# **h5** - An Object Oriented Interface to HDF5

Mario Annau

May 8, 2016

**Abstract**

**h5** provides a flexible object–oriented interface to HDF5 and facilitates fast storage and retrieval of R objects to binary files in a language independent data–format. HDF5 files support partial and parallel I/O and can handle data sets of multiple terrabytes. Using HDF5 to serialize large time series objects can lead to significant I/O speedups compared to binary R files. Additional examples show how time series created from Matlab and Python (**PyTables**) can be read using **h5**.

## 1 Introduction

The Hierarchical Data Format 5 (HDF5) is a binary data format and API created by the **?** to better meet ever–increasing data storage demands of the scientific computing community. HDF5 files store homogeneous, multidimensional data sets organized in groups similar to the folder structure of a file system. As a self–describing file format HDF5 objects can be annotated with meta data using attributes. Compared to R's integrated binary format HDF5 has various advantages.

**Language Independence** HDF5 is implemented in C and includes APIs for a wide range of programming languages like e.g. C++, Fortran, Python and Matlab.

**Partial I/O** HDF5 files support direct access to parts of the file without first parsing the entire contents, thus can process data sets not fitting into memory.

**Optimization** Access performance to parts of the HDF5 file can be further tuned by specifying the memory layout. The defined chunks can be cached in memory to further improve access times for subsequent queries.

### 1.1 Related Work

The CRAN and Bioconductor repositories host three actively maintained packages supporting HDF5 files as shown in Table 1. However, only the **rhdf5** directly supports HDF5 files[1].

[1] **ncdf4** supports the NetCDF 4 format which specifies a layer on top of HDF5; **rgdal** needs to be compiled accordingly and is optimized for geospacial data.

| Package | Repository | First Release | Status |
|---|---|---|---|
| **h5r** | CRAN | 2011-10-23 | Archived |
| **ncdf4** | CRAN | 2010-02-24 | Active |
| **rgdal** | CRAN | 2003-11-24 | Active |
| **hdf5** | CRAN | 2000-02-02 | Archived |
| **rhdf5** | BioC | >10.5 Years | Active |

Table 1: Packages on CRAN and Bioconductor supporting the HDF5 file format.

Although **rhdf5** supports reading/writing of datasets it is lacking various features like direct exposure of HDF5 objects, subsetting data sets using operators or CRAN availability. **h5** fills that gap and provides an easy–to–use object oriented interface to HDF5. It uses the HDF5 **C++** API through **Rcpp** ( **?**) and represents objects like Files, Groups, Datasets and Attributes as S4-classes.

# 2 The h5 Package

## 2.1 Overview

All relevant objects exposed by the HDF5 C++ API are direcly represented in **h5** through S4 classes. The most important ones are H5File, H5Group, DataSet and Attribute.

**H5File** holds a reference to the binary HDF5 file.

**H5Group** can hold various HDF5 objects like `DataSets` and other `H5Groups`.

**DataSet** stores homogeneous data like vectors, matrices and arrays.

**Attribute** stores metadata about other HDF5 objects.

`H5Files` and `H5Groups` can be accessed using the subset operator and a path in a POSIX–like syntax. Applying the subset operator with integer indices on a `DataSet` returns/sets specified parts. `Attributes` are accessed using *h5attr*(). The following example shows how all these objects are created using **h5**. It creates a file in **a**ppend mode, creates a Group and Dataset holding a numeric vector and closes the file.

```
> f <- h5file("test.h5")
> f["testgroup/testset"] <- rnorm(100)
> testattr <- LETTERS[round(runif(100, max=26))]
> h5attr(f["testgroup/testset"], "testattr") <- testattr
> f["testgroup/testset"]
DataSet 'testset' (100)
type: numeric
chunksize: 100
maxdim: UNLIMITED
compression: H5Z_FILTER_DEFLATE
Attributes:
  A testattr
> h5close(f)
```

## 2.2 Data Types

Storing and retrieving data using **h5** requires a mapping of available data types from R to HDF5. Except for the `complex` and `raw` type all basic data types are mapped to HDF5. Although most mappings should be intuitive, the following decisions have been made:

1. 64Bit Integers are converted to double (numeric).
2. Logical values are mapped to an Enumeration Type to save space and support NA values
3. Variable Length (VLen) data types are stored and retrieved as lists of lists.

In addition to data type mappings the representation of `NA` values has been considered. In the case `numeric` types the ANSI/IEEE 754 Floating-Point Standard is applied which is used by R and HDF5. For `integer` the default minimum integer value is used[2]. Since `logical` values are stored as an Enumeration Type NA values are directly represented and retrieved through the type. For `character` we simply use the string `"NA"`.

## 2.3 Supported R Objects

**h5** currently supports storage and retrieval of homogeneous Datasets consisting of only one data type like vectors, matrices and arrays. **HDF5** also supports compound data types which could be used for `data.frame` objects. Support for compound types is planned in the near future.

[2]The minimum value equals to `-.Machine$integer.max-1` or -2147483648 for 32Bit integers.

# 3 Examples

This Section shows the functionality of **h5** with a focus on time series. It covers basic HDF5 dataset manipulations of a datasets and the serialization of **zoo** objects. Finally, we describe how to read time series created from Matlab and Python.

## 3.1 Manipulate Matrix

This example shows how HDF5 data sets can be created, altered, extended and removed[3]. The resulting matrix contains the replaced values in the second column and a third column as a result of *cbind*().

```
> f <- h5file("test.h5")
> f["testmat"] <- matrix(rep(1L, 6), nrow=3)
> f["testmat"][c(1, 3), 2] <- rep(2L, 2)
> cbind(f["testmat"], matrix(7:9, nrow=3))
DataSet 'testmat' (3 x 3)
type: integer
chunksize: 3 x 2
maxdim: UNLIMITED
compression: H5Z_FILTER_DEFLATE
> f["testmat"][]
     [,1] [,2] [,3]
[1,]    1    2    7
[2,]    1    1    8
[3,]    1    2    9
> h5unlink(f, "testmat")
[1] TRUE
> h5close(f)
```

## 3.2 Time Series and Chunking

This example shows how to store and retrieve **zoo** time series with **h5** and the speedup achieved through partial I/O and chunking. For an introduction to chunking see also **?**.

