element or after the last element, the returned current position is minus one or the number of elements in the result set, respectively.

Parameter(s)

`cursor` – pointer to a cursor to get its current position within the result set. If the pointer is NULL (in C), the current position of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C `cursor` is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current position of the cursor in use is returned instead).

Return

`int` – current position of the cursor within the result set.

Example(s)

```c
// show (i.e. get) current working directory
hdfql_execute("SHOW USE DIRECTORY");

// move the cursor in use to the first position within the result set
hdfql_cursor_first(NULL);

// display position of the cursor in use within the result set (should be 0)
printf("Position of cursor is %d\n", hdfql_cursor_get_position(NULL));

// clear the cursor in use
```
hdfql_cursor_clear(NULL);

// display (again) position of the cursor in use within the result set (should be -1)
printf("Position of cursor is %d\n", hdfql_cursor_get_position(NULL));

5.2.15 HDFQL_SUBCURSOR_GET_POSITION

Syntax

int hdfql_subcursor_get_position(const HDFQL_CURSOR *cursor)

Description

Get current position of the subcursor in use within the result subset. The first element of the result subset is at position zero, while the last element is located at the position returned by hdfql_subcursor_get_count - 1. If the cursor that the subcursor belongs to has never been populated or has been initialized or cleared, or in case the result subset is empty, the returned current position is minus one. If the subcursor was moved before the first element or after the last element, the returned current position is minus one or the number of elements in the result subset, respectively.

Parameter(s)

cursor – pointer to a cursor to get the current position of the subcursor in use within the result subset. If the pointer is NULL (in C), the current position of the subcursor of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current position of the subcursor of the cursor in use is returned instead).

Return

int – current position of the subcursor within the result subset.

Example(s)

// create an HDF5 dataset named "my_dataset" of data type variable-length int of two dimensions (size 2x2)
hdfql_execute("CREATE DATASET my_dataset AS VARINT(2, 2)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(((7, 8, 5), (9)), ((6, 1, 2, 3), (4, 0)))");
// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to the first position within the result set
hdfql_cursor_first(NULL);

// display position of the subcursor in use within the result subset (should be -1)
printf("Position of subcursor is %d\n", hdfql_subcursor_get_position(NULL));

// move the subcursor in use to the next position within the result subset (two times)
hdfql_subcursor_next(NULL);

// display (again) position of the subcursor in use within the result subset (should be 1)
printf("Position of subcursor is %d\n", hdfql_subcursor_get_position(NULL));

5.2.16 HDFQL_CURSOR_FIRST

Syntax

int hdfql_cursor_first(HDFQL_CURSOR *cursor)

Description

Move a cursor named cursor to the first position within the result set. In other words, the cursor will point to the first element of the result set and its position is set to zero. If the result set is empty, an error is returned and its position remains unchanged (i.e. remains minus one).

Parameter(s)

cursor – pointer to a cursor to move to the first position within the result set. If the pointer is NULL (in C), the cursor in use is moved to the first position instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the cursor in use is moved to the first position instead).

Return

int – depending on the success in moving the cursor to the first position within the result set, it can either be HDFQL_SUCCESS or HDFQL_ERROR_EMPTY.
Example(s)

```c
// show (i.e. get) current working directory
hdfql_execute("SHOW USE DIRECTORY");

// display position of the cursor in use within the result set (should be -1)
printf("Position of cursor is %d\n", hdfql_cursor_get_position(NULL));

// move the cursor in use to the first position within the result set
hdfql_cursor_first(NULL);

// display (again) position of the cursor in use within the result set (should be 0)
printf("Position of cursor is %d\n", hdfql_cursor_get_position(NULL));
```

5.2.17 HDFQL_SUBCURSOR_FIRST

**Syntax**

```c
int hdfql_subcursor_first(HDFQL_CURSOR *cursor)
```

**Description**

Move the subcursor in use to the first position within the result subset. In other words, the subcursor will point to the first element of the result subset and its position is set to zero. If the result subset is empty, an error is returned and its position remains unchanged (i.e. remains minus one).

**Parameter(s)**

cursor – pointer to a cursor to move the subcursor in use to the first position within the result subset. If the pointer is NULL (in C), the subcursor of the cursor in use is moved to the first position instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C `cursor` is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the subcursor of the cursor in use is moved to the first position instead).

**Return**

int – depending on the success in moving the subcursor to the first position within the result subset, it can either be HDFQL_SUCCESS, HDFQL_ERROR_EMPTY, HDFQL_ERROR_BEFORE_FIRST or HDFQL_ERROR_AFTER_LAST.
Example(s)

```
// create an HDF5 dataset named "my_dataset" of data type variable-length int of two dimensions (size 2x2)
hdfql_execute("CREATE DATASET my_dataset AS VARINT(2, 2)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(((7, 8, 5), (9)), ((6, 1, 2, 3), (4, 0)))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to the first position within the result set
hdfql_cursor_first(NULL);

// display position of the subcursor in use within the result subset (should be -1)
printf("Position of subcursor is %d\n", hdfql_subcursor_get_position(NULL));

// move the subcursor in use to the first position within the result subset
hdfql_subcursor_first(NULL);

// display (again) position of the subcursor in use within the result subset (should be 0)
printf("Position of subcursor is %d\n", hdfql_subcursor_get_position(NULL));
```

5.2.18 HDFQL_CURSOR_LAST

**Syntax**

```
int hdfql_cursor_last(HDFQL_CURSOR *cursor)
```

**Description**

Move a cursor named cursor to the last position within the result set. In other words, the cursor will point to the last element of the result set and its position is set to the value returned by hdfql_cursor_get_count - 1. If the result set is empty, an error is returned and its position remains unchanged (i.e. remains minus one).

**Parameter(s)**

*cursor* – pointer to a cursor to move to the last position within the result set. If the pointer is NULL (in C), the cursor in use is moved to the last position instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql...
Hierarchical Data Format query language (HDFql) __________________________________________________________________________________________________________ Reference Manual

wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the cursor in use is moved to the last position instead).

**Return**

int – depending on the success in moving the cursor to the last position within the result set, it can either be HDFQL_SUCCESS or HDFQL_ERROR_EMPTY.

**Example(s)**

```c
// show (i.e. get) current working directory
hdfql_execute("SHOW USE DIRECTORY");

// display position of the cursor in use within the result set (should be -1)
printf("Position of cursor is %d\n", hdfql_cursor_get_position(NULL));

// move the cursor in use to the last position within the result set
hdfql_cursor_last(NULL);

// display position of the cursor in use within the result set (should be 0)
printf("Position of cursor is %d\n", hdfql_cursor_get_position(NULL));
```

**5.2.19 HDFQL_SUBCURSOR_LAST**

**Syntax**

int hdfql_subcursor_last(HDFQL_CURSOR *cursor)

**Description**

Move the subcursor in use to the last position within the result subset. In other words, the subcursor will point to the last element of the result subset and its position is set to the value returned by hdfql_subcursor_get_count - 1. If the result subset is empty, an error is returned and its position remains unchanged (i.e. remains minus one).

**Parameter(s)**

cursor – pointer to a cursor to move the subcursor in use to the last position within the result subset. If the pointer is NULL (in C), the subcursor of the cursor in use is moved to the last position instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is
mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the subcursor of the cursor in use is moved to the last position instead).

Return

int – depending on the success in moving the subcursor to the last position within the result subset, it can either be HDFQL_SUCCESS, HDFQL_ERROR_EMPTY, HDFQL_ERROR_BEFORE_FIRST or HDFQL_ERROR_AFTER_LAST.

Example(s)

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length int of two dimensions (size 2x2)
hdfql_execute("CREATE DATASET my_dataset AS VARINT(2, 2)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(((7, 8, 5), (9)), ((6, 1, 2, 3), (4, 0)))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to the first position within the result set
hdfql_cursor_first(NULL);

// display position of subcursor in use within the result subset (should be -1)
printf("Position of subcursor is %d\n", hdfql_subcursor_get_position(NULL));

// move the subcursor in use to the last position within the result subset
hdfql_subcursor_last(NULL);

// display (again) position of subcursor in use within the result subset (should be 2)
printf("Position of subcursor is %d\n", hdfql_subcursor_get_position(NULL));
```

5.2.20 HDFQL_CURSOR_NEXT

Syntax

```c
int hdfql_cursor_next(HDFQL_CURSOR *cursor)
```
**Description**

Move a cursor named *cursor* one position forward from its current position. In other words, the cursor will point to the next element of the result set and its position is incremented by one. If the result set is empty or the cursor is in the last position, an error is returned and its position remains unchanged (i.e. remains minus one) or is set to the value returned by `hdfql_cursor_get_count`, respectively.

**Parameter(s)**

*cursor* – pointer to a cursor to move one position forward from its current position. If the pointer is NULL (in C), the cursor in use is moved one position forward from its current position instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C *cursor* is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the cursor in use is moved one position forward from its current position instead).

**Return**

int – depending on the success in moving the cursor one position forward from its current position, it can either be `HDFQL_SUCCESS`, `HDFQL_ERROR_EMPTY` or `HDFQL_ERROR_AFTER_LAST`.

**Example(s)**

```c
// show (i.e. get) current working directory
hdfql_execute("SHOW USE DIRECTORY");

// move the cursor in use to the next position within the result set
hdfql_cursor_next(NULL);

// display position of cursor within the result set (should be 0)
printf("Position of cursor is %d\n", hdfql_cursor_get_position(NULL));
```

### 5.2.21 HDFQL_SUBCURSOR_NEXT

**Syntax**

```c
int hdfql_subcursor_next(HDFQL_CURSOR *cursor)
```

**Description**

Move the subcursor in use one position forward from its current position. In other words, the subcursor will point to the next element of the result subset and its position is incremented by one. If the result subset is empty or the subcursor is in the last position, an error is returned and its position remains unchanged (i.e. remains minus one) or is set to the value returned by `hdfql_subcursor_get_count`, respectively.

**Parameter(s)**

cursor – pointer to a cursor to move the subcursor in use one position forward from its current position. If the pointer is NULL (in C), the subcursor of the cursor in use is moved one position forward from its current position instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C `cursor` is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the subcursor of the cursor in use is moved one position forward from its current position instead).

**Return**

int – depending on the success in moving the subcursor one position forward from its current position, it can either be `HDFQL_SUCCESS`, `HDFQL_ERROR_EMPTY`, `HDFQL_ERROR_BEFORE_FIRST` or `HDFQL_ERROR_AFTER_LAST`.

**Example(s)**

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length int of two dimensions (size 2x2)
hdfql_execute("CREATE DATASET my_dataset AS VARINT(2, 2)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(((7, 8, 5), (9)), ((6, 1, 2, 3), (4, 0)))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to the first position within the result set
hdfql_cursor_first(NULL);

// display position of subcursor in use within the result subset (should be -1)
printf("Position of subcursor is %d\n", hdfql_subcursor_get_position(NULL));

// move the subcursor in use to the next position within the result subset (two times)
hdfql_subcursor_next(NULL);
hdfql_subcursor_next(NULL);
```
// display (again) position of subcursor in use within the result subset (should be 1)
printf("Position of subcursor is %d\n", hdfql_subcursor_get_position(NULL));

5.2.22 HDFQL_CURSOR_PREVIOUS

Syntax

int hdfql_cursor_previous(HDFQL_CURSOR *cursor)

Description

Move a cursor named cursor one position backward from its current position. In other words, the cursor will point to the previous element of the result set and its position is decremented by one. If the result set is empty or the cursor is in the first position, an error is returned and its position remains unchanged (i.e. remains minus one) or is set to minus one, respectively.

Parameter(s)

cursor – pointer to a cursor to move one position backward from its current position. If the pointer is NULL (in C), the cursor in use is moved one position backward from its current position instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the cursor in use is moved one position backward from its current position instead).

Return

int – depending on the success in moving the cursor one position backward from its current position, it can either be HDFQL_SUCCESS, HDFQL_ERROR_EMPTY or HDFQL_ERROR_BEFORE_FIRST.

Example(s)

// create an HDF5 dataset named "my_dataset" of data type float of two dimensions (size 2x10)
hdfql_execute("CREATE DATASET my_dataset AS FLOAT(2, 10)");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to the last position within the result set
hdfql_cursor_last(NULL);

// move the cursor in use to the previous position within the result set
hdfql_cursor_previous(NULL);

// display position of cursor in use within the result set (should be 18 - i.e. 2x10-1-1)
printf("Position of cursor is %d\n", hdfql_cursor_get_position(NULL));

### 5.2.23 HDFQL_SUBCURSOR_PREVIOUS

**Syntax**

```c
int hdfql_subcursor_previous(HDFQL_CURSOR *cursor)
```

**Description**

Move the subcursor in use one position backward from its current position. In other words, the subcursor will point to the previous element of the result subset and its position is decremented by one. If the result subset is empty or the subcursor is in the first position, an error is returned and its position remains unchanged (i.e. remains minus one) or is set to minus one, respectively.

**Parameter(s)**

- `cursor` – pointer to a cursor to move the subcursor in use one position backward from its current position. If the pointer is NULL (in C), the subcursor of the cursor in use is moved one position backward from its current position instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C `cursor` is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the subcursor of the cursor in use is moved one position backward from its current position instead).

**Return**

- int – depending on the success in moving the subcursor one position backward from its current position, it can either be HDFQL_SUCCESS, HDFQL_ERROR_EMPTY, HDFQL_ERROR_BEFORE_FIRST or HDFQL_ERROR_AFTER_LAST.

**Example(s)**

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length int of two dimensions
// (size 2x2)

hdfql_execute("CREATE DATASET my_dataset AS VARINT(2, 2)");
```
// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(((7, 8, 5), (9)), ((6, 1, 2, 3), (4, 0)))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to the first position within the result set
hdfql_cursor_first(NULL);

// move the subcursor in use to the last position within the result subset
hdfql_subcursor_last(NULL);

// move the subcursor in use to the previous position within the result subset (two times)
hdfql_subcursor_previous(NULL);
hdfql_subcursor_previous(NULL);

// display position of the subcursor within the result subset (should be 0 - i.e. 3-1-1-1)
printf("Position of subcursor is %d\n", hdfql_subcursor_get_position(NULL));

5.2.24 HDFQL_CURSOR_ABSOLUTE

Syntax

int hdfql_cursor_absolute(HDFQL_CURSOR *cursor, int position)

Description

Move a cursor named cursor to an absolute position position within the result set. The first element of the result set is at position zero, while the last element is located at the position returned by hdfql_cursor_get_count - 1. An attempt to move the cursor before the first element will return an error and set the position of the cursor to minus one, while an attempt to move the cursor after the last element will return an error and set the position of the cursor to number of elements in the result set.

Parameter(s)

cursor – pointer to a cursor to move to an absolute position within the result set. If the pointer is NULL (in C), the cursor in use is moved to an absolute position instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the cursor in use is moved to an absolute position instead).
position – absolute position to which to move the cursor. If position is positive, the cursor will position itself with reference to the beginning of the result set. If position is negative, the cursor will position itself with reference to the end of the result set.

Return

int – depending on the success in moving the cursor to an absolute position within the result set, it can either be HDFQL_SUCCESS, HDFQL_ERROR_EMPTY, HDFQL_ERROR_BEFORE_FIRST or HDFQL_ERROR_AFTER_LAST.

Example(s)

```c
// create five HDF5 groups named "g1", "g2", "g3", "g4" and "g5"
hdfql_execute("CREATE GROUP g1, g2, g3, g4, g5");

// show (i.e. get) all existing groups and populate cursor in use with these (should be "g1", "g2", "g3", "g4", "g5")
hdfql_execute("SHOW GROUP");

// move the cursor in use to absolute position 2 within the result set
hdfql_cursor_absolute(NULL, 2);

// display current element of the cursor in use within the result set (should be "g3")
printf("Current element of cursor is %s", hdfql_cursor_get_char(NULL));

// move the cursor in use to absolute position -2 within the result set
hdfql_cursor_absolute(NULL, -2);

// display current element of the cursor in use within the result set (should be "g4")
printf("Current element of cursor is %s", hdfql_cursor_get_char(NULL));
```

5.2.25 HDFQL_SUBCURSOR_ABSOLUTE

Syntax

```
int hdfql_subcursor_absolute(HDFQL_CURSOR *cursor, int position)
```

Description

Move the subcursor in use to an absolute position position within the result subset. The first element of the result subset is at position zero, while the last element is located at the position returned by hdfql_subcursor_get_count - 1. An attempt
to move the subcursor before the first element will return an error and set the position of the subcursor to minus one, while an attempt to move the subcursor after the last element will return an error and set the position of the subcursor to number of elements in the result subset.

**Parameter(s)**

cursor – pointer to a cursor to move the subcursor in use to an absolute position within the result subset. If the pointer is NULL (in C), the subcursor of the cursor in use is moved to an absolute position instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the subcursor of the cursor in use is moved to an absolute position instead).

position – absolute position to which to move the subcursor. If position is positive, the subcursor will position itself with reference to the beginning of the result subset. If position is negative, the subcursor will position itself with reference to the end of the result subset.

**Return**

int – depending on the success in moving the subcursor to an absolute position within the result subset, it can either be HDFQL_SUCCESS, HDFQL_ERROR_EMPTY, HDFQL_ERROR_BEFORE_FIRST or HDFQL_ERROR_AFTER_LAST.

**Example(s)**

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length int of two dimensions (size 2x2)
hdfql_execute("CREATE DATASET my_dataset AS VARINT(2, 2)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(((7, 8, 5), (9)), ((6, 1, 2, 3), (4, 0)))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to the first position within the result set
hdfql_cursor_first(NULL);

// move the subcursor in use to absolute position 2 within the result subset
hdfql_subcursor_absolute(NULL, 2);

// display current element of the subcursor in use within the result subset (should be 5)
printf("Current element of subcursor is \%d", hdfql_subcursor_get_int(NULL));
```
// move the subcursor in use to absolute position -2 within the result subset
hdfql_subcursor_absolute(NULL, -2);

// display current element of the subcursor in use within the result subset (should be 8)
printf("Current element of subcursor is %d", hdfql_subcursor_get_int(NULL));

5.2.26 HDFQL_CURSOR_RELATIVE

Syntax

int hdfql_cursor_relative(HDFQL_CURSOR *cursor, int position)

Description

Move a cursor named cursor to a relative position position with respect to its current position. The first element of the result set is at position zero, while the last element is located at the position returned by hdfql_cursor_get_count - 1. An attempt to move the cursor before the first element will return an error and set the position of the cursor to minus one, while an attempt to move the cursor after the last element will return an error and set the position of the cursor to number of elements in the result set.

Parameter(s)

cursor – pointer to a cursor to move to a relative position with respect to its current position. If the pointer is NULL (in C), the cursor in use is moved to a relative position instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the cursor in use is moved to a relative position instead).

position – relative position to which to move the cursor. If position is positive, the cursor will go forward in the result set relative to its current position. If position is negative, the cursor will go backward in the result set relative to its current position.

Return

int – depending on the success in moving the cursor to a relative position with respect to its current position, it can either be HDFQL_SUCCESS, HDFQL_ERROR_EMPTY, HDFQL_ERROR_BEFORE_FIRST or HDFQL_ERROR_AFTER_LAST.
Example(s)

```
// create five HDF5 groups named "g1", "g2", "g3", "g4" and "g5"
hrefql_execute("CREATE GROUP g1, g2, g3, g4, g5");

// show (i.e. get) all existing groups and populate cursor in use with these (should be "g1", "g2", "g3", "g4", "g5")
hrefql_execute("SHOW GROUP");

// move the cursor in use to the first position within the result set
hrefql_cursor_first(NULL);

// move the cursor in use to relative position 2 within the result set
hrefql_cursor_relative(NULL, 2);

// display current element of the cursor within the result set (should be "g3")
printf("Current element of cursor is %s", hrefql_cursor_get_char(NULL));

// move the cursor in use to relative position -2 within the result set
hrefql_cursor_relative(NULL, -2);

// display current element of the cursor within the result set (should be "g1")
printf("Current element of cursor is %s", hrefql_cursor_get_char(NULL));
```

5.2.27 HDFQL_SUBCURSOR_RELATIVE

**Syntax**

```
int hdfql_subcursor_relative(HDFL_CURSOR *cursor, int position)
```

**Description**

Move the subcursor in use to a relative position `position` with respect to its current position. The first element of the result subset is at position zero, while the last element is located at the position returned by `hrefql_subcursor_get_count` - 1. An attempt to move the subcursor before the first element will return an error and set the position of the subcursor to minus one, while an attempt to move the subcursor after the last element will return an error and set the position of the subcursor to number of elements in the result set.
Parameter(s)

cursor – pointer to a cursor to move the subcursor in use to a relative position with respect to its current position. If the pointer is NULL (in C), the subcursor of the cursor in use is moved to a relative position instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the subcursor of the cursor in use is moved to a relative position instead).

position – relative position to which to move the subcursor. If position is positive, the subcursor will go forward in the result set relative to its current position. If position is negative, the subcursor will go backward in the result set relative to its current position.

Return

int – depending on the success in moving the subcursor to a relative position with respect to its current position, it can either be HDFQL_SUCCESS, HDFQL_ERROR_EMPTY, HDFQL_ERROR_BEFORE_FIRST or HDFQL_ERROR_AFTER_LAST.

Example(s)

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length int of two dimensions (size 2x2)
char *command = "CREATE DATASET my_dataset AS VARINT(2, 2)";
hdfql_execute(command);

// insert (i.e. write) values into dataset "my_dataset"
command = "INSERT INTO my_dataset VALUES(((7, 8, 5), (9)), ((6, 1, 2, 3), (4, 0)))";
hdfql_execute(command);

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
command = "SELECT FROM my_dataset"
command = hdfql_execute(command);

// move the cursor in use to the first position within the result set
hdfql_cursor_first(NULL);

// move the subcursor in use to the first position within the result subset
hdfql_subcursor_first(NULL);

// move the subcursor in use to relative position 2 within the result subset
hdfql_subcursor_relative(NULL, 2);

// display current element of the subcursor in use within the result subset (should be 5)
printf("Current element of subcursor is %d", hdfql_subcursor_get_int(NULL));

// move the subcursor in use to relative position -1 within the result subset
```
hdfql_subcursor_relative(NULL, -1);

// display current element of the subcursor in use within the result subset (should be 8)
printf("Current element of subcursor is %d", hdfql_subcursor_get_int(NULL));

### 5.2.28 HDFQL_CURSOR_GET_SIZE

#### Syntax

```c
int hdfql_cursor_get_size(const HDFQL_CURSOR *cursor)
```

#### Description

Get the current element size (in bytes) of a cursor named `cursor`. If the result set is empty or the cursor is located before or after the first or last element of the result set, an error is returned instead.

#### Parameter(s)

- `cursor` – pointer to a cursor to get the current element size (in bytes). If the pointer is NULL (in C), the current element size of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C `cursor` is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element size of the cursor in use is returned instead).

#### Return

- `int` – depending on the success in getting the current element size (in bytes) of the cursor, it can either be ≥ 0 (i.e. the size itself), `HDFQL_ERROR_EMPTY`, `HDFQL_ERROR_BEFORE_FIRST` or `HDFQL_ERROR_AFTER_LAST`.

#### Example(s)

```c
// create an HDF5 group named "my_group"
hdfql_execute("CREATE GROUP my_group");

// show (i.e. get) all existing groups and populate cursor in use with these (should be "my_group")
hdfql_execute("SHOW GROUP");

// move the cursor in use to the first position within the result set
hdfql_cursor_first(NULL);
```
// display current element size (in bytes) of the cursor in use within the result set (should be 8 - i.e. 8x1)
printf("Current element size (in bytes) of cursor is %d\n", hdfql_cursor_get_size(NULL));

5.2.29 HDFQI_SUBCURSOR_GET_SIZE

Syntax

int hdfql_subcursor_get_size(const HDFQL_CURSOR *cursor)

Description

Get the current element size (in bytes) of the subcursor in use. If the result subset it empty or the subcursor is located before or after the first or last element of the result subset, an error is returned instead.

Parameter(s)

cursor – pointer to a cursor to get the current element size (in bytes) of the subcursor in use. If the pointer is NULL (in C), the current element size of the subcursor of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element size of the subcursor of the cursor in use is returned instead).

Return

int – depending on the success in getting the current element size (in bytes) of the subcursor, it can either be ≥ 0 (i.e. the size itself), HDFQL_ERROR_EMPTY, HDFQL_ERROR_BEFORE_FIRST or HDFQL_ERROR_AFTER_LAST.

Example(s)

// create an HDF5 dataset named "my_dataset" of data type variable-length float of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS VARFLOAT(3)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES((5.5, 2.2), (8.1), (4.9, 3.4, 5.6))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to the first position within the result set
hdfql_cursor_first(NULL);

// move the subcursor in use to the first position within the result subset
hdfql_subcursor_first(NULL);

// display current element size (in bytes) of the subcursor within the result subset (should be 4 – i.e. 4x1)
printf("Current element size (in bytes) of subcursor is %d\n", hdfql_subcursor_get_size(NULL));

5.2.30 HDFQL_CURSOR_GET

Syntax

void *hdfql_cursor_get(const HDFQL_CURSOR *cursor)

Description

Get the current element of a cursor named cursor as a generic (typeless) pointer. It is up to the programmer to interpret the returned pointer according to their needs. If the result set is empty or the cursor is located before or after the first or last element of the result set, the returned element is NULL. Of note, this function is only available in C and C++ (the remaining HDFql wrappers – namely Java, Python, C#, Fortran and R – do not support it).

Parameter(s)

cursor – pointer to a cursor to get the current element as a generic (typeless) pointer. If the pointer is NULL (in C), the current element of the cursor in use is returned instead. The equivalent of a NULL pointer in the C++ HDFql wrapper is NULL. While in C cursor is mandatory, in C++ it is optional (when not provided, the current element of the cursor in use is returned instead).

Return

void * – generic (typeless) pointer to the current element of the cursor. If there is no current element, the pointer is NULL.

Example(s)

// create an HDF5 dataset named "my_dataset" of data type float of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS FLOAT(3)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(5.5, 8.1, 4.9)");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display current element of the cursor in use as a float (should be 5.5)
printf("Current element of cursor is %f\n", (float *) hdfql_cursor_get(NULL));

5.2.31 HDFQL_SUBCURSOR_GET

Syntax

void *hdfql_subcursor_get(const HDFQL_CURSOR *cursor)

Description

Get the current element of the subcursor in use as a generic (typeless) pointer. It is up to the programmer to interpret the returned pointer according to their needs. If the result subset it empty or the subcursor is located before or after the first or last element of the result subset, the returned element is NULL. Of note, this function is only available in C and C++ (the remaining HDFql wrappers – namely Java, Python, C#, Fortran and R – do not support it).

Parameter(s)

cursor – pointer to a cursor to get the current element of the subcursor in use as a generic (typeless) pointer. If the pointer is NULL (in C), the current element of the subcursor of the cursor in use is returned instead. The equivalent of a NULL pointer in the C++ HDFql wrapper is NULL. While in C cursor is mandatory, in C++ it is optional (when not provided, the current element of the subcursor of the cursor in use is returned instead).

Return

void * – generic (typeless) pointer to the current element of the subcursor. If there is no current element, the pointer is NULL.
Example(s)

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length float of one
dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS VARFLOAT(3)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES((5.5, 2.2), (8.1), (4.9, 3.4, 5.6))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// move the subcursor in use to next position within the result subset (i.e. first position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as a float (should be 5.5)
printf("Current element of subcursor is %f\n", (float *) hdfql_subcursor_get(NULL));

// move the subcursor in use to next position within the result subset (i.e. second position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as a float (should be 2.2)
printf("Current element of subcursor is %f\n", (float *) hdfql_subcursor_get(NULL));
```

5.2.32 HDFQL_CURSOR_GET_TINYINT

**Syntax**

```c
char *hdfql_cursor_get_tinyint(const HDFQL_CURSOR *cursor)
```

**Description**

Get the current element of a cursor named `cursor` as a `TINYINT`. In other words, the current element is interpreted as a “char” C data type and returned as a pointer of such type. If the result set is empty or the cursor is located before or after the first or last element of the result set, the returned element is NULL.
Parameter(s)

cursor – pointer to a cursor to get the current element as a TINYINT. If the pointer is NULL (in C), the current element of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the cursor in use is returned instead).

Return

char * – pointer to the current element of the cursor. If there is no current element, the pointer will be NULL.

Example(s)

```c
// create an HDF5 dataset named "my_dataset" of data type char of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS TINYINT(3)" );

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(12, 34, 23)" );

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display current element of the cursor in use as a char (should be 12)
printf("Current element of cursor is %d\n", *hdfql_cursor_get_tinyint(NULL));
```

5.2.33 HDFQL_SUBCURSOR_GET_TINYINT

Syntax

char *hdfql_subcursor_get_tinyint(const HDFQL_CURSOR *cursor)

Description

Get the current element of the subcursor in use as a TINYINT. In other words, the current element is interpreted as a “char” C data type and returned as a pointer of such type. If the result subset is empty or the subcursor is located before or after the first or last element of the result subset, the returned element is NULL.
Parameter(s)

cursor – pointer to a cursor to get the current element of the subcursor in use as a TINYINT. If the pointer is NULL (in C), the current element of the subcursor of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the subcursor of the cursor in use is returned instead).

Return

char * – pointer to the current element of the subcursor. If there is no current element, the pointer will be NULL.

Example(s)

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length char of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS VARTINYINT(3)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES((5, 2), (8), (4, 3, 9))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display current element of the cursor in use as a char (should be 5)
printf("Current element of cursor is %d\n", *hdfql_cursor_get_tinyint(NULL));

// move the subcursor in use to next position within the result subset (i.e. first position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as a char (should be 5)
printf("Current element of subcursor is %d\n", *hdfql_subcursor_get_tinyint(NULL));

// move the subcursor in use to next position within the result subset (i.e. second position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as a char (should be 2)
printf("Current element of subcursor is %d\n", *hdfql_subcursor_get_tinyint(NULL));
```
5.2.34 HDFQL_CURSOR_GET_UNSIGNED_TINYINT

Syntax

unsigned char *hdfql_cursor_get_unsigned_tinyint(const HDFQL_CURSOR *cursor)

Description

Get the current element of a cursor named cursor as an UNSIGNED TINYINT. In other words, the current element is interpreted as an “unsigned char” C data type and returned as a pointer of such type. If the result set is empty or the cursor is located before or after the first or last element of the result set, the returned element is NULL.

Parameter(s)

cursor – pointer to a cursor to get the current element as a UNSIGNED TINYINT. If the pointer is NULL (in C), the current element of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the cursor in use is returned instead).

Return

unsigned char * – pointer to the current element of the cursor. If there is no current element, the pointer will be NULL.

Example(s)

```c
// create an HDF5 dataset named "my_dataset" of data type unsigned char of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS UNSIGNED TINYINT(3)");  

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(12, 34, 23)");  

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");  

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);  

// display current element of the cursor in use as an unsigned char (should be 12)
printf("Current element of cursor is \$u\n", *hdfql_cursor_get_unsigned_tinyint(NULL));
```
5.2.35 HDFQL_SUBCURSOR_GET_UNSIGNED_TINYINT

Syntax

unsigned char *hdfql_subcursor_get_unsigned_tinyint(const HDFQL_CURSOR *cursor)

Description

Get the current element of the subcursor in use as an **UNSIGNED TINYINT**. In other words, the current element is interpreted as an “unsigned char” C data type and returned as a pointer of such type. If the result subset is empty or the subcursor is located before or after the first or last element of the result subset, the returned element is NULL.

Parameter(s)

cursor – pointer to a cursor to get the current element of the subcursor in use as an **UNSIGNED TINYINT**. If the pointer is NULL (in C), the current element of the subcursor of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C **cursor** is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the subcursor of the cursor in use is returned instead).

Return

unsigned char * – pointer to the current element of the subcursor. If there is no current element, the pointer will be NULL.

Example(s)

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length unsigned char of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS UNSIGNED VARTINYINT(3) ");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES((5, 2), (8), (4, 3, 9)) ");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset ");

// move the cursor in use to next position within the result set (i.e. first position) 
hdfql_cursor_next(NULL);

// display current element of the cursor in use as an unsigned char (should be 5)
printf("Current element of cursor is %u\n", *hdfql_cursor_get_unsigned_tinyint(NULL));
```
// move the subcursor in use to next position within the result subset (i.e. first position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as an unsigned char (should be 5)
printf("Current element of subcursor is \$u\n", *hdfql_subcursor_get_unsigned_tinyint(NULL));

// move the subcursor in use to next position within the result subset (i.e. second position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as an unsigned char (should be 2)
printf("Current element of subcursor is \$u\n", *hdfql_subcursor_get_unsigned_tinyint(NULL));

5.2.36 HDFQL_CURSOR_GET_SMALLINT

Syntax

short *hdfql_cursor_get_smallint(const HDFQL_CURSOR *cursor)

Description

Get the current element of a cursor named cursor as a SMALLINT. In other words, the current element is interpreted as a “short” C data type and returned as a pointer of such type. If the result set is empty or the cursor is located before or after the first or last element of the result set, the returned element is NULL.

Parameter(s)

cursor – pointer to a cursor to get the current element as a SMALLINT. If the pointer is NULL (in C), the current element of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the cursor in use is returned instead).

Return

short * – pointer to the current element of the cursor. If there is no current element, the pointer will be NULL.

Example(s)

// create an HDF5 dataset named "my_dataset" of data type short of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS SMALLINT(3)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(12, 34, 23)");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display current element of the cursor in use as a short (should be 12)
printf("Current element of cursor is %d\n", *hdfql_cursor_get_smallint(NULL));

5.2.37 HDFQL_SUBCURSOR_GET_SMALLINT

Syntax

short *hdfql_subcursor_get_smallint(const HDFQL_CURSOR *cursor)

Description

Get the current element of the subcursor in use as a SMALLINT. In other words, the current element is interpreted as a “short” C data type and returned as a pointer of such type. If the result subset is empty or the subcursor is located before or after the first or last element of the result subset, the returned element is NULL.

Parameter(s)

cursor – pointer to a cursor to get the current element of the subcursor in use as a SMALLINT. If the pointer is NULL (in C), the current element of the subcursor of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the subcursor of the cursor in use is returned instead).

Return

short * – pointer to the current element of the subcursor. If there is no current element, the pointer will be NULL.
Example(s)

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length short of one
dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS VARSMLINT(3)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES((5, 2), (8), (4, 3, 9))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display current element of the cursor in use as a short (should be 5)
printf("Current element of cursor is %d\n", *hdfql_cursor_get_smallint(NULL));

// move the subcursor in use to next position within the result subset (i.e. first position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as a short (should be 5)
printf("Current element of subcursor is %d\n", *hdfql_subcursor_get_smallint(NULL));

// move the subcursor in use to next position within the result subset (i.e. second position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as a short (should be 2)
printf("Current element of subcursor is %d\n", *hdfql_subcursor_get_smallint(NULL));
```

### 5.2.38 HDFQL_CURSOR_GET_UNSIGNED_SMALLINT

**Syntax**

```c
unsigned short *hdfql_cursor_get_unsigned_smallint(const HDFQL_CURSOR *cursor)
```

**Description**

Get the current element of a cursor named `cursor` as an `UNSIGNED SMALLINT`. In other words, the current element is interpreted as an “unsigned short” C data type and returned as a pointer of such type. If the result set is empty or the cursor is located before or after the first or last element of the result set, the returned element is NULL.
Parameter(s)

cursor – pointer to a cursor to get the current element as an UNSIGNED SMALLINT. If the pointer is NULL (in C), the current element of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the cursor in use is returned instead).

Return

unsigned short * – pointer to the current element of the cursor. If there is no current element, the pointer will be NULL.

Example(s)

```c
// create an HDF5 dataset named "my_dataset" of data type unsigned short of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS UNSIGNED SMALLINT(3)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(12, 34, 23)");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display current element of the cursor in use as an unsigned short (should be 12)
printf("Current element of cursor is %u\n", *hdfql_cursor_get_unsigned_smallint(NULL));
```

5.2.39 HDFQL_SUBCURSOR_GET_UNSIGNED_SMALLINT

Syntax

unsigned short *hdfql_subcursor_get_unsigned_smallint(const HDFQL_CURSOR *cursor)

Description

Get the current element of the subcursor in use as an UNSIGNED SMALLINT. In other words, the current element is interpreted as an “unsigned short” C data type and returned as a pointer of such type. If the result subset is empty or the subcursor is located before or after the first or last element of the result subset, the returned element is NULL.
Parameter(s)

cursor – pointer to a cursor to get the current element of the subcursor in use as an **UNSIGNED SMALLINT**. If the pointer is NULL (in C), the current element of the subcursor of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the subcursor of the cursor in use is returned instead).

Return

unsigned short * – pointer to the current element of the subcursor. If there is no current element, the pointer will be NULL.

Example(s)

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length unsigned short of one dimension (size 3)
    hdfql_execute("CREATE DATASET my_dataset AS UNSIGNED VARSMALLINT(3)"); 

// insert (i.e. write) values into dataset "my_dataset"
    hdfql_execute("INSERT INTO my_dataset VALUES((5, 2), (8), (4, 3, 9))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
    hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
    hdfql_cursor_next(NULL);

// display current element of the cursor in use as an unsigned short (should be 5)
    printf("Current element of cursor is %u\n", *hdfql_cursor_get_unsigned_smallint(NULL));

// move the subcursor in use to next position within the result subset (i.e. first position)
    hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as an unsigned short (should be 5)
    printf("Current element of subcursor is %u\n", *hdfql_subcursor_get_unsigned_smallint(NULL));

// move the subcursor in use to next position within the result subset (i.e. second position)
    hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as an unsigned short (should be 2)
    printf("Current element of subcursor is %u\n", *hdfql_subcursor_get_unsigned_smallint(NULL));
```
5.2.40 HDFQL_CURSOR_GET_INT

Syntax

int *hdfql_cursor_get_int(const HDFQL_CURSOR *cursor)

Description

Get the current element of a cursor named cursor as an INT. In other words, the current element is interpreted as an “int” C data type and returned as a pointer of such type. If the result set is empty or the cursor is located before or after the first or last element of the result set, the returned element is NULL.

Parameter(s)

cursor – pointer to a cursor to get the current element as an INT. If the pointer is NULL (in C), the current element of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the cursor in use is returned instead).

Return

int * – pointer to the current element of the cursor. If there is no current element, the pointer will be NULL.

Example(s)

```c
// create an HDF5 dataset named "my_dataset" of data type int of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS INT(3)"); 

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(12, 34, 23)");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display current element of the cursor in use as an unsigned short (should be 12)
printf("Current element of cursor is \%d\n", *hdfql_cursor_get_int(NULL));
```
5.2.41 HDFQL_SUBCURSOR_GET_INT

**Syntax**

```c
int *hdfql_subcursor_get_int(const HDFQL_CURSOR *cursor)
```

**Description**

Get the current element of the subcursor in use as an **INT**. In other words, the current element is interpreted as an “int” C data type and returned as a pointer of such type. If the result subset is empty or the subcursor is located before or after the first or last element of the result subset, the returned element is NULL.

**Parameter(s)**

`cursor` – pointer to a cursor to get the current element of the subcursor in use as an **INT**. If the pointer is NULL (in C), the current element of the subcursor of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C `cursor` is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the subcursor of the cursor in use is returned instead).

**Return**

`int *` – pointer to the current element of the subcursor. If there is no current element, the pointer will be NULL.

**Example(s)**

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length int of one dimension (size 3)
int hdfql_execute("CREATE DATASET my_dataset AS VARINT(3)");

// insert (i.e. write) values into dataset "my_dataset"
int hdfql_execute("INSERT INTO my_dataset VALUES((5, 2), (8), (4, 3, 9))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
int hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
int hdfql_cursor_next(NULL);
```
```c
// display current element of the cursor in use as an int (should be 5)
printf("Current element of cursor is %d\n", *hdfql_cursor_get_int(NULL));

// move the subcursor in use to next position within the result subset (i.e. first position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as an int (should be 5)
printf("Current element of subcursor is %d\n", *hdfql_subcursor_get_int(NULL));

// move the subcursor in use to next position within the result subset (i.e. second position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as an int (should be 2)
printf("Current element of subcursor is %d\n", *hdfql_subcursor_get_int(NULL));
```

### 5.2.42 HDFQL_CURSOR_GET_UNSIGNED_INT

#### Syntax

```c
unsigned int *hdfql_cursor_get_unsigned_int(const HDFQL_CURSOR *cursor)
```

#### Description

Get the current element of a cursor named `cursor` as an **UNSIGNED INT**. In other words, the current element is interpreted as an “unsigned int” C data type and returned as a pointer of such type. If the result set is empty or the cursor is located before or after the first or last element of the result set, the returned element is NULL.

#### Parameter(s)

`cursor` – pointer to a cursor to get the current element as an **UNSIGNED INT**. If the pointer is NULL (in C), the current element of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C `cursor` is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the cursor in use is returned instead).

#### Return

`unsigned int *` – pointer to the current element of the cursor. If there is no current element, the pointer will be NULL.
Example(s)

```c
// create an HDF5 dataset named "my_dataset" of data type unsigned int of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS UNSIGNED INT(3)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(12, 34, 23)");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display current element of the cursor in use as an unsigned int (should be 12)
printf("Current element of cursor is %u\n", *hdfql_cursor_get_unsigned_int(NULL));
```

5.2.43 HDFQL_SUBCURSOR_GET_UNSIGNED_INT

**Syntax**

```c
unsigned int *hdfql_subcursor_get_unsigned_int(const HDFQL_CURSOR *cursor)
```

**Description**

Get the current element of the subcursor in use as an **UNSIGNED INT**. In other words, the current element is interpreted as an “unsigned int” C data type and returned as a pointer of such type. If the result subset is empty or the subcursor is located before or after the first or last element of the result subset, the returned element is NULL.

**Parameter(s)**

- `cursor` – pointer to a cursor to get the current element of the subcursor in use as an **UNSIGNED INT**. If the pointer is NULL (in C), the current element of the subcursor of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C `cursor` is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the subcursor of the cursor in use is returned instead).

**5.2.44 HDFQL_CURSOR_GET_BIGINT**

**Syntax**

long long *hdfql_cursor_get_bigint(const HDFQL_CURSOR *cursor)
**Description**

Get the current element of a cursor named `cursor` as a `BIGINT`. In other words, the current element is interpreted as a “long long” C data type and returned as a pointer of such type. If the result set is empty or the cursor is located before or after the first or last element of the result set, the returned element is NULL.

**Parameter(s)**

`cursor` – pointer to a cursor to get the current element as a `BIGINT`. If the pointer is NULL (in C), the current element of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C `cursor` is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the cursor in use is returned instead).

**Return**

`long long *` – pointer to the current element of the cursor. If there is no current element, the pointer will be NULL.

**Example(s)**

```c
// create an HDF5 dataset named "my_dataset" of data type long long of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS BIGINT(3) ");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(12, 34, 23) ");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset ");

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display current element of the cursor in use as a long long (should be 12)
printf("Current element of cursor is %lld\n", *hdfql_cursor_get_bigint(NULL));
```

5.2.45 HDFQL_SUBCURSOR_GET_BIGINT

**Syntax**

`long long *hdfql_subcursor_get_bigint(const HDFQL_CURSOR *cursor)`
Description

Get the current element of the subcursor in use as a **BIGINT**. In other words, the current element is interpreted as a “long long” C data type and returned as a pointer of such type. If the result subset is empty or the subcursor is located before or after the first or last element of the result subset, the returned element is NULL.

Parameter(s)

*cursor* – pointer to a cursor to get the current element of the subcursor in use as a **BIGINT**. If the pointer is NULL (in C), the current element of the subcursor of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C *cursor* is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the subcursor of the cursor in use is returned instead).

Return

**long long** * – pointer to the current element of the subcursor. If there is no current element, the pointer will be NULL.

Example(s)

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length long long of one dimension (size 3)
 hdfqlExecute("CREATE DATASET my_dataset AS VARBIGINT(3)");

// insert (i.e. write) values into dataset "my_dataset"
 hdfqlExecute("INSERT INTO my_dataset VALUES((5, 2), (8), (4, 3, 9))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
 hdfqlExecute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
 hdfqlCursorNext(NULL);

// display current element of the cursor in use as a long long (should be 5)
 printf("Current element of cursor is %lld\n", *hdfqlCursorGetBigInt(NULL));

// move the subcursor in use to next position within the result subset (i.e. first position)
 hdfqlSubcursorNext(NULL);

// display current element of the subcursor in use as a long long (should be 5)
 printf("Current element of subcursor is %lld\n", *hdfqlSubcursorGetBigInt(NULL));
```
// move the subcursor in use to next position within the result subset (i.e. second position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as a long long (should be 2)
printf("Current element of subcursor is %lld\n", *hdfql_subcursor_get_bigint(NULL));

5.2.46 HDFQL_CURSOR_GET_UNSIGNED_BIGINT

Syntax

unsigned long long *hdfql_cursor_get_unsigned_bigint(const HDFQL_CURSOR *cursor)

Description

Get the current element of a cursor named cursor as an UNSIGNED BIGINT. In other words, the current element is interpreted as an “unsigned long long” C data type and returned as a pointer of such type. If the result set is empty or the cursor is located before or after the first or last element of the result set, the returned element is NULL.

Parameter(s)

cursor – pointer to a cursor to get the current element as an UNSIGNED BIGINT. If the pointer is NULL (in C), the current element of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the cursor in use is returned instead).

Return

unsigned long long * – pointer to the current element of the cursor. If there is no current element, the pointer will be NULL.

Example(s)

// create an HDF5 dataset named "my_dataset" of data type unsigned long long of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS UNSIGNED BIGINT(3)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(12, 34, 23)");
// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display current element of the cursor in use as an unsigned long long (should be 12)
printf("Current element of cursor is $llu\n", *hdfql_cursor_get_unsigned_bigint(NULL));

5.2.47 HDFQL_SUBCURSOR_GET_UNSIGNED_BIGINT

Syntax

unsigned long long *hdfql_subcursor_get_unsigned_bigint(const HDFQL_CURSOR *cursor)

Description

Get the current element of the subcursor in use as an UNSIGNED BIGINT. In other words, the current element is interpreted as an “unsigned long long” C data type and returned as a pointer of such type. If the result subset is empty or the subcursor is located before or after the first or last element of the result subset, the returned element is NULL.

