RFC: A New Tool to Test Hyperslabs in HDF5
September 2, 2004

1. Introduction

There is a need for improved testing for HDF5, hyperslabs with different combinations of chunking, filters, fill values, datatypes, and complex hyperslabs, etc.. We also need to try to use more realistic datasets than the default regression tests. The parameter space is extremely large (probably infinite), so we need a flexible and extensible method for testing many combinations.

This document describes a new tool to implement some of these tests.

The tool will be a stand alone program, it is not intended to run as part of the regular test suite. The tool should be portable to all supported platforms.

This tool is intended to be run sequentially. It does not test parallel features and might not be useable in parallel environments.

2. Proposed Test Tool

Tool Name: HSTest

Usage:

HSTest
HSTest -I params_file

Description:

HSTest is a test suite being developed for testing of hyperslabs on realistic datasets, with different hyperslabs and chunking. The program should be easily configurable to test different combinations easily. To this end the program takes an input from a parameters file. The user can provide parameters like filters, fill values, data sets and so on through this parameters file. This file is written in a subset of XML language which is easily extensible and easy to use. The tool should generate extensive logs/traces so that it is easy to determine where the error occurred.

3. Details

3.1. Input Parameters

The input of the testing kit would come from the paramters file. As a default the program would assume that the parameters are in a *params.xml* file. The user could also provide the input parameters file through the command line.

The XML grammar of the parameters file is as follows:

```
<!DOCTYPE Parameters [
  <!ELEMENT Params(Rank, DataDims, ChunkFactor, ChunkDims, HSelFactor, Filters,
File, FillValues, WDir)>
  <!ELEMENT DataDims(No+)>
  <!ELEMENT ChunkDims(No+)>
  <!ELEMENT File(Type, Name)>
  <!ELEMENT ChunkFactor(#PCDATA)>
  <!ELEMENT HSelFactor(#PCDATA)>
  <!ELEMENT No(#PCDATA)>
  <!ELEMENT Type(#PCDATA)>
  <!ELEMENT Name(#PCDATA)>
  <!ELEMENT Filters(#PCDATA)>
  <!ELEMENT FillValues(#PCDATA)>
  <!ELEMENT WDir(#PCDATA)>
]>
```

3.2. Modules Required

The tool will be organized in three modules, which are called from a main program. These modules will be designed to be reusable in other tools.

Input Parser: This module would parse the input from the parameters file and convert it into a data structure that is global to all the modules. This would allow all the modules to access the different parameter settings. This data structure has to be easily extensible so that future alterations to requirements would not need a completely new set up. Also the variables in the data structure should correspond properly with the parameters in the parameter file.

DataGenerator: An algorithm that would generate data in such a way that we could reproduce the result given the coordinates of the data, is required to check the accuracy of the results. This should be a separate module so that a user who wishes to generate data a particular way can just replace this module and not touch the other modules.

Requirements for this module are:

- Should have support for arrays with unlimited dimensions and partially initialized datasets.
- Should support different datatypes. Initially, this may be limited to atomic numeric types.
- Should be able to check correctness of block being read back.

Possible algorithms:

- Use the following to generate data.
  Value[i][j] = i + j * MAX_I
  This would need to be generalized for multidimensional arrays
- Have a random number generator that generates a set of i * j random numbers where i is the length of one dimension and j of the other. These random numbers would also be stored in an array of length i*j so that they could matched up against the data read. This approach is possible only when the i * j is a reasonable number. If i * j is

too large, we may need to write that dataset to a file and read it again. This might end up to be time consuming.

HyperSlab Generator: This module randomly generates a set of hyperslabs which would be tested for proper I/O. Currently onle hyperlsabs generated from this module will be used for testing. In future we plan to allow users to specify particular hyperslabs so as to test the nooks and crannies of the system.

Requirements for this module are:

- Should be able to generate strides and blocks.
- Should be able to generate non-rectangular hyperslabs, i.e., unions, etc., of rectangular selections.
- Should be able to do scatter – gather, i.e., read into different shape array.

Main Module: This module coordinates between the various modules and tests the performance of the libraries on the various input parameters.

Requirements for this module are:

- Should detect and log errors, so that the erroneous library can be narrowed down to as much as possible.
- Have options to use filters and compression.
- Should have options to use fill values.

4. Output

The tests should provide output that clearly indicates PASS/FAIL. In the case where a failure is detected, the output and/or logging must give sufficient information to reconstruct the failed case. It may be useful to have a complete trace of all the tests run, to help reconstruct a problem.

The details of the output and logging are TBD.

5. Future Work

In the future, additional features might be added, including:

- Allow for the user to specify exactly which file to be read.
- Allow for more data types to be tested.
- Allow for more options for different chunking.
- Allow different file drivers.