

# SLh5: The S-Lang HDF5 Module Reference Manual, Version 0.3.10

---

Michael S. Noble, [mnoble@space.mit.edu](mailto:mnoble@space.mit.edu)

Aug 13, 2009



# Preface

SLh5 is a S-Lang module for the HDF5 file format and I/O library.

Copyright (C) 2006 Massachusetts Institute of Technology  
Author: Michael S. Noble <mmoble@space.mit.edu>

This software was developed with SLIRP at the MIT Kavli Institute for Astrophysics; it was funded in part by NASA and the Smithsonian Institution, through the AISRP grant NNG05GC23G and Smithsonian Astrophysical Observatory contract SV3-73016.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in the supporting documentation, and that the name of the Massachusetts Institute of Technology not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. The Massachusetts Institute of Technology makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

THE MASSACHUSETTS INSTITUTE OF TECHNOLOGY DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE MASSACHUSETTS INSTITUTE OF TECHNOLOGY BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Examples . . . . .	2
<b>2</b>	<b>High Level API</b>	<b>7</b>
2.1	<code>h5_new</code> . . . . .	7
2.2	<code>h5_open</code> . . . . .	7
2.3	<code>h5_close</code> . . . . .	8
2.4	<code>h5_read</code> . . . . .	9
2.5	<code>h5_write</code> . . . . .	9
2.6	<code>h5_list</code> . . . . .	10
2.7	<code>read_amr</code> . . . . .	10



# Chapter 1

## Introduction

SLh5 is a S-Lang module for the HDF5 file format and software library. HDF5 is widely used in large scale science and engineering projects – such as the FLASH simulator of thermonuclear explosions in stars ( <http://www.flash.uchicago.edu/website/home/> ) – because it combines data and compiler portability with very high performance, scalability to terabyte-size datasets, and support for parallel I/O. The S-Lang interpreter is part of the widely-used, open-source library of the same name, and provides a scripting environment well-suited for scientific and engineering tasks, due to the powerful, compact, fast, and robust multidimensional numerical capabilities that are native to the language.

SLh5 provides a high-level interface to HDF5, supporting the easy creation of S-Lang arrays from HDF5 files or vice versa. The entire HDF5 api is presented in terms of 6 simple functions:

```
h5_new    - create one or more new HDF5 files
h5_open   - open one or more HDF5 files/groups/datasets/attributes
h5_close  - close one or more HDF5 files

h5_read   - read one or more datasets or attributes
h5_write  - write a dataset or attribute
h5_list   - browse an HDF5 container (analogous to h5dump utility)
```

although users will typically need only the latter two or three. Much of the low-level HDF5 library may also be called directly from S-Lang, though there should be little need for user-level scripts to do so (and the low-level wrappers are undocumented here). There is also no need to explicitly close files, or opened datasets, attributes, groups, etc, as SLh5 will automatically do so when the variable referencing them goes out of scope. Moreover, and with the exception of h5\_write, the entire top-level SLh5 interface is vectorized; this allows multiple files to be opened or created, or multiple datasets or attributes to be read, in a single call.

With SLh5 users may thus interact with HDF5 files in scriptable analysis environments such as ISIS ( ), with substantially smaller and cleaner code than is possible in other analysis platforms, e.g. IDL (tm), and with potentially considerable I/O performance advantages.

The present implementation is incomplete, with the major gaps being that partial I/O is not yet supported, and that compound types – i.e. structures – may be read but not written. S-Lang version 2.0.6 or later is required.

## 1.1 Examples

SLh5 can be loaded at runtime into any suitably configured application which embeds the S-Lang interpreter, using either

```
require("h5");
```

or

```
() = evalfile("h5");
```

Now, given the S-Lang variables

```
x = [-50:50];  
y = x^2;  
z = x^3;  
attr = "StringAttribute";
```

one might write them to an HDF5 file as 3 datasets and an attribute via

```
file = "test.h5";  
file_id = h5_new(file);  
h5_write(file_id, x, "x");  
h5_write(file_id, y, "y");  
h5_write(file_id, z, "z");  
dataset_id = H5Dopen(file_id, "z");  
h5_write(dataset_id, attr, "attr", H5I_ATTR);  
h5_close(file_id);
```

They may be read back in various manners, such as

```
x2 = h5_read(file, "x");
```

or

```
file_id = h5_open(file);  
y2 = h5_read(file_id, "y");
```

and

```
attr2 = h5_read("test.h5:/z", "attr");
```

This last statement opens the `test.h5` file and the `z` dataset, using an identifier for the latter as the location from which the `attr` attribute will be read. This is equivalent to

```
attr2 = h5_read("test.h5", "/z/attr");
```

both of which are convenient alternatives to either

```
f = h5_open("test.h5");
d = H5Dopen(f, "z");
attr2 = h5_read(d, "attr");
```

or

```
d = h5_open("test.h5:/z");
attr2 = h5_read(d, "attr");
```

The correctness of the above HDF5 I/O calls may be easily verified through

```
any( x != x2 );
any( y != y2 );
attr == attr2;
```