Parameter(s)

cursor – pointer to a cursor to get the current element of the subcursor in use as an UNSIGNED BIGINT. If the pointer is NULL (in C), the current element of the subcursor of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the subcursor of the cursor in use is returned instead).

Return

unsigned long long * – pointer to the current element of the subcursor. If there is no current element, the pointer will be NULL.

Example(s)

// create an HDF5 dataset named "my_dataset" of data type variable-length unsigned long long of
one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS UNSIGNED VARBIGINT(3)");
// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES((5, 2), (8), (4, 3, 9))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display current element of the cursor in use as an unsigned long long (should be 5)
printf("Current element of cursor is %llu\n", *hdfql_cursor_get_unsigned_bigint(NULL));

// move the subcursor in use to next position within the result subset (i.e. first position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as an unsigned long long (should be 5)
printf("Current element of subcursor is %llu\n", *hdfql_subcursor_get_unsigned_bigint(NULL));

// move the subcursor in use to next position within the result subset (i.e. second position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as an unsigned long long (should be 2)
printf("Current element of subcursor is %llu\n", *hdfql_subcursor_get_unsigned_bigint(NULL));

5.2.48 HDFQL_CURSOR_GET_FLOAT

Syntax

float *hdfql_cursor_get_float(const HDFQL_CURSOR *cursor)

Description

Get the current element of a cursor named cursor as a FLOAT. In other words, the current element is interpreted as a "float" C data type and returned as a pointer of such type. If the result set is empty or the cursor is located before or after the first or last element of the result set, the returned element is NULL.

Parameter(s)

cursor – pointer to a cursor to get the current element as a FLOAT. If the pointer is NULL (in C), the current element of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers
is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the cursor in use is returned instead).

**Return**

float * – pointer to the current element of the cursor. If there is no current element, the pointer will be NULL.

**Example(s)**

```c
// create an HDF5 dataset named "my_dataset" of data type float of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS FLOAT(3)" { );

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(5.5, 8.1, 4.9)" { );

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset" { );

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display current element of the cursor in use as a float (should be 5.5)
printf("Current element of cursor is %f\n", *hdfql_cursor_get_float(NULL));
```

### 5.2.49 HDFQL_SUBCURSOR_GET_FLOAT

**Syntax**

float *hdfql_subcursor_get_float(const HDFQL_CURSOR *cursor)

**Description**

Get the current element of the subcursor in use as a FLOAT. In other words, the current element is interpreted as a “float” C data type and returned as a pointer of such type. If the result subset is empty or the subcursor is located before or after the first or last element of the result subset, the returned element is NULL.

**Parameter(s)**

cursor – pointer to a cursor to get the current element of the subcursor in use as a FLOAT. If the pointer is NULL (in C), the current element of the subcursor of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java,
Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the subcursor of the cursor in use is returned instead).

**Return**

float * – pointer to the current element of the subcursor. If there is no current element, the pointer will be NULL.

**Example(s)**

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length float of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS VARFLOAT(3)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES((7.5, 3.1), (4.5), (4.9, 3.2, 9.7, 8.8))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display current element of the cursor in use as a float (should be 7.5)
printf("Current element of cursor is %f\n", *hdfql_cursor_get_float(NULL));

// move the subcursor in use to next position within the result subset (i.e. first position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as a float (should be 7.5)
printf("Current element of subcursor is %f\n", *hdfql_subcursor_get_float(NULL));

// move the subcursor in use to next position within the result subset (i.e. second position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as a float (should be 3.1)
printf("Current element of subcursor is %f\n", *hdfql_subcursor_get_float(NULL));
```
5.2.50 HDFQL_CURSOR_GET_DOUBLE

Syntax

double *hdfql_cursor_get_double(const HDFQL_CURSOR *cursor)

Description

Get the current element of a cursor named cursor as a DOUBLE. In other words, the current element is interpreted as a “double” C data type and returned as a pointer of such type. If the result set is empty or the cursor is located before or after the first or last element of the result set, the returned element is NULL.

Parameter(s)

cursor – pointer to a cursor to get the current element as a DOUBLE. If the pointer is NULL (in C), the current element of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the cursor in use is returned instead).

Return

double * – pointer to the current element of the cursor. If there is no current element, the pointer will be NULL.

Example(s)

```c
// create an HDF5 dataset named "my_dataset" of data type double of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS DOUBLE(3)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES(5.5, 8.1, 4.9)");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display current element of the cursor in use as a double (should be 5.5)
printf("Current element of cursor is $f\n", *hdfql_cursor_get_double(NULL));
```
5.2.51 HDFQL_SUBCURSOR_GET_DOUBLE

**Syntax**

```c
double *hdfql_subcursor_get_double(const HDFQL_CURSOR *cursor)
```

**Description**

Get the current element of the subcursor in use as a **DOUBLE**. In other words, the current element is interpreted as a “double” C data type and returned as a pointer of such type. If the result subset is empty or the subcursor is located before or after the first or last element of the result subset, the returned element is NULL.

**Parameter(s)**

`cursor` – pointer to a cursor to get the current element of the subcursor in use as a **DOUBLE**. If the pointer is NULL (in C), the current element of the subcursor of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C `cursor` is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the subcursor of the cursor in use is returned instead).

**Return**

`double *` – pointer to the current element of the subcursor. If there is no current element, the pointer will be NULL.

**Example(s)**

```c
// create an HDF5 dataset named "my_dataset" of data type variable-length double of one
dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS VARDOUBLE(3)");

// insert (i.e. write) values into dataset "my_dataset"
hdfql_execute("INSERT INTO my_dataset VALUES((7.5, 3.1), (4.5), (4.9, 3.2, 9.7, 8.8))");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)
hdfql_cursor_next(NULL);

// display current element of the cursor in use as a double (should be 7.5)
printf("Current element of cursor is \$f\n", *hdfql_cursor_get_double(NULL));
```
// move the subcursor in use to next position within the result subset (i.e. first position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as a double (should be 7.5)
printf("Current element of subcursor is %f\n", *hdfql_subcursor_get_double(NULL));

// move the subcursor in use to next position within the result subset (i.e. second position)
hdfql_subcursor_next(NULL);

// display current element of the subcursor in use as a double (should be 3.1)
printf("Current element of subcursor is %f\n", *hdfql_subcursor_get_double(NULL));

5.2.52 HDFQL_CURSOR_GET_CHAR

Syntax

char *hdfql_cursor_get_char(const HDFQL_CURSOR *cursor)

Description

Get the current element of a cursor named cursor as a CHAR. In other words, the current element is interpreted as a "char" C data type and returned as a pointer of such type. If the result set is empty or the cursor is located before or after the first or last element of the result set, the returned element is NULL.

Parameter(s)

cursor – pointer to a cursor to get the current element as a CHAR. If the pointer is NULL (in C), the current element of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the cursor in use is returned instead).

Return

char * – pointer to the current element of the cursor. If there is no current element, the pointer will be NULL.

Example(s)

// create an HDF5 dataset named "my_dataset" of data type char of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS CHAR(3) ");

// insert (i.e. write) values into dataset "my_dataset"

hdfql_execute("INSERT INTO my_dataset VALUES(Red)");

// select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it

hdfql_execute("SELECT FROM my_dataset");

// move the cursor in use to next position within the result set (i.e. first position)

hdfql_cursor_next(NULL);

// display current element of the cursor in use as a char (should be "Red")

printf("Current element of cursor is %sn", hdfql_cursor_get_char(NULL));

5.2.53 HDFQL_SUBCURSOR_GET_CHAR

Syntax

char *hdfql_subcursor_get_char(const HDFQL_CURSOR *cursor)

Description

Get the current element of the subcursor in use as a CHAR. In other words, the current element is interpreted as a “char” C data type and returned as a pointer of such type. If the result subset is empty or the subcursor is located before or after the first or last element of the result subset, the returned element is NULL.

Parameter(s)

cursor – pointer to a cursor to get the current element of the subcursor in use as a CHAR. If the pointer is NULL (in C), the current element of the subcursor of the cursor in use is returned instead. The equivalent of a NULL pointer in C++, Java, Python, C#, Fortran and R HDFql wrappers is NULL, null, None, null, 0 and NULL, respectively. While in C cursor is mandatory, in C++, Java, Python, C#, Fortran and R it is optional (when not provided, the current element of the subcursor of the cursor in use is returned instead).

Return

char * – pointer to the current element of the subcursor. If there is no current element, the pointer will be NULL.
5.2.54 HDFQL_VARIABLE_REGISTER

Syntax

int hdfql_variable_register(const void *variable)

Description

Register a variable named variable for subsequent use. In other words, for HDFql to be able to read or write from/to a user-defined variable it must first be registered. If the operation was successful, variable is registered and a number is assigned to it. This number – calculated by HDFql – starts with zero and is incremented by one every time a new variable is
registered. If variable is registered more than once, only one number is assigned to it (namely the number assigned upon the first registering). Of note, currently up to eight variables can be registered at any given time (trying to register more than this number will raise an HDFQL_ERROR_FULL). In C, C++ and Fortran any variable may be registered as long HDFql can properly read and write values from/to it by having direct access to the memory associated with these – otherwise unexpected errors may arise such as a segmentation fault. The following restrictions apply to other programming languages (supported by HDFql):

- In Java, only a variable that is an array of “byte”, “short”, “int”, “long”, “float”, “double” or “String” data type or its corresponding wrapper class “Byte”, “Short”, “Integer”, “Long”, “Float” or “Double” may be registered. Any attempt to register a variable that is not an array or of the data type/wrapper class previously enumerated will return an error (HDFQL_ERROR_UNEXPECTED_DATA_TYPE).

- In Python, only a variable that is a NumPy array of “int8”, “uint8”, “int16”, “uint16”, “int32”, “uint32”, “int64”, “uint64”, “float32”, “float64”, “Ssize” or “ubyte” data type may be registered. Any attempt to register a variable that is not a NumPy array or of the data type previously enumerated will return an error (HDFQL_ERROR_UNEXPECTED_DATA_TYPE). Please refer to http://www.numpy.org for additional information.

- In C#, only a variable that is an array of “SByte”, “Byte”, “Int16”, “UInt16”, “Int32”, “UInt32”, “Int64”, “UInt64”, “Single”, “Double” or “String” data type or its alias “sbyte”, “byte”, “short”, “ushort”, “int”, “uint”, “long”, “ulong”, “float”, “double” or “string” may be registered. Any attempt to register a variable that is not an array or of the data type/alias previously enumerated will return an error (HDFQL_ERROR_UNEXPECTED_DATA_TYPE).

- In R, only a variable that is a vector, matrix or array of “integer”, “integer64” (through package bit64), “numeric”, “double”, “character” or “raw” data type may be registered. Any attempt to register a variable that is not a vector, matrix or array or of the data type previously enumerated will return an error (HDFQL_ERROR_UNEXPECTED_DATA_TYPE).

An important aspect to remember when working with a variable is that it should not change address from the moment it has been registered until used in the intented operation (e.g. SELECT) or function (e.g. HDFQL_VARIABLE_GET_NUMBER), as HDFql will not be able to identify the variable. In this case, the operation or function will raise an error (HDFQL_ERROR_NOT_REGISTERED). In case a variable needs to change its address (for whatever the reason), first unregister it via the function hdfql_variable_unregister, change its address, and register it again. In general, it is advisable to register a variable just before executing the HDFql operation or function which employs it, and to unregister it as soon as it is no longer used (this is especially relevant in C# where variables are pinned when registered and thus cannot be moved by the Garbage Collector).
Parameter(s)

*variable* – variable to register for subsequent use.

Return

int – depending on the success in registering the variable for subsequent use, it can either be ≥ 0 (i.e. the number assigned to the variable when successfully registered), HDFQL_ERROR_NO_ADDRESS, HDFQL_ERROR_FULL or HDFQL_ERROR_UNEXPECTED_DATA_TYPE.

Example(s)

```c
// declare variables
char script[1024];
short data[3];
int number;

// create an HDF5 dataset named "my_dataset" of data type short of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS SMALLINT(3)");

// populate variable "data" with certain values
data[0] = 21;
data[1] = 18;
data[2] = 75;

// register variable "data" for subsequent use (by HDFql)
number = hdfql_variable_register(data);

// prepare script to insert (i.e. write) values from variable "data" into dataset "my_dataset"
sprintf(script, "INSERT INTO my_dataset VALUES FROM MEMORY %d", number);

// execute script
hdfql_execute(script);

// unregister variable "data" as it is no longer used/needed (by HDFql)
hdfql_variable_unregister(data);
```
5.2.55 HDFQL_VARIABLE_TRANSIENT_REGISTER

Syntax

int hdfql_variable_transient_register(const void *variable)

Description

Register a variable named variable in a transient way for subsequent use. This function is similar to hdfql_variable_register, except that after the execution of a script (via the function hdfql_execute) which uses variable, variable is automatically unregistered (by HDFql) thus alleviating the programmer from doing it.

Parameter(s)

variable – variable to register in a transient way for subsequent use.

Return

int – depending on the success in registering the variable in a transient way for subsequent use, it can either be ≥ 0 (i.e. the number assigned to the variable when successfully registered), HDFQL_ERROR_NO_ADDRESS, HDFQL_ERROR_FULL or HDFQL_ERROR_UNEXPECTED_DATA_TYPE.

Example(s)

```c
// declare variables
char script[1024];
short data[3];
int number;

// create an HDF5 dataset named "my_dataset" of data type short of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS SMALLINT(3)");

// populate variable "data" with certain values
data[0] = 21;
data[1] = 18;
data[2] = 75;

// register variable "data" in a transient way for subsequent use (by HDFql)
number = hdfql_variable_transient_register(data);

// prepare script to insert (i.e. write) values from variable "data" into dataset "my_dataset"
sprintf(script, "INSERT INTO my_dataset VALUES FROM MEMORY %d", number);
```
5.2.56 HDFQL_VARIABLE_UNREGISTER

Syntax

int hdfql_variable_unregister(const void *variable)

Description

Unregister a variable named variable. In other words, HDFql will free up any memory that may have been allocated to manage the variable as well as the number assigned to it (the number may then be assigned to a new variable registered subsequently). In general, it is advisable to unregister a variable as soon as it is no longer used by HDFql (this is especially relevant in C# as variables are unpinned when unregistered and thus may again be moved by the Garbage Collector). If variable has never been registered or has already been unregistered, an error is returned.

Parameter(s)

variable – variable to unregister.

Return

int – depending on the success in unregistering the variable, it can either be HDFQL_SUCCESS, HDFQL_ERROR_NO_ADDRESS or HDFQL_ERROR_NOT_REGISTERED.

Example(s)

// declare variables
char script[1024];
short data[3];
int number;

// create an HDF5 dataset named "my_dataset" of data type short of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset AS SMALLINT(3)");

// populate variable "data" with certain values
 Hierarchical Data Format query language (HDFql)  

5.2.57 HDFQL_VARIABLE_UNREGISTER_ALL

**Syntax**

```c
int hdfql_variable_unregister_all(void)
```

**Description**

Unregister all the variables that may have been registered previously. In other words, HDFql will free up any memory that may have been allocated to manage the variables as well as the numbers assigned to them (the numbers may then be assigned to new variables registered subsequently). In general, it is advisable to unregister variables as soon as they are no longer used by HDFql (this is especially relevant in C# as variables are unpinned when unregistered and thus may again be moved by the Garbage Collector).

**Parameter(s)**

None

**Return**

int – depending on the success in unregistering all the variables that may have been registered previously, it can either be HDFQL_SUCCESS or HDFQL_ERROR_UNKNOWN.
Example(s)

```c
// declare variables
short data0[3];
float data1[5];

// register variable "data0" for subsequent use (by HDFql)
hdfql_variable_register(data0);

// register variable "data1" for subsequent use (by HDFql)
hdfql_variable_register(data1);

// display number of variable "data0" (should be 0)
printf("Number of variable is %d\n", hdfql_variable_get_number(data0));

// display number of variable "data1" (should be 1)
printf("Number of variable is %d\n", hdfql_variable_get_number(data1));

// unregister all the variables (i.e. variables "data0" and "data1") as they are no longer used/needed (by HDFql)
hdfql_variable_unregister_all();
```

5.2.58 HDFQL_VARIABLE_GET_NUMBER

**Syntax**

```c
int hdfql_variable_get_number(const void *variable)
```

**Description**

Get the number of a variable named `variable`. This refers to the number that was calculated by HDFql and assigned to the variable upon registering it with the function `hdfql_variable_register`. If `variable` has never been registered or has been unregistered, an error is returned.

**Parameter(s)**

`variable` – variable to get the number (calculated by HDFql) assigned to it.
Return

int – depending on the success in getting the number assigned to the variable, it can either be ≥ 0 (i.e. the number assigned to the variable), HDFQL_ERROR_NO_ADDRESS or HDFQL_ERROR_NOT_REGISTERED.

Example(s)

```c
// declare variables
short data0[3];
float data1[5];

// register variable "data0" for subsequent use (by HDFql)
hdfql_variable_register(data0);

// register variable "data1" for subsequent use (by HDFql)
hdfql_variable_register(data1);

// display number of variable "data0" (should be 0)
printf("Number of variable is %d\n", hdfql_variable_get_number(data0));

// display number of variable "data1" (should be 1)
printf("Number of variable is %d\n", hdfql_variable_get_number(data1));
```

5.2.59 HDFQL_VARIABLE_GET_DATA_TYPE

Syntax

```c
int hdfql_variable_get_data_type(const void *variable)
```

Description

Get the data type of a variable named variable. This function should help the programmer to better handle the content stored in variable. The data type refers to the result of a DATA QUERY LANGUAGE (DQL) or DATA INTROSPECTION LANGUAGE (DIL) operation redirected into memory – and not the data type of variable declared in the program. If variable has never been registered, populated (through the redirection of the result of a DATA QUERY LANGUAGE (DQL) or DATA INTROSPECTION LANGUAGE (DIL) operation into memory), or in case it has been unregistered, the returned data type is undefined (HDFQL_UNDEFINED). Please refer to Table 6.3 for a complete enumeration of HDFql data types.
Parameter(s)

variable – variable to get its data type.

Return

int – depending on the success in getting the data type of the variable, it can either be HDFQL_TINYINT, HDFQL_UNSIGNED_TINYINT, HDFQL_SMALLINT, HDFQL_UNSIGNED_SMALLINT, HDFQL_INT, HDFQL_UNSIGNED_INT, HDFQL_BIGINT, HDFQL_UNSIGNED_BIGINT, HDFQL_FLOAT, HDFQL_DOUBLE, HDFQL_CHAR, HDFQL_VARTINYINT, HDFQL_UNSIGNED_VARTINYINT, HDFQL_VARSMALLINT, HDFQL_UNSIGNED_VARSMALLINT, HDFQL_VARINT, HDFQL_UNSIGNED_VARINT, HDFQL_VARBIGINT, HDFQL_UNSIGNED_VARBIGINT, HDFQL_VARFLOAT, HDFQL_VARDOUBLE, HDFQL_VARCHAR, HDFQL_OPAQUE, HDFQL_BITFIELD, HDFQL_ENUMERATION, HDFQL_COMPOUND, HDFQL_UNDEFINED, HDFQL_ERROR_NO_ADDRESS or HDFQL_ERROR_NOT_REGISTERED.

Example(s)

```c
// declare variables
cchar script[1024];
cchar data[1024];

// register variable "data" for subsequent use (by HDFql)
hdfql_variable_register(data);

// prepare script to show (i.e. get) current working directory and populate variable "data" with it
sprintf(script, "SHOW USE DIRECTORY INTO MEMORY %d", hdfql_variable_get_number(data));

// execute script
hdfql_execute(script);

// display data type of variable "data" (should be 2097152 – i.e. HDFQL_VARCHAR)
printf("Data type of variable is %d\n", hdfql_variable_get_data_type(data));
```

5.2.60 HDFQL_VARIABLE_GET_COUNT

Syntax

```c
int hdfql_variable_get_count(const void *variable)
```
**Description**

Get the number of elements (i.e. result set size) stored in a variable named `variable`. This function should help the programmer to better handle the content stored in `variable`. If the result set stores data from a dataset or attribute that does not have a dimension (i.e. if it is scalar), the returned number of elements is one. Otherwise, if the result set stores data from a dataset or attribute that has dimensions, the returned number of elements equals the multiplication of all its dimensions’ sizes (e.g. if a variable stores a result set of two dimensions of size 10x3, the number of elements is 30). Of note, in case a hyperslab or point selection is specified (in a DATA QUERY LANGUAGE (DQL) operation) the number of elements of the selection will be returned instead. If `variable` has never been populated (through the redirection of the result of a DATA QUERY LANGUAGE (DQL) or DATA INTROSPECTION LANGUAGE (DIL) operation into memory), the returned number of elements is zero.

**Parameter(s)**

`variable` – variable to get its number of elements (i.e. result set size).

**Return**

int – depending on the success in getting the number of elements of the variable, it can either be $\geq 0$ (i.e. the number of elements), HDFQL_ERROR_NO_ADDRESS or HDFQL_ERROR_NOT_REGISTERED.

**Example(s)**

```c
// declare variables
char script[1024];
int data[5][3];

// create an HDF5 dataset named "my_dataset" of data type int of two dimensions (size 5x3)
hdfql_execute("CREATE DATASET my_dataset AS INT(5, 3)");

// register variable "data" for subsequent use (by HDFql)
hdfql_variable_register(data);

// prepare script to select (i.e. read) data from dataset "my_dataset" and populate variable "data" with it
sprintf(script, "SELECT FROM my_dataset INTO MEMORY %d", hdfql_variable_get_number(data));

// execute script
hdfql_execute(script);

// display number of elements in variable "data" (should be 15 – i.e. 5x3)
```
5.2.61 HDFQL_VARIABLE_GET_SIZE

Syntax

```c
int hdfql_variable_get_size(const void *variable)
```

Description

Get the size (in bytes) of a variable named `variable`. This function should help the programmer to better handle the content stored in `variable`. The size (in bytes) refers to the result of a DATA QUERY LANGUAGE (DQL) or DATA INTROSPECTION LANGUAGE (DIL) operation redirected into memory – and not the size (in bytes) that `variable` has in the program. If `variable` has never been registered or has been unregistered, an error is returned. If `variable` has never been populated (through the redirection of the result of a DATA QUERY LANGUAGE (DQL) or DATA INTROSPECTION LANGUAGE (DIL) operation into memory), the returned size is zero. Please refer to Table 6.3 for a complete enumeration of HDFql data types and their corresponding sizes (in bytes).

Parameter(s)

`variable` – variable to get its size (in bytes).

Return

`int` – depending on the success in getting the size (in bytes) of the variable, it can either be ≥ 0 (i.e. the size itself), HDFQL_ERROR_NO_ADDRESS or HDFQL_ERROR_NOT_REGISTERED.

Example(s)

```c
// declare variables
char script[1024];
int data[5][3];

// create an HDF5 dataset named "my_dataset" of data type int of two dimensions (size 5x3)
hdfql_execute("CREATE DATASET my_dataset AS INT(5, 3)");

// register variable "data" for subsequent use (by HDFql)
hdfql_variable_register(data);
```
// prepare script to select (i.e. read) data from dataset "my_dataset" and populate variable "data" with it
sprintf(script, "SELECT FROM my_dataset INTO MEMORY %d", hdfql_variable_get_number(data));

// execute script
hdfql_execute(script);

// display size (in bytes) of variable "data" (should be 60 - i.e. 5x3x4)
printf("Size (in bytes) of variable is %d\n", hdfql_variable_get_size(data));

5.2.62 HDFQL_VARIABLE_GET_DIMENSION_COUNT

Syntax

int hdfql_variable_get_dimension_count(const void *variable)

Description

Get the number of dimensions of a variable named variable. This function should help the programmer to better handle the content stored in variable. The number of dimensions refers to the result of a DATA QUERY LANGUAGE (DQL) or DATA INTROSPECTION LANGUAGE (DIL) operation redirected into memory – and not the number of dimensions that variable has in the program. If variable has never been registered or has been unregistered, an error is returned. If variable has never been populated (through the redirection of the result of a DATA QUERY LANGUAGE (DQL) or DATA INTROSPECTION LANGUAGE (DIL) operation into memory), the returned number of dimensions is zero.

Parameter(s)

variable – variable to get its number of dimensions.

Return

int – depending on the success in getting the number of dimensions of the variable, it can either be ≥ 0 (i.e. the number of dimensions), HDFQL_ERROR_NO_ADDRESS or HDFQL_ERROR_NOT_REGISTERED.

Example(s)

// declare variables
char script[1024];
int data[5][3];
// create an HDF5 dataset named "my_dataset" of data type int of two dimensions (size 5x3)
hdfql_execute("CREATE DATASET my_dataset AS INT(5, 3)");

// register variable "data" for subsequent use (by HDFql)
hdfql_variable_register(data);

// prepare script to select (i.e. read) data from dataset "my_dataset" and populate variable "data" with it
sprintf(script, "SELECT FROM my_dataset INTO MEMORY %d", hdfql_variable_get_number(data));

// execute script
hdfql_execute(script);

// display number of dimensions of variable "data" (should be 2)
printf("Number of dimensions in variable is %d\n", hdfql_variable_get_dimension_count(data));

5.2.63 HDFQL_VARIABLE_GET_DIMENSION

Syntax

int hdfql_variable_get_dimension(const void *variable, int index)

Description

Get the size of a certain dimension specified in index of a variable named variable. This function should help the programmer to better handle the content stored in variable. The size of a certain dimension refers to the result of a DATASET QUERY LANGUAGE (DQL) or DATA INTROSPECTION LANGUAGE (DIL) operation redirected into memory – and not the size of a certain dimension that variable has in the program. The index of the first dimension is zero (index must be between 0 and the value returned by hdfql_variable_get_dimension_count - 1). If variable has never been registered, populated (through the redirection of the result of a DATASET QUERY LANGUAGE (DQL) or DATA INTROSPECTION LANGUAGE (DIL) operation into memory), or in case it has been unregistered, an error is returned.

Parameter(s)

variable – variable to get the size of one of its dimensions.

index – index of the dimension to get its size.
Return

int – depending on the success in getting the size of a certain dimension of the variable, it can either be ≥ 0 (i.e. the size of a certain dimension itself), HDFQL_ERROR_NO_ADDRESS, HDFQL_ERROR_NOT_REGISTERED or HDFQL_ERROR_OUTSIDE_LIMIT.

Example(s)

```
// declare variables
char script[1024];
int data[5][3];

// create an HDF5 dataset named "my_dataset" of data type int of two dimensions (size 5x3)
hdfql_execute("CREATE DATASET my_dataset AS INT(5, 3)" );

// register variable "data" for subsequent use (by HDFql)
hdfql_variable_register(data);

// prepare script to select (i.e. read) data from dataset "my_dataset" and populate variable "data" with it
sprintf(script, "SELECT FROM my_dataset INTO MEMORY %d", hdfql_variable_get_number(data));

// execute script
hdfql_execute(script);

// display size of the first dimension of variable "data" (should be 5)
printf("Size of first dimension of variable is %d\n", hdfql_variable_get_dimension(0));

// display size of the second dimension of variable "data" (should be 3)
printf("Size of second dimension of variable is %d\n", hdfql_variable_get_dimension(1));
```

5.2.64 HDFQL_MPI_GET_SIZE

Syntax

```
int hdfql_mpi_get_size( void )
```

Description

Get the number (i.e. size) of processes associated to the default MPI communicator (MPI_COMM_WORLD). In other words, this function returns the number of MPI processes that are specified upon launching a program in parallel using
“mpiexec” (or a similar launcher). Of note, this function is basically a wrapper of the MPI function “MPI_Comm_size” (please refer to https://www.mpich.org/static/docs/v3.2/www3/MPI_Comm_size.html or https://www.open-mpi.org/doc/v2.1/man3/MPI_Comm_size.3.php for additional information in case the MPI library used is MPICH (or, alternatively, one of its ABI compatible derivative libraries) or Open MPI).

Parameter(s)

None

Return

int – depending on the success in getting the number of processes associated to the default MPI communicator (MPI_COMM_WORLD), it can either be ≥ 1 (i.e. the number of processes) or HDFQL_UNDEFINED (in case MPI itself was not initialized properly, in case of an HDFql non MPI-based distribution, or if it was executed in Windows as HDFql does not support the parallel HDF5 (PHDF5) library in this platform currently).

Example(s)

```c
// display number (i.e. size) of MPI processes (if the program is launched as, e.g., "mpiexec -n 5 my_program", the message "Number (i.e. size) of MPI processes is 5" will be displayed five times)
printf("Number (i.e. size) of MPI processes is %d\n", hdfql_mpi_get_size());
```

5.2.65 HDFQL_MPI_GET_RANK

Syntax

int hdfql_mpi_get_rank(void)

Description

Get the number (i.e. rank) of the calling process associated to the default MPI communicator (MPI_COMM_WORLD). In other words, this function returns the number of the MPI process assigned to a particular instance of a program that was launched in parallel using “mpiexec” (or a similar launcher). Of note, this function is basically a wrapper of the MPI function “MPI_Comm_rank” (please refer to https://www.mpich.org/static/docs/v3.2/www3/MPI_Comm_rank.html or https://www.open-mpi.org/doc/v2.1/man3/MPI_Comm_rank.3.php for additional information in case the MPI library used is MPICH (or, alternatively, one of its ABI compatible derivative libraries) or Open MPI).
Parameters(s)

None

Return

int – depending on the success in getting the number (i.e. rank) of the calling process associated to the default MPI communicator (MPI_COMM_WORLD), it can either be ≥ 0 (i.e. the number of the calling process) or HDFQL_UNDEFINED (in case MPI itself was not initialized properly, in case of an HDFql non MPI-based distribution, or if in Windows as HDFql does not support the parallel HDF5 (PHDF5) library in this platform currently).

Example(s)

