HDF5 in R: Technical Documentation and Proposals

May 14, 2004

1 Overview

This document describes various aspects of the implementation and design philosophy of the *hdf5* package in R.

It also contains a section on existing bugs and on things that need to get implemented.

1.1 Description

HDF5 is a system for storing large data sets. The acronym stands for Hierarchical Data Format. HDF5 has a three–tiered system. There is the *file*, which contains *group*s and each group contains *datasets*. The datasets are the actual data the other two layers are organizational. The implementation in *rhdf5* follows this structure. There are classes for files, groups and datasets.

A simple example would be the following:

file Experiment1

group Patient1

dataset Mean, StdDev, NPixel

group Patient2

dataset Mean, StdDev, NPixel

For this experiment, each patient represents a different group within the experiment and for each patient we have a number of datasets that represent the different quantities obtained from an Affymetrix `CEL` file.

Each HDF5 dataset contains data that are stored in a common format. Thus, it is natural to think of HDF5 datasets as matrices or arrays but not as data.frames in R. One may be able to usefully think of groups or files as dataframes, at least in some situations. For example, one may want to `apply` a function to the Mean array for all patients in Experiment1.

The different HDF5 objects are represented by symbols in R, just as any R object is. The implementation of an HDF5 object is as a length one vector that contains an external pointer, `EXTPTR`.

This representation introduces a different evaluation model into R. R's standard evaluation model can be thought of as a pass–by–value model. That means that arguments to functions are copied and the function works on a copy of the value not on the original object. With `hdf5` objects this is not true. Functions will effectively get a pointer to the object.

In the next code chunk we see the creating of a file, `mad`. This corresponds to the physical creation of the file named `microarray.h5` in the current working directory. Within that *file* we can create groups and datasets. Groups can be deleted, but individual datasets cannot. The entire HDF5 file can be removed by using `unlink`.

```
> library(rhdf5)

Attaching package 'rhdf5':


        The following object(s) are masked from package:base :

         is.finite is.matrix

> mad <- hdf5.open("microarray.h5")
> class(mad)

[1] "hdf5.file" "hdf5"

> hdf5.group(mad, "chip1")
> class(mad$chip1)

[1] "hdf5.group" "hdf5"

> hdf5.dataset(mad$chip1, "raw", dim = c(534, 534))
> class(mad$chip1$raw)

[1] "hdf5.dataset" "hdf5"

> raw <- mad$chip1$raw
> for (i in 1:534) {
+     raw[i, 1:534] <- runif(534)
+ }
> hdf5.group.rm(mad, "chip1")
```

In the next segment we create a function that alters the value of one of its arguments. Then call the function with two different R objects, one an `hdf5` object and one a regular R object. Notice that the `hdf5` object has its value changed.

It is entirely the users responsibility to ensure the integrity of their data. The *rhdf5* cannot and does not check to see if data values have been altered.

```
> foo <- function(x) x[1, 1] <- 100
> raw[1, 1]

[1] 0.2363524

> foo(raw)
> raw[1, 1]

[1] 100

> x <- matrix(1:10, nc = 2)
> x[1, 1]

[1] 1

> foo(x)
> x[1, 1]

[1] 1
```

There seem to be some options available:

- Copy any object if it is altered. This will be fairly wasteful, and could easily fill up a users hard disk. Perhaps we could use weak references to remove files that are no longer referred to.
- Supply a `duplicate` function and allow users to copy the files that they want to copy.

1.1.1 Size Matters

A fairly basic design strategy is to keep the rather large data sets that are stored as HDF5 objects from ever becoming R objects. That is, we do not want to have them stored in internal R memory unless we have to.

This desire has facilitated some changes to R and caused some design features in the *hdf5* package.

2 HDF5 Dataset Dataspace Overview

An HDF5 dataset is an array of data that are all in a common format. This format can be quite general, but for now we consider only real and integer formats. The data are stored in a particular format on disk and that format is stored in the file header. It cannot ever change. However, translators can be put in place so that when data are read from disk they are automatically translated into a different, specified format. Unfortunately there seem to be no integer to real translators that are built in – we may need to build some.

Data are selected from a dataset using a dataspace. A dataspace must be the same size and have the same dimensions as the dataset that is being read from or written to. The dataspace is conceptually a set of bits indicating whether an element of the dataset is selected or not. Only those elements of the dataset that have been selected in the dataspace are manipulated.

Thus, to select a specific subset of a number of datasets one can use a single dataspace. Applying it to each dataset in turn yields the requisite subset, these can be processed and the necessary values stored. This appears to form the basis of an HDF5 apply type function.

3 Technical Things

The following are classes introduced by *HDF5*.

- `hdf5`, all other HDF5 objects inherit from `hdf5`.
- `hdf5.file`
- `hdf5.group`
- `hdf5.dataset`

Virtually none of the HDF5 functions are methods. This will be improved over time but for now there seems to be little benefit and a lot of work.

Some functionality: Suppose that

- The $ operator works on objects with class `hdf5.file` or with class `hdf5.group`.

4 What is missing

HDF5 does not currently seem to support the deletion of individual datasets or groups, or files. Files can easily be done from the system but we would like to be able to remove datasets. However, HDF5 does not support the removal of individual datasets. This is on their TODO list and we will attempt to take advantage of this functionality when it becomes available.

Do we make HDF5 objects copy on alter? Do we allow a user level interface?

Much of the current interface is to allow HDF5 objects to be treated as if they were R vectors or matrices. However, there are design differences between R and HDF5. It might be useful to simply have an R interface to some basic HDF5 functionality. That is, to write some interface routines that simply operate on HDF5 files as they are without having R interfere except at the external pointer stage.

5 To Do/ WIP

The C–converter and registration routines need to be checked more carefully.

Need to sort out whether, how we are going to have HDF5 files that do not correspond to R types (make sure that we attach R types to the HDF5 files – and that we start to use them) –do we need typeof to become generic?

Need to stop and think about how this relates to relational database things, do they need more types? Do they just have dataframes?

Need to figure out how to use R_alloc to, more or less get char[n][m] type objects to stick into the hdf5 string datasets;

Need to figure out how to do STRSXPs.