We generate a **zoo** object with three series covering one year and a constant interval of one second. The resulting object has 31.5M rows and 4 columns (including the datetime index). The chunk size is chosen so that each chunk covers one day for each series. Only the first day for one instrument (including the datetime index) is retrieved, thus there is no overhead through chunking. Compared to an approach using serialized R objects which needs to read all data elements into memory a speedup of 30 is achieved (see also the

[3]Note, that *h5unlink*() does not remove the actual data from the file. To reduce file size the command line tool h5repack is required.

benchmark results in Section 5.1). Note, that the chunksize has been finely tuned to match the access pattern and speedups are probably lower in real–world examples.

```
> library(zoo)
> datevec <- seq(as.POSIXct("2015-12-01"), as.POSIXct("2016-01-
01"), by = "secs")
> tsdat <- zoo(matrix(rnorm(length(datevec) * 3), ncol=3), or-
der.by=datevec)
> f <- h5file("test.h5", "a")
> f["testseries", chunksize=c(86400, 1)] <- cbind(index(tsdat),
coredata(tsdat))
> h5flush(f)
> tssub <- zoo(f["testseries"][1:86400, 2], or-
der.by=as.POSIXct(f["testseries"][1:86400, 1], origin="1970-01-
01"))
> identical(tssub, tsdat[1:86400, 1, drop=FALSE])
[1] TRUE
> h5close(f)
```

### 3.3 Read Times Series from Matlab

As of version 7.3 Matlab uses an HDF5 based format per default to store data to `.mat` files. Using **h5** we can therefore read any new mat–file. However, we need to transpose any multidimensional data since Matlab reads and writes data directly in column–major order (HDF5 is row–major)[4].

This small example shows how to read a time series data matrix created in Matlab using **h5**. First we need to create and save the matrix in Matlab. Finally, the data set is read and required conversions for the data matrix (transpose) and the time vector (subtraction) is applied.

```
tstart = datenum(2010, 1, 1);
tend = datenum(2016, 1, 1);
td = (tstart:tend)';
tseries = [td, randn(length(td), 3)];
save('ex-matlab.mat', 'tseries', '-v7.3');
```

[4]Since R also stores data in column–major–order **h5** transposes higher dimensional data (matrices, arrays) per default

```
> f <- h5file("ex-matlab.mat", "r")
> dates <- as.Date(f["tseries"][1, 1:3] - 719529)
> zoo(t(f["tseries"][2:4, 1:3]), order.by=dates)
2010-01-01 -0.1319692 -1.2185794 -1.5287349
2010-01-02 -0.4669825  0.1781066  0.4650538
2010-01-03  0.6076260 -0.2878577  0.4175950
```

### 3.4 Read Times Series from Python

This example shows how to read time series created from **PyTables** using **pandas**. The Python code below generates the dataset of interest.

```
from pandas import date_range, DataFrame
from numpy import random
t = date_range('2010-01-01', '2016-01-01', freq='D').date
df = DataFrame(random.standard_normal((len(t), 3)), index=t)
df.to_hdf("ex-pandas.h5", "testset")
```

Objects serialized using **pandas** and **Pytables** have a more complicated structure and dataset names can vary for different `DataFrames`. In this example we read the first three rows including the time index from `axis1` and actual data from `block0_values`.

```
> f <- h5file("ex-pandas.h5", "r")
> dates <- as.Date(f["testset/axis1"][1:3] - 719163, ori-
gin="1970-01-01")
> zoo(f["testset/block0_values"][1:3, ], order.by=dates)
2010-01-01  0.9302118  0.8508929 -1.1483052
2010-01-02 -0.1424808  0.2883631  0.2483735
2010-01-03 -0.7597725 -0.3645527  0.2428528
```

## 4 Conclusion

**h5** provides a flexible interface to handle HDF5 files. It directly exposes HDF5 objects and implements subset operators for easy data handling. In addition to R objects like vectors, matrices and arrays we also showed examples to store and retrieve time series objects. Depending on the use case and chunk size significant speedups can be achieved through partial I/O. Examples showed that **h5** can also be used to exchange data with other programming languages like Matlab and Python.

# 5 Appendix

## 5.1 Benchmark Example Time Series and Chunking

Code to produce benchmark results for the example in Section 3.2. To compare raw I/O performance the conversion to `POSIXct` has been omitted.

```
> library(zoo)
> library(microbenchmark)
> datevec <- seq(as.POSIXct("2015-12-01"), as.POSIXct("2016-01-
01"), by = "secs")
> tsdat <- zoo(matrix(rnorm(length(datevec) * 3), ncol=3), or-
der.by=datevec)
> f <- h5file("testbm.h5", "a")
> f["testseries", chunksize=c(86400, 1)] <- cbind(index(tsdat),
coredata(tsdat))
> h5close(f)
> save(tsdat, file="test.rda")
> readRda <- function() {
+   load("test.rda")
+   tsdat[1:86400, 1:2]
+ }
> readH5 <- function() {
+   f <- h5file("test.h5", "r")
+   f["testseries"][1:86400, 1:2]
+   h5close(f)
+ }
> bm <- microbenchmark(readRda(), readH5(), times = 10L)
> summary(bm)$median[1] / summary(bm)$median[2]
[1] 37.70228
```