```c
// display number (i.e. rank) of the MPI process (if the program is launched as, e.g., "mpiexec -n 3 my_program", the message "Number (i.e. rank) of the MPI process is X" will be displayed three times where X is 0, 1 or 2 (not necessarily in this order))
printf("Number (i.e. rank) of the MPI process is %d\n", hdfql_mpi_get_rank());
```
HDFql is a high-level language to manage HDF5 files in a simple and natural way. It was designed to be similar to SQL (wherever possible) so that its learning effort is kept at minimum while still providing great power and flexibility to the programmer. This chapter describes data types, post-processing options to further transform result sets, redirecting options to read/write data/result sets from/into disparate input/output sources, and operations (i.e. the language itself) available in HDFql. It also introduces text formatting conventions used throughout this chapter to describe HDFql operations (Table 6.1), and a summary of existing operations (Table 6.2). Before continuing, it is highly recommended to first read the HDF5 User’s Guide available at https://support.hdfgroup.org/HDF5/doc/UG/HDF5_Users_Guide.pdf to facilitate the understanding of the current chapter.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Keyword that must be typed exactly as shown</td>
<td>CREATE</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Value that the programmer must supply</td>
<td>dataset_name</td>
</tr>
<tr>
<td>Between brackets ([])</td>
<td>Optional keyword/value</td>
<td>[DATASET]</td>
</tr>
</tbody>
</table>
| Between braces {} | Logical grouping of keywords/values (to ease understanding) | {
| Separated with a pipe | Set of keywords/values from which one must be chosen          | GROUP | DATASET | ATTRIBUTE          |
| Asterisk (*)     | Keyword/value that can be supplied several times | group_name [, group_name]*                  |

Table 6.1 – HDFql operations text formatting conventions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE DIRECTORY</td>
<td>Create a directory</td>
</tr>
<tr>
<td>CREATE FILE</td>
<td>Create an HDF5 file</td>
</tr>
<tr>
<td>CREATE GROUP</td>
<td>Create an HDF5 group</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>CREATE DATASET</td>
<td>Create an HDF5 dataset</td>
</tr>
<tr>
<td>CREATE ATTRIBUTE</td>
<td>Create an HDF5 attribute</td>
</tr>
<tr>
<td>CREATE [SOFT</td>
<td>HARD] LINK</td>
</tr>
<tr>
<td>CREATE EXTERNAL LINK</td>
<td>Create an HDF5 external link</td>
</tr>
<tr>
<td>ALTER DIMENSION</td>
<td>Alter (i.e. change) dimensions of an existing HDF5 dataset</td>
</tr>
<tr>
<td>RENAME DIRECTORY</td>
<td>Rename (or move) an existing directory</td>
</tr>
<tr>
<td>RENAME FILE</td>
<td>Rename (or move) an existing file</td>
</tr>
<tr>
<td>RENAME [GROUP</td>
<td>DATASET</td>
</tr>
<tr>
<td>COPY FILE</td>
<td>Copy an existing file</td>
</tr>
<tr>
<td>COPY [GROUP</td>
<td>DATASET</td>
</tr>
<tr>
<td>DROP DIRECTORY</td>
<td>Drop (i.e. delete) an existing directory</td>
</tr>
<tr>
<td>DROP FILE</td>
<td>Drop (i.e. delete) an existing file</td>
</tr>
<tr>
<td>DROP [GROUP</td>
<td>DATASET</td>
</tr>
<tr>
<td>INSERT</td>
<td>Insert (i.e. write) data into an HDF5 dataset or attribute</td>
</tr>
<tr>
<td>SELECT</td>
<td>Select (i.e. read) data from an HDF5 dataset or attribute</td>
</tr>
<tr>
<td>SHOW FILE VALIDITY</td>
<td>Get validity of a file (i.e. whether it is a valid HDF5 file or not)</td>
</tr>
<tr>
<td>SHOW USE DIRECTORY</td>
<td>Get working directory currently in use</td>
</tr>
<tr>
<td>SHOW USE FILE</td>
<td>Get HDF5 file currently in use</td>
</tr>
<tr>
<td>SHOW ALL USE FILE</td>
<td>Get all HDF5 files in use (i.e. open)</td>
</tr>
<tr>
<td>SHOW USE GROUP</td>
<td>Get HDF5 group currently in use</td>
</tr>
<tr>
<td>SHOW [GROUP</td>
<td>DATASET</td>
</tr>
<tr>
<td>SHOW TYPE</td>
<td>Get type of an HDF5 object (i.e. group, dataset or attribute)</td>
</tr>
<tr>
<td>SHOW DATA TYPE</td>
<td>Get data type of an HDF5 dataset or attribute</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SHOW MASK</td>
<td>Get (filter) mask of an HDF5 dataset</td>
</tr>
<tr>
<td>SHOW ENDIANNESS</td>
<td>Get endianness of an HDF5 dataset or attribute</td>
</tr>
<tr>
<td>SHOW CHARSET</td>
<td>Get charset of an HDF5 dataset or attribute</td>
</tr>
<tr>
<td>SHOW STORAGE TYPE</td>
<td>Get storage type (layout) of an HDF5 dataset</td>
</tr>
<tr>
<td>SHOW STORAGE ALLOCATION</td>
<td>Get storage allocation of an HDF5 dataset</td>
</tr>
<tr>
<td>SHOW STORAGE DIMENSION</td>
<td>Get storage dimensions of an HDF5 dataset</td>
</tr>
<tr>
<td>SHOW DIMENSION</td>
<td>Get dimensions of an HDF5 dataset or attribute</td>
</tr>
<tr>
<td>SHOW ORDER</td>
<td>Get (creation) order strategy of an HDF5 group or dataset</td>
</tr>
<tr>
<td>SHOW TAG</td>
<td>Get tag of an HDF5 dataset or attribute</td>
</tr>
<tr>
<td>SHOW FILL TYPE</td>
<td>Get fill type of an HDF5 dataset</td>
</tr>
<tr>
<td>SHOW FILL VALUE</td>
<td>Get fill values of an HDF5 dataset</td>
</tr>
<tr>
<td>SHOW FILE SIZE</td>
<td>Get size (in bytes) of a file</td>
</tr>
<tr>
<td>SHOW [DATASET</td>
<td>ATTRIBUTE] SIZE</td>
</tr>
<tr>
<td>SHOW HDFQL VERSION</td>
<td>Get version of HDFql library</td>
</tr>
<tr>
<td>SHOW HDF5 VERSION</td>
<td>Get version of HDF5 library used by HDFql</td>
</tr>
<tr>
<td>SHOW PCRE VERSION</td>
<td>Get version of PCRE library used by HDFql</td>
</tr>
<tr>
<td>SHOW ZLIB VERSION</td>
<td>Get version of ZLIB library used by HDFql</td>
</tr>
<tr>
<td>SHOW MPI VERSION</td>
<td>Get version of MPI library used by HDFql</td>
</tr>
<tr>
<td>SHOW DIRECTORY</td>
<td>Get directory names within a directory</td>
</tr>
<tr>
<td>SHOW FILE</td>
<td>Get file names within a directory or check existence of a file</td>
</tr>
<tr>
<td>SHOW EXECUTE STATUS</td>
<td>Get status of the last executed operation</td>
</tr>
<tr>
<td>SHOW LIBRARY BOUNDS</td>
<td>Get library bound values for creating or opening HDF5 files</td>
</tr>
<tr>
<td>SHOW CACHE</td>
<td>Get cache parameters for accessing HDF5 files or datasets</td>
</tr>
<tr>
<td>SHOW ATOMIC</td>
<td>Get atomicity for accessing HDF5 files in an MPI environment</td>
</tr>
<tr>
<td>SHOW EXTERNAL LINK PREFIX</td>
<td>Get prefix to prepend to file names stored in HDF5 external links</td>
</tr>
<tr>
<td>Operation</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SHOW FLUSH</td>
<td>Get status of the automatic flushing</td>
</tr>
<tr>
<td>SHOW DEBUG</td>
<td>Get status of the debug mechanism</td>
</tr>
<tr>
<td>USE DIRECTORY</td>
<td>Use a directory for subsequent operations</td>
</tr>
<tr>
<td>USE FILE</td>
<td>Use (i.e. open) an HDF5 file for subsequent operations</td>
</tr>
<tr>
<td>USE GROUP</td>
<td>Use (i.e. open) an HDF5 group for subsequent operations</td>
</tr>
<tr>
<td>FLUSH</td>
<td>Flush the entire virtual HDF5 file (global) or only the HDF5 file (local)</td>
</tr>
<tr>
<td>CLOSE FILE</td>
<td>Close HDF5 file currently in use</td>
</tr>
<tr>
<td>CLOSE ALL FILE</td>
<td>Close all HDF5 files in use</td>
</tr>
<tr>
<td>CLOSE GROUP</td>
<td>Close HDF5 group currently in use</td>
</tr>
<tr>
<td>SET LIBRARY BOUNDS</td>
<td>Set library bound values for creating and opening HDF5 files</td>
</tr>
<tr>
<td>SET CACHE</td>
<td>Set cache parameters for accessing HDF5 files or datasets</td>
</tr>
<tr>
<td>SET CACHE</td>
<td>Set atomicity for accessing HDF5 files in an MPI environment to enabled or disabled</td>
</tr>
<tr>
<td>SET EXTERNAL LINK PREFIX</td>
<td>Set prefix to prepend to file names stored in HDF5 external links</td>
</tr>
<tr>
<td>SET FLUSH</td>
<td>Set automatic flushing of the entire virtual HDF5 file or only the HDF5 file (local) currently in use to enabled or disabled</td>
</tr>
<tr>
<td>SET DEBUG</td>
<td>Set debug mechanism to enabled or disabled</td>
</tr>
</tbody>
</table>

Table 6.2 – HDFql operations

### 6.1 DATA TYPES

A data type is a classification identifying one of various types of data such as integer, real or string, which determines the possible values for that type, the operations that can be done on values of that type, the meaning of the data, and the way values of that type can be stored. In other words, a data type is a classification of data that tells HDFql how the user
intends to use it. The following table summarizes all existing HDFql data types and how these map with the HDF5 data types.

<table>
<thead>
<tr>
<th>HDFql</th>
<th>HDF5</th>
<th>Range of Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>TINYINT</td>
<td>H5T_NATIVE_CHAR</td>
<td>-128 to 127 (1 byte)</td>
</tr>
<tr>
<td>UNSIGNED TINYINT</td>
<td>H5T_NATIVE_UCHAR</td>
<td>0 to 255 (1 byte)</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>H5T_NATIVE_SHORT</td>
<td>-32,768 to 32,767 (2 bytes)</td>
</tr>
<tr>
<td>UNSIGNED SMALLINT</td>
<td>H5T_NATIVE_USHORT</td>
<td>0 to 65,535 (2 bytes)</td>
</tr>
<tr>
<td>INT</td>
<td>H5T_NATIVE_INT</td>
<td>-2,147,483,648 to 2,147,483,647 (4 bytes)</td>
</tr>
<tr>
<td>UNSIGNED INT</td>
<td>H5T_NATIVE_UINT</td>
<td>0 to 4,294,967,295 (4 bytes)</td>
</tr>
<tr>
<td>BIGINT</td>
<td>H5T_NATIVE_LLONG</td>
<td>-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (8 bytes)</td>
</tr>
<tr>
<td>UNSIGNED BIGINT</td>
<td>H5T_NATIVE_ULLONG</td>
<td>0 to 18,446,744,073,709,551,615 (8 bytes)</td>
</tr>
<tr>
<td>FLOAT</td>
<td>H5T_NATIVE_FLOAT</td>
<td>-3.4E + 38 to 3.4E + 38 (4 bytes)</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>H5T_NATIVE_DOUBLE</td>
<td>-1.79E + 308 to 1.79E + 308 (8 bytes)</td>
</tr>
<tr>
<td>CHAR</td>
<td>H5T_C_S1</td>
<td>0 to 255 (1 byte)</td>
</tr>
<tr>
<td>VARTINYINT</td>
<td>H5T_NATIVE_CHAR</td>
<td>-128 to 127 (size * 1 byte)</td>
</tr>
<tr>
<td>UNSIGNED VARTINYINT</td>
<td>H5T_NATIVE_UCHAR</td>
<td>0 to 255 (size * 1 byte)</td>
</tr>
</tbody>
</table>

Table 6.3 – HDFql data types and their corresponding definitions in HDF

<table>
<thead>
<tr>
<th>Data Type</th>
<th>HDF Type</th>
<th>Range</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARSMALLINT</td>
<td>H5T_NATIVE_SHORT</td>
<td>-32,768 to 32,767 (size * 2 bytes)</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED VARSMALLINT</td>
<td>H5T_NATIVE_USHORT</td>
<td>0 to 65,535 (size * 2 bytes)</td>
<td></td>
</tr>
<tr>
<td>VARINT</td>
<td>H5T_NATIVE_INT</td>
<td>-2,147,483,648 to 2,147,483,647 (size * 4 bytes)</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED VARINT</td>
<td>H5T_NATIVE_UINT</td>
<td>0 to 4,294,967,295 (size * 4 bytes)</td>
<td></td>
</tr>
<tr>
<td>VARBIGINT</td>
<td>H5T_NATIVE_LLONG</td>
<td>-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (size * 8 bytes)</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED VARBIGINT</td>
<td>H5T_NATIVE_ULLONG</td>
<td>0 to 18,446,744,073,709,551,615 (size * 8 bytes)</td>
<td></td>
</tr>
<tr>
<td>VARFLOAT</td>
<td>H5T_NATIVE_FLOAT</td>
<td>-3.4E + 38 to 3.4E + 38 (size * 4 bytes)</td>
<td></td>
</tr>
<tr>
<td>VARDOUBLE</td>
<td>H5T_NATIVE_DOUBLE</td>
<td>-1.79E + 308 to 1.79E + 308 (size * 8 bytes)</td>
<td></td>
</tr>
<tr>
<td>VARCHAR</td>
<td>H5T_C_S1</td>
<td>0 to 255 (size * 1 byte)</td>
<td></td>
</tr>
<tr>
<td>OPAQUE</td>
<td>H5T_OPAQUE</td>
<td>0 to 255 (1 byte)</td>
<td></td>
</tr>
</tbody>
</table>

### 6.1.1 TINYINT

The HDFql TINYINT data type corresponds to the HDF5 HST_NATIVE_CHAR data type. It may store a value between -128 and 127, and occupies 1 byte in memory. Depending on the programming language (supported by HDFql), the TINYINT data type is represented by:

- In C, the “char” data type.
- In C++, the “char” data type.
- In Java, the “byte” data type or its corresponding wrapper class “Byte”.
- In Python, the “int8” NumPy data type.
• In C#, the “SByte” data type or its alias “sbyte”.

• In Fortran, the “INTEGER(KIND = 1)” data type.

• In R\(^2\), the “integer” data type.

### 6.1.2 UNSIGNED TINYINT

The HDFql UNSIGNED TINYINT data type corresponds to the HDF5 H5T_NATIVE_UCHAR data type. It may store a value between 0 and 255, and occupies 1 byte in memory. Depending on the programming language (supported by HDFql), the UNSIGNED TINYINT data type is represented by:

• In C, the “unsigned char” data type.

• In C++, the “unsigned char” data type.

• In Java\(^3\), the “byte” data type or its corresponding wrapper class “Byte”.

• In Python, the “uint8” NumPy data type.

• In C#, the “Byte” data type or its alias “byte”.

• In Fortran\(^4\), the “INTEGER(KIND = 1)” data type.

• In R\(^5\), the “integer” data type.

---

\(^2\) By design, R does not have a data type that may store a value between -128 and 127 with exactly 1 byte in memory. As a substitute, the R “integer” data type may be used with the penalties of more memory being unnecessarily reserved (as this data type occupies 4 bytes in memory) and lower performance (as bytes alignment must be made by HDFql).

\(^3\) By design, Java does not support unsigned data types. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned number in Java.

\(^4\) Although there has been some effort to specify unsigned data types in Fortran, nothing concrete is available. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned number in Fortran.

\(^5\) By design, R does not have a data type that may store a value between 0 and 255 with exactly 1 byte in memory. As a substitute, the R “integer” data type may be used with the penalties of more memory being unnecessarily reserved (as this data type occupies 4 bytes in memory) and lower performance (as bytes alignment must be made by HDFql).
6.1.3 SMALLINT

The HDFql SMALLINT data type corresponds to the HDF5 H5T_NATIVE_SHORT data type. It may store a value between -32,768 and 32,767, and occupies 2 bytes in memory. Depending on the programming language (supported by HDFql), the SMALLINT data type is represented by:

- In C, the “short” data type.
- In C++, the “short” data type.
- In Java, the “short” data type or its corresponding wrapper class “Short”.
- In Python, the “int16” NumPy data type.
- In C#, the “Int16” data type or its alias “short”.
- In Fortran, the “INTEGER(KIND = 2)” data type.
- In R6, the “integer” data type.

6.1.4 UNSIGNED SMALLINT

The HDFql UNSIGNED SMALLINT data type corresponds to the HDF5 H5T_NATIVE_USHORT data type. It may store a value between 0 and 65,535, and occupies 2 bytes in memory. Depending on the programming language (supported by HDFql), the UNSIGNED SMALLINT data type is represented by:

- In C, the “unsigned short” data type.
- In C++, the “unsigned short” data type.
- In Java7, the “short” data type or its corresponding wrapper class “Short”.
- In Python, the “uint16” NumPy data type.

---

6 By design, R does not have a data type that may store a value between -32,768 and 32,767 with exactly 2 bytes in memory. As a substitute, the R “integer” data type may be used with the penalties of more memory being unnecessarily reserved (as this data type occupies 4 bytes in memory) and lower performance (as bytes alignment must be made by HDFql).

7 By design, Java does not support unsigned data types. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned number in Java.
• In C#, the “UInt16” data type or its alias “ushort”.

• In Fortran\textsuperscript{8}, the “INTEGER(KIND = 2)” data type.

• In R\textsuperscript{9}, the “integer” data type.

\subsection*{6.1.5 INT}

The HDFql INT data type corresponds to the HDF5 \texttt{H5T_NATIVE_INT} data type. It may store a value between -2,147,483,648 and 2,147,483,647, and occupies 4 bytes in memory. Depending on the programming language (supported by HDFql), the INT data type is represented by:

• In C, the “int” data type.

• In C++, the “int” data type.

• In Java, the “int” data type or its corresponding wrapper class “Integer”.

• In Python, the “int32” NumPy data type.

• In C#, the “Int32” data type or its alias “int”.

• In Fortran, the “INTEGER(KIND = 4)” or “INTEGER” data types.

• In R, the “integer” data type.

\subsection*{6.1.6 UNSIGNED INT}

The HDFql UNSIGNED INT data type corresponds to the HDF5 \texttt{H5T_NATIVE_UINT} data type. It may store a value between 0 and 4,294,967,295, and occupies 4 bytes in memory. Depending on the programming language (supported by HDFql), the UNSIGNED INT data type is represented by:

\begin{itemize}
\end{itemize}

\textsuperscript{8} Although there has been some effort to specify unsigned data types in Fortran, nothing concrete is available. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned number in Fortran.

\textsuperscript{9} By design, R does not have a data type that may store a value between 0 and 65,535 with exactly 2 bytes in memory. As a substitute, the R “integer” data type may be used with the penalties of more memory being unnecessarily reserved (as this data type occupies 4 bytes in memory) and lower performance (as bytes alignment must be made by HDFql).
• In C, the “unsigned int” data type.

• In C++, the “unsigned int” data type.

• In Java\(^\text{10}\), the “int” data type or its corresponding wrapper class “Integer”.

• In Python, the “uint32” NumPy data type.

• In C#, the “UInt32” data type or its alias “uint”.

• In Fortran\(^\text{11}\), the “INTEGER(KIND = 4)” or “INTEGER” data types.

• In R\(^\text{12}\), the “integer” data type.

### 6.1.7 BIGINT

The HDFql BIGINT data type corresponds to the HDF5 H5T_NATIVE_LLONG data type. It may store a value between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807, and occupies 8 bytes in memory. Depending on the programming language (supported by HDFql), the BIGINT data type is represented by:

• In C, the “long long” data type.

• In C++, the “long long” data type.

• In Java, the “long” data type or its corresponding wrapper class “Long”.

• In Python, the “int64” NumPy data type.

• In C#, the “Int64” data type or its alias “long”.

• In Fortran, the “INTEGER(KIND = 8)” data type.

• In R, the “integer64” bit64 data type.

---

\(^{10}\) By design, Java does not support unsigned data types. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned in Java.

\(^{11}\) Although there has been some effort to specify unsigned data types in Fortran, nothing concrete is available. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned in Fortran.

\(^{12}\) By design, R does not support unsigned data types. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned in R.
6.1.8 **UNSIGNED BIGINT**

The HDFql UNSIGNED BIGINT data type corresponds to the HDF5 H5T_NATIVE_ULLONG data type. It may store a value between 0 and 18,446,744,073,709,551,615, and occupies 8 bytes in memory. Depending on the programming language (supported by HDFql), the UNSIGNED BIGINT data type is represented by:

- In C, the “unsigned long long” data type.
- In C++, the “unsigned long long” data type.
- In Java\(^\text{13}\), the “long” data type or its corresponding wrapper class “Long”.
- In Python, the “uint64” NumPy data type.
- In C#, the “UInt64” data type or its alias “ulong”.
- In Fortran\(^\text{14}\), the “INTEGER(KIND = 8)” data type.
- In R\(^\text{15}\), the “integer64” bit64 data type.

6.1.9 **FLOAT**

The HDFql FLOAT data type corresponds to the HDF5 H5T_NATIVE_FLOAT data type. It may store a value between -3.4E + 38 and 3.4E + 38, and occupies 4 bytes in memory. Depending on the programming language (supported by HDFql), the FLOAT data type is represented by:

- In C, the “float” data type.
- In C++, the “float” data type.

\(^{11}\) By design, Java does not support unsigned data types. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned in Java.

\(^{14}\) Although there has been some effort to specify unsigned data types in Fortran, nothing concrete is available. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned in Fortran.

\(^{15}\) By design, R does not support unsigned data types. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned in R.
In Java, the “float” data type or its corresponding wrapper class “Float”.

In Python, the “float32” NumPy data type.

In C#, the “Single” data type or its alias “float”.

In Fortran, the “REAL(KIND = 4)” or “REAL” data types.

In R\(^{16}\), the “numeric” or “double” data types.

### 6.1.10 DOUBLE

The HDFql DOUBLE data type corresponds to the HDF5 H5T_NATIVE_DOUBLE data type. It may store a value between \(-1.79E + 308\) and \(1.79E + 308\), and occupies 8 bytes in memory. Depending on the programming language (supported by HDFql), the DOUBLE data type is represented by:

- In C, the “double” data type.
- In C++, the “double” data type.
- In Java, the “double” data type or its corresponding wrapper class “Double”.
- In Python, the “float64” NumPy data type.
- In C#, the “Double” data type or its alias “double”.
- In Fortran, the “REAL(KIND = 8)” or “DOUBLE PRECISION” data types.
- In R, the “numeric” or “double” data types.

---

\(^{16}\) By design, R does not have a data type that may store a value between \(-3.4E + 38\) and \(3.4E + 38\) with exactly 4 byte in memory. As a substitute, the R “numeric” or “double” data types may be used with the penalties of more memory being unnecessarily reserved (as this data type occupies 8 bytes in memory) and lower performance (as bytes alignment must be made by HDFql).
6.1.11 CHAR

The HDFql CHAR data type corresponds to the HDF5 H5T_C_S1 data type. It may store a value between 0 and 255, and occupies size * 1 byte in memory (size being the length of the string). The CHAR data type is useful for storing fixed-length strings. Depending on the programming language (supported by HDFql), the CHAR data type is represented by:

- In C, the “unsigned char [size]” data type.
- In C++, the “unsigned char [size]” data type.
- In Java, the “byte [size]” data type or its corresponding wrapper class “Byte [size]”.
- In Python, the “Ssize” NumPy data type.
- In C#, the “Byte [size]” data type or its alias “byte [size]”.
- In Fortran, the “CHARACTER(LEN = size)” data type.
- In R\(^\text{17}\), the “integer” data type.

6.1.12 VARTINYINT

The HDFql VARTINYINT data type corresponds to the HDF5 H5T_NATIVE_CHAR data type. It may store a value between -128 and 127, and occupies size * 1 byte in memory (size being the number of elements composing the VARTINYINT data type). Depending on the programming language (supported by HDFql), the VARTINYINT data type is represented by:

- In C, the “char” data type.
- In C++, the “char” data type.
- In Java, the “byte” data type or its corresponding wrapper class “Byte”.
- In Python, the “int8” NumPy data type.
- In C#, the “SByte” data type or its alias “sbyte”.

\(^{17}\) By design, R does not have a data type that may store a value between 0 and 255 with exactly 1 byte in memory. As a substitute, the R “integer” data type may be used with the penalties of more memory being unnecessarily reserved (as this data type occupies 4 bytes in memory) and lower performance (as bytes alignment must be made by HDFql).
• In Fortran, the “INTEGER(KIND = 1)” data type.

• In R\textsuperscript{18}, the “integer” data type.

### 6.1.13 UNSIGNED VARTINYINT

The HDFql UNSIGNED VARTINYINT data type corresponds to the HDF5 H5T_NATIVE_UCHAR data type. It may store a value between 0 and 255, and occupies $size \times 1$ byte in memory ($size$ being the number of elements composing the VARTINYINT data type). Depending on the programming language (supported by HDFql), the UNSIGNED VARTINYINT data type is represented by:

• In C, the “unsigned char” data type.

• In C++, the “unsigned char” data type.

• In Java\textsuperscript{19}, the “byte” data type or its corresponding wrapper class “Byte”.

• In Python, the “uint8” NumPy data type.

• In C#, the “Byte” data type or its alias “byte”.

• In Fortran\textsuperscript{20}, the “INTEGER(KIND = 1)” data type.

• In R\textsuperscript{21}, the “integer” data type.

\textsuperscript{18} By design, R does not have a data type that may store a value between -128 and 127 with exactly 1 byte in memory. As a substitute, the R “integer” data type may be used with the penalties of more memory being unnecessarily reserved (as this data type occupies 4 bytes in memory) and lower performance (as bytes alignment must be made by HDFql).

\textsuperscript{19} By design, Java does not support unsigned data types. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned number in Java.

\textsuperscript{20} Although there has been some effort to specify unsigned data types in Fortran, nothing concrete is available. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned number in Fortran.

\textsuperscript{21} By design, R does not have a data type that may store a value between 0 and 255 with exactly 1 byte in memory. As a substitute, the R “integer” data type may be used with the penalties of more memory being unnecessarily reserved (as this data type occupies 4 bytes in memory) and lower performance (as bytes alignment must be made by HDFql).
6.1.14 VARSMALLINT

The HDFql VARSMALLINT data type corresponds to the HDF5 H5T_NATIVE_SHORT data type. It may store a value between -32,768 and 32,767, and occupies size * 2 bytes in memory (size being the number of elements composing the VARSMALLINT data type). Depending on the programming language (supported by HDFql), the VARSMALLINT data type is represented by:

- In C, the “short” data type.
- In C++, the “short” data type.
- In Java, the “short” data type or its corresponding wrapper class “Short”.
- In Python, the “int16” NumPy data type.
- In C#, the “Int16” data type or its alias “short”.
- In Fortran, the “INTEGER(KIND = 2)” data type.
- In R\(^{22}\), the “integer” data type.

6.1.15 UNSIGNED VARSMALLINT

The HDFql UNSIGNED VARSMALLINT data type corresponds to the HDF5 H5T_NATIVE_USHORT data type. It may store a value between 0 and 65,535, and occupies size * 2 bytes in memory (size being the number of elements composing the VARSMALLINT data type). Depending on the programming language (supported by HDFql), the UNSIGNED VARSMALLINT data type is represented by:

- In C, the “unsigned short” data type.
- In C++, the “unsigned short” data type.
- In Java\(^{23}\), the “short” data type or its corresponding wrapper class “Short”.

---

\(^{22}\) By design, R does not have a data type that may store a value between -32,768 and 32,767 with exactly 2 bytes in memory. As a substitute, the R “integer” data type may be used with the penalties of more memory being unnecessarily reserved (as this data type occupies 4 bytes in memory) and lower performance (as bytes alignment must be made by HDFql).

\(^{23}\) By design, Java does not support unsigned data types. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned number in Java.
In Python, the “uint16” NumPy data type.

In C#, the “UInt16” data type or its alias “ushort”.

In Fortran\(^\text{24}\), the “INTEGER(KIND = 2)” data type.

In R\(^\text{25}\), the “integer” data type.

### 6.1.16 VARINT

The HDFql VARINT data type corresponds to the HDF5 H5T_NATIVE_INT data type. It may store a value between -2,147,483,648 and 2,147,483,647, and occupies size * 4 bytes in memory (size being the number of elements composing the VARINT data type). Depending on the programming language (supported by HDFql), the VARINT data type is represented by:

- In C, the “int” data type.
- In C++, the “int” data type.
- In Java, the “int” data type or its corresponding wrapper class “Integer”.
- In Python, the “int32” NumPy data type.
- In C#, the “Int32” data type or its alias “int”.
- In Fortran, the “INTEGER(KIND = 4)” data type.
- In R, the “integer” data type.

\(^\text{24}\) Although there has been some effort to specify unsigned data types in Fortran, nothing concrete is available. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned number in Fortran.

\(^\text{25}\) By design, R does not have a data type that may store a value between 0 and 65,535 with exactly 2 bytes in memory. As a substitute, the R “integer” data type may be used with the penalties of more memory being unnecessarily reserved (as this data type occupies 4 bytes in memory) and lower performance (as bytes alignment must be made by HDFql).
6.1.17 UNSIGNED VARINT

The HDFql UNSIGNED VARINT data type corresponds to the HDF5 H5T_NATIVE_UINT data type. It may store a value between 0 and 4,294,967,295, and occupies size * 4 bytes in memory (size being the number of elements composing the UNSIGNED VARINT data type). Depending on the programming language (supported by HDFql), the UNSIGNED VARINT data type is represented by:

- In C, the “unsigned int” data type.
- In C++, the “unsigned int” data type.
- In Java, the “int” data type or its corresponding wrapper class “Integer”.
- In Python, the “uint32” NumPy data type.
- In C#, the “UInt32” data type or its alias “uint”.
- In Fortran, the “INTEGER(KIND = 4)” data type.
- In R, the “integer” data type.

6.1.18 VARBIGINT

The HDFql VARBIGINT data type corresponds to the HDF5 H5T_NATIVE_LLONG data type. It may store a value between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807, and occupies size * 8 bytes in memory (size being the number of elements composing the VARBIGINT data type). Depending on the programming language (supported by HDFql), the VARBIGINT data type is represented by:

- In C, the “long long” data type.
- In C++, the “long long” data type.

---

26 By design, Java does not support unsigned data types. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned number in Java.

27 Although there has been some effort to specify unsigned data types in Fortran, nothing concrete is available. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned number in Fortran.

28 By design, R does not support unsigned data types. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned number in R.
• In Java, the “long” data type or its corresponding wrapper class “Long”.
• In Python, the “int64” NumPy data type.
• In C#, the “Int64” data type or its alias “long”.
• In Fortran, the “INTEGER(KIND = 8)” data type.
• In R, the “integer64” bit64 data type.

6.1.19 UNSIGNED VARBIGINT

The HDFql UNSIGNED VARBIGINT data type corresponds to the HDF5 H5T_NATIVE_ULLONG data type. It may store a value between 0 and 18,446,744,073,709,551,615, and occupies size * 8 bytes in memory (size being the number of elements composing the UNSIGNED VARBIGINT data type). Depending on the programming language (supported by HDFql), the UNSIGNED VARBIGINT data type is represented by:

• In C, the “unsigned long long” data type.
• In C++, the “unsigned long long” data type.
• In Java\(^\text{29}\), the “long” data type or its corresponding wrapper class “Long”.
• In Python, the “uint64” NumPy data type.
• In C#, the “UInt64” data type or its alias “ulong”.
• In Fortran\(^\text{30}\), the “INTEGER(KIND = 8)” data type.
• In R\(^\text{31}\), the “integer64” bit64 data type.

\(^{29}\) By design, Java does not support unsigned data types. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned number in Java.

\(^{30}\) Although there has been some effort to specify unsigned data types in Fortran, nothing concrete is available. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned number in Fortran.

\(^{31}\) By design, R does not support unsigned data types. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned number in R.
6.1.20 VARFLOAT

The HDFql VARFLOAT data type corresponds to the HDF5 H5T_NATIVE_FLOAT data type. It may store a value between -3.4E + 38 and 3.4E + 38, and occupies size * 4 bytes in memory (size being the number of elements composing the VARFLOAT data type). Depending on the programming language (supported by HDFql), the VARFLOAT data type is represented by:

- In C, the “float” data type.
- In C++, the “float” data type.
- In Java, the “float” data type or its corresponding wrapper class “Float”.
- In Python, the “float32” NumPy data type.
- In C#, the “Single” data type or its alias “float”.
- In Fortran, the “REAL(KIND = 4)” data type.
- In R\(^{32}\), the “numeric” or “double” data types.

6.1.21 VARDDOUBLE

The HDFql VARDDOUBLE data type corresponds to the HDF5 H5T_NATIVE_DOUBLE data type. It may store a value between -1.79E + 308 and 1.79E + 308, and occupies size * 8 bytes in memory (size being the number of elements composing the VARDDOUBLE data type). Depending on the programming language (supported by HDFql), the VARDDOUBLE data type is represented by:

- In C, the “double” data type.
- In C++, the “double” data type.
- In Java, the “double” data type or its corresponding wrapper class “Double”.
- In Python, the “float64” NumPy data type.

---

\(^{32}\) By design, R does not have a data type that may store a value between -3.4E + 38 and 3.4E + 38 with exactly 4 byte in memory. As a substitute, the R “numeric” or “double” data types may be used with the penalties of more memory being unnecessarily reserved (as this data type occupies 8 bytes in memory) and lower performance (as bytes alignment must be made by HDFql).
• In C#, the “Double” data type or its alias “double”.

• In Fortran, the “REAL(KIND = 8)” or “DOUBLE PRECISION” data types.

• In R, the “numeric” or “double” data types.

### 6.1.22 VARCHAR

The HDFql VARCHAR data type corresponds to the HDF5 HST_C_S1 data type. It may store a value between 0 and 255, and occupies size * 1 byte in memory (size being the length of the string). The VARCHAR data type is useful for storing variable-length strings. Depending on the programming language (supported by HDFql), the VARCHAR data type is represented by:

• In C, the “unsigned char *” data type.

• In C++, the “unsigned char *” data type.

• In Java, the “String” object.

• In Python, the “Ssize” NumPy data type.

• In C#, the “String” data type or its alias “string”.

• In Fortran, the “CHARACTER(LEN = *)” data type.

• In R, the “character” data type.

### 6.1.23 OPAQUE

The HDFql OPAQUE data type corresponds to the HDF5 HST_OPAQUE data type. It may store a value between 0 and 255, and occupies 1 byte in memory. The OPAQUE data type is useful for representing data that should not be interpreted/rearranged by the HDF5 library when reading/writing it from/into in an HDF5 dataset or attribute. Depending on the programming language (supported by HDFql), the OPAQUE data type is represented by:

• In C, the “unsigned char” data type.

• In C++, the “unsigned char” data type.
• In Java\textsuperscript{33}, the “byte” data type or its corresponding wrapper class “Byte”.

• In Python, the “ubyte” NumPy data type.

• In C#, the “Byte” data type or its alias “byte”.

• In Fortran, the “CHARACTER” data type.

• In R, the “raw” data type.

### 6.2 POST-PROCESSING

Post-processing options enable transforming results of a query according to the programmer’s needs such as ordering or truncating. These options may be used to create a (linear) pipeline to further process result sets returned by DATA QUERY LANGUAGE (DQL) and DATA INTROSPECTION LANGUAGE (DIL) operations. In case a pipeline is composed of two or more options, the order in which they are used affects the final outcome (e.g. usage of ORDER ASC followed by TOP 2 in a result set composed of 4, 2, 3 and 1, returns 1 and 2; usage of these same two options inversed – i.e. TOP 2 followed by ORDER ASC – returns 2 and 4 instead). The next subsections describe the post-processing options provided by HDFql.

<table>
<thead>
<tr>
<th>Post-processing Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDER</td>
<td>Order (i.e. sort) a result set in an ascending, descending or reverse way</td>
</tr>
<tr>
<td>TOP</td>
<td>Truncate a result set after a certain given position in a topmost way</td>
</tr>
<tr>
<td>BOTTOM</td>
<td>Truncate a result set after a certain given position in a bottommost way</td>
</tr>
<tr>
<td>FROM TO</td>
<td>Retain a result set within a certain given range</td>
</tr>
<tr>
<td>STEP</td>
<td>Step (i.e. jump) the result set at every given position</td>
</tr>
</tbody>
</table>

Table 6.4 – HDFql post-processing options

\[\text{\textsuperscript{33} By design, Java does not support unsigned data types. Therefore, the programmer is responsible for making the conversion from a signed number to its equivalent unsigned number in Java.}\]
6.2.1 ORDER

Syntax

ORDER {{ASC | DESC | REV} | (, {ASC | DESC | REV}) | {{ASC | DESC | REV}, {ASC | DESC | REV}}}

Description

Order (i.e. sort) a result set in an ascending, descending or reverse way by specifying either the keyword ASC, DESC or REV respectively. When in an ascending or descending order, HDFql automatically uses all available CPU cores to speed-up the task completion. Additionally, if the result set is of data type HDFQL_CHAR, HDFQL_VARTINYINT, HDFQL_UNSIGNED_VARTINYINT, HDFQL_VARSMALLINT, HDFQL_UNSIGNED_VARSMALLINT, HDFQL_VARINT, HDFQL_UNSIGNED_VARINT, HDFQL_VARBIGINT, HDFQL_UNSIGNED_VARBIGINT, HDFQL_VARFLOAT, HDFQL_VARDOUBLE or HDFQL_OPAQUE, the result subset can be ordered (i.e. sorted) in an ascending, descending or reverse way by specifying a comma (,) and either the keyword ASC, DESC or REV, respectively.

Parameter(s)

None

Return

The result set and/or subset is ordered (i.e. sorted) in an ascending, descending or reverse way depending on whether the keyword ASC, DESC or REV is specified respectively.

Example(s)

```bash
# create an HDF5 dataset named "my_dataset0" of data type float of four dimensions (size 5x8x4x7)
CREATE DATASET my_dataset0 AS FLOAT(5, 8, 4, 7)

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with these (should be 5, 8, 4, 7)
SHOW DIMENSION my_dataset0

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with these in ascending order (should be 4, 5, 7, 8)
```

34 Through a parallelized Quicksort algorithm.
SHOW DIMENSION my_dataset0 ORDER ASC

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with these in
descending order (should be 8, 7, 5, 4)
SHOW DIMENSION my_dataset0 ORDER DESC

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with these in
reversed order (should be 7, 4, 8, 5)
SHOW DIMENSION my_dataset0 ORDER REV

# create an HDF5 dataset named "my_dataset1" of data type double of two dimensions (size 3x2)
CREATE DATASET my_dataset1 AS DOUBLE(3, 2)

# insert (i.e. write) values into dataset "my_dataset1"
INSERT INTO my_dataset1 VALUES((3.2, 1.3), (0, 0.2), (9.1, 6.5))

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with it (should be 3.2, 1.3, 0, 0.2, 9.1, 6.5)
SELECT FROM my_dataset1

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with it in
ascending order (should be 0, 0.2, 1.3, 3.2, 6.5, 9.1)
SELECT FROM my_dataset1 ORDER ASC

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with it in
descending order (should be 9.1, 6.5, 3.2, 1.3, 0.2, 0)
SELECT FROM my_dataset1 ORDER DESC

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with it in
reversed order (should be 6.5, 9.1, 0.2, 0, 1.3, 3.2)
SELECT FROM my_dataset1 ORDER REV

# create an HDF5 dataset named "my_dataset2" of data type variable-length double of one
dimension (size 3)
CREATE DATASET my_dataset2 AS VARDOUBLE(3)

# insert (i.e. write) values into dataset "my_dataset2"
INSERT INTO my_dataset2 VALUES((3.2, 1.3), (0, 0.2), (9.1, 7.4, 6.5))

# select (i.e. read) data from dataset "my_dataset2" and populate cursor in use with it (should be 3.2, 1.3, 0, 0.2, 9.1, 7.4, 6.5)
SELECT FROM my_dataset2
Hierarchical Data Format query language (HDFql) ___________________________________________ Reference Manual

# select (i.e. read) data from dataset "my_dataset2" and populate cursor in use with it in ascending order on the result subset only (should be 1.3, 3.2, 0, 0.2, 6.5, 7.4, 9.1)
SELECT FROM my_dataset2 ORDER , ASC

# select (i.e. read) data from dataset "my_dataset2" and populate cursor in use with it in descending order on the result subset only (should be 3.2, 1.3, 0.2, 0, 9.1, 7.4, 6.5)
SELECT FROM my_dataset2 ORDER , DESC

# select (i.e. read) data from dataset "my_dataset2" and populate cursor in use with it in reversed order on the result set only (should be 9.1, 7.4, 6.5, 0, 0.2, 3.2, 1.3)
SELECT FROM my_dataset2 ORDER REV

# select (i.e. read) data from dataset "my_dataset2" and populate cursor in use with it in reversed order on the result subset only (should be 1.3, 3.2, 0.2, 0, 6.5, 7.4, 9.1)
SELECT FROM my_dataset2 ORDER , REV

# select (i.e. read) data from dataset "my_dataset2" and populate cursor in use with it in reversed order on both the result set and result subset (should be 6.5, 7.4, 9.1, 0.2, 0, 1.3, 3.2)
SELECT FROM my_dataset2 ORDER REV, REV

6.2.2 TOP

Syntax

TOP {top_value | {, subtop_value} | {top_value, subtop_value}}

Description

Truncate a result set after position top_value in a topmost way. In other words, all elements after position top_value are discarded from the result set. Additionally, if the result set is of data type HDFQL_CHAR, HDFQL_VARTINYINT, HDFQL_UNSIGNED_VARTINYINT, HDFQL_VARSMALLINT, HDFQL_UNSIGNED_VARSMALLINT, HDFQL_VARINT, HDFQL_UNSIGNED_VARINT, HDFQL_VARBIGINT, HDFQL_UNSIGNED_VARBIGINT, HDFQL_VARFLOAT, HDFQL_VARDOUBLE or HDFQL_OPAQUE, the result subset can be truncated in a topmost way by specifying a comma (,) and subtop_value.

Parameter(s)

top_value – optional integer that specifies the position of the truncation of a result set in a topmost way. If negative, the TOP option will behave as the BOTTOM option with a positive top_value.
**subtop_value** – optional integer that specifies the position of the truncation of a result set in a topmost way. If negative, the TOP option will behave as the BOTTOM option with a positive **subtop_value**. Of note, this parameter is only applicable for a result set of one of the aforementioned data types and ignored otherwise.

**Return**

The result set and/or subset is truncated in a topmost way in function of the position provided.

**Example(s)**

```bash
# create an HDF5 dataset named "my_dataset0" of data type float of four dimensions (size 5x8x4x7)
CREATE DATASET my_dataset0 AS FLOAT(5, 8, 4, 7)

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with these (should be 5, 8, 4, 7)
SHOW DIMENSION my_dataset0

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with the topmost (i.e. first) dimension (should be 5)
SHOW DIMENSION my_dataset0 TOP 1

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with the two topmost dimensions (should be 5, 8)
SHOW DIMENSION my_dataset0 TOP 2

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with the two bottommost dimensions (should be 4, 7)
SHOW DIMENSION my_dataset0 TOP -2

# create an HDF5 dataset named "my_dataset1" of data type variable-length int of one dimension (size 3) with initial values of 12, 14 and 16 for the first position, 18 for the second position, and 20, 22, 24 and 26 for the third position
CREATE DATASET my_dataset1 AS VARINT(3) VALUES((12, 14, 16), (18), (20, 22, 24, 26))

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with it (should be 12, 14, 16, 18, 20, 22, 24, 26)
SELECT FROM my_dataset1

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with values of the topmost (i.e. first) position (should be 12, 14, 16)
SELECT FROM my_dataset1 TOP 1
```
# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with values of the two topmost positions (should be 12, 14, 18, 20, 22)
SELECT FROM my_dataset1 TOP , 2

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with the topmost value of the two bottommost positions (should be 18, 20)
SELECT FROM my_dataset1 TOP -2, 1

### 6.2.3 BOTTOM

**Syntax**

BOTTOM \{bottom_value | \{subbottom_value\} | \{bottom_value, subbottom_value\}\}

**Description**

Truncate a result set after position \textit{bottom\_value} in a bottommost way. In other words, all elements before position \textit{bottom\_value} are discarded from the result set. Additionally, if the result set is of data type HDFQL\_CHAR, HDFQL\_VARTINYINT, HDFQL\_UNSIGNED\_VARTINYINT, HDFQL\_VARSMALLINT, HDFQL\_UNSIGNED\_VARSMALLINT, HDFQL\_VARINT, HDFQL\_UNSIGNED\_VARINT, HDFQL\_VARBIGINT, HDFQL\_UNSIGNED\_VARBIGINT, HDFQL\_VARFLOAT, HDFQL\_VARDDOUBLE or HDFQL\_OPAQUE, the result subset can be truncated in a bottommost way by specifying a comma (,) and \textit{subbottom\_value}.

**Parameter(s)**

\textit{bottom\_value} – optional integer that specifies the position of the truncation of a result set in a bottommost way. If negative, the BOTTOM option will behave as the TOP option with a positive \textit{bottom\_value}.

\textit{subbottom\_value} – optional integer that specifies the position of the truncation of a result set in a bottommost way. If negative, the BOTTOM option will behave as the TOP option with a positive \textit{subbottom\_value}. Of note, this parameter is only applicable for a result set of one of the aforementioned data types and ignored otherwise.

**Return**

The result set and/or subset is truncated in a bottommost way in function of the position provided.
Example(s)

```sql
# create an HDF5 dataset named "my_dataset0" of data type float of four dimensions (size 5x8x4x7)
CREATE DATASET my_dataset0 AS FLOAT(5, 8, 4, 7)

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with these (should be 5, 8, 4, 7)
SHOW DIMENSION my_dataset0

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with the bottommost (i.e. last) dimension (should be 7)
SHOW DIMENSION my_dataset0 BOTTOM 1

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with the two bottommost dimensions (should be 4, 7)
SHOW DIMENSION my_dataset0 BOTTOM 2

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with the two topmost dimensions (should be 5, 8)
SHOW DIMENSION my_dataset0 BOTTOM -2

# create an HDF5 dataset named "my_dataset1" of data type variable-length int of one dimension (size 3) with initial values of 12, 14 and 16 for the first position, 18 for the second position, and 20, 22, 24 and 26 for the third position
CREATE DATASET my_dataset1 AS VARINT(3) VALUES((12, 14, 16), (18), (20, 22, 24, 26))

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with it (should be 12, 14, 16, 18, 20, 22, 24, 26)
SELECT FROM my_dataset1

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with values of the bottommost (i.e. last) position (should be 20, 22, 24, 26)
SELECT FROM my_dataset1 BOTTOM 1

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with values of the two bottommost positions (should be 14, 16, 18, 24, 26)
SELECT FROM my_dataset1 BOTTOM 2

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with the bottommost value of the two topmost positions (should be 16, 18)
SELECT FROM my_dataset1 BOTTOM -2, 1
```
6.2.4 FROM TO

Syntax

FROM {from_value | {, subfrom_value} | {from_value, subfrom_value}} TO {to_value | {, subto_value} | {to_value, subto_value}}

Description

Retain a result set from from_value to to_value. In other words, all elements before position from_value and after position to_value are discarded from the result set. The first element of the result set is at position zero, while the last element is located at the position returned by hdfql_cursor_get_count - 1. Additionally, if the result set is of data type HDFQL_CHAR, HDFQL_VARTINYINT, HDFQL_UNSIGNED_VARTINYINT, HDFQL_VARSMALLINT, HDFQL_UNSIGNED_VARSMALLINT, HDFQL_VARINT, HDFQL_UNSIGNED_VARINT, HDFQL_VARBIGINT, HDFQL_UNSIGNED_VARBIGINT, HDFQL_VARFLOAT, HDFQL_VARDDOUBLE or HDFQL_OPAQUE, the result subset can be retained by specifying a comma (,), subfrom_value and/or subto_value.

Parameter(s)

from_value – optional integer that specifies the starting position to retain elements of a result set. If negative, the FROM option will retain elements of a result set starting from its end.

subfrom_value – optional integer that specifies the starting position to retain elements of a result set. If negative, the FROM option will retain elements of a result set starting from its end. Of note, this parameter is only applicable for a result set of one of the aforementioned data types and ignored otherwise.

to_value – optional integer that specifies the ending position to retain elements of a result set. If negative, the TO option will retain elements of a result set starting from its end.

subto_value – optional integer that specifies the ending position to retain elements of a result set. If negative, the TO option will retain elements of a result set starting from its end. Of note, this parameter is only applicable for a result set of one of the aforementioned data types and ignored otherwise.

Return

The result set and/or subset is retained in function of the position provided.
Example(s)

```plaintext
# create an HDF5 dataset named "my_dataset0" of data type float of four dimensions (size 5x8x4x7)
CREATE DATASET my_dataset0 AS FLOAT(5, 8, 4, 7)

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with these (should be 5, 8, 4, 7)
SHOW DIMENSION my_dataset0

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with the first, second and third dimensions (should be 5, 8, 4)
SHOW DIMENSION my_dataset0 FROM 0 TO 2

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with the second and third dimensions (should be 8, 4)
SHOW DIMENSION my_dataset0 FROM 1 TO 2

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with the second, third and fourth dimensions (should be 8, 4, 7)
SHOW DIMENSION my_dataset0 FROM -3 TO -1

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with the second and third dimensions (should be 8, 4)
SHOW DIMENSION my_dataset0 FROM 1 TO -2

# create an HDF5 dataset named "my_dataset1" of data type variable-length int of one dimension (size 3) with initial values of 12, 14 and 16 for the first position, 18 for the second position, and 20, 22, 24 and 26 for the third position
CREATE DATASET my_dataset1 AS VARINT(3) VALUES((12, 14, 16), (18), (20, 22, 24, 26))

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with it (should be 12, 14, 16, 18, 20, 22, 24, 26)
SELECT FROM my_dataset1

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with values of the second position (should be 18)
SELECT FROM my_dataset1 FROM 1 TO 1

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with values of the second and third positions (should be 18, 20, 22, 24, 26)
SELECT FROM my_dataset1 FROM -2 TO -1
```
# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with the second and third values of all positions (should be 14, 16, 22, 24)

```
SELECT FROM my_dataset1 FROM , 1 TO , 2
```

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with second bottommost and bottommost values of the first position (should be 14, 16)

```
SELECT FROM my_dataset1 FROM 0, -2 TO 0, -1
```

## 6.2.5 STEP

### Syntax

STEP `{step_value | {, substep_value} | {step_value, substep_value}}`

### Description

Step (i.e. jump) the result set at every `step_value` position. In other words, all elements between steps are discarded from the result set. Additionally, if the result set is of data type HDFQL_CHAR, HDFQL_VARTINYINT, HDFQL_UNSIGNED_VARTINYINT, HDFQL_VARSMALLINT, HDFQL_UNSIGNED_VARSMALLINT, HDFQL_VARINT, HDFQL_UNSIGNED_VARINT, HDFQL_VARBIGINT, HDFQL_UNSIGNED_VARBIGINT, HDFQL_VARFLOAT, HDFQL_VARDOUBLE or HDFQL_OPAQUE, the result subset can be stepped (i.e. jumped) by specifying a comma (,) and `substep_value`.

### Parameter(s)

- `step_value` – optional integer that specifies the position to step (i.e. jump) a result set. If `step_value` is negative, the STEP option will step (i.e. jump) the result set starting from its end.
- `substep_value` – optional integer that specifies the position to step (i.e. jump) a result set. If `substep_value` is negative, the STEP option will step (i.e. jump) the result set starting from its end. Of note, this parameter is only applicable for a result set of one of the aforementioned data types and ignored otherwise.

### Return

The result set and/or subset is stepped (i.e. jumped) in function of the position provided.

### Example(s)

```
# create an HDF5 dataset named "my_dataset0" of data type float of four dimensions (size 5x8x4x7)
```
CREATE DATASET my_dataset0 AS FLOAT(5, 8, 4, 7)

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with these
# (should be 5, 8, 4, 7)
SHOW DIMENSION my_dataset0

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with these
# (should be 5, 8, 4, 7)
SHOW DIMENSION my_dataset0 STEP 1

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with every
# second dimension (should be 5, 4)
SHOW DIMENSION my_dataset0 STEP 2

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with every
# second dimension starting from the end (should be 8, 7)
SHOW DIMENSION my_dataset0 STEP -2

# show (i.e. get) dimensions of dataset "my_dataset0" and populate cursor in use with every
# third dimension (should be 5, 7)
SHOW DIMENSION my_dataset0 STEP 3

# create an HDF5 dataset named "my_dataset1" of data type variable-length int of one dimension
# (size 3) with initial values of 12, 14 and 16 for the first position, 18 for the second
# position, and 20, 22, 24 and 26 for the third position
CREATE DATASET my_dataset1 AS VARINT(3) VALUES((12, 14, 16), (18), (20, 22, 24, 26))

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with it (should
# be 12, 14, 16, 18, 20, 22, 24, 26)
SELECT FROM my_dataset1

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with values of
# every second position (should be 12, 14, 16, 20, 22, 24, 26)
SELECT FROM my_dataset1 STEP 2

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with every
# third value of all positions (should be 12, 18, 20, 26)
SELECT FROM my_dataset1 STEP 3

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with every
# second value of every second position (should be 12, 16, 22, 26)
SELECT FROM my_dataset1 STEP 2,-2
6.3 REDIRECTING

Redirecting options enable reading data from the cursor in use, a (text or binary) file or memory (i.e. user-defined variable) and writing it into an HDF5 dataset or attribute through a CREATE DATASET, CREATE ATTRIBUTE or INSERT operation. It also enables writing result sets (i.e. data) returned by DATA QUERY LANGUAGE (DQL) and DATA INTROSPECTION LANGUAGE (DIL) operations into the cursor in use, a (text or binary) file or memory. The next subsections describe the redirecting options provided by HDFql.

<table>
<thead>
<tr>
<th>Redirecting Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM</td>
<td>Read data from the cursor in use, a file or memory and write it into an HDF5 dataset or attribute</td>
</tr>
<tr>
<td>INTO</td>
<td>Write result sets into the cursor in use, a file or memory</td>
</tr>
</tbody>
</table>

Table 6.5 – HDFql redirecting options

6.3.1 FROM

Syntax

FROM {CURSOR | {[DOS | UNIX] [TEXT] FILE file_name [SEPARATOR {separator_value | {, subseparator_value} | {separator_value, subseparator_value}]}]} | {BINARY FILE file_name} | {MEMORY variable_number [SIZE variable_size]}}

Description

Read data from the cursor in use (default behavior when no redirecting option is specified), a (text or binary) file or memory (i.e. user-defined variable) and write it into an HDF5 dataset or attribute through a CREATE DATASET, CREATE ATTRIBUTE or INSERT operation. This procedure (which is known as input redirecting option) can be performed from:

- The cursor in use. Example: “CREATE DATASET my_dataset AS FLOAT VALUES FROM CURSOR” or “INSERT INTO my_dataset VALUES FROM CURSOR”.
- A text file using optional parameters such as which terminator to use – DOS (CR+LF) or UNIX (LF) – for the end of line (EOL), or the separator to use between elements (of the data). Example: “CREATE DATASET my_dataset AS FLOAT VALUES FROM TEXT FILE my_file.txt” or “INSERT INTO my_dataset VALUES FROM TEXT FILE my_file.txt”.
• A binary file. Example: “CREATE DATASET my_dataset AS FLOAT VALUES FROM BINARY FILE my_file.bin” or “INSERT INTO my_dataset VALUES FROM BINARY FILE my_file.bin”.

• A user-defined variable that was previously registered through the function hdfql_variable_register. Example: “CREATE DATASET my_dataset AS FLOAT VALUES FROM MEMORY 0” or “INSERT INTO my_dataset VALUES FROM MEMORY 2”. Of note, when working in Java, HDFql has to copy each element of the Java variable into the HDF5 dataset or attribute (managed by the underlying HDFql C library) as the JVM does not provide a direct access to the memory associated to the variable, which induces a performance penalty. This penalty is not present when working in other programming languages supported by HDFql – namely C, C++, Python, C#, Fortan and R – as these provide a way for the underlying HDFql C library to access the variable directly.

**Parameter(s)**

- **file_name** – optional string that specifies the name of a text or binary file to read data from.
- **separator_value** – optional string that specifies the separator to use between elements (of the data) when reading these from a text file. If not specified, its default value is a comma (,).
- **subseparator_value** – optional string that specifies the subseparator to use between elements (of the data) when reading these from a text file. The subseparator is only applicable when the data type of the HDF5 dataset or attribute is either HDFQL_VARTINYINT, HDFQL_UNSIGNED_VARTINYINT, HDFQL_VARSMALLINT, HDFQL_UNSIGNED_VARSMALLINT, HDFQL_VARINT, HDFQL_UNSIGNED_VARINT, HDFQL_VARBIGINT, HDFQL_UNSIGNED_VARBIGINT, HDFQL_VARFLOAT, HDFQL_VARDOUBLE or HDFQL_OPAQUE, and ignored otherwise. If not specified, its default value is a space.
- **variable_number** – optional integer that specifies the number of the variable whose data will be written into the HDF5 dataset or attribute. The number is returned by the function hdfql_variable_register upon registering the variable or, subsequently, returned by the function hdfql_variable_get_number.
- **variable_size** – optional integer that specifies the maximum size (in bytes) of the data stored in the variable to write into the HDF5 dataset or attribute. In other words, only the first variable_size bytes of the data stored in the variable will be written into the dataset or attribute. Of note, variable_size may be smaller than the size (in bytes) of the dataset or attribute (in this case, the remainder of the dataset or attribute will be zeroed if a number or emptied if a string).

**Example(s)**

```
# use (i.e. open) an HDF5 file named "my_file.h5"
USE FILE my_file.h5

# show (i.e. get) HDF5 file currently in use and populate cursor in use with it
```
SHOW USE FILE

# create an HDF5 dataset named "my_dataset0" of data type variable-length char with initial values from the cursor in use
CREATE DATASET my_dataset0 AS VARCHAR VALUES FROM CURSOR

# select (i.e. read) data from dataset "my_dataset0" and populate cursor in use with it (should be "my_file.h5")
SELECT FROM my_dataset0

# create an HDF5 dataset named "my_dataset1" of data type char of one dimension (size 3)
CREATE DATASET my_dataset1 AS TINYINT(3)

# insert (i.e. write) values from a text file named "my_file0.txt" into dataset "my_dataset1" (assume that the file "my_file0.txt" exists and contains "65,66,67")
INSERT INTO my_dataset1 VALUES FROM FILE my_file0.txt

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with it (should be 65, 66, 67)
SELECT FROM my_dataset1

# insert (i.e. write) values from a text file named "my_file1.txt" into dataset "my_dataset1" (assume that the file "my_file1.txt" exists and contains "90**92**94")
INSERT INTO my_dataset1 VALUES FROM TEXT FILE my_file1.txt SEPARATOR **

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with it (should be 90, 92, 94)
SELECT FROM my_dataset1

# insert (i.e. write) values from binary file "my_file.bin" into dataset "my_dataset1" (assume that the file "my_file.bin" exists and contains "ABC")
INSERT INTO my_dataset1 VALUES FROM BINARY FILE my_file.bin

# select (i.e. read) data from dataset "my_dataset1" and populate cursor in use with it (should be 65, 66, 67)
SELECT FROM my_dataset1

// declare variables
char script[1024];
double data[3][2];
int x;
int y;

// create an HDF5 dataset named "my_dataset2" of data type double of two dimensions (size 3x2)
hdfql_execute("CREATE DATASET my_dataset2 AS DOUBLE(3, 2)");

// populate variable "data" with certain values
data[0][0] = 3.2;
data[0][1] = 1.3;
data[1][0] = 0;
data[1][1] = 0.2;
data[2][0] = 9.1;
data[2][1] = 6.5;

// register variable "data" for subsequent use (by HDFql)
hdfql_variable_register(data);

// prepare script to insert (i.e. write) values from variable "data" into dataset "my_dataset2"
sprintf(script, "INSERT INTO my_dataset2 VALUES FROM MEMORY %d",
hdfql_variable_get_number(data));

// execute script
hdfql_execute(script);

// unregister variable "data" as it is no longer used/needed (by HDFql)
hdfql_variable_unregister(data);

// select (i.e. read) data from dataset "my_dataset2" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset2");

// display content of cursor in use (should be 3.2, 1.3, 0, 0.2, 9.1, 6.5)
while(hdfql_cursor_next(NULL) == HDFQL_SUCCESS)
{
    printf("%f\n", *hdfql_cursor_get_double(NULL));
}

// declare variables
char script[1024];
HDFQL_VARIABLE_LENGTH data[3];

// create an HDF5 dataset named "my_dataset3" of data type variable-length double of one
dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset3 AS VARDDOUBLE(3)");  

// allocate memory in variable "data"
.data[0].address = malloc(2 * sizeof(double));
data[0].count = 2;
data[1].address = malloc(3 * sizeof(double));
data[1].count = 3;
data[2].address = malloc(1 * sizeof(double));
data[2].count = 1;

// populate variable "data" with certain values
*((double *) data[0].address + 0) = 3.2;
*((double *) data[0].address + 1) = 1.3;
*((double *) data[1].address + 0) = 0;
*((double *) data[1].address + 1) = 0.2;
*((double *) data[1].address + 2) = 9.1;
*((double *) data[2].address + 0) = 6.5;

// register variable "data" for subsequent use (by HDFql)
hdfql_variable_register(data);

// prepare script to insert (i.e. write) values from variable "data" into dataset "my_dataset3"
sprintf(script, "INSERT INTO my_dataset3 VALUES FROM MEMORY %d", hdfql_variable_get_number(data));

// execute script
hdfql_execute(script);

// unregister variable "data" as it is no longer used/needed (by HDFql)
hdfql_variable_unregister(data);

// select (i.e. read) data from dataset "my_dataset3" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset3");

// display content of cursor in use (should be 3.2, 1.3, 0, 0.2, 9.1, 6.5)
while(hdfql_cursor_next(NULL) == HDFQL_SUCCESS)
{
    while(hdfql_subcursor_next(NULL) == HDFQL_SUCCESS)
    {
        printf("%f\n", *hdfql_subcursor_get_double(NULL));
    }
}

// release memory allocated in variable "data"
free(data[0].address);
free(data[1].address);
free(data[2].address);

// declare variables
char script[1024];
char *data[3];

// create an HDF5 dataset named "my_dataset4" of data type variable-length char of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset4 AS VARCHAR(3)");

// allocate memory in variable "data"
data[0] = malloc(13 * sizeof(char));
data[1] = malloc(5 * sizeof(char));
data[2] = malloc(7 * sizeof(char));

// populate variable "data" with certain values
strcpy(data[0], "Hierarchical");
strcpy(data[1], "Data");
strcpy(data[2], "Format");

// register variable "data" for subsequent use (by HDFql)
hdfql_variable_register(data);

// prepare script to insert (i.e. write) values from variable "data" into dataset "my_dataset4"
sprintf(script, "INSERT INTO my_dataset4 VALUES FROM MEMORY %d",
hdfql_variable_get_number(data));

// execute script
hdfql_execute(script);

// unregister variable "data" as it is no longer used/needed (by HDFql)
hdfql_variable_unregister(data);

// select (i.e. read) data from dataset "my_dataset4" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset4");

// display content of cursor in use (should be "Hierarchical", "Data", "Format")
while(hdfql_cursor_next(NULL) == HDFQL_SUCCESS)
{
    printf("%s\n", hdfql_cursor_get_char(NULL));
}
6.3.2 INTO

**Syntax**

```
INTO {CURSOR | [TRUNCATE] [DOS | UNIX] [TEXT] FILE file_name [SEPARATOR {separator_value | {, subseparator_value} | {separator_value, subseparator_value}}] [SPLIT split_value] | [TRUNCATE] BINARY FILE file_name | {MEMORY variable_number [SIZE variable_size]}}
```

**Description**

Write result sets (i.e. data) returned by DATA QUERY LANGUAGE (DQL) and DATA INTROSPECTION LANGUAGE (DIL) operations into the cursor in use (default behavior when no redirecting option is specified), a (text or binary) file or memory (i.e. user-defined variable). This procedure (which is known as output redirecting option) can be performed into:

- The cursor in use. Example: “SELECT FROM my_dataset INTO CURSOR” or “SHOW USE DIRECTORY INTO CURSOR”.

- A text file using optional parameters such as which terminator to use – DOS (CR+LF) or UNIX (LF) – for the end of line (EOL), which separator to use between elements (of the result set), or the number of elements to write per line before starting writing remaining elements in a new line. Example: “SELECT FROM my_dataset INTO TEXT FILE my_file.txt” or “SHOW USE DIRECTORY INTO TEXT FILE output.txt”.

- A binary file. Example: “SELECT FROM my_dataset INTO BINARY FILE my_file.bin” or “SHOW USE DIRECTORY INTO BINARY FILE output.bin”. When redirecting data of type HDFQL_VARTINYINT, HDFQL_UNSIGNED_VARTINYINT, HDFQL_VARSMALLINT, HDFQL_UNSIGNED_VARSMALLINT, HDFQL_VARINT, HDFQL_UNSIGNED_VARINT, HDFQL_VARBIGINT, HDFQL_UNSIGNED_VARBIGINT, HDFQL_VARFLOAT or HDFQL_VARDOUBLE into a binary file, each result subset to be written is preceded by its number of elements (as a C “unsigned int” data type with a 4 bytes size). This is to enable a correct interpretation/parsing of the binary file when reading it afterwards.

- A user-defined variable that was previously registered through the function hdfql_variable_register. Example: “SELECT FROM my_dataset INTO MEMORY 0” or “SHOW USE DIRECTORY INTO MEMORY 2”. When redirecting data of type HDFQL_VARTINYINT, HDFQL_UNSIGNED_VARTINYINT, HDFQL_VARSMALLINT, HDFQL_UNSIGNED_VARSMALLINT,
HDFQL_VARINT, HDFQL_UNSIGNED_VARINT, HDFQL_VARBIGINT, HDFQL_UNSIGNED_VARBIGINT, HDFQL_VARFLOAT, HDFQL_VARDOUBLE or HDFQLVARCHAR into a user-defined variable, the programmer is responsible for releasing the memory (allocated by HDFql) afterwards. Of note, when working in Java, HDFql has to copy each element of the result set (managed by the underlying HDFql C library) into the Java variable as the JVM does not provide a direct access to the memory associated to the variable, which induces a performance penalty. This penalty is not present when working in other programming languages supported by HDFql – namely C, C++, Python, C#, Fortan and R – as these provide a way for the underlying HDFql C library to access the memory of the variable directly.

When redirecting a result set into a file that already exists, the result set is appended to it. To overwrite an existing file, specify the keyword TRUNCATE (all data stored in the file will be permanently lost).

Parameter(s)

*file_name* – optional string that specifies the name of a text or binary file to redirect (i.e. write) a result set into.

*separater_value* – optional string that specifies the separator to use between elements (of the result set) when redirecting (i.e. writing) these in a text file. If not specified, its default value is a comma (,).

*subseparater_value* – optional string that specifies the subseparator to use between elements (of the result subset) when redirecting (i.e. writing) these in a text file. The subseparator is only applicable when the data type of the result set is either HDFQL_VARTINYINT, HDFQL_UNSIGNED_VARTINYINT, HDFQL_VARSMALLINT, HDFQL_UNSIGNED_VARSMALLINT, HDFQL_VARINT, HDFQL_UNSIGNED_VARINT, HDFQL_VARBIGINT, HDFQL_UNSIGNED_VARBIGINT, HDFQL_VARFLOAT, HDFQL_VARDOUBLE or HDFQL_OPAQUE, and ignored otherwise. If not specified, its default value is a space.

*split_value* – optional integer that specifies the number of elements (of the result set) to redirect (i.e. write) per line before starting writing remaining elements in a new line in a text file. If *split_value* is specified it must be equal to or greater than zero (otherwise an error will be raised). Otherwise, if it is not specified, no splitting is done which means that all elements (of the result set) are redirectered (i.e. written) in the same line.

*variable_number* – optional integer that specifies the number of the variable that will store the result set (i.e. data) returned by DATA QUERY LANGUAGE (DQL) and DATA INTROSPECTION LANGUAGE (DIL) operations. The number is returned by the function hdfql_variable_register upon registering the variable or, subsequently, returned by the function hdfql_variable_get_number.

*variable_size* – optional integer that specifies the maximum size (in bytes) to use from the variable to store the result set (i.e. data). In other words, only the first *variable_size* bytes of the variable will be used to store the result set. Of note, *variable_size* may be smaller than the size (in bytes) of the result set (in this case, the remainder of the result set is discarded). If *variable_size* is specified it must be equal to or greater than zero (otherwise an error will be raised).
Otherwise, if it is not specified, the variable must have enough space to store the entire result set (otherwise an error may occur such as a segmentation fault).

Example(s)

```sql
CREATE DATASET my_dataset0 AS TINYINT(3)

INSERT INTO my_dataset0 VALUES(65, 66, 67)

SELECT FROM my_dataset0

SELECT FROM my_dataset0 INTO CURSOR

SELECT FROM my_dataset0 INTO FILE my_file0.txt

SELECT FROM my_dataset0 INTO TEXT FILE my_file1.txt SEPARATOR **

SELECT FROM my_dataset0 INTO UNIX TEXT FILE my_file2.txt SPLIT 2

SELECT FROM my_dataset0 INTO TRUNCATE BINARY FILE my_file.bin

// declare variables
char script[1024];
double data[3][2];
int x;
int y;
```
// create an HDF5 dataset named "my_dataset1" of data type double of two dimensions (size 3x2)
hdfql_execute("CREATE DATASET my_dataset1 AS DOUBLE(3, 2)");

// insert (i.e. write) values into dataset "my_dataset1"
hdfql_execute("INSERT INTO my_dataset1 VALUES((3.2, 1.3), (0, 0.2), (9.1, 6.5))");

// register variable "data" for subsequent use (by HDFql)
hdfql_variable_register(data);

// prepare script to select (i.e. read) data from dataset "my_dataset1" and populate variable "data" with it
sprintf(script, "SELECT FROM my_dataset1 INTO MEMORY %d", hdfql_variable_get_number(data));

// execute script
hdfql_execute(script);

// unregister variable "data" as it is no longer used/needed (by HDFql)
hdfql_variable_unregister(data);

// display content of variable "data" (should be 3.2, 1.3, 0, 0.2, 9.1, 6.5)
for(x = 0; x < 3; x++)
{
    for(y = 0; y < 2; y++)
    {
        printf("%d
", data[x][y]);
    }
}

// declare variables
char script[1024];
HDFQL_VARIABLE_LENGTH data[3];
int x;
int y;
int count;

// create an HDF5 dataset named "my_dataset2" of data type variable-length double of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset2 AS VARDouble(3)");

// insert (i.e. write) values into dataset "my_dataset2"
hdfql_execute("INSERT INTO my_dataset2 VALUES((3.2, 1.3), (0, 0.2), (9.1, 7.4, 6.5))");
// register variable "data" for subsequent use (by HDFql)
  hdfql_variable_register(data);

// prepare script to select (i.e. read) data from dataset "my_dataset2" and populate variable "data" with it
  sprintf(script, "SELECT FROM my_dataset2 INTO MEMORY %d", hdfql_variable_get_number(data));

// execute script
  hdfql_execute(script);

// unregister variable "data" as it is no longer used/needed (by HDFql)
  hdfql_variable_unregister(data);

// display content of cursor in use (should be 3.2, 1.3, 0, 0.2, 9.1, 7.4, 6.5)
  for(x = 0; x < 3; x++)
  {
    count = data[x].count;
    for(y = 0; y < count; y++)
    {
      printf("%f
");
    }
  }

// release memory allocated (by HDFql) in variable "data"
  for(x = 0; x < 3; x++)
  {
    free(data[x].address);
  }

// declare variables
  char script[1024];
  char *data[3];
  int x;

// create an HDF5 dataset named "my_dataset3" of data type variable-length char of one dimension (size 3)
  hdfql_execute("CREATE DATASET my_dataset3 AS VARCHAR(3)");

// insert (i.e. write) values into dataset "my_dataset3"
  hdfql_execute("INSERT INTO my_dataset3 VALUES("Hierarchical", "Data", "Format");
}
// register variable "data" for subsequent use (by HDFql)
hdfql_variable_register(data);

// prepare script to select (i.e. read) data from dataset "my_dataset3" and populate variable "data" with it
sprintf(script, "SELECT FROM my_dataset3 INTO MEMORY %d", hdfql_variable_get_number(data));

// execute script
hdfql_execute(script);

// unregister variable "data" as it is no longer used/needed (by HDFql)
hdfql_variable_unregister(data);

// display content of cursor in use (should be "Hierarchical", "Data", "Format")
for(x = 0; x < 3; x++)
{
    printf("%s
", data[x]);
}

// release memory allocated (by HDFql) in variable "data"
for(x = 0; x < 3; x++)
{
    free(data[x]);
}

### 6.4 DATA DEFINITION LANGUAGE (DDL)

Data Definition Language (DDL) is, generally speaking, syntax for defining and modifying structures that store data. In HDFql, the DDL assembles the operations that enable the creation, alteration, renaming, copying and deletion of HDF5 files, groups, datasets, attributes and links. These operations begin either with the keyword CREATE, ALTER, RENAME, COPY or DROP.

#### 6.4.1 CREATE DIRECTORY

**Syntax**

```plaintext
CREATE DIRECTORY directory_name [, directory_name]*
```
Description

Create a directory named directory_name. Multiple directories can be created at once by separating these with a comma (,). If directory_name already exists, it will not be overwritten, no subsequent directories are created, and an error is raised. In case directory_name has intermediate directories that do not exist, besides directory_name being created, all these intermediate directories will be created on the fly (e.g. when creating the directory “my_directory/my_subdirectory/my_subsubdirectory”, besides “my_subsubdirectory” being created, “my_directory” and “my_subdirectory” will be created in case they do not exist).

Parameter(s)

directory_name – mandatory string that specifies the name of the directory to create. Multiple directories are separated with a comma (,). As a general rule, in case directory_name is composed of spaces, special characters or reserved keywords (e.g. SELECT), it should be surrounded by double-quotes ("). Otherwise, if it is not surrounded by double-quotes, the directory will not be created and an error is raised. This rule also applies to any other HDFql operation that works with directory names (e.g. RENAME DIRECTORY).

Return

Nothing

Example(s)

```
# create a directory named "my_directory0" (the directory will not be overwritten if it already exists)
CREATE DIRECTORY my_directory0

# create a directory named "my_directory1" in a root directory named "data" (neither directory will be overwritten if they already exist; directory "data" will be created on the fly if it does not exist)
CREATE DIRECTORY /data/my_directory1

# create two directories named "my_directory2" and "my_directory3" (neither directory will be overwritten if they already exist)
CREATE DIRECTORY my_directory2, my_directory3

# create a directory named "this is a long directory name" (the directory will not be overwritten if it already exists)
CREATE DIRECTORY "this is a long directory name"
```
6.4.2  CREATE FILE

Syntax

CREATE [TRUNCATE] [PARALLEL] FILE file_name [, file_name]*

[LIBRARY BOUNDS [FROM {EARLIEST | LATEST | V18 | DEFAULT}] [TO {LATEST | V18 | DEFAULT}]]

Description

Create an HDF5 file named file_name. Multiple files can be created at once by separating these with a comma (,). If file_name already exists, it will not be overwritten, no subsequent files are created, and an error is raised. To overwrite an existing file, specify the keyword TRUNCATE (all data stored in the file will be permanently lost). In case the keyword PARALLEL is specified, HDFql creates the file using all the MPI processes specified upon launching the program (that employs HDFql). In case the keyword LIBRARY BOUNDS is specified, HDFql creates the file using these bounds (instead of the library bounds that may have been set through the operation SET LIBRARY BOUNDS).

Parameter(s)

file_name – mandatory string that specifies the name of the HDF5 file to create. Multiple files are separated with a comma (,). As a general rule, in case file_name is composed of spaces, special characters or reserved keywords (e.g. SELECT), it should be surrounded by double-quotes ("). Otherwise, if it is not surrounded by double-quotes, the file will not be created and an error is raised. This rule also applies to any other HDFql operation that works with file names (e.g. RENAME FILE).

Return

Nothing

Example(s)

```
# create an HDF5 file named "my_file0.h5" (the file will not be overwritten if it already exists)
CREATE FILE my_file0.h5

# create an HDF5 file named "my_file1.h5" in a root directory named "data" (the file will not be overwritten if it already exists)
```

35 This option is not allowed in Windows as HDFql does not support the parallel HDF5 (PHDF5) library in this platform currently.
CREATE FILE /data/my_file1.h5

# create two HDF5 files named "my_file2.h5" and "my_file3.h5" (both files will be overwritten if they already exist)
CREATE TRUNCATE FILE my_file2.h5, my_file3.h5

# create an HDF5 file named "my_file4.h5" (the file will not be overwritten if it already exists) with the latest version of the HDF5 library
CREATE FILE my_file4.h5 LIBRARY BOUNDS FROM LATEST TO LATEST

# create an HDF5 file named "this is a long file name.h5" (the file will not be overwritten if it already exists)
CREATE FILE "this is a long file name.h5"

# create an HDF5 file named "my_file5.h5" (the file will not be overwritten if it already exists) in parallel (i.e. all the MPI processes specified upon launching the program (that employs HDFql) will collectively create the file - e.g. if the program is launched as "mpiexec -n 3 my_program", all three MPI processes will participate in the creation of the file)
CREATE PARALLEL FILE my_file5.h5

6.4.3 CREATE GROUP

Syntax

CREATE [TRUNCATE] GROUP [file_name] group_name [, [file_name] group_name] *

[ORDER {TRACKED | INDEXED}]

[STORAGE COMPACT object_max_compact DENSE object_min_dense]

[ATTRIBUTE [ORDER {TRACKED | INDEXED}] [STORAGE COMPACT attribute_max_compact DENSE attribute_min_dense]]

Description

Create an HDF5 group named group_name. Multiple groups can be created at once by separating these with a comma (,). If group_name already exists, it will not be overwritten, no subsequent groups are created, and an error is raised. To overwrite an existing group, specify the keyword TRUNCATE (all data stored in the group will be permanently lost). In case group_name has intermediate groups that do not exist, besides group_name being created, all these intermediate groups will be created on the fly (e.g. when creating the group “my_group/my_subgroup/my_subsubgroup”, besides
“my_subsubgroup” being created, “my_group” and “my_subgroup” will be created in case they do not exist). By default, 
\_group\_name does not track objects (i.e. groups, datasets, (soft) links or external links) stored within it by their creation order. To track the creation order of objects stored in \_group\_name, the keyword ORDER TRACKED must be specified. In case the keyword ORDER INDEXED is specified, objects stored within \_group\_name are also tracked by their creation order and using an index (to speed-up retrieval of object names). By default, \_group\_name does not track attributes stored within it by their creation order. To track the creation order of attributes stored in \_group\_name, the keyword ATTRIBUTE ORDER TRACKED must be specified. In case the keyword ATTRIBUTE ORDER INDEXED is specified, attributes stored within \_group\_name are also tracked by their creation order and using an index (to speed-up retrieval of attribute names).

\textbf{Parameter(s)}

\_file\_name – optional string that specifies the name of the HDF5 file in which the group is created. If \_file\_name is specified, the file is opened on the fly, the group is created within it and, afterwards, the file is closed. Otherwise, if it is not specified, the group is created in the file currently in use. As a general rule, in case \_file\_name is composed of spaces, special characters or reserved keywords (e.g. SELECT), it should be surrounded by double-quotes ("). Otherwise, if it is not surrounded by double-quotes, the group will not be created and an error is raised. This rule also applies to any other HDFql operation that works with group names (e.g. RENAME GROUP).

\_group\_name – mandatory string that specifies the name of the HDF5 group to create. Multiple groups are separated with a comma (,). As a general rule, in case \_group\_name is composed of spaces, special characters or reserved keywords (e.g. SELECT), it should be surrounded by double-quotes ("). Otherwise, if it is not surrounded by double-quotes, the group will not be created and an error is raised. This rule also applies to any other HDFql operation that works with group names (e.g. RENAME GROUP).

\_object\_max\_compact – optional integer that specifies the maximum number of links (i.e. objects) to store in the compact format. In case the number of links (stored in \_group\_name) exceeds \_object\_max\_compact, the storage of links switches to the dense format. If not specified, its default value is 8 (defined by the HDF5 library).

\_object\_min\_dense – optional integer that specifies the minimum number of links (i.e. objects) to store in the dense format. In case the number of links (stored in \_group\_name) falls below \_object\_min\_dense, the storage of links switches to the compact format. If not specified, its default value is 6 (defined by the HDF5 library).

\_attribute\_max\_compact – optional integer that specifies the maximum number of attributes to store in the compact format. In case the number of attributes (stored in \_group\_name) exceeds \_attribute\_max\_compact, the storage of attributes switches to the dense format. If not specified, its default value is 8 (defined by the HDF5 library).
attribute_min_dense – optional integer that specifies the minimum number of attributes to store in the dense format. In case the number of attributes (stored in group_name) falls below attribute_min_dense, the storage of attributes switches to the compact format. If not specified, its default value is 6 (defined by the HDF5 library).

**Return**

Nothing

**Example(s)**

```plaintext
# create an HDF5 group named "my_group0" (the group will not be overwritten if it already exists)
CREATE GROUP my_group0

# create an HDF5 group named "my_subgroup0" in a root group named "my_group1" (neither group will be overwritten if they already exist; group "my_group1" will be created on the fly if it does not exist)
CREATE GROUP /my_group1/my_subgroup0

# create two HDF5 groups named "my_group2" and "my_group3" (both groups will be overwritten if they already exist)
CREATE TRUNCATE GROUP my_group2, my_group3

# create an HDF5 group named "this is a long group name" (the group will not be overwritten if it already exists)
CREATE GROUP "this is a long group name"

# create an HDF5 group named "my_group4" that tracks the objects’ (i.e. groups and datasets) creation order within the group and using compact storage
CREATE GROUP my_group4 ORDER TRACKED STORAGE COMPACT 10 DENSE 7

# create an HDF5 group named "my_group5" that indexes the attributes’ creation order
CREATE GROUP my_group5 ATTRIBUTE ORDER INDEXED

# use (i.e. open) an HDF5 file named "my_file.h5"
USE FILE my_file.h5

# create an HDF5 group named "my_group6" in the HDF5 file currently in use (i.e. file "my_file.h5")
```
**6.4.4 CREATE DATASET**

**Syntax**

```
CREATE GROUP my_group6

# close HDF5 file currently in use (i.e. file "my_file.h5")
CLOSE FILE

# create an HDF5 group named "my_group7" in file "my_file.h5"
CREATE GROUP my_file.h5 my_group7
```

```
CREATE GROUP my_group6

# close HDF5 file currently in use (i.e. file "my_file.h5")
CLOSE FILE

# create an HDF5 group named "my_group7" in file "my_file.h5"
CREATE GROUP my_file.h5 my_group7
```

**Description**

Create an HDF5 dataset named `dataset_name`. Multiple datasets can be created at once by separating these with a comma (,). If `dataset_name` already exists, it will not be overwritten, no subsequent datasets are created, and an error is raised. To overwrite an existing dataset, specify the keyword TRUNCATE (all data stored in the dataset will be permanently lost). In case `dataset_name` has intermediate groups that do not exist, besides `dataset_name` being created, all these intermediate groups will be created on the fly (e.g. when creating the dataset “my_group/my_subgroup/my_dataset”,...
besides “my_dataset” being created as a dataset, “my_group” and “my_subgroup” will be created as groups in case they do not exist). By default, \textit{dataset\_name} does not track attributes stored within it by their creation order. To track the creation order of attributes stored in \textit{dataset\_name}, the keyword ATTRIBUTE ORDER TRACKED must be specified. In case the keyword ATTRIBUTE ORDER INDEXED is specified, attributes stored within \textit{dataset\_name} are also tracked by their creation order and using an index (to speed-up retrieval of attribute names).

By default, if no storage type (layout) is specified and (1) the dataset is not extendible and (2) no HDF5 pre-defined filter is used, the dataset will be created as contiguous. To specify a certain storage type (layout), one of the following keywords may be employed:

- CONTIGUOUS – the data is stored in the HDF5 file in one contiguous block.
- COMPACT – the data is stored in the object header of the dataset. This storage type (layout) should only be used for data with a size limit of 65520 bytes (otherwise an error is raised).
- CHUNKED – the data is stored in equal-sized blocks or chunks of a pre-defined size. This storage type (layout) should be used when the dataset is extendible and/or HDF5 pre-defined filters are specified (otherwise an error is raised).

By default, if no storage allocation is specified, the dataset will have an early, incremental or late storage allocation depending on whether its storage type (layout) is compact, chunked or contiguous, respectively. To specify a certain storage allocation, one of the following keywords may be employed:

- EARLY – the space necessary to store the entire dataset is immediately allocated (i.e. reserved) in the HDF5 file.
- INCREMENTAL – the space necessary to store the dataset is incrementally allocated (i.e. reserved) according to the ongoing needs in the HDF5 file.
- LATE – the space necessary to store the entire dataset is only allocated (i.e. reserved) in the HDF5 file when data is written into the dataset for the first time.

To create an extendible dataset\footnote{An extendible HDF5 dataset is one whose one or more dimensions can grow. These dimensions start with an initial size and may be increased in a later stage. To be able to create an extendible dataset, the storage type (layout) of the dataset must be chunked (otherwise an error is raised). In case the storage type (layout) is not specified, HDFql will automatically set it to chunked and calculate an appropriate value for the chunk size.}, the keyword TO may be employed when specifying the dimensions that are extendible (i.e. that can grow) along with the initial size of the dimension (\textit{dataset\_dim}) and the maximum size (\textit{dataset\_max\_dim}) that it may grow to. If a dimension is expected to grow infinitely, the keyword UNLIMITED should be specified. Of note, when a dimension has an initial size of one and is expected to grow infinitely, the keyword TO along with \textit{dataset\_dim} and \textit{dataset\_max\_dim} may simply be replaced by the keyword UNLIMITED.
In case the keyword ENABLE is specified, one or more HDF5 pre-defined filters\(^{37}\) may be used to create a (linear) pipeline by additionally specifying one or more of the following keywords:

- **NBIT** – Compresses the data of an n-bit data type (including arrays and the n-bit fields of compound data types) by packing n-bit data on output (i.e. stripping off all unused bits) and unpacking on input (i.e. restoring the extra bits required by the computation). This filter may only be used for integer and floating-point data types (otherwise an error is raised).

- **SCALEOFFSET** – Compresses the data by performing a scale and/or offset operation on each element and truncates the result to a minimum number of bits. This filter may only be used for integer and floating-point data types (otherwise an error is raised).

- **SHUFFLE** – Rearranges the bytes in the chunk by de-interlacing a block of data, which may lead to a better compression ratio. This filter is usually used in conjunction with the ZLIB filter.

- **ZLIB** – Compresses the data using the ZLIB library which is based on the Deflate lossless data compression algorithm.

- **FLETCHER32** – Adds a checksum to each chunk to detect data corruption. In case a chunk gets corrupted, any attempt to read it afterwards will raise an error.

**Parameter(s)**

*chunk_dim* – optional integer that specifies the chunk size of the dimension in question. Multiple chunk sizes are separated with a comma (,). If *chunk_dim* is specified it must be equal to or greater than one (otherwise an error will be raised). Otherwise, if it is not specified and in case the keyword CHUNKEKED is specified, HDFql will automatically calculate an appropriate value\(^{38}\) and assign it to *chunk_dim*.

*file_name* – optional string that specifies the name of the HDF5 file in which the dataset is created. If *file_name* is specified, the file is opened on the fly, the dataset is created within it and, afterwards, the file is closed. Otherwise, if it is not specified, the dataset is created in the file currently in use. As a general rule, in case *file_name* is composed of spaces, special characters or reserved keywords (e.g. SELECT), it should be surrounded by double-quotes (“”). Otherwise, if it is not surrounded by double-quotes, the dataset will not be created and an error is raised. This rule also applies to any other HDFql operation that works with dataset names (e.g. RENAME DATASET).

---

\(^{37}\) To be able use HDF5 pre-defined filters the storage type (layout) of the HDF5 dataset must be chunked (otherwise an error is raised). In case the storage type (layout) is not specified, HDFql will automatically set it to chunked and calculate an appropriate value for the chunk size.

\(^{38}\) This calculated value may not be optimal as it is based on a best guess approach with the main purpose of alleviating the programmer from specifying it. In case performance is critical, the chunk size of the dimension in question should be explicitly specified taking into account how the data (stored in the HDF5 dataset) is accessed as it greatly influences performance (HDFql does not have enough information on how this access is ultimately done).
**dataset_name** – mandatory string that specifies the name of the HDF5 dataset to create. Multiple datasets are separated with a comma (,). As a general rule, in case **dataset_name** is composed of spaces, special characters or reserved keywords (e.g. SELECT), it should be surrounded by double-quotes ("."). Otherwise, if it is not surrounded by double-quotes, the dataset will not be created and an error is raised. This rule also applies to any other HDFql operation that works with dataset names (e.g. RENAME DATASET).

**dataset_dim** – optional integer that specifies the size of the dimension. Multiple dimensions are separated with a comma (,). If not specified, the size of the dimension is zero.

**dataset_max_dim** – optional integer that specifies the maximum size of the dimension. Multiple dimensions are separated with a comma (,). To specify an unlimited size, the keyword UNLIMITED should be specified for this purpose. If **dataset_max_dim** is specified it must be equal to or greater than **dataset_dim** and the keyword CHUNKED should be specified (otherwise an error will be raised).

**tag_value** – optional string that specifies the value of a tag attached to the HDF5 dataset. Of note, the specification of a tag is only available for a dataset of data type HDFQL_OPAQUE (any other data type will raise an error).

**fill_value** – optional integer, float or string that specifies the (default) value to return in case of reading the HDF5 dataset when no data has ever been written into it. Multiple fill values are separated with a comma (,). If not specified, the dataset will be zeroed or emptied depending on whether the dataset is a number or a string, respectively.

**attribute_max_compact** – optional integer that specifies the maximum number of attributes to store in the compact format. In case the number of attributes (stored in **dataset_name**) exceeds **attribute_max_compact**, the storage of attributes switches to the dense format. If not specified, its default value is 8 (defined by the HDF5 library).

**attribute_min_dense** – optional integer that specifies the minimum number of attributes to store in the dense format. In case the number of attributes (stored in **dataset_name**) falls below **attribute_min_dense**, the storage of attributes switches to the compact format. If not specified, its default value is 6 (defined by the HDF5 library).

**nbit_precision_value** – optional integer that specifies the precision of the N-bit filter.

**nbit_offset_value** – optional integer that specifies the offset of the N-bit filter.

**scaleoffset_value** – optional integer that specifies the offset of the scale-offset filter. The **scaleoffset_value** must be equal to or greater than zero (otherwise an error is raised). In case the HDF5 dataset is of integer data type, **scaleoffset_value** specifies the number of bits to retain (of note, if **scaleoffset_value** is zero, the HDF5 library automatically calculates the number of bits required for lossless compression). In case the dataset is of floating-point data type, **scaleoffset_value** specifies the number of digits after the decimal point to retain.
**zlib_level** – optional integer that specifies the compression level of the ZLIB filter. The **zlib_level** must be between 0 (no compression) and 9 (best compression) (otherwise an error is raised). If not specified and in case the keyword ZLIB is specified, its default value is 9.

**initial_value** – optional integer, float or string to write into the created HDF5 dataset. Multiple values are separated with a comma (,).

**input_redirecting_option** – optional option that specifies a file or memory to read data from in order to write it into the created HDF5 dataset (please refer to the subsection FROM for additional information).

**Return**

Nothing

**Example(s)**

```hdfql
# create an HDF5 dataset named "my_dataset0" of data type int (the dataset will not be overwritten if it already exists)
CREATE DATASET my_dataset0 AS INT

# create an HDF5 dataset named "my_dataset1" of data type char in a root group named "my_group" (the dataset will not be overwritten if it already exists)
CREATE DATASET /my_group/my_dataset1 AS CHAR

# create two HDF5 datasets named "my_dataset2" and "my_dataset3" of data type short (both datasets will be overwritten if they already exist)
CREATE TRUNCATE DATASET my_dataset2, my_dataset3 AS SMALLINT

# create an HDF5 dataset named "this is a long dataset name" of data type float (the dataset will not be overwritten if it already exists)
CREATE DATASET "this is a long dataset name" AS FLOAT

# create an HDF5 dataset named "my_dataset4" of data type unsigned long long using the big endian representation
CREATE DATASET my_dataset4 AS BIG ENDIAN UNSIGNED BIGINT

# create an HDF5 dataset named "my_dataset5" of data type int using the little endian representation with an initial value of 80178
CREATE DATASET my_dataset5 AS LITTLE ENDIAN INT VALUES(80178)
```
# create an HDF5 dataset named "my_dataset6" of data type char using an ASCII representation
CREATE DATASET my_dataset6 AS ASCII CHAR

# create an HDF5 dataset named "my_dataset7" of data type float of one dimension (size 1024) with a fill value (i.e. default) of 85.2
CREATE DATASET my_dataset7 AS FLOAT(1024) FILL(85.2)

# create a compact HDF5 dataset named "my_dataset8" of data type double of three dimensions (size 2x5x10)
CREATE COMPACT DATASET my_dataset8 AS DOUBLE(2, 5, 10)

# create a chunked (size 20x100) HDF5 dataset named "my_dataset9" of data type unsigned char of two dimensions (size 500x1000)
CREATE CHUNKEO(20, 100) DATASET my_dataset9 AS UNSIGNED TINYINT(500, 1000)

# create an HDF5 dataset named "my_dataset10" of data type int of two dimensions (size 20x400) using the N-bit data compression filter
CREATE DATASET my_dataset10 AS INT(20, 400) ENABLE NBIT PRECISION 16 OFFSET 4

# create an HDF5 dataset named "my_dataset11" of data type float of one dimension (size 500000) using both the ZLIB data compression and Fletcher32 checksum error detection filters
CREATE DATASET my_dataset11 AS FLOAT(500000) ENABLE ZLIB LEVEL 5 FLETCHER32

# create an HDF5 dataset named "my_dataset12" of data type variable-length float
CREATE DATASET my_dataset12 AS VARFLOAT

# create an HDF5 dataset named "my_dataset13" of data type variable-length short of one dimension (size 5) with initial values from a text file named "my_file.txt"
CREATE DATASET my_dataset13 AS VARSMALLINT(5) VALUES FROM FILE my_file.txt

# create an HDF5 dataset named "my_dataset14" of data type variable-length char with an initial value of "Hierarchical Data Format"
CREATE DATASET my_dataset14 AS VARCHAR VALUES("Hierarchical Data Format")

# create an HDF5 dataset named "my_dataset15" of data type opaque
CREATE DATASET my_dataset15 AS OPAQUE
# create an HDF5 dataset named "my_dataset16" of data type opaque of one dimension (size 6) with initial (ASCII) values of 72, 68, 70, 0, 113 and 108 (i.e. "HDF\0ql")

```hdfql```
CREATE DATASET my_dataset16 AS OPAQUE(6) VALUES(72, 68, 70, 0, 113, 108)
```hdfql```

# create an HDF5 dataset named "my_dataset17" of data type opaque of two dimensions (size 10x1024) with a tag value "Raw data"

```hdfql```
CREATE DATASET my_dataset17 AS OPAQUE(10, 1024) TAG "Raw data"
```hdfql```

# create a chunked (size 2) HDF5 dataset named "my_dataset18" of data type float of one dimension (size 5 and extendible up to 10)

```hdfql```
CREATE CHUNKED(2) DATASET my_dataset18 AS FLOAT(5 TO 10)
```hdfql```

# create a chunked (with an automatically calculated size) HDF5 dataset named "my_dataset19" of data type variable-length int of one dimension (size 1 and extendible to an unlimited size)

```hdfql```
CREATE CHUNKED DATASET my_dataset19 AS VARINT(UNLIMITED)
```hdfql```

# create a chunked (with an automatically calculated size) HDF5 dataset named "my_dataset20" of data type double of three dimensions (first dimension with size 3 and extendible up to 5; second dimension with size 7; third dimension with size 20 and extendible to an unlimited size)

```hdfql```
CREATE CHUNKED DATASET my_dataset20 AS DOUBLE(3 TO 5, 7, 20 TO UNLIMITED)
```hdfql```

# use (i.e. open) an HDF5 file named "my_file.h5"

```hdfql```
USE FILE my_file.h5
```hdfql```

# create an HDF5 dataset named "my_dataset21" of data type double in the HDF5 file currently in use (i.e. file "my_file.h5")

```hdfql```
CREATE DATASET my_dataset21 AS DOUBLE
```hdfql```

# close HDF5 file currently in use (i.e. file "my_file.h5")

```hdfql```
CLOSE FILE
```hdfql```

# create an HDF5 dataset named "my_dataset22" of data type int in file "my_file.h5"

```hdfql```
CREATE DATASET my_file.h5 my_dataset22 AS INT
```hdfql```
6.4.5 CREATE ATTRIBUTE

Syntax

CREATE [TRUNCATE] ATTRIBUTE [file_name] attribute_name [, [file_name] attribute_name]* AS [NATIVE | LITTLE ENDIAN | BIG ENDIAN | ASCII | UTF8] {TINYINT | UNSIGNED TINYINT | SMALLINT | UNSIGNED SMALLINT | INT | UNSIGNED INT | BIGINT | UNSIGNED BIGINT | FLOAT | DOUBLE | CHAR | VARTINYINT | UNSIGNED VARTINYINT | VARSMALLINT | UNSIGNED VARSMALLINT | VARINT | UNSIGNED VARINT | VARBIGINT | UNSIGNED VARBIGINT | VARFLOAT | VARDouble | VARCHAR | OPAQUE} [{attribute_dim [, attribute_dim]*}] [TAG tag_value] [VALUES {{initial_value [, initial_value]*} | input Redirecting option}]

Description

Create an HDF5 attribute named attribute_name. Multiple attributes can be created at once by separating these with a comma (.). If attribute_name already exists, it will not be overwritten, no subsequent attributes are created, and an error is raised. To overwrite an existing attribute, specify the keyword TRUNCATE (all data stored in the attribute will be permanently lost).

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file in which the attribute is created. If file_name is specified, the file is opened on the fly, the attribute is created within it and, afterwards, the file is closed. Otherwise, if it is not specified, the attribute is created in the file currently in use. As a general rule, in case file_name is composed of spaces, special characters or reserved keywords (e.g. SELECT), it should be surrounded by double-quotes (“”). Otherwise, if it is not surrounded by double-quotes, the attribute will not be created and an error is raised. This rule also applies to any other HDFql operation that works with attribute names (e.g. RENAME ATTRIBUTE).

attribute_name – mandatory string that specifies the name of the HDF5 attribute to create. Multiple attributes are separated with a comma (.). As a general rule, in case attribute_name is composed of spaces, special characters or reserved keywords (e.g. SELECT), it should be surrounded by double-quotes (“”). Otherwise, if it is not surrounded by double-quotes, the attribute will not be created and an error is raised. This rule also applies to any other HDFql operation that works with attribute names (e.g. RENAME ATTRIBUTE).

attribute_dim – optional integer that specifies the size of the dimension. Multiple dimensions are separated with a comma (,).
**tag_value** – optional string that specifies the value of a tag attached to the HDF5 attribute. Of note, the specification of a tag is only available for an attribute of data type **HDFQL_OPAQUE** (any other data type will raise an error).

**initial_value** – optional integer, float or string to write into the created HDF5 attribute. Multiple values are separated with a comma (,). In case **initial_value** is not specified, the element in question will be zeroed or emptied depending on whether the attribute is a number or a string, respectively.

**input_redirecting_option** – optional option that specifies a file or memory to read data from in order to write it into the created HDF5 attribute (please refer to the subsection **FROM** for additional information).

**Return**

Nothing

**Example(s)**

```bash
# create an HDF5 attribute named "my_attribute0" of data type int (the attribute will not be overwritten if it already exists)
CREATE ATTRIBUTE my_attribute0 AS INT

# create an HDF5 attribute named "my_attribute1" of data type char in a root object (either a group or dataset) named "my_object" (the attribute will not be overwritten if it already exists)
CREATE ATTRIBUTE /my_object/my_attribute1 AS CHAR

# create two HDF5 attributes named "my_attribute2" and "my_attribute3" of data type short (both attributes will be overwritten if they already exist)
CREATE TRUNCATE ATTRIBUTE my_attribute2, my_attribute3 AS SMALLINT

# create an HDF5 attribute named "this is a long attribute name" of data type float (the attribute will not be overwritten if it already exists)
CREATE ATTRIBUTE "this is a long attribute name" AS FLOAT

# create an HDF5 attribute named "my_attribute4" of data type unsigned long long using the big endian representation
CREATE ATTRIBUTE my_attribute4 AS BIG ENDIAN UNSIGNED BIGINT

# create an HDF5 attribute named "my_attribute5" of data type int using the little endian representation with an initial value of 80178
CREATE ATTRIBUTE my_attribute5 AS LITTLE ENDIAN INT VALUES(80178)
```
# create an HDF5 attribute named "my_attribute6" of data type char using an UTF8 representation
CREATE ATTRIBUTE my_attribute6 AS UTF8 CHAR

# create an HDF5 attribute named "my_attribute7" of data type float of one dimension (size 512)
CREATE ATTRIBUTE my_attribute7 AS FLOAT(512)

# create an HDF5 attribute named "my_attribute8" of data type unsigned char of two dimensions (size 500x1000)
CREATE ATTRIBUTE my_attribute8 AS UNSIGNED TINYINT(500, 1000)

# create an HDF5 attribute named "my_attribute9" of data type variable-length float
CREATE ATTRIBUTE my_attribute9 AS VARFLOAT

# create an HDF5 attribute named "my_attribute10" of data type variable-length short of one dimension (size 5) with initial values from a text file named "my_file.txt"
CREATE ATTRIBUTE my_attribute10 AS VARSMALLINT(5) VALUES FROM FILE my_file.txt

# create an HDF5 attribute named "my_attribute11" of data type variable-length char with an initial value of "Hierarchical Data Format"
CREATE ATTRIBUTE my_attribute11 AS VARCHAR VALUES("Hierarchical Data Format")

# create an HDF5 attribute named "my_attribute12" of data type opaque
CREATE ATTRIBUTE my_attribute12 AS OPAQUE

# create an HDF5 attribute named "my_attribute13" of data type opaque of one dimension (size 6) with initial (ASCII) values 72, 68, 70, 0, 113 and 108 (i.e. "HDF\0ql")
CREATE ATTRIBUTE my_attribute13 AS OPAQUE(6) VALUES(72, 68, 70, 0, 113, 108)

# create an HDF5 attribute named "my_attribute14" of data type opaque of two dimensions (size 10x1024) with a tag value "Raw data"
CREATE ATTRIBUTE my_attribute14 AS OPAQUE(10, 1024) TAG "Raw data"

# use (i.e. open) an HDF5 file named "my_file.h5"
USE FILE my_file.h5
## 6.4.6 CREATE [SOFT | HARD] LINK

### Syntax

```hdfql
CREATE [TRUNCATE] [SOFT | HARD] LINK [file_name] link_name [, [file_name] link_name]* TO object_name [, object_name]*
```

### Description

Create an HDF5 soft or hard link named `link_name` to a group or dataset named `object_name`. Multiple links can be created at once by separating these with a comma (,). If `link_name` already exists, it will not be overwritten, no subsequent links are created, and an error is raised. To overwrite an existing link, specify the keyword TRUNCATE. If neither the keyword SOFT nor HARD is specified, a soft link is created by default. To create a hard link, the keyword HARD must be specified.

### Parameter(s)

- **file_name** – optional string that specifies the name of the HDF5 file in which the soft or hard link is created. If `file_name` is specified, the file is opened on the fly, the soft or hard link is created within it and, afterwards, the file is closed. Otherwise, if it is not specified, the soft or hard link is created in the file currently in use. As a general rule, in case `file_name` is composed of spaces, special characters or reserved keywords (e.g. SELECT), it should be surrounded by double-quotes ("). Otherwise, if it is not surrounded by double-quotes, the link will not be created and an error is raised. This rule also applies to any other HDFql operation that works with link names (e.g. RENAME LINK).

- **link_name** – mandatory string that specifies the name of the HDF5 soft or hard link to create. Multiple links are separated with a comma (,). As a general rule, in case `link_name` is composed of spaces, special characters or reserved keywords (e.g. SELECT), it should be surrounded by double-quotes ("). Otherwise, if it is not surrounded by double-quotes, the link will
not be created and an error is raised. This rule also applies to any other HDFql operation that works with link names (e.g. RENAME LINK).

object_name – mandatory string that specifies the name of the HDF5 group or dataset that link_name points to. Multiple objects are separated with a comma (,). As a general rule, in case object_name is composed of spaces, special characters or reserved keywords (e.g. SELECT), it should be surrounded by double-quotes ("). Otherwise, if it is not surrounded by double-quotes, the link will not be created and an error is raised. This rule also applies to any other HDFql operation that works with link names (e.g. RENAME LINK).

Return

Nothing

Example(s)

```
# create an HDF5 group named "my_group0"
CREATE GROUP my_group0

# create an HDF5 dataset named "my_dataset0" of data type variable-length unsigned int
CREATE DATASET my_dataset0 AS UNSIGNED VARI

# create an HDF5 soft link named "my_link0" to group "my_group0" (the soft link will not be overwritten if it already exists)
CREATE LINK my_link0 TO my_group0

# create an HDF5 soft link named "my_link1" to dataset "my_dataset0" (the soft link will not be overwritten if it already exists)
CREATE SOFT LINK my_link1 TO my_dataset0

# create two HDF5 soft links named "my_link2" and "my_link3" to dataset "my_dataset0" and group "my_group0" respectively (both soft links will be overwritten if they already exist)
CREATE TRUNCATE SOFT LINK my_link2, my_link3 TO my_dataset0, my_group0

# create an HDF5 soft link named "this is a long link name" to dataset "my_dataset0" (the soft link will not be overwritten if it already exists)
CREATE LINK "this is a long link name" TO my_dataset0
```

```
# create an HDF5 dataset named "my_dataset1" of data type variable-length unsigned int
CREATE DATASET my_dataset1 AS UNSIGNED VARINT

# create an HDF5 hard link named "my_link4" to group "my_group1" (the hard link will not be overwritten if it already exists)
CREATE HARD LINK my_link4 TO my_group1

# create an HDF5 hard link named "my_link5" to dataset "my_dataset1" (the hard link will not be overwritten if it already exists)
CREATE HARD LINK my_link5 TO my_dataset1

# create two HDF5 hard links named "my_link6" and "my_link7" to dataset "my_dataset1" and group "my_group1" respectively (both hard links will be overwritten if they already exist)
CREATE TRUNCATE HARD LINK my_link6, my_link7 TO my_dataset1, my_group1

# use (i.e. open) an HDF5 file named "my_file.h5"
USE FILE my_file.h5

# create a HDF5 soft link named "my_link8" to object "my_object0" in the HDF5 file currently in use (i.e. file "my_file.h5")
CREATE LINK my_link8 TO my_object0

# close HDF5 file currently in use (i.e. file "my_file.h5")
CLOSE FILE

# create a HDF5 soft link named "my_link9" to object "my_object1" in file "my_file.h5"
CREATE LINK my_file.h5 my_link9 TO my_object1

### 6.4.7 CREATE EXTERNAL LINK

**Syntax**

```
CREATE [TRUNCATE] EXTERNAL LINK [file_name] link_name [, [file_name] link_name]* TO target_file_name
object_name [, target_file_name object_name]*
```
Description

Create an HDF5 external link named \textit{link\_name} to a group or dataset named \textit{object\_name} belonging to another HDF5 file named \textit{target\_file\_name}. Multiple external links can be created at once by separating these with a comma (,). If \textit{link\_name} already exists, it will not be overwritten, no subsequent external links are created, and an error is raised. To overwrite an existing external link, specify the keyword TRUNCATE.

Parameter(s)

\textit{file\_name} – optional string that specifies the name of the HDF5 file in which the external link is created. If \textit{file\_name} is specified, the file is opened on the fly, the external link is created within it and, afterwards, the file is closed. Otherwise, if it is not specified, the external link is created in the file currently in use. As a general rule, in case \textit{file\_name} is composed of spaces, special characters or reserved keywords (e.g. SELECT), it should be surrounded by double-quotes (""). Otherwise, if it is not surrounded by double-quotes, the link will not be created and an error is raised. This rule also applies to any other HDFql operation that works with external link names (e.g. RENAME EXTERNAL LINK).

\textit{link\_name} – mandatory string that specifies the name of the HDF5 external link to create. Multiple external links are separated with a comma (,). As a general rule, in case \textit{link\_name} is composed of spaces, special characters or reserved keywords (e.g. SELECT), it should be surrounded by double-quotes (""). Otherwise, if it is not surrounded by double-quotes, the link will not be created and an error is raised. This rule also applies to any other HDFql operation that works with external link names (e.g. RENAME EXTERNAL LINK).

\textit{target\_file\_name} – mandatory string that specifies the name of the HDF5 file where \textit{object\_name} is stored and \textit{link\_name} points to. Multiple files are separated with a comma (,). As a general rule, in case \textit{target\_file\_name} is composed of spaces, special characters or reserved keywords (e.g. SELECT), it should be surrounded by double-quotes (""). Otherwise, if it is not surrounded by double-quotes, the link will not be created and an error is raised. This rule also applies to any other HDFql operation that works with link external names (e.g. RENAME EXTERNAL LINK).

\textit{object\_name} – mandatory string that specifies the name of the HDF5 group or dataset (stored in \textit{file\_name}) that \textit{link\_name} points to. As a general rule, in case \textit{object\_name} is composed of spaces, special characters or reserved keywords (e.g. SELECT), it should be surrounded by double-quotes (""). Otherwise, if it is not surrounded by double-quotes, the link will not be created and an error is raised. This rule also applies to any other HDFql operation that works with external link names (e.g. RENAME EXTERNAL LINK).

Return

Nothing
Example(s)

```plaintext
# use (i.e. open) an HDF5 file named "my_file0.h5"
USE FILE my_file0.h5

# create an HDF5 group named "my_group0"
CREATE GROUP my_group0

# create an HDF5 dataset named "my_dataset0" of data type variable-length unsigned int
CREATE DATASET my_dataset0 AS UNSIGNED VARINT

# create an HDF5 external link named "my_link0" to object "my_group0" in file "my_file0.h5"
# (the external link will not be overwritten if it already exists)
CREATE EXTERNAL LINK my_link0 TO my_file0.h5 my_group0

# create an HDF5 external link named "my_link1" to object "my_object0" in file "my_file1.h5"
# (the external link will be overwritten if it already exists)
CREATE TRUNCATE EXTERNAL LINK my_link1 TO my_file1.h5 my_object0

# create two HDF5 external links named "my_link2" and "my_link3" to object "my_object1" in file "my_file1.h5" and object "my_object2" in file "my_file2.h5" respectively (neither external links will be overwritten if they already exist)
CREATE EXTERNAL LINK my_link2, my_link3 TO my_file1.h5 my_object1, my_file2.h5 my_object2

# create an HDF5 external link named "this is a long external link name" to object "my_object3" in file "my_file3.h5" (the external link will not be overwritten if it already exists)
CREATE EXTERNAL LINK "this is a long external link name" TO my_file3.h5 my_object3

# use (i.e. open) an HDF5 file named "my_file4.h5"
USE FILE my_file4.h5

# create an HDF5 external link named "my_link4" in the HDF5 file currently in use (i.e. file "my_file4.h5") to object "my_object4" in file "my_file5.h5"
CREATE EXTERNAL LINK my_link4 TO my_file5.h5 my_object4

# close HDF5 file currently in use (i.e. file "my_file4.h5")
CLOSE FILE

# create an HDF5 external link named "my_link5" in file "my_file4.h5" to object "my_object5" in file "my_file6.h5"
CREATE EXTERNAL LINK my_file4.h5 my_link5 TO my_file6.h5 my_object5
```
6.4.8 ALTER DIMENSION

Syntax

ALTER DIMENSION [file_name] dataset_name [, [file_name] dataset_name]* TO dataset_dim [, dataset_dim]*

Description

Alter (i.e. change) the dimensions of an existing HDF5 dataset named dataset_name. Multiple datasets can have their dimensions altered at once by separating these with a comma (,). If dataset_name was not found or its dimensions could not be altered (due to its storage type not being HDFQL_CHUNKED or for unknown/unexpected reasons), no subsequent datasets are altered, and an error is raised.

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file which stores the dataset to alter (i.e. change) dimensions. If file_name is specified, the file is opened on the fly, the dimensions of the dataset are altered and, afterwards, the file is closed. Otherwise, if it is not specified, the dataset to alter the dimensions is stored in the file currently in use.

dataset_name – mandatory string that specifies the name of the HDF5 dataset whose dimensions are to be altered (i.e. changed). Multiple datasets are separated with a comma (,).

dataset_dim – mandatory integer that specifies the new size for the dimension in question. Multiple dimensions are separated with a comma (,). Depending on the prefix of the value specified in dataset_dim, one of the following behaviors applies:

- If its prefix is “+”, the dimension will have its size increased by this value.
- If its prefix is “-”, the dimension will have its size decreased by this value.
- In case its prefix is neither “+” nor “-”, the dimension will have exactly the size of this value.

To preserve the value of a certain dimension (i.e. for its size not to be altered), it should be skipped with a comma (,).

Return

Nothing
Example(s)

```hdfql
# create an HDF5 dataset named "my_dataset" of data type double of three dimensions (first
dimension with size 2 and extendible up to 10; second dimension with size 7; third dimension
with size 20 and extendible to an unlimited size)
CREATE CHUNKED DATASET my_dataset AS DOUBLE(2 TO 10, 7, 20 TO UNLIMITED)

# show (i.e. get) current dimensions of dataset "my_dataset" (should be 2, 7, 20)
SHOW DIMENSION my_dataset

# alter (i.e. change) dimensions of dataset "my_dataset" to set its first dimension size to 6,
and increase the third dimension size by 10 (the second dimension size remains intact)
ALTER DIMENSION my_dataset TO 6, 7, +10

# show (i.e. get) current dimensions of dataset "my_dataset" (should be 6, 7, 30)
SHOW DIMENSION my_dataset

# alter (i.e. change) dimensions of dataset "my_dataset" to increase its first dimension size
by 2, to set the second dimension size to 3, and to decrease the third dimension size by 5
ALTER DIMENSION my_dataset TO +2, 3, -5

# show (i.e. get) current dimensions of dataset "my_dataset" (should be 8, 3, 25)
SHOW DIMENSION my_dataset
```

6.4.9 RENAME DIRECTORY

Syntax

```hdfql
RENAME DIRECTORY directory_name [, directory_name]* AS new_directory_name [, new_directory_name]*
```

Description

Rename (or move) an existing directory named `directory_name` as `new_directory_name`. Multiple directories can be
renamed (or moved) at once by separating these with a comma (,). If `new_directory_name` already exists, it will not be
overwritten, no subsequent directories are renamed (or moved), and an error is raised.

Parameter(s)

- `directory_name` – mandatory string that specifies the name of the directory to rename (or move). Multiple directories are
  separated with a comma (,).
**new_directory_name** – mandatory string that specifies the new name and/or location (in the file system) to use for renaming and/or moving **directory_name**. Multiple directories are separated with a comma (,).

**Return**

Nothing

**Example(s)**

```
# rename a directory named "my_directory0" as "my_directory1" (the directory "my_directory1"
# will not be overwritten if it already exists)
RENAME DIRECTORY my_directory0 AS my_directory1

# rename two directories named "my_directory2" and "my_directory3" as "my_directory4" and
# "my_directory5" respectively (neither directory will be overwritten if it already exists)
RENAME DIRECTORY my_directory2, my_directory3 AS my_directory4, my_directory5

# move a directory named "my_directory6" into a root directory named "data" and rename it as
# "my_directory7" (the directory "my_directory7" will not be overwritten if it already exists)
RENAME DIRECTORY my_directory6 AS /data/my_directory7

# move a directory named "my_directory8" into a relative directory named "backup" (the
directory "my_directory8" will not be overwritten if it already exists)
RENAME DIRECTORY my_directory8 AS backup/
```

### 6.4.10 RENAME FILE

**Syntax**

```
RENAME [TRUNCATE] FILE file_name [, file_name]* AS new_file_name [, new_file_name]*
```

**Description**

Rename (or move) an existing file named **file_name** as **new_file_name**. Multiple files can be renamed (or moved) at once by separating these with a comma (,). If **new_file_name** already exists, it will not be overwritten, no subsequent files are renamed (or moved), and an error is raised. To overwrite an existing file, specify the keyword TRUNCATE (all data stored in the file will be permanently lost).
Parameter(s)

`file_name` – mandatory string that specifies the name of the file to rename (or move). Multiple files are separated with a comma (,).

`new_file_name` – mandatory string that specifies the new name and/or location (in the file system) to use for renaming and/or moving `file_name`. Multiple files are separated with a comma (,).

Return

Nothing

Example(s)

```bash
# rename a file named "my_file0.h5" as "my_file1.h5" (the file "my_file1.h5" will not be overwritten if it already exists)
RENAME FILE my_file0.h5 AS my_file1.h5

# rename a file named "my_file2.h5" as "my_file3.h5" (the file "my_file3.h5" will be overwritten if it already exists)
RENAME TRUNCATE FILE my_file2.h5 AS my_file3.h5

# rename two files named "my_file4.h5" and "my_file5.h5" as "my_file6.h5" and "my_file7.h5" respectively (both files "my_file6.h5" and "my_file7.h5" will be overwritten if they already exist)
RENAME TRUNCATE FILE my_file4.h5, my_file5.h5 AS my_file6.h5, my_file7.h5

# move a file named "my_file8.h5" into a root directory named "data" and rename it as "my_file9.h5" (the file "my_file9.h5" will not be overwritten if it already exists in this directory)
RENAME FILE my_file8.h5 AS /data/my_file9.h5

# move a file named "my_file10.h5" into a relative directory named "backup" (the file "my_file10.h5" will not be overwritten if it already exists in this directory)
RENAME FILE my_file10.h5 AS backup/
```
6.4.11 RENAME [GROUP | DATASET | ATTRIBUTE | [SOFT] LINK | EXTERNAL LINK]

Syntax

RENAME [TRUNCATE] [GROUP | DATASET | ATTRIBUTE | [SOFT] LINK | EXTERNAL LINK] [file_name] object_name [, [file_name] object_name]* AS new_object_name [, new_object_name]*

Description

Rename (or move) an existing HDF5 group, dataset, attribute, (soft) link or external link named object_name as new_object_name. Multiple groups, datasets, attributes, (soft) links or external links can be renamed (or moved) at once by separating these with a comma (,). If new_object_name already exists, it will not be overwritten, no subsequent objects are renamed (or moved), and an error is raised. To overwrite an existing object, specify the keyword TRUNCATE (all data stored in the object will be permanently lost). In case (1) a group and an attribute or (2) a dataset and an attribute with identical names (object_name) are stored in the same location (i.e. group) and neither the keyword GROUP, DATASET nor ATTRIBUTE is specified, the object to be renamed is the group or dataset, respectively. To explicitly rename an object according to its type, the keyword GROUP, DATASET, ATTRIBUTE, [SOFT] LINK or EXTERNAL LINK must be specified. While the renaming (or moving) of groups and datasets to a different location is supported by the HDF5 library, this is not the case for attributes; HDFql overcomes this limitation by (1) creating a new attribute with the same characteristics as the existing one (e.g. data type, number of dimensions) using the new specified location and name, (2) writing the data from the existing attribute to the newly created attribute, and (3) deleting the existing attribute.

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file which stores the object to rename (or move). If file_name is specified, the file is opened on the fly, the object is renamed (or moved) and, afterwards, the file is closed. Otherwise, if it is not specified, the object to rename (or move) is stored in the file currently in use.

object_name – mandatory string that specifies the name of the object to rename (or move). Multiple objects are separated with a comma (,).

new_object_name – mandatory string that specifies the new name and/or location (within the HDF5 file) to use for renaming and/or moving object_name. Multiple objects are separated with a comma (,).

Return

Nothing
Example(s)

```hdfql
# create two HDF5 groups named "my_group0" and "my_group1"
CREATE GROUP my_group0, my_group1

# create two HDF5 datasets named "my_dataset" and "my_common" of data type short
CREATE DATASET my_dataset, my_common AS SMALLINT

# create two HDF5 attributes named "my_attribute" and "my_common" of data type float
CREATE ATTRIBUTE my_attribute, my_common AS FLOAT

# rename an object named "my_group0" as "my_group" (the object "my_group" will not be overwritten if it already exists)
RENAME my_group0 AS my_group

# move an object named "my_group1" into object "my_group" and rename it as "my_subgroup" (the object "my_subgroup" will be overwritten if it already exists in object "my_group")
RENAME TRUNCATE my_group1 AS my_group/my_subgroup

# move two objects named "my_dataset" and "my_attribute" into objects "my_group" and "my_group/my_subgroup" respectively (both objects "my_dataset" and "my_attribute" will not be overwritten if they already exist in objects "my_group" and "my_group/my_subgroup")
RENAME my_dataset, my_attribute AS my_group/, my_group/my_subgroup/

# rename attribute "my_common" as "my_attribute" (the attribute "my_attribute" will not be overwritten if it already exists)
RENAME ATTRIBUTE my_common AS my_attribute

# rename dataset "my_common" as "my_dataset" (the dataset "my_dataset" will not be overwritten if it already exists)
RENAME DATASET my_common AS my_dataset
```

6.4.12 COPY FILE

Syntax

```text
COPY [TRUNCATE] FILE file_name [, file_name]* TO new_file_name [, new_file_name]*
```
Description

Copy an existing file named `file_name` to `new_file_name`. Multiple files can be copied at once by separating these with a comma (,). If `new_file_name` already exists, it will not be overwritten, no subsequent files are copied, and an error is raised. To overwrite an existing file, specify the keyword TRUNCATE (all data stored in the file will be permanently lost).

Parameter(s)

`file_name` – mandatory string that specifies the name of the file to copy. Multiple files are separated with a comma (,).

`new_file_name` – mandatory string that specifies the new name and/or location (in the file system) to use for copying `file_name`. Multiple files are separated with a comma (,).

Return

Nothing

Example(s)

```bash
# copy a file named "my_file0.h5" to "my_file1.h5" (the file "my_file1.h5" will not be overwritten if it already exists)
COPY FILE my_file0.h5 TO my_file1.h5

# copy a file named "my_file2.h5" to "my_file3.h5" (the file "my_file3.h5" will be overwritten if it already exists)
COPY TRUNCATE FILE my_file2.h5 TO my_file3.h5

# copy two files named "my_file4.h5" and "my_file5.h5" to "my_file6.h5" and "my_file7.h5" respectively (both files "my_file6.h5" and "my_file7.h5" will be overwritten if they already exist)
COPY TRUNCATE FILE my_file4.h5, my_file5.h5 TO my_file6.h5, my_file7.h5

# copy a file named "my_file8.h5" into a root directory named "data" and rename it as "my_file9.h5" (the file "my_file9.h5" will not be overwritten if it already exists in this directory)
COPY FILE my_file8.h5 TO /data/my_file9.h5

# copy a file named "my_file10.h5" into a relative directory named "backup" (the file "my_file10.h5" will not be overwritten if it already exists in this directory)
COPY FILE my_file10.h5 TO backup/
```
6.4.13 COPY [GROUP | DATASET | ATTRIBUTE | [SOFT] LINK | EXTERNAL LINK]

Syntax

COPY [TRUNCATE] [GROUP | DATASET | ATTRIBUTE | [SOFT] LINK | EXTERNAL LINK] [file_name] object_name [, [file_name] object_name]* TO [target_file_name] new_object_name [, [target_file_name] new_object_name]*

Description

Copy an existing HDF5 group, dataset, attribute, (soft) link or external link named object_name to new_object_name. Multiple groups, datasets, attributes, (soft) links or external links can be copied at once by separating these with a comma (,). If new_object_name already exists, it will not be overwritten, no subsequent objects are copied, and an error is raised. To overwrite an existing object, specify the keyword TRUNCATE (all data stored in the object will be permanently lost). In case (1) a group and an attribute or (2) a dataset and an attribute with identical names (object_name) are stored in the same location (i.e. group) and neither the keyword GROUP, DATASET nor ATTRIBUTE is specified, the object to be copied is the group or dataset, respectively. To explicitly copy an object according to its type, the keyword GROUP, DATASET, ATTRIBUTE, [SOFT] LINK or EXTERNAL LINK must be specified.

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file which stores the object to copy. If file_name is specified, the file is opened on the fly, the object is copied and, afterwards, the file is closed. Otherwise, if it is not specified, the object to copy is stored in the file currently in use.

object_name – mandatory string that specifies the name of the object to copy. Multiple objects are separated with a comma (,).

target_file_name – optional string that specifies the name of the HDF5 file in which to copy the object. Multiple files are separated with a comma (,).

new_object_name – mandatory string that specifies the new name and/or location (within the HDF5 file or in another HDF5 file specified by target_file_name) to use for copying object_name. Multiple objects are separated with a comma (,).

Return

Nothing
Example(s)

```hdfql
# create two HDF5 groups named "my_group0" and "my_group1"
CREATE GROUP my_group0, my_group1

# create two HDF5 datasets named "my_dataset0" and "my_common" of data type short
CREATE DATASET my_dataset0, my_common AS SMALLINT

# create two HDF5 attributes named "my_attribute0" and "my_common" of data type float
CREATE ATTRIBUTE my_attribute0, my_common AS FLOAT

# copy an object named "my_group0" to "my_group2" (the object "my_group2" will not be overwritten if it already exists)
COPY my_group0 TO my_group2

# copy an object named "my_group1" into object "my_group0" and rename it as "my_subgroup0" (the object "my_subgroup0" will be overwritten if it already exists in object "my_group0")
COPY TRUNCAST my_group1 TO my_group0/my_subgroup0

# copy two objects named "my_dataset0" and "my_attribute0" into objects "my_group0" and "my_group0/my_subgroup0" respectively (both objects "my_dataset0" and "my_attribute0" will not be overwritten if they already exist in objects "my_group0" and "my_group0/my_subgroup0")
COPY my_dataset0, my_attribute0 TO my_group0/, my_group0/my_subgroup0/

# copy attribute "my_common" to "my_attribute1" (the attribute "my_attribute1" will not be overwritten if it already exists)
COPY ATTRIBUTE my_common TO my_attribute1

# copy dataset "my_common" to "my_dataset1" (the dataset "my_dataset1" will not be overwritten if it already exists)
COPY DATASET my_common TO my_dataset1

# copy an object named "my_group3" from the file currently in use to "my_group4" in an HDF5 file named "my.file0.h5" (the object "my_group4" will not be overwritten if it already exists in the file)
COPY my_group3 TO my_file0.h5 my_group4

# copy an object named "my_group5" from an HDF5 file named "my_file1.h5" to "my_group6" in the file currently in use (the object "my_group6" will not be overwritten if it already exists in the file)
COPY my_file1.h5 my_group5 TO my_group6
```
# copy an object named "my_group7" from an HDF5 file named "my_file2.h5" to "my_group8" in an HDF5 file named "my_file3.h5" (the object "my_group8" will not be overwritten if it already exists in the file)

```
COPY my_file2.h5 my_group7 TO my_file3.h5 my_group8
```

## 6.4.14 DROP DIRECTORY

### Syntax

```
DROP DIRECTORY directory_name [, directory_name]*
```

### Description

Drop (i.e. delete) an existing directory named `directory_name`. Multiple directories can be dropped at once by separating these with a comma (,). If `directory_name` contains directories or files (i.e. if it is not empty), it will not be dropped, no subsequent directories are dropped, and an error is raised.

### Parameter(s)

`directory_name` – mandatory string that specifies the name of the directory to drop (i.e. delete). Multiple directories are separated with a comma (,).

### Return

Nothing

### Example(s)

```
# create two directories named "my_directory0" and "my_directory1" within the current working directory
CREATE DIRECTORY my_directory0, my_directory1

# create two directories named "my_subdirectory0" and "my_subdirectory1" within the directory "my_directory0"
CREATE DIRECTORY my_directory0/my_subdirectory0, my_directory0/my_subdirectory1

# drop (i.e. delete) directory "my_directory1" within the current working directory
DROP DIRECTORY my_directory1

# drop (i.e. delete) directory "my_subdirectory0" within directory "my_directory0"
```

DROP DIRECTORY my_directory0/my_subdirectory0

6.4.15 DROP FILE

Syntax

DROP FILE file_name [, file_name]*

Description

Drop (i.e. delete) an existing file named file_name. Multiple files can be dropped at once by separating these with a comma (,). If file_name was not found or could not be dropped (due to unknown/unexpected reasons), no subsequent files are dropped, and an error is raised.

Parameter(s)

file_name – mandatory string that specifies the name of the file to drop (i.e. delete). Multiple files are separated with a comma (,).

Return

Nothing

Example(s)

# create two HDF5 files named "my_file0.h5" and "my_file1.h5" within the current working directory
CREATE FILE my_file0.h5, my_file1.h5

# create two HDF5 files named "my_file2.h5" and "my_file3.h5" within a directory named "my_directory"
CREATE FILE my_directory/my_file2.h5, my_directory/my_file3.h5

# drop (i.e. delete) file "my_file1.h5" within the current working directory
DROP FILE my_file1.h5

# drop (i.e. delete) file "my_file2.h5" within directory "my_directory"
DROP FILE my_directory/my_file2.h5
6.4.16 DROP [GROUP | DATASET | ATTRIBUTE | [SOFT] LINK | EXTERNAL LINK]

Syntax


Description

Drop (i.e. delete) an existing HDF5 group, dataset, attribute, (soft) link or external link named object_name. Multiple groups, datasets, attributes, (soft) links or external links can be dropped at once by separating these with a comma (,). If object_name was not found or could not be dropped (due to unknown/unexpected reasons), no subsequent objects are dropped, and an error is raised. In case (1) a group and an attribute or (2) a dataset and an attribute with identical names (object_name) are stored in the same location (i.e. group) and neither the keyword GROUP, DATASET nor ATTRIBUTE is specified, the object to be dropped is the group or dataset, respectively. To explicitly drop an object according to its type, the keyword GROUP, DATASET, ATTRIBUTE, [SOFT] LINK or EXTERNAL LINK must be specified.

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file which stores the object to drop (i.e. delete). If file_name is specified, the file is opened on the fly, the object is dropped and, afterwards, the file is closed. Otherwise, if it is not specified, the object to drop is stored in the file currently in use.

object_name – mandatory string that specifies the name of the object to drop (i.e. delete). Multiple objects are separated with a comma (,).

regular_expression – optional string that specifies the regular expression which only names of objects that comply with it are dropped. If regular_expression includes “**”, recursive search is performed.

depth_value – optional integer that specifies the maximum recursiveness limit.

Return

Nothing

Example(s)

```bash
# create three HDF5 groups named "my_group0", "my_group1" and "my_group2"
```
CREATE GROUP my_group0, my_group1, my_group2

# create two HDF5 datasets named "my_dataset0" and "my_dataset1" of data type short in group "my_group2"
CREATE DATASET my_group2/my_dataset0, my_group2/my_dataset1 AS SMALLINT

# create two HDF5 datasets named "my_dataset2" and "my_common" of data type short
CREATE DATASET my_dataset2, my_common AS SMALLINT

# create two HDF5 attributes named "my_attribute0" and "my_common" of data type float
CREATE ATTRIBUTE my_attribute0, my_common AS FLOAT

# drop (i.e. delete) an object named "my_group0" (and all objects that may eventually be stored in it)
DROP my_group0

# drop (i.e. delete) attribute "my_common"
DROP ATTRIBUTE my_common

# drop (i.e. delete) all existing datasets in group "my_group2" (should be "my_dataset2" and "my_dataset3")
DROP DATASET my_group2/

# drop (i.e. delete) all existing groups (should be "my_group1" and "my_group2")
DROP GROUP

# drop (i.e. delete) all existing objects (should be "my_dataset2", "my_common" and "my_attribute0")
DROP /

6.5 DATA MANIPULATION LANGUAGE (DML)

Data Manipulation Language (DML) is, generally speaking, syntax for defining and modifying data stored in structures. In HDFql, the DML is composed of only one operation (INSERT), which enables the insertion (i.e. writing) of data into HDF5 datasets or attributes. Moreover, it supports REDIRECTING options to redirect the input source according to the programmer’s needs.
6.5.1 INSERT

**Syntax**

```
INSERT [DIRECTLY [MASK mask_value] [SIZE data_size]] INTO [PARALLEL] [DATASET | ATTRIBUTE] [file_name]
object_name [[[start]:[stride]:[count]:[block] [, [start]:[stride]:[count]:[block]]* | {start}:[commit]]
{[OR | AND | XOR | NOTA | NOTB] [start]:[stride]:[count]:[block] [, [start]:[stride]:[count]:[block]]* | {coord [, coord]* ; coord [, coord]*} | 
{chunk_number [, chunk_number]*}] [{file_name} object_name ([[start]:[stride]:[count]:[block] [, 
{start]:[stride]:[count]:[block]]* | {start]:[stride]:[count]:[block]]* | {coord [, coord]* ; coord [, coord]*} | 
{chunk_number [, chunk_number]*}])]
VALUES {(value [, value]*) | input redirecting option]}
```

**Description**

Insert (i.e. write) data into an HDF5 dataset or attribute named `object_name`. Multiple datasets or attributes can be written at once by separating these with a comma (,). If `object_name` was not found or could not be written (due to unknown/unexpected reasons), no subsequent objects are written, and an error is raised. In case a dataset and an attribute with identical names (`object_name`) are stored in the same location (i.e. group) and neither the keyword DATASET nor ATTRIBUTE is specified, the object that will have data inserted into it is the dataset. To explicitly insert data into an object according to its type, the keyword DATASET or ATTRIBUTE must be specified. In case the keyword DIRECTLY\(^39\) is specified, HDFql inserts data chunks directly into the dataset bypassing several internal processing steps of the HDF5 library itself (e.g. data conversion, filter pipeline), which can lead to a much faster writing. In case the keyword PARALLEL\(^40\) is specified, HDFql inserts data into a dataset in parallel using all the MPI processes specified upon launching the program (that employs HDFql).

By default, the entire `object_name` is written when performing an insert operation. To write only a subset (i.e. portion) of `object_name`, hyperslab and point selections can be used\(^41\). To enable a (regular) hyperslab selection, the `start`, `stride`, `count` and `block` parameters may be specified and separated with a colon (:). For each dimension of `object_name`, a set of such parameters may be specified and each set should be separated with a comma (,). Multiple hyperslab selections can

\(^{39}\) Only available for HDF5 datasets as, by design, direct insert (i.e. write) for HDF5 attributes is not supported by the HDF5 library. Moreover, the library does not support inserting (i.e. writing) data directly into a dataset of data type variable-length or in parallel (by specifying the keyword PARALLEL).

\(^{40}\) This option is not allowed in Windows as HDFql does not support the parallel HDF5 (PHDF5) library in this platform currently. Moreover, due to a limitation of the PHDF5 library, inserting (i.e. writing) data into an HDF5 dataset of data type variable-length in parallel is not supported.

\(^{41}\) Only available for HDF5 datasets as, by design, both hyperslab and point selections for HDF5 attributes are not supported by the HDF5 library.
be enabled at once (in this case, the hyperslab will be considered irregular). This is enabled by using the following boolean operators:

- **OR** – adds the new selection to the existing selection.

- **AND** – retains only the overlapping portions of the new selection and the existing selection.

- **XOR** – retains only the elements that are members of the new selection or the existing selection, excluding elements that are members of both selections.

- **NOTA** – retains only elements of the new selection that are not in the existing selection.

- **NOTB** – retains only elements of the existing selection that are not in the new selection.

To enable a point selection, a set of coordinates may be specified. Each coordinate is separated with a comma (,). More than one set of coordinates (i.e. points) may be specified and each set should be separated with a semicolon (;). Of note, hyperslab and point selections cannot be used both at the same time (i.e. be mixed) in an insert operation. Since hyperslab and point selections can be complicated to set up, it is highly recommended to first read [https://support.hdfgroup.org/HDF5/doc1.8/UG/HDF5_Users_Guide-Responsive%20HTML5/index.html#HDF5_Users_Guide%2FDataspace%2FHDF5_Dataspaces_and_Partial_I_O.htm%23TOC_7_4_1_Data_Selection](https://support.hdfgroup.org/HDF5/doc1.8/UG/HDF5_Users_Guide-Responsive%20HTML5/index.html#HDF5_Users_Guide%2FDataspace%2FHDF5_Dataspaces_and_Partial_I_O.htm%23TOC_7_4_1_Data_Selection) and eventually enable the debug mechanism (through the operation SET DEBUG) when working with these to obtain debug information in case of errors.

HDFql provides several ways of inserting data into a dataset or attribute, namely either from a cursor (e.g. “INSERT INTO my_dataset”), direct values (e.g. “INSERT INTO my_dataset VALUES(0, 2, 4, 6, 8)”), or an input redirecting option (e.g. “INSERT INTO my_dataset VALUES FROM FILE my_file.txt”).

**Parameter(s)**

- **mask_value** – optional integer that specifies which filters have been applied to the data chunk. A filter is skipped if the bit corresponding to the position of the filter in the pipeline is turned on. If **mask_value** is specified it must be equal to or greater than zero (otherwise an error will be raised). Otherwise, if it is not specified and in case the keyword DIRECTLY is specified, its default value is 0 (meaning that all filters have been applied to the data chunk).

- **data_size** – optional integer that specifies the size (in bytes) of the data to insert (i.e. write) into the HDF5 dataset. If **data_size** is specified it must be greater than zero (otherwise an error will be raised). Otherwise, if it is not specified and in case the keyword DIRECTLY is specified, HDFql automatically calculates the size by multiplying all storage dimensions of the dataset with its data type size.
**file_name** – optional string that specifies the name of the HDF5 file in which the HDF5 dataset or attribute to insert (i.e. write) data into is stored. If *file_name* is specified, the file is opened on the fly, the dataset or attribute is inserted and, afterwards, the file is closed. Otherwise, if it is not specified, the dataset or attribute (whose data is to be inserted) is stored in the file currently in use.

**object_name** – mandatory string that specifies the name of the HDF5 dataset or attribute to insert (i.e. write) data into. Multiple datasets or attributes are separated with a comma (,).

**start** – optional integer that specifies the starting location of the hyperslab selection. If not specified, its default value is 0 (i.e. the first position of the dimension in question). If negative, its value will be the last position of the dimension in question minus the value of start.

**stride** – optional integer that specifies the number of elements to separate each block to be selected. If not specified, its default value is equal to the value of block.

**count** – optional integer that specifies the number of blocks to select along each dimension. If not specified, its default value is 1.

**block** – optional integer that specifies the size of the block selected (i.e. number of elements) from the HDF5 dataset. If not specified, its default value is the size of the dimension in question minus the value of start divided by the value of count.

**coord** – optional integer that specifies the point of interest (i.e. to insert) for the point selection. If negative, its value will be the last position of the dimension in question minus the value of coord.

**chunk_number** – optional integer that specifies the number of the chunk to insert (i.e. write) data into. Multiple chunk numbers are separated with a comma (,). If *chunk_number* is specified it must either be between 0 and the storage dimension in question - 1 (otherwise an error will be raised) or negative (in this case its value will be the last position of the storage dimension in question minus the value of chunk_number). Otherwise, if it is not specified and in case the keyword DIRECTLY is specified, its default value is 0 (i.e. first chunk of the storage dimension in question).

**value** – optional integer, float or string to insert (i.e. write) into the HDF5 dataset or attribute. Multiple values are separated with a comma (,). In case value is not specified, the element in question will be zeroed or emptied depending on whether the dataset/attribute is a number or a string, respectively.

**input_redirecting_option** – optional option that specifies a file or memory to read data from in order to write it into an HDF5 dataset or attribute (please refer to the subsection FROM for additional information).
Return

Nothing

Example(s)

```
# create an HDF5 dataset named "my_dataset0" of data type short of one dimension (size 3)
CREATE DATASET my_dataset0 AS SMALLINT(3)

# create an HDF5 dataset named "my_dataset1" of data type int of one dimension (size 5)
CREATE DATASET my_dataset1 AS INT(5)

# insert (i.e. write) values 65, 66 and 67 into dataset "my_dataset0"
INSERT INTO my_dataset0 VALUES(65, 66, 67)

# select (i.e. read) data from dataset "my_dataset0" and populate cursor in use with it (should be 65, 66, 67)
SELECT FROM my_dataset0

# insert (i.e. write) values into dataset "my_dataset1" from cursor in use (should be 65, 66, 67, 0, 0)
INSERT INTO my_dataset1

# create an HDF5 attribute named "my_attribute0" of data type short
CREATE ATTRIBUTE my_attribute0 AS SMALLINT

# insert (i.e. write) value 95 into attribute "my_attribute0"
INSERT INTO my_attribute0 VALUES(95)

# create an HDF5 attribute named "my_attribute1" of data type unsigned short of one dimension (size 2)
CREATE ATTRIBUTE my_attribute1 AS UNSIGNED SMALLINT(2)

# insert (i.e. write) values 95 and 97 into attribute "my_attribute1"
INSERT INTO my_attribute1 VALUES(95, 97)

# create an HDF5 dataset named "my_dataset2" of data type float of one dimension (size 512)
CREATE DATASET my_dataset2 AS FLOAT(512)

# insert (i.e. write) values into dataset "my_dataset2" from a text file named "my_file0.txt" that has values separated with "," (i.e. default separator)
```
INSERT INTO my_dataset2 VALUES FROM FILE my_file0.txt

# insert (i.e. write) values into dataset "my_dataset2" from a text file named "my_file1.txt" that has a DOS-based end of line (EOL) terminator and values separated with "**"
INSERT INTO my_dataset2 VALUES FROM DOS TEXT FILE my_file1.txt SEPARATOR **

# insert (i.e. write) values into dataset "my_dataset2" from a binary file named "my_file.bin"
INSERT INTO my_dataset2 VALUES FROM BINARY FILE my_file.bin

# create an HDF5 dataset named "my_dataset3" of data type short of one dimension (size 5)
CREATE DATASET my_dataset3 AS SMALLINT(5)

# insert (i.e. write) value 9 into position #3 of dataset "my_dataset3" using a hyperslab selection
INSERT INTO my_dataset3(3:::) VALUES(9)

# select (i.e. read) data from dataset "my_dataset3" and populate cursor in use with it (should be 0, 0, 0, 9, 0)
SELECT FROM my_dataset3

# insert (i.e. write) value 9 into position #4 of dataset "my_dataset3" using a hyperslab selection
INSERT INTO my_dataset3(-1::) VALUES(7)

# select (i.e. read) data from dataset "my_dataset3" and populate cursor in use with it (should be 0, 0, 0, 9, 7)
SELECT FROM my_dataset3

# insert (i.e. write) values 5 and 3 into positions #1 and #2 of dataset "my_dataset3" using a hyperslab selection
INSERT INTO my_dataset3(1::2) VALUES(5, 3)

# select (i.e. read) data from dataset "my_dataset3" and populate cursor in use with it (should be 0, 5, 3, 9, 7)
SELECT FROM my_dataset3

# create an HDF5 dataset named "my_dataset4" of data type int of two dimensions (size 3x3)
CREATE DATASET my_dataset4 AS INT(3, 3)

# insert (i.e. write) value 8 into position #2 of the first dimension and position #1 of the second dimension of dataset "my_dataset4" using a hyperslab selection
INSERT INTO my_dataset4(2:::, 1::) VALUES(8)
# select (i.e. read) data from dataset "my_dataset4" and populate cursor in use with it (should be 0, 0, 0, 0, 0, 0, 8, 0)

SELECT FROM my_dataset4

# insert (i.e. write) value 4 into position #2 of the first dimension and position #0 of the second dimension, and value 6 into position #2 of the first dimension and position #2 of the second dimension of dataset "my_dataset4" using a hyperslab selection

INSERT INTO my_dataset4(2:::0:2:2:1) VALUES(4, 6)

# select (i.e. read) data from dataset "my_dataset4" and populate cursor in use with it (should be 0, 0, 0, 0, 4, 8, 6)

SELECT FROM my_dataset4

# create an HDF5 dataset named "my_dataset5" of data type short of one dimension (size 10)

CREATE DATASET my_dataset5 AS SMALLINT(10)

# insert (i.e. write) values 90, 91 and 92 into positions #2, #3 and #4, value 93 into position#5, and values 94 and 95 into positions #7 and #8 of dataset "my_dataset5" using an irregular hyperslab selection

INSERT INTO my_dataset5(2:::3::1 OR 4:::2:1 OR 7:::2:1) VALUES(90, 91, 92, 93, 94, 95)

# select (i.e. read) data from dataset "my_dataset5" and populate cursor in use with it (should be 0, 0, 90, 91, 92, 93, 0, 94, 95, 0)

SELECT FROM my_dataset5

# create an HDF5 dataset named "my_dataset6" of data type long long of one dimension (size 15)

CREATE DATASET my_dataset6 AS BIGINT(15)

# insert (i.e. write) values 75 and 77 into positions #5 and #6 of dataset "my_dataset6" using an irregular hyperslab selection

INSERT INTO my_dataset6(3:::4:1 AND 5:::3:1) VALUES(75, 77, 79, 81, 83, 85, 87)

# select (i.e. read) data from dataset "my_dataset6" and populate cursor in use with it (should be 0, 0, 0, 0, 75, 77, 0, 0, 0, 0, 0, 0, 0, 0, 0)

SELECT FROM my_dataset6

# create an HDF5 dataset named "my_dataset7" of data type float of one dimension (size 8)

CREATE DATASET my_dataset7 AS FLOAT(8)

# insert (i.e. write) values 7.5, 7.7 and 7.9 into positions #2, #4 and #7 of dataset "my_dataset7" using a point selection

INSERT INTO my_dataset7(2; 4; ?) VALUES(7.5, 7.7, 7.9)
# select (i.e. read) data from dataset "my_dataset7" and populate cursor in use with it (should be 0, 0, 7.5, 0, 7.7, 0, 0, 7.9)

```sql
SELECT FROM my_dataset7
```

# create an HDF5 dataset named "my_dataset8" of data type double of two dimensions (size 4x3)

```sql
CREATE DATASET my_dataset8 AS DOUBLE(4, 3)
```

# insert (i.e. write) value 15.2 into position #1 of the first dimension and position #2 of the second dimension, and value 18.5 into position #3 of the first dimension and position #0 of the second dimension of dataset "my_dataset8" using a point selection

```sql
INSERT INTO my_dataset8(1, 2; 3, 0) VALUES(15.2, 18.5)
```

# select (i.e. read) data from dataset "my_dataset8" and populate cursor in use with it (should be 0, 0, 0, 0, 15.2, 0, 0, 18.5, 0, 0)

```sql
SELECT FROM my_dataset8
```

# use (i.e. open) an HDF5 file named "my_file.h5"

```sql
USE FILE my_file.h5
```

# create a HDF5 dataset named "my_dataset9" of data type double in the HDF5 file currently in use (i.e. file "my_file.h5")

```sql
CREATE DATASET my_dataset9 AS DOUBLE
```

# insert (i.e. write) value 6.5 into dataset "my_dataset9"

```sql
INSERT INTO my_dataset9 VALUES(6.5)
```

# select (i.e. read) data from dataset "my_dataset9" and populate cursor in use with it (should be 6.5)

```sql
SELECT FROM my_dataset9
```

# close HDF5 file currently in use (i.e. file "my_file.h5")

```sql
CLOSE FILE
```

# insert (i.e. write) value 3.2 into dataset "my_dataset9" in file "my_file.h5"

```sql
INSERT INTO my_file.h5 my_dataset9 VALUES(3.2)
```

# select (i.e. read) data from dataset "my_dataset9" in file "my_file.h5" and populate cursor in use with it (should be 3.2)

```sql
SELECT FROM my_file.h5 my_dataset9
```

# create a chunked (size 2) HDF5 dataset named "my_dataset10" of data type int of one dimension (size 6)

CREATE CHUNKE D(2) DATASET my_dataset10 AS INT(6)

# insert (i.e. write) values 60 and 61 directly into chunk #0 of dataset "my_dataset10" using a (filter) mask equal to 8

INSERT DIRECTLY MASK 8 INTO my_dataset10 VALUES(60, 61)

# insert (i.e. write) values 62 and 63 directly into chunk #1 of dataset "my_dataset10" using a (filter) mask equal to 255 (i.e. 0xFF)

INSERT DIRECTLY MASK 0xFF INTO my_dataset10(1) VALUES(62, 63)

# insert (i.e. write) values 64 and 65 directly into chunk #2 of dataset "my_dataset10" using a (filter) mask equal to 0 (i.e. default value)

INSERT DIRECTLY INTO my_dataset10(2) VALUES(64, 65)

# select (i.e. read) data from dataset "my_dataset10" and populate cursor in use with it (should be 60, 61, 62, 63, 64, 65)

SELECT FROM my_dataset10

// declare variables
char script[1024];
double data[3];

// create an HDF5 dataset named "my_dataset11" of data type double of one dimension (size 3)

hdfql_execute("CREATE DATASET my_dataset11 AS DOUBLE(3)"");

// populate variable "data" with certain values
data[0] = 21.1;
data[1] = 18.8;
data[2] = 75.6;

// register variable "data" for subsequent use (by HDFql)

hdfql_variable_register(data);

// prepare script to insert (i.e. write) values from variable "data" into dataset "my_dataset11"

sprintf(script, "INSERT INTO my_dataset11 VALUES FROM MEMORY %d",

hdfql_variable_get_number(data));

// execute script

hdfql_execute(script);
// unregister variable "data" as it is no longer used/needed (by HDFql)
hdfql_variable_unregister(data);

// declare variables
char script[1024];
HDFQL_VARIABLE_LENGTH data[3];

// create an HDF5 dataset named "my_dataset1" of data type variable-length double of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset1 AS VARDOUBLE(3)");

// allocate memory in variable "data"
data[0].address = malloc(2 * sizeof(double));
data[0].count = 2;
data[1].address = malloc(3 * sizeof(double));
data[1].count = 3;
data[2].address = malloc(1 * sizeof(double));
data[2].count = 1;

// populate variable "data" with certain values
*((double *) data[0].address + 0) = 3.2;
*((double *) data[0].address + 1) = 1.3;
*((double *) data[1].address + 0) = 0;
*((double *) data[1].address + 1) = 0.2;
*((double *) data[1].address + 2) = 9.1;
*((double *) data[2].address + 0) = 6.5;

// register variable "data" for subsequent use (by HDFql)
hdfql_variable_register(data);

// prepare script to insert (i.e. write) values from variable "data" into dataset "my_dataset1"
sprintf(script, "INSERT INTO my_dataset1 VALUES FROM MEMORY %d",
hdfql_variable_get_number(data));

// execute script
hdfql_execute(script);

// unregister variable "data" as it is no longer used/needed (by HDFql)
hdfql_variable_unregister(data);
// select (i.e. read) data from dataset "my_dataset12" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset12");

// display content of cursor in use (should be 3.2, 1.3, 0, 0.2, 9.1, 6.5)
while(hdfql_cursor_next(NULL) == HDFQL_SUCCESS)
{
    while(hdfql_subcursor_next(NULL) == HDFQL_SUCCESS)
    {
        printf("%f\n", *hdfql_subcursor_get_double(NULL));
    }
}

// release memory allocated in variable "data"
free(data[0].address);
free(data[1].address);
free(data[2].address);

// declare variables
char script[1024];
char *data[3];

// create an HDF5 dataset named "my_dataset13" of data type variable-length char of one
dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset13 AS VARCHAR(3)");  

// allocate memory in variable "data"
data[0] = malloc(13 * sizeof(char));
data[1] = malloc(5 * sizeof(char));
data[2] = malloc(7 * sizeof(char));

// populate variable "data" with certain values
strcpy(data[0], "Hierarchical");
strcpy(data[1], "Data");
strcpy(data[2], "Format");

// register variable "data" for subsequent use (by HDFql)
hdfql_variable_register(data);

// prepare script to insert (i.e. write) values from variable "data" into dataset
"my_dataset13"
sprintf(script, "INSERT INTO my_dataset13 VALUES FROM MEMORY %d",
hdfql_variable_get_number(data));
// execute script
hdfql_execute(script);

// unregister variable "data" as it is no longer used/needed (by HDFql)
hdfql_variable_unregister(data);

// select (i.e. read) data from dataset "my_dataset13" and populate cursor in use with it
hdfql_execute("SELECT FROM my_dataset13");

// display content of cursor in use (should be "Hierarchical", "Data", "Format")
while(hdfql_cursor_next(NULL) == HDFQL_SUCCESS)
{
    printf("%s\n", hdfql_cursor_get_char(NULL));
}

// release memory allocated in variable "data"
free(data[0]);
free(data[1]);
free(data[2]);

// assume that the following program is launched in parallel using four MPI processes (e.g. "mpiexec -n 4 my_program")

// declare variables
char script[1024];
int rank;

// create an HDF5 file named "my_file.h5" in parallel
hdfql_execute("CREATE PARALLEL FILE my_file.h5");

// use (i.e. open) HDF5 file "my_file.h5" in parallel
hdfql_execute("USE PARALLEL FILE my_file.h5");

// create an HDF5 dataset named "my_dataset14" of data type int of one dimension (size 4)
hdfql_execute("CREATE DATASET my_dataset14 AS INT(4)";

// get number (i.e. rank) of the MPI process (should be between 0 and 3)
rank = hdfql_mpi_get_rank();

// prepare script to insert (i.e. write) values 0, 10, 20 and 30 in parallel into positions #0 (by MPI process rank 0), #1 (by MPI process rank 1), #2 (by MPI process rank 2) and #3 (by MPI
6.6 DATA QUERY LANGUAGE (DQL)

Data Query Language (DQL) is, generally speaking, syntax for retrieving data stored in structures. In HDFql, the DQL is composed of only one operation (SELECT). It enables retrieval (i.e. reading) of data stored in HDF5 datasets or attributes according to certain criteria. Moreover, it supports both POST-PROCESSING and REDIRECTING options to further transform and redirect the result of the operation according to the programmer’s needs.

6.6.1 SELECT

Syntax

```
SELECT [DIRECTLY] FROM [PARALLEL] [DATASET | ATTRIBUTE] [file_name] object_name
[(start):[stride]:[count]:[block] [, [start]:[stride]:[count]:[block]]* [, {OR | AND | XOR | NOTA | NOTB} [start]:[stride]:[count]:[block] [, [start]:[stride]:[count]:[block]]*] | {coord [, coord]* [, coord [, coord]*] | {chunk_number [, chunk_number]*])

[CACHE {SLOTS {slots_value | FILE | DEFAULT}} [SIZE {size_value | FILE | DEFAULT}] [PREEMPTION {preemption_value | FILE | DEFAULT}]

[output_redirecting_option]
```

Description

Select (i.e. read) data from an HDF5 dataset or attribute named `object_name`. In case the keyword CACHE is specified, the dataset is read using cache parametrized with the values `slots_value`, `size_value` and `preemption_value` (instead of the dataset cache parameters that may have been set through the operation SET CACHE). In case a dataset and an attribute with identical names (`object_name`) are stored in the same location (i.e. group) and neither the keyword DATASET nor ATTRIBUTE is specified, the object for which data will be read is the dataset. To explicitly read data from an object...
according to its type, the keyword DATASET or ATTRIBUTE must be specified. In case the keyword DIRECTLY\(^{42}\) is specified, HDFql reads data chunks directly from the dataset bypassing several internal processing steps of the HDF5 library itself (e.g. data conversion, filter pipeline), which can lead to a much faster reading. In case the keyword PARALLEL\(^{43}\) is specified, HDFql reads data from a dataset in parallel using all the MPI processes specified upon launching the program (that employs HDFql).

By default, the entire \textit{object}\_\textit{name} is read when performing a select operation. To read only a subset (i.e. portion) of \textit{object}\_\textit{name}, hyperslab and point selections can be used\(^{44}\). To enable a (regular) hyperslab selection, the \textit{start}, \textit{stride}, \textit{count} and \textit{block} parameters may be specified and separated with a colon (\:). For each dimension of \textit{object}\_\textit{name}, a set of such parameters may be specified and each set should be separated with a comma (,). Multiple hyperslab selections can be enabled at once (in this case, the hyperslab will be considered irregular). This is enabled by using the following boolean operators:

- \textbf{OR} – adds the new selection to the existing selection.
- \textbf{AND} – retains only the overlapping portions of the new selection and the existing selection.
- \textbf{XOR} – retains only the elements that are members of the new selection or the existing selection, excluding elements that are members of both selections.
- \textbf{NOTA} – retains only elements of the new selection that are not in the existing selection.
- \textbf{NOTB} – retains only elements of the existing selection that are not in the new selection.

To enable a point selection, a set of coordinates may be specified. Each coordinate is separated with a comma (,). More than one set of coordinates (i.e. points) may be specified and each set should be separated with a semicolon (;). Of note, hyperslab and point selections cannot be used both at the same time (i.e. be mixed) in a select operation. Since hyperslab and point selections can be complicated to set up, it is highly recommended to first read \url{https://support.hdfgroup.org/HDF5/doc1.8/UG/HDF5_Users_Guide-Responsive%20HTML5/index.html#t=HDF5_Users_Guide%2FDataspaces%2FHDF5_Dataspaces_and_Partial_I_O.htm%23T OC_7_4_1_Data_Selectionbc-7} and eventually enable the debug mechanism (through the operation \texttt{SET DEBUG}) when working with these to obtain debug information in case of errors.

---

\(^{42}\) Only available for HDF5 datasets as, by design, direct selection (i.e. read) for HDF5 attributes is not supported by the HDF5 library. Moreover, the library does not support selecting (i.e. reading) data directly from a dataset of data type variable-length or in parallel (by specifying the keyword PARALLEL).

\(^{43}\) This option is not allowed in Windows as HDFql does not support the parallel HDF5 (PHDF5) library in this platform currently.

\(^{44}\) Only available for HDF5 datasets as, by design, both hyperslab and point selections for HDF5 attributes are not supported by the HDF5 library.
HDFql provides several ways of writing result sets that was read from a dataset or attribute, namely either to a cursor (e.g. “SELECT FROM my_dataset”) or an output redirecting option (e.g. “SELECT FROM my_dataset INTO FILE my_file.txt”).

**Parameter(s)**

*file_name* – optional string that specifies the name of the HDF5 file in which the HDF5 dataset or attribute to select (i.e. read) data from is stored. If *file_name* is specified, the file is opened on the fly, the dataset or attribute is selected and, afterwards, the file is closed. Otherwise, if it is not specified, the dataset or attribute (whose data is to be selected is stored) in the file currently in use.

*object_name* – mandatory string that specifies the name of the HDF5 dataset or attribute to select (i.e. read) data from.

*start* – optional integer that specifies the starting location of the hyperslab selection. If not specified, its default value is 0 (i.e. the first position of the dimension in question). If negative, its value will be the last position of the dimension in question minus the value of *start*.

*stride* – optional integer that specifies the number of elements to separate each block to be selected. If not specified, its default value is equal to the value of *block*.

*count* – optional integer that specifies the number of blocks to select along each dimension. If not specified, its default value is 1.

*block* – optional integer that specifies the size of the block (i.e. number of elements) selected from the HDF5 dataset. If not specified, its default value is the size of the dimension in question minus the value of *start* divided by the value of *count*.

*coord* – optional integer that specifies the point of interest (i.e. to select) for the point selection. If negative, its value will be the last position of the dimension in question minus the value of *coord*.

*chunk_number* – optional integer that specifies the number of the chunk to select (i.e. read) data from. Multiple chunk numbers are separated with a comma (,). If *chunk_number* is specified it must either be between 0 and the storage dimension in question - 1 (otherwise an error will be raised) or negative (in this case its value will be the last position of the storage dimension in question minus the value of *chunk_number*). Otherwise, if it is not specified and in case the keyword DIRECTLY is specified, its default value is 0 (i.e. first chunk of the storage dimension in question).

*slots_value* – optional integer that specifies the number of chunk slots in the raw data chunk cache for accessing the HDF5 dataset. Due to the hashing strategy, its value should ideally be a prime number. In case the keyword DEFAULT is specified, its value is 521 (i.e. default value defined by the HDF5 library). In case the keyword FILE is specified, its value will be as defined in the cache slots parameter upon using (i.e. opening) the file. In case the keyword SLOTS is not specified, its
current value remains intact. Of note, if object_name is an HDF5 attribute then the cache is ignored (i.e. it is not applicable for accessing attributes).

size_value – optional integer that specifies the total size of the raw data chunk cache in bytes for accessing the HDF5 dataset. In case the keyword DEFAULT is specified, its value is 1048576 (i.e. 1 MB – default value defined by the HDF5 library). In case the keyword FILE is specified, its value will be as defined in the cache size parameter upon using (i.e. opening) the file. In case the keyword SIZE is not specified, its current value remains intact. Of note, if object_name is an HDF5 attribute then the cache is ignored (i.e. it is not applicable for accessing attributes).

preemption_value – optional float that specifies the chunk preemption policy for accessing the HDF5 dataset. Its value must be between 0 and 1. It indicates the weighting according to which chunks which have been fully read or written are penalized when determining which chunks to flush from cache. In case the keyword DEFAULT is specified, its value is 0.75 (i.e. default value defined by the HDF5 library). In case the keyword FILE is specified, its value will be as defined in the cache preemption parameter upon using (i.e. opening) the file. In case the keyword PREEMPTION is not specified, its current value remains intact. Of note, if object_name is an HDF5 attribute then the cache is ignored (i.e. it is not applicable for accessing attributes).

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The data selected (i.e. read) from an HDF5 dataset or attribute as an HDFQL_TINYINT (in case the data type of the dataset or attribute is HDFQL_TINYINT), HDFQL_UNSIGNED_TINYINT (in case the data type of the dataset or attribute is HDFQL_UNSIGNED_TINYINT), HDFQL_SMALLINT (in case the data type of the dataset or attribute is HDFQL_SMALLINT), HDFQL_UNSIGNED_SMALLINT (in case the data type of the dataset or attribute is HDFQL_UNSIGNED_SMALLINT), HDFQL_INT (in case the data type of the dataset or attribute is HDFQL_INT), HDFQL_UNSIGNED_INT (in case the data type of the dataset or attribute is HDFQL_UNSIGNED_INT), HDFQL_BIGINT (in case the data type of the dataset or attribute is HDFQL_BIGINT), HDFQL_UNSIGNED_BIGINT (in case the data type of the dataset or attribute is HDFQL_UNSIGNED_BIGINT), HDFQL_FLOAT (in case the data type of the dataset or attribute is HDFQL_FLOAT), HDFQL_DOUBLE (in case the data type of the dataset or attribute is HDFQL_DOUBLE), HDFQL_CHAR (in case the data type of the dataset or attribute is HDFQL_CHAR), HDFQL_VARTINYINT (in case the data type of the dataset or attribute is HDFQL_VARTINYINT), HDFQL_UNSIGNED_VARTINYINT (in case the data type of the dataset or attribute is HDFQL_UNSIGNED_VARTINYINT).
HDFQL_UNSIGNED_VARTINYINT), HDFQL_VARSMALLINT (in case the data type of the dataset or attribute is HDFQL_VARSMALLINT), HDFQL_UNSIGNED_VARSMALLINT (in case the data type of the dataset or attribute is HDFQL_UNSIGNED_VARSMALLINT), HDFQL_VARINT (in case the data type of the dataset or attribute is HDFQL_VARINT), HDFQL_UNSIGNED_VARINT (in case the data type of the dataset or attribute is HDFQL_UNSIGNED_VARINT), HDFQL_VARBIGINT (in case the data type of the dataset or attribute is HDFQL_VARBIGINT), HDFQL_UNSIGNED_VARBIGINT (in case the data type of the dataset or attribute is HDFQL_UNSIGNED_VARBIGINT), HDFQL_VARFLOAT (in case the data type of the dataset or attribute is HDFQL_VARFLOAT), HDFQL_VARDOUBLE (in case the data type of the dataset or attribute is HDFQL_VARDOUBLE), HDFQL_VARCHAR (in case the data type of the dataset or attribute is HDFQL_VARCHAR) or HDFQL_OPAQUE (in case the data type of the dataset or attribute is HDFQL_OPAQUE).

Example(s)

```sql
# create an HDF5 dataset named "my_dataset0" of data type short of one dimension (size 3) with initial values of 65, 66 and 77
CREATE DATASET my_dataset0 AS SMALLINT(3) VALUES(65, 66, 67)

# select (i.e. read) data from dataset "my_dataset0" and populate cursor in use with it (should be 65, 66, 67)
SELECT FROM my_dataset0

# create an HDF5 attribute named "my_attribute0" of data type short
CREATE ATTRIBUTE my_attribute0 AS SMALLINT

# select (i.e. read) data from attribute "my_attribute0" and populate cursor in use with it (should be 0)
SELECT FROM my_attribute0

# create an HDF5 attribute named "my_attribute1" of data type unsigned short of one dimension (size 2) with initial values of 95 and 97
CREATE ATTRIBUTE my_attribute1 AS UNSIGNED SMALLINT(2) VALUES(95, 97)

# select (i.e. read) data from attribute "my_attribute1" and populate cursor in use with it (should be 95, 97)
SELECT FROM my_attribute1

# create an HDF5 dataset named "my_dataset1" of data type float of one dimension (size 512)
CREATE DATASET my_dataset1 AS FLOAT(512)

# select (i.e. read) data from dataset "my_dataset1" and write it into a text file named
```
"my_file0.txt" using default separator ",",

```
SELECT FROM my_dataset1 INTO FILE my_file0.txt
```

# select (i.e. read) data from dataset "my_dataset1" and write it into a text file named "my_file1.txt" using a DOS-based end of line (EOL) terminator and separator "**"
```
SELECT FROM my_dataset1 INTO DOS TEXT FILE my_file1.txt SEPARATOR **
```

# select (i.e. read) data from dataset "my_dataset1" and write it into a binary file named "my_file.bin"
```
SELECT FROM my_dataset1 INTO BINARY FILE my_file.bin
```
count of 2 and block of 1) and populate cursor in use with it (should be 4, 6)

```
SELECT FROM my_dataset3(2:::, 0:2;2:1)
```

# create an HDF5 dataset named "my_dataset4" of data type short of one dimension (size 10)

```
CREATE DATASET my_dataset4 AS SMALLINT(10)
```

# insert (i.e. write) values 0, 0, 90, 91, 93, 0, 94, 95 and 0 into dataset "my_dataset4"

```
INSERT INTO my_dataset4 VALUES(0, 0, 90, 91, 93, 0, 94, 95, 0)
```

# select (i.e. read) data from dataset "my_dataset4" using an irregular hyperslab selection
# (starting from position #2 with a count of 3 and block of 1; starting from position #4 with a count of 2 and block of 1; starting from position #7 with a count of 2 and block of 1) and populate cursor in use with it (should be 90, 91, 93, 94, 95)

```
SELECT FROM my_dataset4(2::3:1 OR 4::2:1 OR 7::2:1)
```

# create an HDF5 dataset named "my_dataset5" of data type long long of one dimension (size 15)

```
CREATE DATASET my_dataset5 AS BIGINT(15)
```

# insert (i.e. write) values 0, 0, 0, 0, 0, 75, 77, 0, 0, 0, 0, 0, 0 and 0 into dataset "my_dataset5"

```
INSERT INTO my_dataset5 VALUES(0, 0, 0, 0, 0, 75, 77, 0, 0, 0, 0, 0, 0, 0, 0)
```

# select (i.e. read) data from dataset "my_dataset5" using an irregular hyperslab selection
# (starting from position #3 with a count of 4 and block of 1; starting from position #5 with a count of 3 and block of 1) and populate cursor in use with it (should be 75, 77)

```
SELECT FROM my_dataset5(3::4:1 AND 5::3:1)
```

# create an HDF5 dataset named "my_dataset6" of data type float of one dimension (size 8)

```
CREATE DATASET my_dataset6 AS FLOAT(8)
```

# insert (i.e. write) values 0, 0, 7.5, 0, 7.7, 0, 0 and 7.9 into dataset "my_dataset6"

```
INSERT INTO my_dataset6 VALUES(0, 0, 7.5, 0, 7.7, 0, 0, 7.9)
```

# select (i.e. read) data from dataset "my_dataset6" using a point selection (positions #2, #4 and #7) and populate cursor in use with it (should be 7.5, 7.7, 7.9)

```
SELECT FROM my_dataset6(2; 4; 7)
```

# create an HDF5 dataset named "my_dataset7" of data type double of two dimensions (size 4x3)

```
CREATE DATASET my_dataset7 AS DOUBLE(4, 3)
```

# insert (i.e. write) values 0, 0, 0, 0, 15.2, 0, 0, 18.5 and 0 into dataset "my_dataset7"

```
INSERT INTO my_dataset7 VALUES(0, 0, 0, 0, 15.2, 0, 0, 18.5, 0)
```
# select (i.e. read) data from dataset "my_dataset7" using a point selection (position #1 of the first dimension and position #2 of the second dimension, position #3 of the first dimension and position #0 of the second dimension) and populate cursor in use with it (should be 15.2, 18.5)

```sql
SELECT FROM my_dataset7(1, 2; 3, 0)
```

# use (i.e. open) an HDF5 file named "my_file.h5"

```sql
USE FILE my_file.h5
```

# create an HDF5 dataset named "my_dataset8" of data type double in the HDF5 file currently in use (i.e. file "my_file.h5")

```sql
CREATE DATASET my_dataset8 AS DOUBLE
```

# insert (i.e. write) value 6.5 into dataset "my_dataset8"

```sql
INSERT INTO my_dataset8 VALUES(6.5)
```

# select (i.e. read) data from dataset "my_dataset8" and populate cursor in use with it (should be 6.5)

```sql
SELECT FROM my_dataset8
```

# close HDF5 file currently in use (i.e. file "my_file.h5")

```sql
CLOSE FILE
```

# insert (i.e. write) value 3.2 into dataset "my_dataset8" in file "my_file.h5"

```sql
INSERT INTO my_file.h5 my_dataset8 VALUES(3.2)
```

# select (i.e. read) data from dataset "my_dataset8" in file "my_file.h5" and populate cursor in use with it (should be 3.2)

```sql
SELECT FROM my_file.h5 my_dataset8
```

# create a chunked (size 2) HDF5 dataset named "my_dataset9" of data type int of one dimension (size 6)

```sql
CREATE CHUNKED(2) DATASET my_dataset9 AS INT(6)
```

# insert (i.e. write) values 60, 61, 62, 63, 64 and 65 into dataset "my_dataset9"

```sql
INSERT INTO my_dataset9 VALUES(60, 61, 62, 63, 64, 65)
```

# select (i.e. read) data directly from chunk #0 of dataset "my_dataset9" (should be 60, 61)

```sql
SELECT DIRECTLY FROM my_dataset9
```
# select (i.e. read) data directly from chunk #1 of dataset "my_dataset9" (should be 62, 63)

```
SELECT DIRECTLY FROM my_dataset9(1)
```

# select (i.e. read) data directly from chunk #2 of dataset "my_dataset9" (should be 64, 65)

```
SELECT DIRECTLY FROM my_dataset9(2)
```
// create an HDF5 dataset named "my_dataset1" of data type variable-length double of one
dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset1 AS VARDOUBLE(3)");

// insert (i.e. write) values into dataset "my_dataset1"
hdfql_execute("INSERT INTO my_dataset1 VALUES((3.2, 1.3), (0, 0.2), (9.1, 7.4, 6.5))");

// register variable "data" for subsequent use (by HDFql)
hdfql_variable_register(data);

// prepare script to select (i.e. read) data from dataset "my_dataset1" and populate variable
"data" with it
sprintf(script, "SELECT FROM my_dataset1 INTO MEMORY %d", hdfql_variable_get_number(data));

// execute script
hdfql_execute(script);

// unregister variable "data" as it is no longer used/needed (by HDFql)
hdfql_variable_unregister(data);

// display content of cursor in use (should be 3.2, 1.3, 0, 0.2, 9.1, 7.4, 6.5)
for(x = 0; x < 3; x++)
{
    count = data[x].count;
    for(y = 0; y < count; y++)
    {
        printf("%f\n", *((double *) data[x].address + y));
    }
}

// release memory allocated (by HDFql) in variable "data"
for(x = 0; x < 3; x++)
{
    free(data[x].address);
}

// declare variables
char script[1024];
char *data[3];
int x;
// create an HDF5 dataset named "my_dataset12" of data type variable-length char of one dimension (size 3)
hdfql_execute("CREATE DATASET my_dataset12 AS VARCHAR(3)");

// insert (i.e. write) values into dataset "my_dataset12"
hdfql_execute("INSERT INTO my_dataset12 VALUES("Hierarchical", "Data", "Format")");

// register variable "data" for subsequent use (by HDFql)
hdfql_variable_register(data);

// prepare script to select (i.e. read) data from dataset "my_dataset12" and populate variable "data" with it
sprintf(script, "SELECT FROM my_dataset12 INTO MEMORY %d", hdfql_variable_get_number(data));

// execute script
hdfql_execute(script);

// unregister variable "data" as it is no longer used/needed (by HDFql)
hdfql_variable_unregister(data);

// display content of cursor in use (should be "Hierarchical", "Data", "Format")
for(x = 0; x < 3; x++)
{
    printf("%s\n", data[x]);
}

// release memory allocated (by HDFql) in variable "data"
for(x = 0; x < 3; x++)
{
    free(data[x]);
}

// assume that 1) the following program is launched in parallel using four MPI processes (e.g. "mpiexec -n 4 my_program"), 2) an HDF5 file named "my_file.h5" containing a dataset named "my_dataset13" of data type int of one dimension (size 4) already exists, and 3) the dataset stores the values 0, 10, 20 and 30 in positions #0, #1, #2 and #3 respectively

// declare variables
char script[1024];
int rank;

// use (i.e. open) an HDF5 file named "my_file.h5" in parallel
hdfql_execute("USE PARALLEL FILE my_file.h5");

// get number (i.e. rank) of the MPI process (should be between 0 and 3)
rank = hdfql_mpi_get_rank();

// prepare script to select (i.e. read) in parallel positions #0 (by MPI process rank 0), #1
(by MPI process rank 1), #2 (by MPI process rank 2) and #3 (by MPI process rank 3) from dataset
"my_dataset13" using a point selection
sprintf(script, "SELECT FROM PARALLEL my_dataset13(%d)", rank);

// execute script
hdfql_execute(script);

// move the cursor in use to the first position within the result set
hdfql_cursor_first(NULL);

// display value selected (i.e. read) by each MPI process (should display message "Value read
by MPI process rank X is Y" four times, where X is 0 and Y is 0, X is 1 and Y is 10, X is 2 and
Y is 20, or X is 3 and Y is 30 (not necessarily in this order))
printf("Value read by MPI process rank %d is %d\n", rank, *hdfql_cursor_get_int(NULL));

### 6.7 DATA INTROSPECTION LANGUAGE (DIL)

HDFql has certain operations that retrieve information about the internals of HDF5 files but also about HDFql itself and the runtime environment. These operations are part of the Data Introspection Language (DIL) and they all begin with the keyword SHOW. Moreover, these operations support both POST-PROCESSING and REDIRECTING options to further transform and redirect the result of operations according to the programmer’s needs. Typically, a DIL operation has the following syntactical form:

```
SHOW operation_name [post_processing_option [post_processing_option]*] [output_redirecting_option]
```

### 6.7.1 SHOW FILE VALIDITY

**Syntax**

```
SHOW FILE VALIDITY file_name [, file_name]*
       [post_processing_option [post_processing_option]*]
```
[output_redirecting_option]

Description

Show (i.e. get) validity of a file named file_name. Multiple files' validities can be checked at once by separating these with a comma (,). If file_name was not found or its validity could not be checked (due to unknown/unexpected reasons), no subsequent files are checked, and an error is raised.

Parameter(s)

file_name – mandatory string that specifies the name of the file whose validity is to be obtained. Multiple files are separated with a comma (,).

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST_PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The validity of a file as an HDFQL_INT, which can either be HDFQL_YES or HDFQL_NO depending on whether the file is a valid HDF5 file or not.

Example(s)

```plaintext
# create an HDF5 file named "my_file.h5"
CREATE FILE my_file.h5

# show (i.e. get) validity of file "my_file.h5" (should be 0 – i.e. HDFQL_YES)
SHOW FILE VALIDITY my_file.h5

# show (i.e. get) validity of a file named "not_an_hdf_file.xml" (should be -1 – i.e. HDFQL_NO)
(assume that the file "not_an_hdf_file.xml" exists and contains XML text)
SHOW FILE VALIDITY not_an_hdf_file.xml

# show (i.e. get) validity of both files "my_file.h5" and "not_an_hdf_file.xml" at once (should be 0, -1)
SHOW FILE VALIDITY my_file.h5, not_an_hdf_file.xml
```
6.7.2 SHOW USE DIRECTORY

Syntax

SHOW USE DIRECTORY

[post_processing_option [post_processing_option]*]

[output_redirecting_option]

Description

Show (i.e. get) working directory currently in use.

Parameter(s)

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The working directory currently in use as an HDFQL_VARCHAR.

Example(s)

# set working directory currently in use to "/"
USE DIRECTORY /

# show (i.e. get) current working directory (should be "/")
SHOW USE DIRECTORY

# create a directory named "my_directory"
CREATE DIRECTORY my_directory
# set working directory currently in use to "my_directory" (more precisely "/my_directory")
USE DIRECTORY my_directory

# show (i.e. get) current working directory (should be "/my_directory")
SHOW USE DIRECTORY

# create two directories named "my_subdirectory0" and "my_subdirectory1" (both directories will be created in directory "/my_directory")
CREATE DIRECTORY my_subdirectory0, my_subdirectory1

# set directory currently in use to "my_subdirectory0" (more precisely "/my_directory/my_subdirectory0")
USE DIRECTORY my_subdirectory0

# show (i.e. get) current working directory (should be "/my_directory/my_subdirectory0")
SHOW USE DIRECTORY

# set directory currently in use to "my_subdirectory1" located one level up (more precisely "/my_directory/my_subdirectory1")
USE DIRECTORY ../my_subdirectory1

# show (i.e. get) current working directory (should be "/my_directory/my_subdirectory1")
SHOW USE DIRECTORY

# set directory currently in use two levels up (should be "/")
USE DIRECTORY ../..

# show (i.e. get) current working directory (should be "/")
SHOW USE DIRECTORY

## 6.7.3 SHOW USE FILE

**Syntax**

SHOW USE FILE

=post_processing_option [post_processing_option]*

=output_redirecting_option*
**Description**

Show (i.e. get) HDF5 file currently in use.

**Parameter(s)**

- `post_processing_option` – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

- `output_redirecting_option` – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

**Return**

The HDF5 file currently in use as an HDFQL_VARCHAR or nothing (in case no file is in use).

**Example(s)**

```sql
# show (i.e. get) HDF5 file currently in use (i.e. open) (should be empty)
SHOW USE FILE

# use (i.e. open) four HDF5 files named "my_file0.h5", "my_file1.h5", "my_file2.h5" and "my_file3.h5"
USE FILE my_file0.h5, my_file1.h5, my_file2.h5, my_file3.h5

# show (i.e. get) HDF5 file currently in use (i.e. open) (should be "my_file3.h5")
SHOW USE FILE

# close HDF5 file currently in use (i.e. file "my_file3.h5")
CLOSE FILE

# show (i.e. get) HDF5 file currently in use (i.e. open) (should be my_"file2.h5")
SHOW USE FILE

# close HDF5 file "my_file1.h5"
CLOSE FILE my_file1.h5

# show (i.e. get) HDF5 file currently in use (i.e. open) (should be "my_file2.h5")
SHOW USE FILE
```
```hdfql
# close HDF5 file currently in use (i.e. file "my_file2.h5")
CLOSE FILE

# show (i.e. get) HDF5 file currently in use (i.e. open) (should be "my_file0.h5")
SHOW USE FILE

# close HDF5 file currently in use (i.e. file "my_file0.h5")
CLOSE FILE

# show (i.e. get) HDF5 file currently in use (i.e. open) (should be empty)
SHOW USE FILE
```

### 6.7.4 SHOW ALL USE FILE

**Syntax**

```hdfql
SHOW ALL USE FILE

[post_processing_option [post_processing_option] ]

[output_redirecting_option]
```

**Description**

Show (i.e. get) all HDF5 files in use (i.e. open).

**Parameter(s)**

*post_processing_option* – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

*output_redirecting_option* – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

**Return**

All HDF5 files in use (i.e. open) as an HDFQL_VARCHAR or nothing (in case no files are in use).
Example(s)

```bash
# show (i.e. get) all HDF5 files in use (i.e. open) (should be empty)
SHOW ALL USE FILE

# use (i.e. open) three HDF5 files named "my_file0.h5", "my_file1.h5" and "my_file2.h5"
USE FILE my_file0.h5, my_file1.h5, my_file2.h5

# show (i.e. get) all HDF5 files in use (i.e. open) (should be "my_file2.h5", "my_file1.h5", "my_file0.h5")
SHOW ALL USE FILE

# close all HDF5 files in use (i.e. open)
CLOSE ALL FILE

# show (i.e. get) all HDF5 files in use (i.e. open) (should be empty)
SHOW ALL USE FILE
```

6.7.5 SHOW USE GROUP

Syntax

SHOW USE GROUP

```bash
[post_processing_option [post_processing_option]*]

[output_redirecting_option]
```

Description

Show (i.e. get) HDF5 group currently in use.

Parameter(s)

`post_processing_option` – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.
output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

**Return**

The HDF5 group currently in use as an HDFQL_VARCHAR or nothing (in case no file is in use).

**Example(s)**

```bash
# use (i.e. open) an HDF5 file named "my_file.h5"
USE FILE my_file.h5

# show (i.e. get) current working group (should be "/")
SHOW USE GROUP

# create an HDF5 group named "my_group"
CREATE GROUP my_group

# set group currently in use to "my_group" (more precisely "/my_group")
USE GROUP my_group

# show (i.e. get) current working group (should be "/my_group")
SHOW USE GROUP

# create two HDF5 groups named "my_subgroup0" and "my_subgroup1" (both groups will be created in group "/my_group")
CREATE GROUP my_subgroup0, my_subgroup1

# set group currently in use to "my_subgroup0" (more precisely "/my_group/my_subgroup0")
USE GROUP my_subgroup0

# show (i.e. get) current working group (should be "/my_group/my_subgroup0")
SHOW USE GROUP

# set group currently in use to "." (the group currently in use will not change as "." refers to the current working group itself)
USE GROUP .

# show (i.e. get) current working group (should be "/my_group/my_subgroup0")
SHOW USE GROUP

# set group currently in use to "my_subgroup1" located one level up (more precisely
```
6.7.6 SHOW [GROUP | DATASET | ATTRIBUTE | [SOFT] LINK | EXTERNAL LINK]

Syntax

SHOW [GROUP | DATASET | ATTRIBUTE | [SOFT] LINK | EXTERNAL LINK] [[file_name] object_name]

[LIKE regular_expression [DEEP deep_value [, deep_value]*]]

[WHERE condition]

[ORDER CREATION]

[post_processing_option [post_processing_option]*]

[output redirecting_option]

Description

Show (i.e. get) HDF5 objects (i.e. groups, datasets, attributes, (soft) links or external links) within an HDF5 group or dataset named object_name or check the existence of an object named object_name. If object_name is not specified, all objects are returned – to return only objects of type group, dataset, attribute, (soft) link or external link, specify the keyword GROUP, DATASET, ATTRIBUTE, [SOFT] LINK or EXTERNAL LINK respectively. Otherwise, if object_name is specified and the keyword LIKE is not specified, one of the following behaviors applies:

- If it ends with "/", object_name will be treated as a group or dataset, and all groups, datasets or attributes stored in object_name are returned.
- If it does not end with "/", object_name will be checked for its existence. If it does exist, object_name is returned; otherwise, if it does not exist, an error is raised.

If the keyword LIKE is specified, only objects with names complying with a regular expression named regular_expression will be returned (in HDFql, regular expressions are the ones specified by PCRE which closely follow PERL5 syntax – please refer to http://www.pcre.org and http://perldoc.perl.org/perlre.html for additional information). If regular_expression
includes “***”, recursive search is performed (i.e. HDFql will search in all existing groups and subgroups for objects). To limit the recursiveness, the keyword DEEP may be specified along with a value deep_value representing the maximum recursiveness limit.

A special type of ordering can be performed using the keyword ORDER CREATION allowing HDF5 objects (i.e. groups, datasets, attributes, (soft) links or external links) to be returned according to their time of creation45 – in contrast to the default behavior which returns objects in an ascending order.

**Parameter(s)**

- **file_name** – optional string that specifies the name of the HDF5 file which stores the objects (i.e. groups, datasets, attributes, (soft) links or external links) to show (i.e. get) or check for their existence. If file_name is specified, the file is opened on the fly, the objects are obtained or checked for their existence and, afterwards, the file is closed. Otherwise, if it is not specified, the objects to obtain or check for their existence are stored in the file currently in use.

- **object_name** – optional string that specifies the name of the HDF5 group or dataset to show (i.e. get) existing objects (i.e. groups, datasets, attributes, (soft) links or external links) within object_name or check the existence of an object named object_name.

- **regular_expression** – optional string that specifies the regular expression which only names of objects that comply with it are returned. If regular_expression includes “***”, recursive search is performed.

- **deep_value** – optional integer that specifies the maximum recursiveness limit.

- **condition** – to be defined.

- **post_processing_option** – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

- **output redirecting_option** – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

---

45 This assumes that the HDF5 group or dataset storing the objects was created with the option of tracking objects by their time of creation. Otherwise, if the group or dataset was not created with the option of tracking objects by their time of creation, the keyword ORDER CREATION is ignored. Please refer to the CREATE GROUP and CREATE DATASET operations for additional information.
Return

The HDF5 objects (i.e. groups, datasets, attributes, (soft) links or external links) within an HDF5 group or dataset or the existence of an object as an HDFQL_VARCHAR.

Example(s)

```hdf
# set group currently in use to "/" (i.e. the root of the HDF5 file)
USE GROUP /

# create two HDF5 groups named "my_group0" and "my_group1" (both groups will be created in group "/")
CREATE GROUP my_group0, my_group1

# create one HDF5 dataset named "my_dataset0" of data type unsigned short (it will be created in group "/")
CREATE DATASET my_dataset0 AS UNSIGNED SMALLINT

# create one HDF5 dataset named "my_dataset1" of data type short (it will be created in group "/my_group0")
CREATE DATASET my_group0/my_dataset1 AS SMALLINT

# create two HDF5 attributes named "my_attribute0" and "my_attribute1" of data type long long (both attributes will be created in group "/")
CREATE ATTRIBUTE my_attribute0, my_attribute1 AS BIGINT

# create one HDF5 attribute named "my_attribute2" of data type char (it will be created in group "/my_group0")
CREATE ATTRIBUTE my_group0/my_attribute2 AS TINYINT

# create one HDF5 attribute named "my_attribute3" of data type unsigned char (it will be created in dataset "/my_dataset0")
CREATE ATTRIBUTE my_dataset0/my_attribute3 AS UNSIGNED TINYINT

# show (i.e. get) all HDF5 objects existing in group "/" (should be "my_group0", "my_group1", "my_dataset0", "my_attribute0", "my_attribute1")
SHOW

# show (i.e. get) all HDF5 groups existing in group "/" (should be "my_group0", "my_group1")
SHOW GROUP

# show (i.e. get) all HDF5 datasets existing in group "/" (should be "my_dataset0")
SHOW DATASET
```
# check if HDF5 object "my_groupX" exists (should raise an error)
SHOW my_groupX

# check if HDF5 object "my_group0" exists (should be "my_group0")
SHOW my_group0

# show (i.e. get) all HDF5 objects existing within group "my_group0" (should be "my_dataset1", "my_attribute2")
SHOW my_group0/

# show (i.e. get) all HDF5 attributes existing within group "my_group0" (should be "my_attribute2")
SHOW ATTRIBUTE my_group0/

# show (i.e. get) all HDF5 objects existing within dataset "my_dataset0" (should be "my_attribute3")
SHOW my_dataset0/

# create an HDF5 group named "my_group1" that tracks the objects’ (i.e. groups and datasets) creation order within the group
CREATE GROUP my_group1 ORDER TRACKED

# create two HDF5 groups named "my_subgroup1" and "my_subgroup0" (both groups will be created in group "/my_group1")
CREATE GROUP my_group1/my_subgroup1, my_group1/my_subgroup0

# create two HDF5 datasets named "my_dataset1" and "my_dataset0" of data type float (both datasets will be created in group "/my_group1")
CREATE DATASET my_group1/my_dataset1, my_group1/my_dataset0 AS FLOAT

# show (i.e. get) all HDF5 objects existing within group "my_group1" (should be "my_dataset0", "my_dataset1", "my_subgroup0", "my_subgroup1")
SHOW my_group1/

# show (i.e. get) all HDF5 objects existing within group "my_group1" ordered by their time of creation (should be "my_subgroup1", "my_subgroup0", "my_dataset1", "my_dataset0")
SHOW my_group1/ ORDER CREATION

# create an HDF5 dataset named "my_dataset1" of data type double that tracks the attributes’ creation order within the dataset
CREATE DATASET my_dataset1 AS DOUBLE ATTRIBUTE ORDER TRACKED
# create two HDF5 attributes named "my_attribute2" and "my_attribute0" of data type int (both attributes will be created in dataset "/my_dataset1")

CREATE ATTRIBUTE my_dataset1/my_attribute2, my_dataset1/my_attribute0 AS INT

# create an HDF5 attribute named "my_attribute1" of data type short (it will be created in dataset "/my_dataset1")

CREATE ATTRIBUTE my_dataset1/my_attribute1 AS SMALLINT

# show (i.e. get) all HDF5 objects existing within dataset "my_dataset1" (should be "my_attribute0", "my_attribute1", "my_attribute2")

SHOW my_dataset1/

# show (i.e. get) all HDF5 objects existing within dataset "my_dataset1" ordered by their time of creation (should be "my_attribute2", "my_attribute0", "my_attribute1")

SHOW my_dataset1/ ORDER CREATION

# create an HDF5 group named "my_group2"

CREATE GROUP my_group2

# create two HDF5 groups named "my_subgroup0" and "my_subgroup1" (both groups will be created in group "/my_group2")

CREATE GROUP my_group2/my_subgroup0, my_group2/my_subgroup1

# create three HDF5 groups in one go named "my_group3" (in root group "/"), "my_subgroup0" (in group "my_group3") and "my_subsubgroup0" (in group "my_group3/my_subgroup0")

CREATE GROUP my_group3/my_subgroup0/my_subsubgroup0

# create an HDF5 dataset named "my_dataset2" (in root group "/") of data type double

CREATE DATASET my_dataset2 AS DOUBLE

# create an HDF5 dataset named "my_dataset0" (in group "my_group2") of data type int

CREATE DATASET my_group2/my_dataset0 AS INT

# create an HDF5 dataset named "my_dataset1" (in group "my_group2") of data type short

CREATE DATASET my_group2/my_dataset1 AS SMALLINT

# create an HDF5 dataset named "my_dataset0" (in group "my_group3") of data type float

CREATE DATASET my_group3/my_dataset0 AS FLOAT

# create an HDF5 dataset named "my_dataset0" (in group "my_group3/my_subgroup0") of data type char
```sql
CREATE DATASET my_group3/my_subgroup0/my_dataset0 AS TINYINT

CREATE ATTRIBUTE my_attribute3 AS BIGINT

CREATE ATTRIBUTE my_attribute4 AS UNSIGNED INT

CREATE ATTRIBUTE my_group2/my_attribute0, my_group3/my_attribute0 AS VARFLOAT

CREATE ATTRIBUTE my_dataset2/my_attribute0 AS VARCHAR

SHOW LIKE 3

SHOW ATTRIBUTE LIKE 3

SHOW LIKE **

SHOW DATASET LIKE **

SHOW DATASET LIKE ** DEEP 1
```

# show (i.e. get) all HDF5 objects recursively starting from group "my_group3" (should be "my_attribute0", "my_dataset0", "my_subgroup0", "my_subgroup0/my_dataset0", "my_subgroup0/my_dataset0/my_dataset0", "my_subgroup0/my_dataset0/my_dataset0/my_dataset0")

```sql
SHOW LIKE **

SHOW DATASET LIKE **

SHOW DATASET LIKE ** DEEP 1
```
"my_subgroup0/my_subsubgroup0")
SHOW my_group3 LIKE **

# show (i.e. get) all HDF5 groups recursively starting from group "my_group3" (should be "my_subgroup0", "my_subgroup0/my_subsubgroup0")
SHOW GROUP my_group3 LIKE **

# show (i.e. get) all HDF5 objects recursively starting from group "/" that has "2" in their names (should be "my_dataset2", "my_group2")
SHOW LIKE **/2

# show (i.e. get) all HDF5 groups recursively starting from group "/" that has "1" or "2" in their names (should be "my_group2", "my_group2/my_subgroup1")
SHOW GROUP LIKE **/1|2

# show (i.e. get) all HDF5 objects recursively starting from group "/" that starts with "sub" in their names (should be "my_group2/my_subgroup0", "my_group2/my_subgroup1", "my_group3/my_subgroup0", "my_group3/my_subgroup0/my_subsubgroup0")
SHOW LIKE **/"my_sub

# use (i.e. open) an HDF5 file named "my_file.h5"
USE FILE my_file.h5

# create an HDF5 group named "my_group3" in the HDF5 file currently in use (i.e. file "my_file.h5")
CREATE GROUP my_group3

# create two HDF5 datasets named "my_dataset3" and "my_dataset4" of data type double both in the HDF5 file currently in use (i.e. file "my_file.h5")
CREATE DATASET my_dataset3, my_dataset4 AS DOUBLE

# show (i.e. get) all HDF5 objects existing in group "/" of the HDF5 file currently in use (i.e. file "my_file.h5") (should be "my_group3", "my_dataset3", "my_dataset4")
SHOW /

# close HDF5 file currently in use (i.e. file "my_file.h5")
CLOSE FILE

# show (i.e. get) all HDF5 objects existing in group "/" of file "my_file.h5" (should be "my_group3", "my_dataset3", "my_dataset4")
SHOW my_file.h5 /
6.7.7 SHOW TYPE

Syntax

SHOW TYPE [file_name] object_name [, [file_name] object_name]*

   [post_processing_option [post_processing_option] ]*

   [output_redirecting_option]

Description

Show (i.e. get) type of an object named object_name. Multiple objects’ types can be obtained at once by separating these with a comma (,). If object_name was not found or its type could not be checked (due to unknown/unexpected reasons), no subsequent objects are checked, and an error is raised.

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file which stores the object to show (i.e. get) the type. If file_name is specified, the file is opened on the fly, the type of the object is obtained and, afterwards, the file is closed. Otherwise, if it is not specified, the object (whose type is to be obtained) is stored in the file currently in use.

object_name – name of the object whose type is to be obtained. Multiple objects are separated with a comma (,).

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

outputRedirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The type of an object as an HDFQL_INT, which can either be HDFQL_GROUP, HDFQL_DATASET, HDFQL_ATTRIBUTE, HDFQL_GROUP | HDFQL_SOFT_LINK, HDFQL_DATASET | HDFQL_SOFT_LINK, HDFQL_GROUP | HDFQL_EXTERNAL_LINK, or HDFQL_DATASET | HDFQL_EXTERNAL_LINK depending on whether the object is a group, dataset, attribute, group and (soft) link at the same time, dataset and (soft) link at the same time, group and external link at the same time, or dataset and external link at the same time, respectively.
Example(s)

```hdfql
# create an HDF5 group named "my_object0"
CREATE GROUP my_object0

# create an HDF5 dataset named "my_object1" of data type double
CREATE DATASET my_object1 AS DOUBLE

# create an HDF5 attribute named "my_object2" of data type float
CREATE ATTRIBUTE my_object2 AS FLOAT

# create an HDF5 soft link named "my_object3" to object "my_object0"
CREATE SOFT LINK my_object3 TO my_object0

# create an HDF5 external link named "my_object4" to object "my_object" (assumed to be a dataset) in file "my_file.h5"
CREATE EXTERNAL LINK my_object4 TO my_file.h5 my_object

# show (i.e. get) type of object "my_object0" (should be 4 - i.e. HDFQL_GROUP)
SHOW TYPE my_object0

# show (i.e. get) type of object "my_object1" (should be 8 - i.e. HDFQL_DATASET)
SHOW TYPE my_object1

# show (i.e. get) type of object "my_object2" (should be 16 - i.e. HDFQL_ATTRIBUTE)
SHOW TYPE my_object2

# show (i.e. get) type of both objects "my_object0" and "my_object2" at once (should be 4, 16)
SHOW TYPE my_object0, my_object2

# show (i.e. get) type of object "my_object3" (should be 36 - i.e. HDFQL_GROUP | HDFQL_SOFT_LINK)
SHOW TYPE my_object3

# show (i.e. get) type of object "my_object4" (should be 136 - i.e. HDFQL_DATASET | HDFQL_EXTERNAL_LINK)
SHOW TYPE my_object4
```
6.7.8  SHOW DATA TYPE

Syntax

SHOW [DATASET | ATTRIBUTE] DATA TYPE [file_name] object_name [, [file_name] object_name]*

[post_processing_option [post_processing_option]*/

[output_redirecting_option]

Description

Show (i.e. get) data type of an HDF5 dataset or attribute named object_name. Multiple objects’ data types can be obtained at once by separating these with a comma (,). If object_name was not found or its data type could not be checked (due to unknown/unexpected reasons), no subsequent objects are checked, and an error is raised. In case a dataset and an attribute with identical names (object_name) are stored in the same location (i.e. group) and neither the keyword DATASET nor ATTRIBUTE is specified, the data type returned belongs to the dataset. To explicitly get the data type of object_name according to its type, the keyword DATASET or ATTRIBUTE must be specified.

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file which stores the dataset or attribute to show (i.e. get) the data type. If file_name is specified, the file is opened on the fly, the data type of the dataset or attribute is obtained and, afterwards, the file is closed. Otherwise, if it is not specified, the dataset or attribute (whose data type is to be obtained) is stored in the file currently in use.

object_name – mandatory string that specifies the name of the HDF5 dataset or attribute whose data type is to be obtained. Multiple datasets or attributes are separated with a comma (,).

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).
Return

The data type of an HDF5 dataset or attribute as an HDFQL_INT, which can either be HDFQL_TINYINT, HDFQL_UNSIGNED_TINYINT, HDFQL_SMALLINT, HDFQL_UNSIGNED_SMALLINT, HDFQL_INT, HDFQL_UNSIGNED_INT, HDFQL_BIGINT, HDFQL_UNSIGNED_BIGINT, HDFQL_FLOAT, HDFQL_DOUBLE, HDFQL_CHAR, HDFQL_VARTINYINT, HDFQL_UNSIGNED_VARTINYINT, HDFQL_VARSMALLINT, HDFQL_UNSIGNED_VARSMALLINT, HDFQL_VARINT, HDFQL_UNSIGNED_VARINT, HDFQL_VARBIGINT, HDFQL_UNSIGNED_VARBIGINT, HDFQL_VARFLOAT, HDFQL_VARDOUBLE, HDFQL_VARCHAR, HDFQL_OPAQUE, HDFQL_BITFIELD, HDFQL_ENUMERATION, HDFQL_COMPOUND or HDFQL_UNDEFINED (please refer to Table 6.3 for additional information about data types).

Example(s)

```bash
# create an HDF5 dataset named "my_dataset0" of data type double
CREATE DATASET my_dataset0 AS DOUBLE

# show (i.e. get) data type of dataset "my_dataset0" (should be 512 – i.e. HDFQL_DOUBLE)
SHOW DATA TYPE my_dataset0

# create an HDF5 dataset named "my_dataset1" of data type float
CREATE DATASET my_dataset1 AS FLOAT

# show (i.e. get) data type of dataset "my_dataset1" (should be 256 – i.e. HDFQL_FLOAT)
SHOW DATA TYPE my_dataset1

# create an HDF5 dataset named "my_common" of data type short
CREATE DATASET my_common AS SMALLINT

# create an HDF5 attribute named "my_common" of data type int
CREATE ATTRIBUTE my_common AS INT

# show (i.e. get) data type of dataset "my_common" (should be 4 – i.e. HDFQL_SMALLINT)
SHOW DATA TYPE my_common

# show (i.e. get) data type of dataset "my_common" (should be 4 – i.e. HDFQL_SMALLINT)
SHOW DATASET DATA TYPE my_common

# show (i.e. get) data type of attribute "my_common" (should be 16 – i.e. HDFQL_INT)
SHOW ATTRIBUTE DATA TYPE my_common
```
6.7.9  SHOW MASK

Syntax

SHOW MASK [file_name] dataset_name[(chunk_number [, chunk_number]*)] [, [file_name]
  dataset_name[(chunk_number [, chunk_number]*)]]*

  [post_processing_option [post_processing_option]*]

  [output_redirecting_option]

Description

Show (i.e. get) (filter) mask of an HDF5 dataset named dataset_name. Multiple datasets’ masks can be obtained at once by separating these with a comma (,). If dataset_name was not found or its mask could not be checked (due to unknown/unexpected reasons), no subsequent datasets are checked, and an error is raised.

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file which stores the dataset to show (i.e. get) the (filter) mask. If file_name is specified, the file is opened on the fly, the mask of the dataset is obtained and, afterwards, the file is closed. Otherwise, if it is not specified, the dataset (whose mask is to be obtained) is stored in the file currently in use.

dataset_name – mandatory string that specifies the name of the HDF5 dataset whose (filter) mask is to be obtained. Multiple datasets are separated with a comma (,).

chunk_number – optional integer that specifies the number of the chunk to show (i.e. get) its (filter) mask. Multiple chunk numbers are separated with a comma (,). If chunk_number is specified it must either be between 0 and the storage dimension in question - 1 (otherwise an error will be raised) or negative (in this case its value will be the last position of the storage dimension in question minus the value of chunk_number). Otherwise, if it is not specified, its default value is 0 (i.e. first chunk of the storage dimension in question).

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).
Return

The (filter) mask of an HDF5 dataset as an HDFQL_UNSIGNED_INT.

Example(s)

```hdfql
# create a chunked (size 2) HDF5 dataset named "my_dataset" of data type int of one dimension (size 6)
CREATE CHUNKED(2) DATASET my_dataset AS INT(6)

# insert (i.e. write) values 60 and 61 directly into chunk #0 of dataset "my_dataset" using a (filter) mask equal to 8
INSERT DIRECTLY MASK 8 INTO my_dataset VALUES(60, 61)

# insert (i.e. write) values 62 and 63 directly into chunk #1 of dataset "my_dataset" using a (filter) mask equal to 255 (i.e. 0xFF)
INSERT DIRECTLY MASK 0xFF INTO my_dataset(1) VALUES(62, 63)

# insert (i.e. write) values 64 and 65 directly into chunk #2 of dataset "my_dataset" using a (filter) mask equal to 0 (i.e. default value)
INSERT DIRECTLY INTO my_dataset(2) VALUES(64, 65)

# select (i.e. read) data from dataset "my_dataset" and populate cursor in use with it (should be 60, 61, 62, 63, 64, 65)
SELECT FROM my_dataset

# show (i.e. get) (filter) mask of chunks #0, #1 and #2 of dataset "my_dataset" and populate cursor in use with it (should be 8, 255, 0)
SHOW MASK my_dataset(0), my_dataset(1), my_dataset(2)
```

6.7.10 SHOW ENDIANNESS

Syntax

```
SHOW [DATASET | ATTRIBUTE] ENDIANNESS [file_name] object_name [, [file_name] object_name]*

[past_processing_option [post_processing_option]*]

[output_redirecting_option]
```
Description

Show (i.e. get) endianness of an HDF5 dataset or attribute named object_name. Multiple objects’ endiannesses can be obtained at once by separating these with a comma (,). If object_name was not found or its endianness could not be checked (due to unknown/unexpected reasons), no subsequent objects are checked, and an error is raised. In case a dataset and an attribute with identical names (object_name) are stored in the same location (i.e. group) and neither the keyword DATASET nor ATTRIBUTE is specified, the endianness returned belongs to the dataset. To explicitly get the endianness of object_name according to its type, the keyword DATASET or ATTRIBUTE must be specified.

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file which stores the dataset or attribute to show (i.e. get) the endianness. If file_name is specified, the file is opened on the fly, the endianness of the dataset or attribute is obtained and, afterwards, the file is closed. Otherwise, if it is not specified, the dataset or attribute (whose endianness is to be obtained) is stored in the file currently in use.

object_name – mandatory string that specifies the name of the HDF5 dataset or attribute whose endianness is to be obtained. Multiple datasets or attributes are separated with a comma (,).

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The endianness of an HDF5 dataset or attribute as an HDFQL_INT, which can either be HDFQL_LITTLE_ENDIAN, HDFQL_BIG_ENDIAN or HDFQL_UNDEFINED depending on whether the endianness is little, big or undefined (i.e. endianness is not applicable to object_name) respectively.

Example(s)

```
# create an HDF5 dataset named "my_dataset0" of data type int using the native endian representation (of the machine)
CREATE DATASET my_dataset0 AS INT
```
# show (i.e. get) endianness of dataset "my_dataset0" (should be 1 or 2 - i.e. HDFQL_LITTLE_ENDIAN or HDFQL_BIG_ENDIAN - depending on whether the dataset was created in a little or big endian machine respectively)

```
SHOW ENDIANNESS my_dataset0
```

# create an HDF5 dataset named "my_dataset1" of data type long long using the little endian representation

```
CREATE DATASET my_dataset1 AS LITTLE ENDIAN BIGINT
```

# show (i.e. get) endianness of dataset "my_dataset1" (should be 1 - i.e. HDFQL_LITTLE_ENDIAN)

```
SHOW ENDIANNESS my_dataset1
```

# create an HDF5 dataset named "my_common" of data type short using the big endian representation

```
CREATE DATASET my_common AS BIG ENDIAN SMALLINT
```

# create an HDF5 attribute named "my_common" of data type int using the little endian representation

```
CREATE ATTRIBUTE my_common AS LITTLE ENDIAN INT
```

# show (i.e. get) endianness of dataset "my_common" (should be 2 - i.e. HDFQL_BIG_ENDIAN)

```
SHOW ENDIANNESS my_common
```

# show (i.e. get) endianness of dataset "my_common" (should be 2 - i.e. HDFQL_BIG_ENDIAN)

```
SHOW DATASET ENDIANNESS my_common
```

# show (i.e. get) endianness of attribute "my_common" (should be 1 - i.e. HDFQL_LITTLE_ENDIAN)

```
SHOW ATTRIBUTE ENDIANNESS my_common
```

## 6.7.11 SHOW CHARSET

### Syntax

```
SHOW [DATASET | ATTRIBUTE] CHARSET [file_name] object_name [, [file_name] object_name]*
```

```
[post_processing_option [post_processing_option]*]
```

```
[output_redirecting_option]
```
Description

Show (i.e. get) charset of an HDF5 dataset or attribute named object_name. Multiple objects' charsets can be obtained at once by separating these with a comma (,). If object_name was not found or its charset could not be checked (due to unknown/unexpected reasons), no subsequent objects are checked, and an error is raised. In case a dataset and an attribute with identical names (object_name) are stored in the same location (i.e. group) and neither the keyword DATASET nor ATTRIBUTE is specified, the charset returned belongs to the dataset. To explicitly get the charset of object_name according to its type, the keyword DATASET or ATTRIBUTE must be specified.

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file which stores the dataset or attribute to show (i.e. get) the charset. If file_name is specified, the file is opened on the fly, the charset of the dataset or attribute is obtained and, afterwards, the file is closed. Otherwise, if it is not specified, the dataset or attribute (whose charset is to be obtained) is stored in the file currently in use.

object_name – mandatory string that specifies the name of the HDF5 dataset or attribute whose charset is to be obtained. Multiple datasets or attributes are separated with a comma (,).

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The charset of an HDF5 dataset or attribute as an HDFQL_INT, which can either be HDFQL_ASCII, HDFQL_UTF8 or HDFQL_UNDEFINED depending on whether the charset is ASCII, UTF8 or undefined (i.e. object_name is neither of data type HDFQL_CHAR nor HDFQL_VARCHAR) respectively.

Example(s)

```bash
# create an HDF5 dataset named "my_dataset0" of data type char
CREATE DATASET my_dataset0 AS CHAR

# show (i.e. get) charset of dataset "my_dataset0" (should be 1 - i.e. HDFQL_ASCII)
```
**SHOW CHARSET** my_dataset0

# create an HDF5 dataset named "my_dataset1" of data type char of one dimension (size 20)
**CREATE DATASET** my_dataset1 AS UTF8 CHAR(20)

# show (i.e. get) charset of dataset "my_dataset1" (should be 2 - i.e. HDFQL_UTF8)
**SHOW CHARSET** my_dataset1

# create an HDF5 dataset named "my_common" of data type short
**CREATE DATASET** my_common AS UTF8 CHAR

# create an HDF5 attribute named "my_common" of data type variable-length char
**CREATE ATTRIBUTE** my_common AS ASCII VARCHAR

# show (i.e. get) charset of dataset "my_common" (should be 2 - i.e. HDFQL_UTF8)
**SHOW CHARSET** my_common

# show (i.e. get) data type of dataset "my_common" (should be 2 - i.e. HDFQL_UTF8)
**SHOW DATASET CHARSET** my_common

# show (i.e. get) charset of attribute "my_common" (should be 1 - i.e. HDFQL_ASCII)
**SHOW ATTRIBUTE CHARSET** my_common

### 6.7.12 SHOW STORAGE TYPE

**Syntax**

**SHOW STORAGE TYPE** [file_name] dataset_name [, [file_name] dataset_name]*

[post_processing_option [post_processing_option]*]

[output_redirecting_option]

**Description**

Show (i.e. get) storage type (layout) of an HDF5 dataset named *dataset_name*. Multiple datasets’ storage types can be obtained at once by separating these with a comma (,). If *dataset_name* was not found or its storage type could not be checked (due to unknown/unexpected reasons), no subsequent datasets are checked, and an error is raised.
**Parameter(s)**

`file_name` – optional string that specifies the name of the HDF5 file which stores the dataset to show (i.e. get) the storage type (layout). If `file_name` is specified, the file is opened on the fly, the storage type of the dataset is obtained and, afterwards, the file is closed. Otherwise, if it is not specified, the dataset (whose storage type is to be obtained) is stored in the file currently in use.

`dataset_name` – mandatory string that specifies the name of the HDF5 dataset whose storage type (layout) is to be obtained. Multiple datasets are separated with a comma (,).

`post_processing_option` – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

`output_redirecting_option` – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

**Return**

The storage type (layout) of an HDF5 dataset as an **HDFQL_INT**, which can either be **HDFQL_CONTIGUOUS**, **HDFQL_COMPACT** or **HDFQL_CHUNKED** depending on whether the storage is contiguous, compact or chunked respectively.

**Example(s)**

```sql
# create an HDF5 dataset named "my_dataset0" of data type unsigned int
CREATE DATASET my_dataset0 AS UNSIGNED INT

# show (i.e. get) storage type (layout) of dataset "my_dataset0" (should be 1 – i.e. HDFQL_CONTIGUOUS)
SHOW STORAGE TYPE my_dataset0

# create an HDF5 dataset named "my_dataset1" of data type int of two dimensions (size 5x7)
CREATE CONTIGUOUS DATASET my_dataset1 AS INT(5, 7)

# show (i.e. get) storage type (layout) of dataset "my_dataset1" (should be 1 – i.e. HDFQL_CONTIGUOUS)
SHOW STORAGE TYPE my_dataset1
```

# create an HDF5 dataset named "my_dataset2" of data type double of one dimension (size 8)
CREATE COMPACT DATASET my_dataset2 AS DOUBLE(8)

# show (i.e. get) storage type (layout) of dataset "my_dataset2" (should be 2 - i.e.
HDFQL_COMPACT)
SHOW STORAGE TYPE my_dataset2

# create an HDF5 dataset named "my_dataset3" of data type float of three dimensions (size
3x5x20)
CREATE CHUNKED DATASET my_dataset3 AS FLOAT(3, 5, 20)

# show (i.e. get) storage type (layout) of dataset "my_dataset3" (should be 4 - i.e.
HDFQL_CHUNKED)
SHOW STORAGE TYPE my_dataset3

6.7.13 SHOW STORAGE ALLOCATION

Syntax

SHOW STORAGE ALLOCATION [file_name] dataset_name [, [file_name] dataset_name]*

[post_processing_option [post_processing_option]*]

[output_redirecting_option]

Description

Show (i.e. get) storage allocation of an HDF5 dataset named dataset_name. Multiple datasets’ storage allocation can be obtained at once by separating these with a comma (,). If dataset_name was not found or its storage allocation could not be checked (due to unknown/unexpected reasons), no subsequent datasets are checked, and an error is raised.

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file which stores the dataset to show (i.e. get) the storage allocation. If file_name is specified, the file is opened on the fly, the storage allocation of the dataset is obtained and, afterwards, the file is closed. Otherwise, if it is not specified, the dataset (whose storage allocation is to be obtained) is stored in the file currently in use.

dataset_name – mandatory string that specifies the name of the HDF5 dataset whose storage allocation is to be obtained. Multiple datasets are separated with a comma (,).
post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The storage allocation of an HDF5 dataset as an HDFQL_INT, which can either be HDFQL_EARLY, HDFQL_INCREMENTAL or HDFQL_LATE depending on whether the storage allocation is early, incremental or late respectively.

Example(s)

```plaintext
# create an HDF5 dataset named "my_dataset0" of data type unsigned int
CREATE DATASET my_dataset0 AS UNSIGNED INT

# show (i.e. get) storage allocation of dataset "my_dataset0" (should be 4 – i.e. HDFQL_LATE)
SHOW STORAGE ALLOCATION my_dataset0

# create an HDF5 dataset named "my_dataset1" of data type int of two dimensions (size 5x7)
CREATE CONTIGUOUS DATASET my_dataset1 AS INT(5, 7)

# show (i.e. get) storage allocation of dataset "my_dataset1" (should be 4 – i.e. HDFQL_LATE)
SHOW STORAGE ALLOCATION my_dataset1

# create an HDF5 dataset named "my_dataset2" of data type double of one dimension (size 8)
CREATE COMPACT DATASET my_dataset2 AS DOUBLE(8)

# show (i.e. get) storage allocation of dataset "my_dataset2" (should be 1 – i.e. HDFQL_EARLY)
SHOW STORAGE ALLOCATION my_dataset2

# create an HDF5 dataset named "my_dataset3" of data type float of three dimensions (size 3x5x20)
CREATE CHUNKED DATASET my_dataset3 AS FLOAT(3, 5, 20)

# show (i.e. get) storage allocation of dataset "my_dataset3" (should be 2 – i.e. HDFQL_INCREMENTAL)
SHOW STORAGE ALLOCATION my_dataset3
```
6.7.14 SHOW STORAGE DIMENSION

Syntax

SHOW STORAGE DIMENSION [file_name] dataset_name

[post_processing_option [post_processing_option] *]

[output_redirecting_option]

Description

Show (i.e. get) storage dimensions of an HDF5 dataset named dataset_name.

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file which stores the dataset to show (i.e. get) the storage dimensions. If file_name is specified, the file is opened on the fly, the storage dimensions of the dataset is obtained and, afterwards, the file is closed. Otherwise, if it is not specified, the dataset (whose storage dimensions is to be obtained) is stored in the file currently in use.

dataset_name – mandatory string that specifies the name of the HDF5 dataset whose storage dimensions are to be obtained.

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The storage dimensions of an HDF5 dataset as an HDFQL_BIGINT or nothing (in case the dataset is not chunked – i.e. its storage type is not HDFQL_CHUNKED).
Example(s)

```hdfql
# create an HDF5 dataset named "my_dataset0" of data type unsigned int
CREATE DATASET my_dataset0 AS UNSIGNED INT

# show (i.e. get) storage dimensions of dataset "my_dataset0" (should be empty)
SHOW STORAGE DIMENSION my_dataset0

# create an HDF5 dataset named "my_dataset1" of data type int of two dimensions (size 5x7)
CREATE DATASET my_dataset1 AS INT(5, 7)

# show (i.e. get) storage dimensions of dataset "my_dataset1" (should be empty)
SHOW STORAGE DIMENSION my_dataset1

# create an HDF5 dataset named "my_dataset2" of data type double of one dimension (size 8)
CREATE CHUNKED DATASET my_dataset2 AS DOUBLE(8)

# show (i.e. get) storage dimensions of dataset "my_dataset2" (should be 8)
SHOW STORAGE DIMENSION my_dataset2

# create an HDF5 dataset named "my_dataset3" of data type float of three dimensions (size 3x5x20)
CREATE CHUNKED(1, 2, 10) DATASET my_dataset3 AS FLOAT(3, 5, 20)

# show (i.e. get) storage dimensions of dataset "my_dataset3" (should be 1, 2, 10)
SHOW STORAGE DIMENSION my_dataset3
```

6.7.15 SHOW DIMENSION

Syntax

```
SHOW [DATASET | ATTRIBUTE] [MAX] DIMENSION [file_name] object_name

[post_processing_option [post_processing_option] *]

[output redirecting_option]
```

Description

Show (i.e. get) dimensions of an HDF5 dataset or attribute named `object_name`. In case a dataset and an attribute with identical names (`object_name`) are stored in the same location (i.e. group) and neither the keyword `DATASET` nor
ATTRIBUTE is specified, the dimensions returned belong to the dataset. To explicitly get the dimensions of object_name according to its type, the keyword DATASET or ATTRIBUTE must be specified. If object_name does not have a dimension (i.e. if it is scalar), the returned value is one. By default, the returned dimensions refer to the ones that a dataset or an attribute currently has; to return the maximum dimensions that a dataset or an attribute may grow to, the keyword MAX must be specified. If the maximum dimension is unlimited, the returned value is HDFQL_UNLIMITED.

**Parameter(s)**

file_name – optional string that specifies the name of the HDF5 file which stores the dataset or attribute to show (i.e. get) the dimensions. If file_name is specified, the file is opened on the fly, the dimensions of the dataset or attribute is obtained and, afterwards, the file is closed. Otherwise, if it is not specified, the dataset or attribute (whose dimensions is to be obtained) is stored in the file currently in use.

object_name – mandatory string that specifies the name of the HDF5 dataset or attribute whose dimensions are to be obtained.

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

**Return**

The dimensions of an HDF5 dataset or attribute as an HDFQL_BIGINT.

**Example(s)**

```
# create an HDF5 dataset named "my_dataset0" of data type unsigned int
CREATE DATASET my_dataset0 AS UNSIGNED INT

# show (i.e. get) dimensions of dataset "my_dataset0" (should be 1)
SHOW DIMENSION my_dataset0

# show (i.e. get) maximum dimensions of dataset "my_dataset0" (should be 1)
SHOW MAX DIMENSION my_dataset0

# create an HDF5 dataset named "my_dataset1" of data type double of one dimension (size 15)
```
```
CREATE DATASET my_dataset1 AS DOUBLE(15)

# show (i.e. get) dimensions of dataset "my_dataset1" (should be 15)
SHOW DIMENSION my_dataset1

# show (i.e. get) maximum dimensions of dataset "my_dataset1" (should be 15)
SHOW MAX DIMENSION my_dataset1

# create an HDF5 attribute named "my_attribute0" of data type int of one dimension (size 1)
CREATE ATTRIBUTE my_attribute0 AS INT(1)

# show (i.e. get) dimensions of attribute "my_attribute0" (should be 1)
SHOW DIMENSION my_attribute0

# show (i.e. get) maximum dimensions of attribute "my_attribute0" (should be 1)
SHOW MAX DIMENSION my_attribute0

# create an HDF5 attribute named "my_attribute1" of data type short of two dimensions (size 2x3)
CREATE ATTRIBUTE my_attribute1 AS SMALLINT(2, 3)

# show (i.e. get) dimensions of attribute "my_attribute1" (should be 2, 3)
SHOW DIMENSION my_attribute1

# show (i.e. get) maximum dimensions of attribute "my_attribute1" (should be 2, 3)
SHOW MAX DIMENSION my_attribute1

# create an HDF5 dataset named "my_dataset2" of data type float of three dimensions (first dimension with size 2 and extendible up to 10; second dimension with size 5; third dimension with size 20 and extendible to an unlimited size)
CREATE CHUNKED DATASET my_dataset2 AS FLOAT(3 TO 10, 5, 20 TO UNLIMITED, UNLIMITED)

# show (i.e. get) dimensions of dataset "my_dataset2" (should be 3, 20, 1)
SHOW DIMENSION my_dataset2

# show (i.e. get) maximum dimensions of dataset "my_dataset2" (should be 10, 5, -1, -1)
SHOW MAX DIMENSION my_dataset2
```
6.7.16 SHOW ORDER

Syntax

SHOW [ATTRIBUTE] ORDER [file_name] object_name [, [file_name] object_name]*

[post_processing_option [post_processing_option]***]

[output_redirecting_option]

Description

Show (i.e. get) (creation) order strategy of an HDF5 group or dataset named object_name. Multiple objects’ order strategies can be obtained at once by separating these with a comma (,). If object_name was not found or its order strategy could not be checked (due to unknown/unexpected reasons), no subsequent objects are checked, and an error is raised. By default, the returned order strategy refers to objects (i.e. groups, datasets, (soft) links or external links) within a group; to return the order strategy of attributes within a group or dataset, the keyword ATTRIBUTE must be specified.

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file which stores the group or dataset to show (i.e. get) the (creation) order strategy. If file_name is specified, the file is opened on the fly, the order strategy of the group or dataset is obtained and, afterwards, the file is closed. Otherwise, if it is not specified, the group or dataset (whose order strategy is to be obtained) is stored in the file currently in use.

object_name – mandatory string that specifies the name of the HDF5 group or dataset whose (creation) order strategy is to be obtained. Multiple groups or datasets are separated with a comma (,).

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).
**Return**

The (creation) order strategy of an HDF5 group or dataset as an HDFQL_INT, which can either be HDFQL_TRACKED, HDFQL_INDEXED or HDFQL_UNDEFINED depending on whether the order is tracked, indexed or undefined (i.e. object_name was created without any order strategy) respectively.

**Example(s)**

```hl7
# create an HDF5 group named "my_group0"
CREATE GROUP my_group0

# show (i.e. get) (creation) order strategy of objects within group "my_group0" (should be -1 - i.e. HDFQL_UNDEFINED)
SHOW ORDER my_group0

# show (i.e. get) (creation) order strategy of attributes within group "my_group0" (should be -1 - i.e. HDFQL_UNDEFINED)
SHOW ATTRIBUTE ORDER my_group0

# create an HDF5 group named "my_group1" that tracks both the objects’ (i.e. groups and datasets) and the attributes’ creation order within the group
CREATE GROUP my_group1 ORDER TRACKED ATTRIBUTE ORDER INDEXED

# show (i.e. get) (creation) order strategy of objects within group "my_group1" (should be 1 - i.e. HDFQL_TRACKED)
SHOW ORDER my_group1

# show (i.e. get) (creation) order strategy of attributes within group "my_group1" (should be 2 - i.e. HDFQL_INDEXED)
SHOW ATTRIBUTE ORDER my_group1

# create an HDF5 dataset named "my_dataset0" of data type int that tracks the attributes’ creation order within the dataset
CREATE DATASET my_dataset0 AS INT ATTRIBUTE ORDER TRACKED

# show (i.e. get) (creation) order strategy of attributes within dataset "my_dataset0" (should be 1 - i.e. HDFQL_TRACKED)
SHOW ATTRIBUTE ORDER my_dataset0

# show (i.e. get) (creation) order strategy of attributes within both group "my_group1" and dataset "my_dataset0" at once (should be 2, 1)
SHOW ATTRIBUTE ORDER my_group1, my_dataset0
```
### 6.7.17 SHOW TAG

#### Syntax

```plaintext
SHOW [DATASET | ATTRIBUTE] TAG [file_name] object_name [, [file_name] object_name]*

[post_processing_option [post_processing_option] *]

[output_redirecting_option]
```

#### Description

Show (i.e. get) tag of an HDF5 dataset or attribute named `object_name`. Multiple objects’ tags can be obtained at once by separating these with a comma (,). If `object_name` was not found or its tag could not be checked (due to its data type not being HDFQL_OPAQUE or for unknown/unexpected reasons), no subsequent objects are checked, and an error is raised. In case a dataset and an attribute with identical names (`object_name`) are stored in the same location (i.e. group) and neither the keyword DATASET nor ATTRIBUTE is specified, the tag returned belongs to the dataset. To explicitly get the tag of `object_name` according to its type, the keyword DATASET or ATTRIBUTE must be specified.

#### Parameter(s)

- `file_name` – optional string that specifies the name of the HDF5 file which stores the dataset or attribute to show (i.e. get) the tag. If `file_name` is specified, the file is opened on the fly, the tag of the dataset or attribute is obtained and, afterwards, the file is closed. Otherwise, if it is not specified, the dataset or attribute (whose tag is to be obtained) is stored in the file currently in use.

- `object_name` – mandatory string that specifies the name of the HDF5 dataset or attribute whose tag is to be obtained. Multiple datasets or attributes are separated with a comma (,).

- `post_processing_option` – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

- `output_redirecting_option` – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

#### Return

The tag of an HDF5 dataset or attribute as an HDFQL_VARCHAR.
Example(s)

```hdf
# create an HDF5 dataset named "my_dataset0" of data type opaque
CREATE DATASET my_dataset0 AS OPAQUE

# show (i.e. get) tag of dataset "my_dataset0" (should be empty)
SHOW TAG my_dataset0

# create an HDF5 dataset named "my_dataset1" of data type opaque of one dimension (size 15)
# with a tag value "my_tag1"
CREATE DATASET my_dataset1 AS OPAQUE(15) TAG my_tag1

# show (i.e. get) tag of dataset "my_dataset1" (should be "my_tag1")
SHOW TAG my_dataset1

# create an HDF5 attribute named "my_attribute0" of data type opaque of two dimensions (size 3x5)
# with a tag value "Hierarchical Data Format"
CREATE ATTRIBUTE my_attribute0 AS OPAQUE(3, 5) TAG "Hierarchical Data Format"

# show (i.e. get) tag of attribute "my_attribute0" (should be "Hierarchical Data Format")
SHOW TAG my_attribute0

# create an HDF5 dataset named "my_common" of data type opaque with a tag value "Dataset tag"
CREATE DATASET my_common AS OPAQUE TAG "Dataset tag"

# create an HDF5 attribute named "my_common" of data type opaque of one dimension (size 10)
# with a tag value "Attribute tag"
CREATE ATTRIBUTE my_common AS OPAQUE(10) TAG "Attribute tag"

# show (i.e. get) tag of dataset "my_common" (should be "Dataset tag")
SHOW TAG my_common

# show (i.e. get) tag of dataset "my_common" (should be "Dataset tag")
SHOW DATASET TAG my_common

# show (i.e. get) tag of attribute "my_common" (should be "Attribute tag")
SHOW ATTRIBUTE TAG my_common
```
6.7.18 SHOW FILL TYPE

Syntax

SHOW FILL TYPE [file_name] dataset_name [, [file_name] dataset_name]*

[post_processing_option [post_processing_option]*]

[output_redirecting_option]

Description

Show (i.e. get) fill type of an HDF5 dataset named dataset_name. Multiple datasets’ fill types can be obtained at once by separating these with a comma (,). If dataset_name was not found or its fill type could not be checked (due to unknown/unexpected reasons), no subsequent datasets are checked, and an error is raised.

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file which stores the dataset to show (i.e. get) the fill type. If file_name is specified, the file is opened on the fly, the fill type of the dataset is obtained and, afterwards, the file is closed. Otherwise, if it is not specified, the dataset (whose fill type is to be obtained) is stored in the file currently in use.

dataset_name – mandatory string that specifies the name of the HDF5 dataset whose fill type is to be obtained. Multiple datasets are separated with a comma (,).

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The fill type of an HDF5 dataset as an HDFQL_INT, which can either be HDFQL_FILL_DEFAULT, HDFQL_FILL_USER_DEFINED or HDFQL_FILL_UNDEFINED depending on whether the fill is default, user defined or undefined respectively.
Example(s)

```plaintext
# create an HDF5 dataset named "my_dataset0" of data type int
CREATE DATASET my_dataset0 AS INT

# show (i.e. get) fill type of dataset "my_dataset0" (should be 1 - i.e. HDFQL_FILL_DEFAULT)
SHOW FILL TYPE my_dataset0

# create an HDF5 dataset named "my_dataset1" of data type int with a fill value "20"
CREATE DATASET my_dataset1 AS INT FILL(20)

# show (i.e. get) fill type of dataset "my_dataset1" (should be 2 - i.e. HDFQL_FILL_USER_DEFINED)
SHOW FILL TYPE my_dataset1

# create an HDF5 dataset named "my_dataset2" of data type variable-length char of one dimension (size 5) with an undefined fill value
CREATE DATASET my_dataset2 AS VARCHAR(5) FILL UNDEFINED

# show (i.e. get) fill type of dataset "my_dataset2" (should be 4 - i.e. HDFQL_FILL_UNDEFINED)
SHOW FILL TYPE my_dataset2
```

6.7.19 SHOW FILL VALUE

Syntax

SHOW FILL VALUE [file_name] dataset_name

   [post_processing_option [post_processing_option]*]

   [output_redirecting_option]

Description

Show (i.e. get) fill values of an HDF5 dataset named dataset_name.

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file which stores the dataset to show (i.e. get) the fill values. If file_name is specified, the file is opened on the fly, the fill values of the dataset are obtained and, afterwards, the
file is closed. Otherwise, if it is not specified, the dataset (whose fill values is to be obtained) is stored in the file currently in use.

dataset_name – mandatory string that specifies the name of the HDF5 dataset whose fill values are to be obtained.

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The fill values of an HDF5 dataset as an HDFQL_TINYINT (in case the data type of the dataset is HDFQL_TINYINT), HDFQL_UNSIGNED_TINYINT (in case the data type of the dataset is HDFQL_UNSIGNED_TINYINT), HDFQL_SMALLINT (in case the data type of the dataset is HDFQL_SMALLINT), HDFQL_UNSIGNED_SMALLINT (in case the data type of the dataset is HDFQL_UNSIGNED_SMALLINT), HDFQL_INT (in case the data type of the dataset is HDFQL_INT), HDFQL_UNSIGNED_INT (in case the data type of the dataset is HDFQL_UNSIGNED_INT), HDFQL_BIGINT (in case the data type of the dataset is HDFQL_BIGINT), HDFQL_UNSIGNED_BIGINT (in case the data type of the dataset is HDFQL_UNSIGNED_BIGINT), HDFQL_FLOAT (in case the data type of the dataset is HDFQL_FLOAT), HDFQL_DOUBLE (in case the data type of the dataset is HDFQL_DOUBLE), HDFQL_CHAR (in case the data type of the dataset is HDFQL_CHAR), HDFQL_VARTINYINT (in case the data type of the dataset is HDFQL_VARTINYINT), HDFQL_UNSIGNED_VARTINYINT (in case the data type of the dataset is HDFQL_UNSIGNED_VARTINYINT), HDFQL_VARSMALLINT (in case the data type of the dataset is HDFQL_VARSMALLINT), HDFQL_UNSIGNED_VARSMALLINT (in case the data type of the dataset is HDFQL_UNSIGNED_VARSMALLINT), HDFQL_VARINT (in case the data type of the dataset is HDFQL_VARINT), HDFQL_UNSIGNED_VARINT (in case the data type of the dataset is HDFQL_UNSIGNED_VARINT), HDFQL_VARBIGINT (in case the data type of the dataset is HDFQL_VARBIGINT), HDFQL_UNSIGNED_VARBIGINT (in case the data type of the dataset is HDFQL_UNSIGNED_VARBIGINT), HDFQL_VARFLOAT (in case the data type of the dataset is HDFQL_VARFLOAT), HDFQL_VARDOUBLE (in case the data type of the dataset is HDFQL_VARDOUBLE), HDFQL_VARCHAR (in case the data type of the dataset is HDFQL_VARCHAR) or HDFQL_OPAQUE (in case the data type of the dataset is HDFQL_OPAQUE).

Example(s)

```sql
# create an HDF5 dataset named "my_dataset0" of data type int
CREATE DATASET my_dataset0 AS INT
```
# show (i.e. get) fill values of dataset "my_dataset0" (should be 0)
SHOW FILL VALUE my_dataset0

# create an HDF5 dataset named "my_dataset1" of data type int with a fill value "20"
CREATE DATASET my_dataset1 AS INT FILL(20)

# show (i.e. get) fill values of dataset "my_dataset1" (should be 20)
SHOW FILL VALUE my_dataset1

# create an HDF5 dataset named "my_dataset2" of data type variable-length char of one dimension
(size 5) with a fill value "Hierarchical Data Format"
CREATE DATASET my_dataset2 AS VARCHAR(5) FILL("Hierarchical Data Format")

# show (i.e. get) fill values of dataset "my_dataset2" (should be "Hierarchical Data Format")
SHOW FILL VALUE my_dataset2

# create an HDF5 dataset named "my_dataset3" of data type variable-length int with fill values
"100" and "200"
CREATE DATASET my_dataset3 AS VARINT FILL(100, 200)

# show (i.e. get) fill values of dataset "my_dataset3" (should be 100, 200)
SHOW FILL VALUE my_dataset3

6.7.20 SHOW FILE SIZE

Syntax

SHOW FILE SIZE [file_name [, file_name]*]

[post_processing_option [post_processing_option]*]

[outputRedirecting_option]

Description

Show (i.e. get) size (in bytes) of a file named file_name. Multiple files’ sizes can be obtained at once by separating several file names with a comma (,). If file_name is not specified, the size of the (HDF5) file currently in use is returned. Otherwise, if file_name is specified, its size is returned. If file_name was not found or its size could not be checked (due to unknown/unexpected reasons), no subsequent files are checked, and an error is raised.
Parameter(s)

`file_name` – optional string that specifies the name of the file whose size (in bytes) is to be obtained. Multiple files are separated with a comma (,).

`post_processing_option` – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

`output_redirecting_option` – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The size (in bytes) of a file as an HDFQL_UNSIGNED_BIGINT.

Example(s)

```plaintext
# create an HDF5 file named "my_file.h5"
CREATE FILE my_file.h5

# show (i.e. get) size of file "my_file.h5" (should be 800)
SHOW FILE SIZE my_file.h5

# use (i.e. open) HDF5 file "my_file.h5"
USE FILE my_file.h5

# create an HDF5 group named "my_group"
CREATE GROUP my_group

# flush the entire virtual HDF5 file (global) currently in use
FLUSH

# show (i.e. get) size of the file currently in use (should be greater than 800)
SHOW FILE SIZE
```
6.7.21 SHOW [DATASET | ATTRIBUTE] SIZE

Syntax

SHOW [DATASET | ATTRIBUTE] SIZE [file_name] object_name [, [file_name] object_name]*

[post_processing_option [post_processing_option]*]

[output Redirecting_option]

Description

Show (i.e. get) size (in bytes) of an HDF5 dataset or attribute named object_name. Multiple objects’ sizes can be obtained at once by separating these with a comma (,). If object_name was not found or its size could not be checked (due to unknown/unexpected reasons), no subsequent objects are checked, and an error is raised. In case a dataset and an attribute with identical names (object_name) are stored in the same location (i.e. group) and neither the keyword DATASET nor ATTRIBUTE is specified, the size returned belongs to the dataset. To explicitly get the size of object_name according to its type, the keyword DATASET or ATTRIBUTE must be specified.

Parameter(s)

file_name – optional string that specifies the name of the HDF5 file which stores the dataset or attribute to show (i.e. get) the size (in bytes). If file_name is specified, the file is opened on the fly, the size of the dataset or attribute is obtained and, afterwards, the file is closed. Otherwise, if it is not specified, the dataset or attribute (whose size is to be obtained) is stored in the file currently in use.

object_name – mandatory string that specifies the name of the HDF5 dataset or attribute whose size (in bytes) is to be obtained. Multiple datasets or attributes are separated with a comma (,).

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output Redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The size (in bytes) of an HDF5 dataset or attribute as an HDFQL_UNSIGNED_BIGINT.
Example(s)

```hdfql
# create an HDF5 dataset named "my_dataset0" of data type float
CREATE DATASET my_dataset0 AS FLOAT

# show (i.e. get) size (in bytes) of dataset "my_dataset0" (should be 4)
SHOW SIZE my_dataset0

# create an HDF5 dataset named "my_dataset1" of data type long long of one dimension (size 3)
CREATE DATASET my_dataset1 AS BIGINT(3)

# show (i.e. get) size (in bytes) of dataset "my_dataset1" (should be 24 - i.e. 8x3)
SHOW SIZE my_dataset1

# create an HDF5 dataset named "my_common" of data type variable-length short with initial values of 10, 20, 30, 40, 50 and 60
CREATE DATASET my_common AS VARSMLINT VALUES(10, 20, 30, 40, 50, 60)

# create an HDF5 attribute named "my_common" of data type double of two dimensions (size 2x3)
CREATE ATTRIBUTE my_common AS DOUBLE(2, 3)

# show (i.e. get) size (in bytes) of dataset "my_common" (should be 12 - i.e. 2x6)
SHOW SIZE my_common

# show (i.e. get) size (in bytes) of dataset "my_common" (should be 12 - i.e. 2x6)
SHOW DATASET SIZE my_common

# show (i.e. get) size (in bytes) of attribute "my_common" (should be 48 - i.e. 8x2x3)
SHOW ATTRIBUTE SIZE my_common
```

6.7.22 SHOW HDFQL VERSION

Syntax

SHOW HDFQL VERSION

    [post_processing_option [post_processing_option]*]

    [output_redirecting_option]
Description

Show (i.e. get) version of HDFql library. The format of the version returned is MAJOR.MINOR.REVISION.

Parameter(s)

$post_processing_option$ – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

$output_redirecting_option$ – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The version of HDFql library as an HDFQL_VARCHAR.

Example(s)

```
# show (i.e. get) version of HDFql library (should be something similar to "2.1.0")
SHOW HDFQL VERSION
```

6.7.23 SHOW HDF5 VERSION

Syntax

SHOW HDF5 VERSION

[post_processing_option [post_processing_option]*]

[output_redirecting_option]

Description

Show (i.e. get) version of the HDF5 library used by HDFql. The format of the version returned is MAJOR.MINOR.REVISION. The HDF5 library refers to the library used to compile HDFql, and which is shipped with HDFql (and not the HDF5 library that may be installed in the machine).
**Parameter(s)**

`post_processing_option` – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

`output_redirecting_option` – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

**Return**

The version of the HDF5 library used by HDFql as an HDFQL_VARCHAR.

**Example(s)**

```bash
# show (i.e. get) version of the HDF5 library used by HDFql (should be something similar to "1.8.21")
SHOW HDF5 VERSION
```

**6.7.24 SHOW PCRE VERSION**

**Syntax**

`SHOW PCRE VERSION`

```bash
[post_processing_option] [post_processing_option]*
[output_redirecting_option]
```

**Description**

Show (i.e. get) version of the PCRE library used by HDFql. The format of the version returned is MAJOR.MINOR. The PCRE library refers to the library used to compile HDFql, and which is shipped with HDFql (and not the PCRE library that may be installed in the machine).
Parameter(s)

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The version of the PCRE library used by HDFql as an HDFQL_VARCHAR.

Example(s)

```sh
# show (i.e. get) version of the PCRE library used by HDFql (should be something similar to "8.41")
SHOW PCRE VERSION
```

6.7.25 SHOW ZLIB VERSION

Syntax

SHOW ZLIB VERSION

    [post_processing_option] [post_processing_option]*

    [output_redirecting_option]

Description

Show (i.e. get) version of the ZLIB library used by HDFql. The format of the version returned is MAJOR.MINOR.REVISION. The ZLIB library refers to the library used to compile HDFql, and which is shipped with HDFql (and not the ZLIB library that may be installed in the machine).
Parameter(s)

`post_processing_option` – optional option that transforms the result set according to the programmer's needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

`output_redirecting_option` – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The version of the ZLIB library used by HDFql as an `HDFQL_VARCHAR`.

Example(s)

```sql
# show (i.e. get) version of the ZLIB library used by HDFql (should be something similar to "1.2.11")
SHOW ZLIB VERSION
```

## 6.7.26 SHOW MPI VERSION

Syntax

```
SHOW MPI VERSION
   [post_processing_option [post_processing_option]*]
   [output_redirecting_option]
```

Description

Show (i.e. get) version of the MPI library used by HDFql. The information returned depends on the MPI library loaded by HDFql at runtime (which must be previously installed in the machine). Please refer to https://www.mpich.org/static/docs/v3.2/www3/MPI_Get_library_version.html or https://www.open-mpi.org/doc/v2.1/man3/MPI_Get_library_version.3.php for additional information in case the MPI library used is MPICH (or, alternatively, one of its ABI compatible derivative libraries) or Open MPI.
Parameter(s)

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The version of the MPI library used by HDFql as an HDFQL_VARCHAR or nothing (in case of using an HDFql non MPI-based distribution or if in Windows as HDFql does not support the parallel HDF5 (PHDF5) library in this platform currently).

Example(s)

```
# show (i.e. get) version of the MPI library used by HDFql (e.g. if the MPI library used is Open MPI, it should be something similar to "Open MPI v2.1.3, package: Open MPI dummy@machine Distribution, ident: 2.1.3, repo rev: v2.1.2-129-gcfd8f3f, Mar 13, 2018")
SHOW MPI VERSION
```

6.7.27 SHOW DIRECTORY

Syntax

SHOW DIRECTORY [directory_name]

   [post_processing_option [post_processing_option]*]

   [output Redirecting_option]

Description

Show (i.e. get) directory names within a directory named directory_name or check the existence of a directory named directory_name.
**Parameter(s)**

directory_name – optional string that specifies the name of the directory whose directory names are to be obtained or the name of the directory to check for its existence. If directory_name is not specified, all directory names within the current working directory are returned. Otherwise, if directory_name is specified, one of the following behaviors applies:

- If it ends with “/”, all directory names within directory_name are returned.
- If it does not end with “/”, directory_name will be checked for its existence as a directory. If it exists, directory_name is returned; otherwise, if it does not exist, an error is raised.

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

**Return**

The directory names within a directory or the existence of a directory as an HDFQL_VARCHAR.

**Example(s)**

```
# create three directories named "my_directory0", "my_directory1" and "my_directory2" within the current working directory
CREATE DIRECTORY my_directory0, my_directory1, my_directory2

# create two directories named "my_subdirectory0" and "my_subdirectory1" within the directory "my_directory0"
CREATE DIRECTORY my_directory0/my_subdirectory0, my_directory0/my_subdirectory1

# show (i.e. get) directory names within the current working directory (should be "my_directory0", "my_directory1" and "my_directory2")
SHOW DIRECTORY

# show (i.e. get) directory names within directory "my_directory0" (should be "my_subdirectory0" and "my_subdirectory1")
SHOW DIRECTORY my_directory0/
```
6.7.28 SHOW FILE

Syntax

SHOW FILE [object_name]

[post_processing_option [post_processing_option]*]

[output_redirecting_option]

Description

Show (i.e. get) file names within a directory named object_name or check the existence of a file named object_name.

Parameter(s)

object_name – optional string that specifies the name of the directory whose file names are to be obtained or the name of the file to check for its existence. If object_name is not specified, all file names within the current working directory are returned. Otherwise, if object_name is specified, one of the following behaviors applies:

- If it ends with “/”, object_name will be treated as a directory and all file names within this directory are returned.
- If it does not end with “/”, object_name will be checked for its existence as a file. If it exists, object_name is returned; otherwise, if it does not exist, an error is raised.

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The file names within a directory or the existence of a file as an HDFQL_VARCHAR.
Example(s)

```bash
# create three HDF5 files named "my_file0.h5", "my_file1.h5" and "my_file2.h5" within the current working directory
CREATE FILE my_file0.h5, my_file1.h5, my_file2.h5

# create two HDF5 files named "my_file3.h5" and "my_file4.h5" within a directory named "my_directory"
CREATE FILE my_directory/my_file3.h5, my_directory/my_file4.h5

# show (i.e. get) file names within the current working directory (should be "my_file0.h5", "my_file1.h5" and "my_file2.h5")
SHOW FILE

# show (i.e. get) file names within directory "my_directory" (should be "my_file3.h5" and "my_file4.h5")
SHOW FILE my_directory/

# check the existence of a file named "my_file0.h5" within the current working directory (should be "my_file0.h5" - i.e. it exists)
SHOW FILE my_file0.h5
```

6.7.29 SHOW EXECUTE STATUS

**Syntax**

SHOW EXECUTE STATUS

```
[post_processing_option [post_processing_option]*]

[output Redirecting_option]
```

**Description**

Show (i.e. get) status of the last executed operation.

**Parameter(s)**

*post_processing_option* – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.
**outputRedirectingOption** – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter **CURSOR** and subsection **INTO** for additional information).

**Return**

The status of the last executed operation as an **HDFQL_INT**.

**Example(s)**

```
# show (i.e. get) current working directory (this operation will succeed since it is syntactically correct)
SHOW USE DIRECTORY

# show (i.e. get) status of the last executed operation (should be 0 – i.e. HDFQL_SUCCESS)
SHOW EXECUTE STATUS

# show (i.e. get) current working directory (this operation will fail since it is syntactically incorrect due to a typo in "SHOWX")
SHOWX USE DIRECTORY

# show (i.e. get) status of the last executed operation (should be -1 – i.e. HDFQL_ERROR_PARSE)
SHOW EXECUTE STATUS
```

### 6.7.30 SHOW LIBRARY BOUNDS

**Syntax**

```
SHOW [USE FILE] LIBRARY BOUNDS [FROM | TO]
          [post_processing_option [post_processing_option] *]
          [output_redirecting_option]
```

**Description**

Show (i.e. get) library bound values for creating or opening HDF5 files. If neither the keyword FROM nor TO is specified, all library bound values (i.e. from and to) are returned. To return a specific library bound value, the keyword FROM or TO must be specified. In case the keyword USE FILE is not specified, the library bound values returned refers to creating or opening files by default; otherwise, if it is specified, the library bound values of the file currently in use are returned.
Parameter(s)

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The library bound values for creating or opening HDF5 files as an HDFQL_INT.

Example(s)

```}
# show (i.e. get) library bound values from (i.e. lower bound) and to (i.e. upper bound)
(should be EARLIEST and LATEST - i.e. default values defined by the HDF5 library)
SHOW LIBRARY BOUNDS

# show (i.e. get) library bound value from (i.e. lower bound) (should be EARLIEST)
SHOW LIBRARY BOUNDS FROM

# show (i.e. get) library bound value to (i.e. upper bound) (should be LATEST)
SHOW LIBRARY BOUNDS TO

# set library bound from (i.e. lower bound) to LATEST (the library bound to - i.e. upper bound - remains intact) for subsequent usage (i.e. creating or opening HDF5 files)
SET LIBRARY BOUNDS FROM LATEST

# show (i.e. get) library bound values from (i.e. lower bound) and to (i.e. upper bound)
(should be LATEST and LATEST)
SHOW LIBRARY BOUNDS

# set both library bounds from (i.e. lower bound) and to (i.e. upper bound) to DEFAULT for subsequent usage (i.e. creating or opening HDF5 files)
SET LIBRARY BOUNDS FROM DEFAULT TO DEFAULT

# show (i.e. get) library bound values from (i.e. lower bound) and to (i.e. upper bound)
(should be EARLIEST and LATEST - i.e. default values defined by the HDF5 library)
SHOW LIBRARY BOUNDS
```

# use (i.e. open) an HDF5 file named "my_file0.h5" with library bounds from (i.e. lower bound) and to (i.e. upper bound) set to EARLIEST and LATEST respectively

USE FILE my_file0.h5

# show (i.e. get) library bound value from (i.e. lower bound) (should be EARLIEST)

SHOW USE FILE LIBRARY BOUNDS FROM

# show (i.e. get) library bound value to (i.e. upper bound) (should be LATEST)

SHOW USE FILE LIBRARY BOUNDS TO

# set library bound from (i.e. lower bound) to LATEST (the library bound to - i.e. upper bound - remains intact) for subsequent usage (i.e. creating or opening HDF5 files)

SET LIBRARY BOUNDS FROM LATEST

# use (i.e. open) an HDF5 file named "my_file1.h5" with both library bounds from (i.e. lower bound) and to (i.e. upper bound) set to LATEST respectively

USE FILE my_file1.h5

# show (i.e. get) library bound values from (i.e. lower bound) and to (i.e. upper bound) of the file currently in use (should be LATEST and LATEST)

SHOW USE FILE LIBRARY BOUNDS

# use (i.e. open) an HDF5 file named "my_file2.h5" with library bounds from (i.e. lower bound) and to (i.e. upper bound) set to EARLIEST (i.e. default value defined by the HDF5 library) and LATEST respectively

USE FILE my_file2.h5 LIBRARY BOUNDS FROM DEFAULT

# show (i.e. get) library bound values from (i.e. lower bound) and to (i.e. upper bound) of the file currently in use (should be EARLIEST and LATEST)

SHOW USE FILE LIBRARY BOUNDS

6.7.31 SHOW CACHE

Syntax

SHOW [[USE] FILE | DATASET] CACHE [SLOTS | SIZE | PREEMPTION]

   [post_processing_option [post_processing_option]*]

   [output Redirecting_option]
Description

Show (i.e. get) cache parameter values for accessing HDF5 files or datasets. If neither the keyword SLOTS, SIZE nor PREEMPTION is specified, all cache parameter values (i.e. for slots, size and preemption) are returned. To return a specific cache parameter value, the keyword SLOTS, SIZE or PREEMPTION must be specified. If neither the keyword FILE, USE FILE nor DATASET is specified, the cache parameters returned refers to files by default (optionally, the keyword FILE may be specified to make the purpose of this operation clearer). To explicitly return cache parameters of datasets or the file currently in use, the keyword DATASET or USE FILE must be specified.

Parameter(s)

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The cache parameter values for accessing HDF5 files or datasets as an HDFQL_DOUBLE (when returning all cache parameter values or the preemption cache parameter value only) or HDFQL_INT (when returning the slots or size cache parameter value).

Example(s)

```
# show (i.e. get) cache parameter values for accessing HDF5 files (should be 521, 1048576, 0.75)
SHOW CACHE

# show (i.e. get) cache preemption value for accessing HDF5 files (should be 0.75)
SHOW CACHE PREEMPTION

# show (i.e. get) cache parameter values for accessing HDF5 files (should be 521, 1048576, 0.75)
SHOW FILE CACHE

# show (i.e. get) cache slots value for accessing HDF5 datasets (should be 521)
SHOW DATASET CACHE SLOTS
```
# use (i.e. open) an HDF5 file named "my_file0.h5" with cache parameters values previously set (i.e. with slots, size and preemption values of 521, 1048576 and 0.75 respectively)

USE FILE my_file0.h5

# show (i.e. get) cache parameter values for accessing the HDF5 file currently in use (should be 521, 1048576, 0.75)

SHOW USE FILE CACHE

# use (i.e. open) an HDF5 file named "my_file1.h5" with cache slots, size and preemption values of 1523, 262144 and 0.6 respectively

USE FILE my_file1.h5 CACHE SLOTS 1523 SIZE 262144 PREEMPTION 0.6

# show (i.e. get) cache parameter values for accessing the HDF5 file currently in use (should be 1523, 262144, 0.6)

SHOW USE FILE CACHE

# use (i.e. open) an HDF5 file named "my_file2.h5" with a cache preemption value of 0.9

USE FILE my_file2.h5 CACHE PREEMPTION 0.9

# show (i.e. get) cache parameter values for accessing the HDF5 file currently in use (should be 521, 1048576, 0.9)

SHOW USE FILE CACHE

## 6.7.32 SHOW ATOMIC

### Syntax

SHOW [USE FILE] ATOMIC

   [post_processing_option [post_processing_option]*]

   [outputRedirecting_option]

### Description

Show (i.e. get) atomicity for accessing HDF5 files in an MPI environment. In case the keyword USE FILE is not specified, the atomicity returned refers to files that are subsequently opened; otherwise, if it is specified, the atomicity of the file currently in use is returned.
Parameter(s)

`post_processing_option` – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

`output_redirecting_option` – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The status of the atomicity for accessing HDF5 files as an `HDFQL_INT`, which can either be `HDFQL_ENABLED` or `HDFQL_DISABLED` depending on whether the atomicity for accessing files is enabled or disabled respectively.

Example(s)

```plaintext
# enable atomicity for accessing HDF5 files in an MPI environment
SET ATOMIC ENABLE

# show (i.e. get) atomicity for accessing HDF5 files in an MPI environment (should be 0 – i.e. HDFQL_ENABLED)
SHOW ATOMIC

# use (i.e. open) an HDF5 file named "my_file0.h5" with atomicity for accessing it in an MPI environment
USE PARALLEL FILE my_file0.h5

# show (i.e. get) atomicity of the HDF5 file currently in use in an MPI environment (should be 0 – i.e. HDFQL_ENABLED)
SHOW USE FILE ATOMIC

# disable atomicity for accessing HDF5 files in an MPI environment
SET ATOMIC DISABLE

# show (i.e. get) atomicity for accessing HDF5 files in an MPI environment (should be -1 – i.e. HDFQL_DISABLED)
SHOW ATOMIC

# use (i.e. open) an HDF5 file named "my_file1.h5" without atomicity for accessing it in an MPI environment
USE PARALLEL FILE my_file1.h5
```
# show (i.e. get) atomicity of the HDF5 file currently in use in an MPI environment (should be -1 - i.e. HDFQL_DISABLED)

SHOW USE FILE ATOMIC

# use (i.e. open) an HDF5 file named "my_file2.h5" with atomicity for accessing it in an MPI environment

USE ATOMIC PARALLEL FILE my_file2.h5

# show (i.e. get) atomicity of the HDF5 file currently in use in an MPI environment (should be 0 - i.e. HDFQL_ENABLED)

SHOW USE FILE ATOMIC

### 6.7.33 SHOW EXTERNAL LINK PREFIX

**Syntax**

SHOW EXTERNAL LINK PREFIX

    [post_processing_option [post_processing_option]*]

    [output redirecting_option]

**Description**

Show (i.e. get) prefix to prepend to file names stored in HDF5 external links.

**Parameter(s)**

*post_processing_option* – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

*output redirecting_option* – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).
Return

The prefix to prepend to file names stored in HDF5 external links as an HDFQL_VARCHAR or nothing (in case no prefix is specified).

Example(s)

```
# set external link prefix to "/target"
SET EXTERNAL LINK PREFIX /target

# show (i.e. get) external link prefix (should be "/target")
SHOW EXTERNAL LINK PREFIX

# set external link prefix to default (i.e. empty)
SET EXTERNAL LINK PREFIX DEFAULT

# show (i.e. get) external link prefix (should be empty)
SHOW EXTERNAL LINK PREFIX
```

6.7.34 SHOW FLUSH

Syntax

```
SHOW FLUSH

   [post_processing_option [post_processing_option]*/

   [output_redirecting_option]
```

Description

Show (i.e. get) status of the automatic flushing.

Parameter(s)

`post_processing_option` – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.
output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

**Return**

The status of the automatic flushing as an HDFQL_INT, which can either be HDFQL_GLOBAL, HDFQL_LOCAL or HDFQL_DISABLED depending on whether the automatic flushing of the entire virtual HDF5 file (global) or only the HDF5 file (local) currently in use is enabled or disabled respectively.

**Example(s)**

```plaintext
# enable automatic flushing of the entire virtual HDF5 file (global) currently in use
SET FLUSH ENABLE

# show (i.e. get) status of the automatic flushing (should be 1 – i.e. HDFQL_GLOBAL)
SHOW FLUSH

# enable automatic flushing of only the HDF5 file (local) currently in use
SET FLUSH LOCAL ENABLE

# show (i.e. get) status of the automatic flushing (should be 2 – i.e. HDFQL_LOCAL)
SHOW FLUSH

# disable automatic flushing of the entire virtual HDF5 file (global) or only the HDF5 file (local) currently in use
SET FLUSH DISABLE

# show (i.e. get) status of the automatic flushing (should be -1 – i.e. HDFQL_DISABLED)
SHOW FLUSH
```

### 6.7.35 SHOW DEBUG

**Syntax**

SHOW DEBUG

[post_processing_option [post_processing_option]*]

[output_redirecting_option]
Description

Show (i.e. get) status of the debug mechanism.

Parameter(s)

post_processing_option – optional option that transforms the result set according to the programmer’s needs such as ordering or truncating (please refer to the section POST-PROCESSING for additional information). Multiple post processing options are separated with a space.

output_redirecting_option – optional option that specifies a (text or binary) file or memory (i.e. user-defined variable) to write the result set into. If not specified, the cursor in use is populated with the result set instead (please refer to the chapter CURSOR and subsection INTO for additional information).

Return

The status of the debug mechanism as an HDFQL_INT, which can either be HDFQL_ENABLED or HDFQL_DISABLED depending on whether the debug mechanism is enabled or disabled respectively.

Example(s)

```bash
# enable debug mechanism (i.e. debug messages will be displayed when executing operations)
SET DEBUG ENABLE

# show (i.e. get) status of the debug mechanism (should be 0 – i.e. HDFQL_ENABLED)
SHOW DEBUG

# disable debug mechanism (i.e. debug messages will not be displayed when executing operations)
SET DEBUG DISABLE

# show (i.e. get) status of the debug mechanism (should be -1 – i.e. HDFQL_DISABLED)
SHOW DEBUG
```

6.8 MISCELLANEOUS

This section assembles all remaining HDFql operations that – due to their heterogeneous nature and functionality – do not fit in the previous sections about the language for data definition, manipulation, querying and introspection.
6.8.1 USE DIRECTORY

Syntax

USE DIRECTORY directory_name

Description

Use a directory named directory_name for subsequent operations. This will change the current working directory to directory_name thus avoiding the need to explicitly provide the full path of this directory when working within it (i.e. subsequent operations are done relatively to this directory, unless otherwise specified). If directory_name was not found or could not be opened (due to unknown/unexpected reasons), an error is raised.

Parameter(s)

directory_name – mandatory string that specifies the name of the directory to use for subsequent operations.

Return

Nothing

Example(s)

```
# set working directory currently in use to "/
USE DIRECTORY /

# show (i.e. get) current working directory (should be "/")
SHOW USE DIRECTORY

# create a directory named "my_directory"
CREATE DIRECTORY my_directory

# set working directory currently in use to "my_directory" (more precisely "/my_directory")
USE DIRECTORY my_directory

# show (i.e. get) current working directory (should be "/my_directory")
SHOW USE DIRECTORY

# create two directories named "my_subdirectory0" and "my_subdirectory1" (both directories will be created in directory "/my_directory")
CREATE DIRECTORY my_subdirectory0, my_subdirectory1
```
# set directory currently in use to "my_subdirectory0" (more precisely "/my_directory/my_subdirectory0")
USE DIRECTORY my_subdirectory0

# show (i.e. get) current working directory (should be "/my_directory/my_subdirectory0")
SHOW USE DIRECTORY

# set directory currently in use to "my_subdirectory1" located one level up (more precisely "/my_directory/my_subdirectory1")
USE DIRECTORY ../my_subdirectory1

# show (i.e. get) current working directory (should be "/my_directory/my_subdirectory1")
SHOW USE DIRECTORY

# set directory currently in use two levels up (should be "/")
USE DIRECTORY ../..

# show (i.e. get) current working directory (should be "/")
SHOW USE DIRECTORY

## 6.8.2 USE FILE

### Syntax

USE [READONLY] [[ATOMIC] PARALLEL] FILE file_name [, file_name]*

[LIBRARY BOUNDS [FROM {EARLIEST | LATEST | V18 | DEFAULT}] [TO {LATEST | V18 | DEFAULT}]]

[CACHE [SLOTS {slots_value | DEFAULT}] [SIZE {size_value | DEFAULT}] [PREEMPTION {preemption_value | DEFAULT}]]

### Description

Use (i.e. open) an HDF5 file named file_name for subsequent operations. Multiple files can be opened at once by separating these with a comma (,). If file_name was not found or could not be opened (due to unknown/unexpected reasons), no subsequent files are opened, and an error is raised. By default, the file is opened with read/write permissions. To open a file with read only permission, the keyword READONLY should be specified (any subsequent attempt to write into this file will raise an error). HDFql tracks opened files in a stack fashion (i.e. LIFO) meaning that the most recently
opened file is the one currently in use. In case the keyword PARALLEL is specified, HDFql opens the file using all the MPI processes specified upon launching the program (that employs HDFql). In case the keyword ATOMIC is specified, all file access operations will appear atomic, guaranteeing sequential consistency in an MPI environment (i.e. the operations will behave as though they were performed in a serial order consistent with the program order). In case the keyword LIBRARY BOUNDS is specified, HDFql opens the file using these bounds (instead of the library bounds that may have been set through the operation SET LIBRARY BOUNDS). In case the keyword CACHE is specified, HDFql opens the file using cache parametrized with the slots_value, size_value and preemption_value values (instead of the file cache parameters that may have been set through the operation SET CACHE).

**Parameter(s)**

file_name – mandatory string that specifies the name of the HDF5 file to use (i.e. open) for subsequent operations. Multiple files are separated with a comma (,).

slots_value – optional integer that specifies the number of chunk slots in the raw data chunk cache for accessing the HDF5 file. Due to the hashing strategy, its value should ideally be a prime number. In case the keyword DEFAULT is specified, its value is 521 (i.e. default value defined by the HDF5 library). In case the keyword SLOTS is not specified, its current value remains intact.

size_value – optional integer that specifies the total size of the raw data chunk cache in bytes for accessing the HDF5 file. In case the keyword DEFAULT is specified, its value is 1048576 (i.e. 1 MB – default value defined by the HDF5 library). In case the keyword SIZE is not specified, its current value remains intact.

preemption_value – optional float that specifies the chunk preemption policy for accessing the HDF5 file. Its value must be between 0 and 1. It indicates the weighting according to which chunks which have been fully read or written are penalized when determining which chunks to flush from cache. In case the keyword DEFAULT is specified, its value is 0.75 (i.e. default value defined by the HDF5 library). In case the keyword PREEMPTION is not specified, its current value remains intact.

**Return**

Nothing

---

46 This option is not allowed in Windows as HDFql does not support the parallel HDF5 (PHDF5) library in this platform currently.
Example(s)

# use (i.e. open) an HDF5 file named "my_file0.h5" located in the current working directory
USE FILE my_file0.h5

# use (i.e. open) an HDF5 file named "my_file1.h5" located in a root directory named "data"
USE FILE /data/my_file1.h5

# use (i.e. open) two HDF5 files named "my_file2.h5" and "my_file3.h5" with read only permissions (both files are located in the current working directory)
USE READONLY FILE my_file2.h5, my_file3.h5

# use (i.e. open) an HDF5 file named "my_file4.h5" located in the parent directory with the latest version of the HDF5 library
USE FILE ../my_file4.h5 LIBRARY BOUNDS FROM LATEST TO LATEST

# use (i.e. open) an HDF5 file named "my_file5.h5" located in the current working directory with cache slots, size and preemption values of 1523, 262144 and 0.6 respectively
USE FILE my_file5.h5 CACHE SLOTS 1523 SIZE 262144 PREEMPTION 0.6

# use (i.e. open) an HDF5 file named "my_file6.h5" located in the current working directory with the earliest version of the HDF5 library and a cache preemption value of 0.9
USE FILE my_file6.h5 LIBRARY BOUNDS FROM EARLIEST CACHE PREEMPTION 0.9

# use (i.e. open) an HDF5 file named "my_file7.h5" located in the current working directory in parallel (i.e. all the MPI processes specified upon launching the program (that employs HDFql) will collectively open the file - e.g. if the program is launched as "mpiexec -n 4 my_program", all the four MPI processes will participate in the opening of the file)
USE PARALLEL FILE my_file7.h5

# use (i.e. open) an HDF5 file named "my_file8.h5" located in the current working directory in parallel with atomicity for accessing it
USE ATOMIC PARALLEL FILE my_file8.h5

6.8.3 USE GROUP

Syntax

USE GROUP group_name
Description

Use (i.e. open) an HDF5 group named `group_name` for subsequent operations. This will change the current working group to `group_name` thus avoiding the need to explicitly provide the full path of this group when working within it (i.e. subsequent operations are done relatively to this group, unless otherwise specified). If `group_name` was not found or could not be opened (due to unknown/unexpected reasons), an error is raised. Upon using (i.e. opening) an HDF5 file, the group currently in use is “/” (i.e. the root of the HDF5 file).

Parameter(s)

`group_name` – mandatory string that specifies the name of the HDF5 group to use (i.e. open) for subsequent operations. Besides the name of the group to be used for subsequent operations, `group_name` may be composed of special tokens (that are not part of the name of the group itself). These are:

- “/” to separate multiple groups. Example: “USE GROUP my_group/my_subgroup/my_subsubgroup”.
- “.” to refer to the group currently in use. Example: “USE GROUP .”.
- “..” to go up one level from the group currently in use. Example: “USE GROUP ..”.

Return

Nothing

Example(s)

```bash
# set group currently in use to "/" (i.e. the root of the HDF5 file)
USE GROUP /

# create two HDF5 groups named "my_group0" and "my_group1" (both groups will be created in group "/")
CREATE GROUP my_group0, my_group1

# create an HDF5 dataset named "my_dataset0" of data type double (it will be created in group "/")
CREATE DATASET my_dataset0 AS DOUBLE

# set group currently in use to "my_group0" (more precisely "/my_group0")
USE GROUP my_group0

# create an HDF5 dataset named "my_dataset1" of data type double (it will be created in group
```
"/my_group0")

CREATE DATASET my_dataset1 AS DOUBLE

# create an HDF5 group named "my_subgroup0" (it will be created in group "/my_group0")
CREATE GROUP my_subgroup0

# create an HDF5 dataset named "my_dataset2" of data type variable-length double (it will be
created in group "/my_group0/my_subgroup0")
CREATE DATASET my_subgroup0/my_dataset2 AS VARDOUBLE

# create an HDF5 attribute named "my_attribute0" of data type float (it will be created in
group "/")
CREATE ATTRIBUTE ../my_attribute0 AS FLOAT

# set group currently in use to "my_subgroup0" (more precisely "/my_group0/my_subgroup0")
USE GROUP my_subgroup0

# create an HDF5 attribute named "my_attribute1" of data type char (it will be created in group
"/my_group1")
CREATE ATTRIBUTE ../../my_group1/my_attribute1 AS CHAR

# create an HDF5 attribute named "my_attribute2" of data type variable-length char (it will be
created in group "/")
CREATE ATTRIBUTE /my_attribute2 AS VARCHAR

# set group currently in use to "." (the group currently in use will not change - i.e. it
remains "/my_group0/my_subgroup0" - as "." refers to the current working group itself)
USE GROUP .

# create an HDF5 attribute named "my_attribute3" of data type int (it will be created in group
"/my_group0/my_subgroup0")
CREATE ATTRIBUTE my_attribute3 AS INT

# set group currently in use one level up (should be "/my_group0")
USE GROUP..

# create an HDF5 attribute named "my_attribute4" of data type short (it will be created in
group "/my_group0")
CREATE ATTRIBUTE my_attribute4 AS SMALLINT
6.8.4 FLUSH

Syntax

FLUSH [GLOBAL | LOCAL]

Description

Flush the entire virtual HDF5 file (global) or the specific HDF5 file (local) currently in use. All buffered data will be written into the disk. If neither the keyword GLOBAL nor LOCAL is specified, a global flush is performed by default (optionally, the keyword GLOBAL may be specified to make the purpose of this operation clearer). To perform a local flush, the keyword LOCAL must be specified. If no file is currently used, no flush is performed.

Parameter(s)

None

Return

Nothing

Example(s)

```
# flush the entire virtual HDF5 file (global) currently in use
FLUSH

# flush the entire virtual HDF5 file (global) currently in use
FLUSH GLOBAL

# flush only the HDF5 file (local) currently in use
FLUSH LOCAL
```

6.8.5 CLOSE FILE

Syntax

CLOSE FILE [file_name [, file_name]*]
**Description**

Close a certain HDF5 file used (i.e. opened) or the HDF5 file currently in use. Multiple files can be closed at once by separating these with a comma (,). If `file_name` is not in use (i.e. open) or it is not possible to close it (due to unknown/unexpected reasons), no subsequent files are closed, and an error is raised. Before closing a file, all buffered data will be written into it. After closing a file, the file in use will be the one most recently used before the closed file. If `file_name` is specified, it will be closed regardless of whether it is the file currently in use or not. Otherwise, if `file_name` is not specified, the file currently in use will be closed.

**Parameter(s)**

`file_name` – optional string that specifies the name of the HDF5 file to close. Multiple files are separated with a comma (,). If `file_name` is specified it must match exactly the name of the file when it was opened (otherwise no file will be closed and an error is raised).

**Return**

Nothing

**Example(s)**

```bash
# use (i.e. open) four HDF5 files named "my_file0.h5", "my_file1.h5", "my_file2.h5" and "my_file3.h5"
USE FILE my_file0.h5, my_file1.h5, my_file2.h5, my_file3.h5

# show (i.e. get) HDF5 file currently in use (i.e. open) (should be "my_file3.h5")
SHOW USE FILE

# close HDF5 file currently in use (i.e. file "my_file3.h5")
CLOSE FILE

# show (i.e. get) HDF5 file currently in use (i.e. open) (should be "my_file2.h5")
SHOW USE FILE

# close HDF5 file "my_file1.h5"
CLOSE FILE my_file1.h5

# show (i.e. get) HDF5 file currently in use (i.e. open) (should be "my_file2.h5")
SHOW USE FILE

# close HDF5 file currently in use (i.e. file "my_file2.h5")
```
CLOSE FILE

# show (i.e. get) HDF5 file currently in use (i.e. open) (should be "my_file0.h5")
SHOW USE FILE

# close HDF5 file currently in use (i.e. file "my_file0.h5")
CLOSE FILE

# show (i.e. get) HDF5 file currently in use (i.e. open) (should be empty)
SHOW USE FILE

6.8.6 CLOSE ALL FILE

Syntax

CLOSE ALL FILE

Description

Close all HDF5 files in use. All buffered data will be written into the respective files before closing them. If it is not possible to close a file (due to unknown/unexpected reasons), no subsequent files are closed, and an error is raised.

Parameter(s)

None

Return

Nothing

Example(s)

# use (i.e. open) three HDF5 files named "my_file0.h5", "my_file1.h5" and "my_file2.h5"
USE FILE my_file0.h5, my_file1.h5, my_file2.h5

# show (i.e. get) all HDF5 files in use (i.e. open) (should be "my_file2.h5", "my_file1.h5", "my_file0.h5")
SHOW ALL USE FILE

# close all HDF5 files in use (i.e. open)
### 6.8.7 CLOSE GROUP

**Syntax**

```
CLOSE GROUP
```

**Description**

Close the HDF5 group currently in use. After closing it, the group currently in use will be “/” (i.e. the root of the HDF5 file). If no file is currently used, no group is closed.

**Parameter(s)**

None

**Return**

Nothing

**Example(s)**

```bash
# use (i.e. open) an HDF5 file named "my_file.h5"
USE FILE my_file.h5

# show (i.e. get) current working group (should be "/")
SHOW USE GROUP

# create an HDF5 group named "my_group"
CREATE GROUP my_group

# set group currently in use to "my_group" (more precisely "/my_group")
USE GROUP my_group

# show (i.e. get) current working group (should be "/my_group")
SHOW USE GROUP
```
# create an HDF5 dataset named "my_dataset" of data type double (more precisely "/my_group/my_dataset")
      \texttt{CREATE\ DATASET\ my_dataset\ AS\ DOUBLE}

# set group currently in use to "/" (i.e. the root of the HDF5 file)
      \texttt{CLOSE\ GROUP}

# show (i.e. get) current working group (should be "/")
      \texttt{SHOW\ USE\ GROUP}

# create an HDF5 dataset named "my_dataset" of data type int (more precisely "/ my_dataset")
      \texttt{CREATE\ DATASET\ my_dataset\ AS\ INT}

### 6.8.8 SET LIBRARY BOUNDS

**Syntax**

\texttt{SET\ LIBRARY\ BOUNDS\ [FROM\ \{EARLIEST\ |\ LATEST\ |\ V18\ |\ DEFAULT\}]\ [TO\ \{LATEST\ |\ V18\ |\ DEFAULT\}]}

**Description**

Set library bounds\(^{47}\) for creating and opening HDF5 files. In other words, it sets bounds on library versions to be used when creating objects (the object format versions are determined indirectly from the HDF5 library versions specified in the call). All files that are subsequently created or opened (through the operations \texttt{CREATE\ FILE} or \texttt{USE\ FILE}) will use the default bound values defined by the HDF5 library or user-defined bound values. These bounds are:

- **From** – lower bound on the range of possible library versions used to create the object. The library version indirectly specifies the earliest object format version that can be used when creating objects in an HDF5 file. In case the keyword \texttt{DEFAULT} is specified, its value is \texttt{EARLIEST} (i.e. default value defined by the HDF5 library). In case the keyword \texttt{FROM} is not specified (i.e. the lower bound), its current value remains intact.

- **To** – upper bound on the range of possible library versions used to create the object. The library version indirectly specifies the latest object format version that can be used when creating objects in an HDF5 file. In case the keyword

\(^{47}\) By default (i.e. upon initialization of the HDFql library), the library bounds from (i.e. lower bound) and to (i.e. upper bound) are set to \texttt{EARLIEST} and \texttt{LATEST} respectively.
DEFAULT is specified, its value is LATEST (i.e. default value defined by the HDF5 library). In case the keyword TO is not specified (i.e. the upper bound), its current value remains intact.

Parameter(s)

None

Return

Nothing

Example(s)

```
# use (i.e. open) an HDF5 file named "my_file0.h5" with library bounds from (i.e. lower bound) and to (i.e. upper bound) set to EARLIEST and LATEST respectively (default values defined by the HDF5 library)
USE FILE my_file0.h5

# set library bound from (i.e. lower bound) to LATEST (the library bound to - i.e. upper bound - remains intact) for subsequent usage (i.e. creating or opening HDF5 files)
SET LIBRARY BOUNDS FROM LATEST

# use (i.e. open) an HDF5 file named "my_file1.h5" with both library bounds from (i.e. lower bound) and to (i.e. upper bound) set to LATEST
USE FILE my_file1.h5

# set library bound to (i.e. upper bound) to V18 (the library bound from - i.e. lower bound - remains intact) for subsequent usage (i.e. creating or opening HDF5 files)
SET LIBRARY BOUNDS TO V18

# use (i.e. open) an HDF5 file named "my_file2.h5" with library bounds from (i.e. lower bound) and to (i.e. upper bound) set to LATEST and V18 respectively
USE FILE my_file2.h5

# set both library bounds from (i.e. lower bound) and to (i.e. upper bound) to DEFAULT for subsequent usage (i.e. creating or opening HDF5 files)
SET LIBRARY BOUNDS FROM DEFAULT TO DEFAULT

# use (i.e. open) an HDF5 file named "my_file3.h5" with library bounds from (i.e. lower bound) and to (i.e. upper bound) set to EARLIEST and LATEST respectively (default values defined by the HDF5 library)
USE FILE my_file3.h5
```
6.8.9 SET CACHE

Syntax

```plaintext
SET [FILE | DATASET] CACHE [SLOTS {slots_value | FILE | DEFAULT}] [SIZE {size_value | FILE | DEFAULT}] [PREEMPTION {preemption_value | FILE | DEFAULT}]
```

Description

Set cache parameters for accessing HDF5 files or datasets. All files or datasets that are subsequently opened or read (through the operations USE FILE or SELECT respectively) will use the default cache parameter values defined by the HDF5 library or user-defined cache parameter values. These cache parameters are:

- **Slots** – number of chunk slots in the raw data chunk cache.
- **Size** – total size of the raw data chunk cache in bytes.
- **Preemption** – chunk preemption policy.

If neither the keyword FILE nor DATASET is specified, the setting of the cache parameters refers to files by default (optionally, the keyword FILE may be specified to make the purpose of this operation clearer). To explicitly set the cache parameters to datasets, the keyword DATASET must be specified.

**Parameter(s)**

- **slots_value** – number of chunk slots in the raw data chunk cache for accessing HDF5 files or datasets. Due to the hashing strategy, its value should ideally be a prime number. In case the keyword FILE is specified, its value will be as defined in the cache slots parameter upon using (i.e. opening) the file. In case the keyword DEFAULT is specified, its value is 521 (i.e. default value defined by the HDF5 library). In case the keyword SLOTS is not specified, its current value remains intact.

- **size_value** – total size of the raw data chunk cache in bytes for accessing HDF5 files or datasets. In case the keyword FILE is specified, its value will be as defined in the cache size parameter upon using (i.e. opening) the file. In case the keyword DEFAULT is specified, its value is 1048576 (i.e. 1 MB – default value defined by the HDF5 library). In case the keyword SIZE is not specified, its current value remains intact.

- **preemption_value** – chunk preemption policy for accessing HDF5 files or datasets. Its value must be between 0 and 1. It indicates the weighting according to which chunks which have been fully read or written are penalized when determining

---

48 By default (i.e. upon initialization of the HDFql library), the cache parameters slots, size and preemption are set to 521, 1048576 and 0.75 respectively.
which chunks to flush from cache. In case the keyword FILE is specified, its value will be as defined in the cache preemption parameter upon using (i.e. opening) the file. In case the keyword DEFAULT is specified, its value is 0.75 (i.e. default value defined by the HDF5 library). In case the keyword PREEMPTION is not specified, its current value remains intact.

**Return**

Nothing

**Example(s)**

```plaintext
# use (i.e. open) an HDF5 file named "my_file0.h5" with cache slots, size and preemption values of 521, 1048576 and 0.75 respectively (default values defined by the HDF5 library)
USE FILE my_file0.h5

# set cache slots and preemption values to 2297 and 0.9 respectively (the cache size value remains intact) for subsequent usage (i.e. opening HDF5 files)
SET CACHE SLOTS 2297 PREEMPTION 0.9

# use (i.e. open) an HDF5 file named "my_file1.h5" with cache slots, size and preemption values of 2297, 1048576 and 0.9 respectively
USE FILE my_file1.h5

# set cache slots, size and preemption values to 1523, 262144 and 0.6 respectively for subsequent usage (i.e. opening HDF5 files)
SET FILE CACHE SLOTS 1523 SIZE 262144 PREEMPTION 0.6

# use (i.e. open) an HDF5 file named "my_file2.h5" with cache slots, size and preemption values of 1523, 262144 and 0.6 respectively
USE FILE my_file2.h5

# set cache size value to 1048576 (default value defined by the HDF5 library) and preemption value to 0.4 (the cache slots value remains intact) for subsequent usage (i.e. opening HDF5 files)
SET FILE CACHE SIZE DEFAULT PREEMPTION 0.4

# use (i.e. open) an HDF5 file named "my_file3.h5" with cache slots, size and preemption values of 1523, 1048576 and 0.4 respectively
USE FILE my_file3.h5
```
## Hierarchical Data Format query language (HDFql) Reference Manual

# select (i.e. read) an HDF5 dataset named "my_dataset0" with cache slots, size and preemption values of 521, 1048576 and 0.75 respectively (default values defined by the HDF5 library)

```sql
SELECT FROM my_dataset0
```

# set cache slots and preemption values to 2297 and 0.9 respectively (the cache size value remains intact) for subsequent selection (i.e. reading HDF5 datasets)

```sql
SET DATASET CACHE SLOTS 2297 PREEMPTION 0.9
```

# select (i.e. read) an HDF5 dataset named "my_dataset1" with cache slots, size and preemption values of 2297, 1048576 and 0.9 respectively

```sql
SELECT FROM my_dataset1
```

# set cache slots, size and preemption values to 1523, 262144 and 0.6 respectively for subsequent selection (i.e. reading HDF5 datasets)

```sql
SET DATASET CACHE SLOTS 1523 SIZE 262144 PREEMPTION 0.6
```

# select (i.e. read) an HDF5 dataset named "my_dataset2" with cache slots, size and preemption values of 1523, 262144 and 0.6 respectively

```sql
SELECT FROM my_dataset2
```

# set cache size value to 1048576 (default value defined by the HDF5 library) and preemption value to 0.4 (the cache slots value remains intact) for subsequent selection (i.e. reading HDF5 datasets)

```sql
SET DATASET CACHE SIZE DEFAULT PREEMPTION 0.4
```

# select (i.e. read) an HDF5 dataset named "my_dataset3" with cache slots, size and preemption values of 1523, 1048576 and 0.4 respectively

```sql
SELECT FROM my_dataset3
```

# set cache slots, size and preemption values to 3089, 2048 and 0.85 respectively for subsequent usage (i.e. opening HDF5 files)

```sql
SET FILE CACHE SLOTS 3089 SIZE 2048 PREEMPTION 0.85
```

# set cache slots value to 521 (default value defined by the HDF5 library), size value to 1024, and preemption value to 0.85 (defined by the cache preemption value for HDF5 files) for subsequent selection (i.e. reading HDF5 datasets)

```sql
SET DATASET CACHE SLOTS DEFAULT SIZE 1024 PREEMPTION FILE
```

# select (i.e. read) an HDF5 dataset named "my_dataset4" with cache slots, size and preemption values of 521, 1024 and 0.85 respectively

```sql
SELECT FROM my_dataset4
```
6.8.10 SET ATOMIC

Syntax

SET [USE FILE] ATOMIC {ENABLE | DISABLE | DEFAULT}

Description

Set atomicity\(^{49}\) for accessing HDF5 files in an MPI environment to enabled or disabled. All files that are subsequently opened (through the operation USE FILE) will have access operations performed in an atomic fashion or not accordingly. If enabled, all file access operations will appear atomic, guaranteeing sequential consistency in an MPI environment (i.e. the operations will behave as though they were performed in a serial order consistent with the program order). If disabled, no enforcement of atomic file access will be done. In case the keyword DEFAULT is specified, the atomicity for accessing files is set to disabled (i.e. equivalent to specifying the keyword DISABLE). In case the keyword USE FILE is specified, subsequent access operations of the file currently in use will be performed in an atomic fashion or not accordingly.

Parameter(s)

None

Return

Nothing

Example(s)

```
# enable atomicity for accessing HDF5 files in an MPI environment
SET ATOMIC ENABLE

# use (i.e. open) an HDF5 file named "my_file0.h5" with atomicity for accessing it in an MPI environment
USE PARALLEL FILE my_file0.h5

# disable atomicity for accessing HDF5 files in an MPI environment
SET ATOMIC DISABLE

# use (i.e. open) an HDF5 file named "my_file1.h5" without atomicity for accessing it in an MPI environment
```

\(^{49}\) By default (i.e. upon initialization of the HDFql library), the atomicity for accessing HDF5 files is set to disabled.


USE PARALLEL FILE my_file1.h5

# enable atomicity for accessing the HDF5 file currently in use (i.e. file "my_file1.h5") in an MPI environment
SET USE FILE ATOMIC

# use (i.e. open) an HDF5 file named "my_file2.h5" with atomicity for accessing it in an MPI environment
USE ATOMIC PARALLEL FILE my_file2.h5

6.8.11 SET EXTERNAL LINK PREFIX

Syntax

SET EXTERNAL LINK PREFIX {prefix_value | DEFAULT}

Description

Set prefix\(^{50}\) to prepend to file names stored in HDF5 external links. In other words, before resolving a file name stored in an external link, the prefix prefix_value is prepended to the name. In case the keyword DEFAULT is specified, the prefix to resolve file names is set to empty (i.e. nothing is prepended).

Parameter(s)

prefix_value – optional string that specifies the prefix to prepend to file names stored in HDF5 external links.

Return

Nothing

Example(s)

# set directory currently in use to "/data"
USE DIRECTORY /data

# create two HDF5 files named "my_file0.h5" and "my_file1.h5" in the directory currently in use (i.e. directory "/data") and in a root directory named "target" respectively

\(^{50}\) By default (i.e. upon initialization of the HDFql library), the prefix is set to empty (i.e. nothing is prepended).
CREATE FILE my_file0.h5, /target/my_file1.h5

# create an HDF5 dataset named "my_dataset" (in file "my_file1.h5" located in root directory "target") of data type float with an initial value of 7.8
CREATE DATASET /target/my_file1.h5 my_dataset AS FLOAT VALUES(7.8)

# use (i.e. open) HDF5 file "my_file0.h5" located in the directory currently in use (i.e. directory "/data")
USE FILE my_file0.h5

# create an HDF5 external link named "my_link" in file "my_file1.h5" to object "my_dataset" in file "my_file1.h5"
CREATE EXTERNAL LINK my_link TO my_file1.h5 my_dataset

# select (i.e. read) data from object "my_link" and populate cursor in use with it (should raise an error since "my_link" is a dangling link due to "my_file1.h5" being located in directory "/target" and not in the directory currently in use)
SELECT FROM my_link

# set external link prefix to "/target"
SET EXTERNAL LINK PREFIX /target

# select (i.e. read) data from object "my_link" and populate cursor in use with it (should be 7.8)
SELECT FROM my_link

6.8.12 SET FLUSH

**Syntax**

```
SET FLUSH ((GLOBAL | LOCAL) ENABLE | DISABLE | DEFAULT)
```

**Description**

Set automatic flushing\(^{51}\) of the entire virtual HDF5 file (global) or only the HDF5 file (local) currently in use to enabled or disabled. If enabled, automatic flushing (i.e. all buffered data is written into the disk) will subsequently occur whenever an operation modifying the file is executed. If neither the keyword GLOBAL nor LOCAL is specified, automatic global flushing is set by default (optionally, the keyword GLOBAL may be specified to make the purpose of this operation clearer). To set

\(^{51}\) By default (i.e. upon initialization of the HDFql library), the automatic flushing is set to disabled.
automatic local flushing, the keyword LOCAL must be specified. In case the keyword DEFAULT is specified, the automatic flushing is set to disabled (i.e. equivalent to specifying the keyword DISABLE).

Parameter(s)

None

Return

Nothing

Example(s)

```bash
# enable automatic flushing of the entire virtual HDF5 file (global) currently in use
SET FLUSH ENABLE

# enable automatic flushing of the entire virtual HDF5 file (global) currently in use
SET FLUSH GLOBAL ENABLE

# enable automatic flushing of only the HDF5 file (local) currently in use
SET FLUSH LOCAL ENABLE

# disable automatic flushing of the entire virtual HDF5 file (global) or only the HDF5 file (local) currently in use
SET FLUSH DISABLE
```

6.8.13 SET DEBUG

Syntax

```bash
SET DEBUG {ENABLE | DISABLE | DEFAULT}
```

Description

Set debug mechanism\(^{52}\) to enabled or disabled. If enabled, debug messages will be displayed when executing operations, which should help the programmer to have a better understanding of the parameters HDFql is receiving, the operation performed, and the return value of this operation. Additionally, debug messages of the HDF5 library itself are displayed in

\(^{52}\) By default (i.e. upon initialization of the HDFql library), the debug mechanism is set to disabled.
case of an error. In case the keyword DEFAULT is specified, the debug mechanism is set to disabled (i.e. equivalent to specifying the keyword DISABLE).

**Parameter(s)**

None

**Return**

Nothing

**Example(s)**

```
# enable debug mechanism (i.e. debug messages will be displayed when executing operations)
SET DEBUG ENABLE

# disable debug mechanism (i.e. debug messages will not be displayed when executing operations)
SET DEBUG DISABLE
```
Application Programming Interface (API)

An Application Programming Interface (API) specifies how software components should interact with each other. In practice, an API comes in the form of a library that includes specifications for functions, data structures, object classes, constants and variables. A good API makes it easier to develop a program by providing all the building blocks.

Attribute

An (HDF5) attribute is a metadata object describing the nature and/or intended usage of a primary data object. A primary data object may be a group, dataset or committed data type. Attributes are assumed to be very small as data objects go, so storing them as standard (HDF5) datasets would be inefficient.

Cursor

A cursor is a control structure that is used to iterate through the results returned by a query (that was previously executed). It can be seen as an effective means to abstract the programmer from low-level implementation details of accessing data stored in specific structures. Besides offering several ways to traverse result sets according to specific needs, cursors in HDFql also store result sets returned by DATA QUERY LANGUAGE (DQL) and DATA INTROSPECTION LANGUAGE (DIL) operations.

Dataset

A (HDF5) dataset is an object composed of a collection of data elements and metadata that stores a description of the data elements, data layout and all other information necessary to write and read the data. A dataset may be a multidimensional array of data elements and it may have zero or more attributes.
Data type

A data type is a classification identifying one of various types of data such as integer, real or string, which determines the possible values for that type, the operations that can be done on values of that type, the meaning of the data, and the way values of that type can be stored. In other words, a data type is a classification of data that tells HDFql how the user intends to use it.

Endianness

Endianness refers to the ordering of packing bytes into words when stored in memory. In big endian format, whenever addressing memory or storing words bytewise, the most significant byte – i.e. the byte containing the most significant bit – is stored first (has the lowest address); subsequently, the following bytes are stored in order of decreasing significance, with the least significant byte – i.e. the one containing the least significant bit – stored last (having the highest address). The little endian format reverses this order: the sequence addresses/stores the least significant byte first (lowest address) and the most significant byte last (highest address).

Group

A (HDF5) group is a container structure which can hold zero or more objects (i.e. datasets, (soft) links, external links and other groups) and have zero or more attributes (attached to it). Every object must be a member of at least one group, except the root group, which (as the sole exception) may not belong to any group.

Hierarchical Data Format (HDF)

The Hierarchical Data Format (HDF) is the name of a set of file formats and libraries designed to store and organize large amounts of numerical data. It is a versatile data model that can represent complex data objects and a wide variety of metadata. HDF is supported by The HDF Group, whose mission is to ensure continued development of HDF technologies and the continued accessibility of data currently stored in this file format.

Hyperslab

A hyperslab allows reading or writing a portion (subset) of a dataset (as opposed to its entirety). It can be a selection of logically contiguous collection of points in a dataspace, or it can be a regular pattern of points or blocks in a dataspace.
Message Passing Interface (MPI)

The Message Passing Interface (MPI) is a standardized means of exchanging messages between multiple computers running a parallel program across distributed memory. It was designed by a group of researchers from academia and industry to work on a wide variety of parallel computing architectures. MPI fosters the development of a parallel software industry, and encourages development of portable and scalable large-scale parallel applications.

Operation

An operation is an action that can be performed in HDFql such as to create an HDF5 file or read data from a dataset. Operations can be seen as the HDFql language itself. In HDFql, many operations exist and these are categorized into DATA DEFINITION LANGUAGE (DDL), DATA MANIPULATION LANGUAGE (DML), DATA QUERY LANGUAGE (DQL), DATA INTROSPECTION LANGUAGE (DIL) and MISCELLANEOUS.

Parallel HDF5 (PHDF5)

The Parallel HDF5 (PHDF5) is a parallel version of HDF5 which is the name of a set of file formats and libraries designed to store and organize large amounts of numerical data. It leverages MPI to effectively manipulate HDF5 files in parallel (e.g. write a dataset) across multiple computers. In HDFql, PHDF5 can be explicitly used through the CREATE FILE, USE FILE, INSERT and SELECT operations.

Post-processing

Post-processing options enable transforming results of a query according to the programmer’s needs such as ordering or truncating. These options are optional and may be used to create a (linear) pipeline to further process result sets returned by DATA QUERY LANGUAGE (DQL) and DATA INTROSPECTION LANGUAGE (DIL) operations.

Result set

A result set stores the results of data type HDFQL_TINYINT, HDFQL_UNSIGNED_TINYINT, HDFQL_SMALLINT, HDFQL_UNSIGNED_SMALLINT, HDFQL_INT, HDFQL_UNSIGNED_INT, HDFQL_BIGINT, HDFQL_UNSIGNED_BIGINT,
HDFQL_FLOAT, HDFQL_DOUBLE and HDFQLVARCHAR) returned by DATA QUERY LANGUAGE (DQL) and DATA INTROSPECTION LANGUAGE (DIL) operations.

Result subset

A result subset stores the results (of data type HDFQL_CHAR, HDFQL_VARTINYINT, HDFQL_UNSIGNED_VARTINYINT, HDFQL_VARSMALLINT, HDFQL_UNSIGNED_VARSMALLINT, HDFQL_VARINT, HDFQL_UNSIGNED_VARINT, HDFQL_VARBIGINT, HDFQL_UNSIGNED_VARBIGINT, HDFQL_VARFLOAT, HDFQL_VARDOUBLE and HDFQL_OPAQUE) returned by DATA QUERY LANGUAGE (DQL) and DATA INTROSPECTION LANGUAGE (DIL) operations.

Subcursor

A subcursor is meant to complement (i.e. help) cursors in the task of storing data of type HDFQL_CHAR, HDFQL_VARTINYINT, HDFQL_UNSIGNED_VARTINYINT, HDFQL_VARSMALLINT, HDFQL_UNSIGNED_VARSMALLINT, HDFQL_VARINT, HDFQL_UNSIGNED_VARINT, HDFQL_VARBIGINT, HDFQL_UNSIGNED_VARBIGINT, HDFQL_VARFLOAT, HDFQL_VARDOUBLE and HDFQL_OPAQUE. In practice, when a result set is of one of these data types, only the first element of the result set is stored in the cursor (as expected), while all elements of the result set are stored in the subcursor. In other words, each position of the cursor stores the first element of the result set and also points to a subcursor that in turn stores all the elements of the result set. Similar to cursors, HDFql subcursors offer several ways to traverse result subsets.