As noted earlier, SLh5 is vectorized, which allows multiple operations to be performed with a single call. For example,

```
just_two = h5_read("test.h5", ["y","x"]);
```

returns the x and y dataset arrays, in reverse write order, while both

```
three_1 = h5_read("test.h5", ["x","y","z"]);
```

and

```
three_2 = h5_read("test.h5", "*");
```

read all three dataset arrays at once, the second using wildcard shorthand notation. Each of these vectored calls returns an array of arrays, the results of which might be verified via

```
any( three_1[0] != three_2[0] );
any( y != three_1[1] );
any(three_1[2] != three_2[2] or three_2[2] != z);
```

Now let's look at some scientific examples using the ISIS ( ) astrophysical modeling and analysis system (and assuming `Isis_Append_Semicolon = 1`). Consider

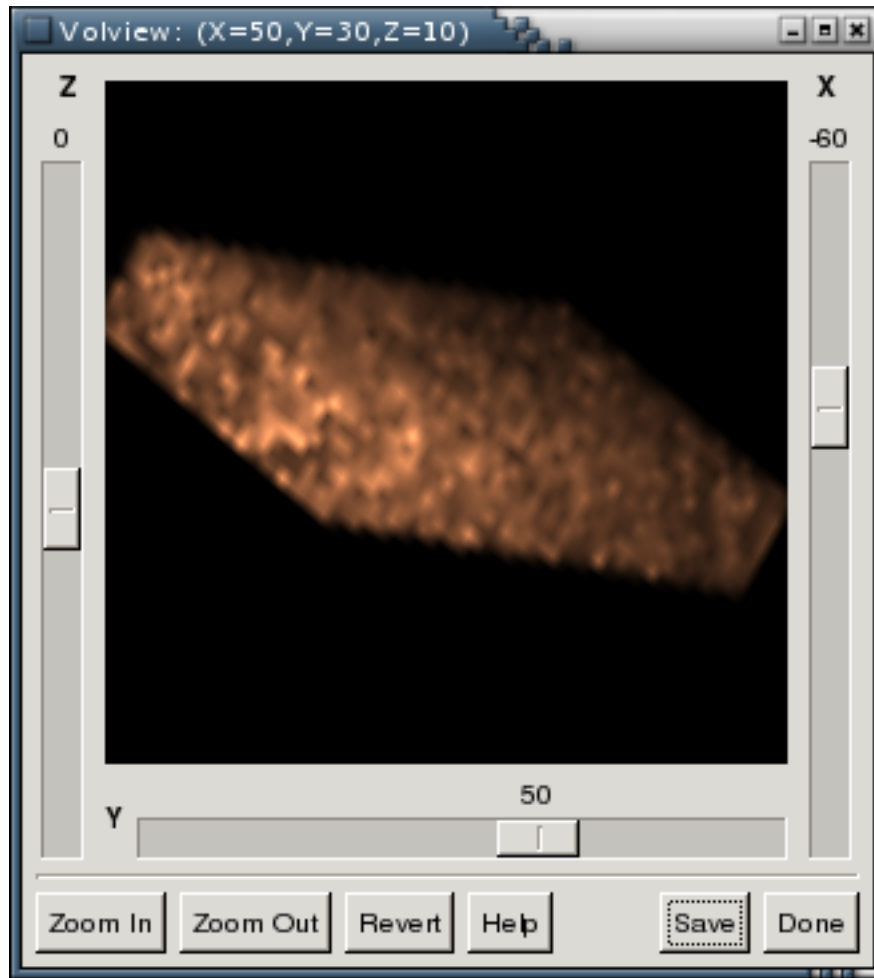
```
isis> slab = urand(10, 30, 50)
```

which generates a 3D grid of uniformly-distributed, double-precision random numbers

```
isis> slab
Double_Type [10,30,50]
```

We can use `volview`, our prototype 3D volume visualizer, to get an idea of what this slab looks like:

```
isis> require("volview")
isis> volview(slab)
```



The slab may be saved as an HDF5 file via

```
isis> h5_write("slab.h5", slab)
```

and recreated from this file via

```
isis> slab_copy = h5_read("slab.h5")
```

Verifying that the original and copy are equivalent is again as easy as

```
isis> any(slab != slab_copy)
0
```

As another example, consider the S-Lang function

```
define cone()
```

```
{
  variable r, h, nsteps;
  switch(_NARGS)
  { case 3: (r, h, nsteps) = (); }
  { usage("cone(radius, height, number_of_steps)"); }

  r = r * 1.0; h = h * 1.0;
  variable hsteps = [ 0.0 : h : h / nsteps ];
  variable rsteps = (r*hsteps/h)^2;
  variable xysteps = [-r: r: 2*r/nsteps]^2, xsq = xysteps;
  nsteps = length(xysteps);
  variable c = Double_Type[length(rsteps), nsteps, nsteps];

  variable i = 0;
  foreach(rsteps) {
    variable rsq = ();
    variable j = 0;
    foreach (xysteps) {
      variable ysq = ();
      c[i, j, *] = (ysq + xsq <= rsq);
      j++;
    }
    i++;
  }
  return c;
}
```

which defines a cone in 3D space, with a given size and discretization (number of subdivisions along the Z axis). The command

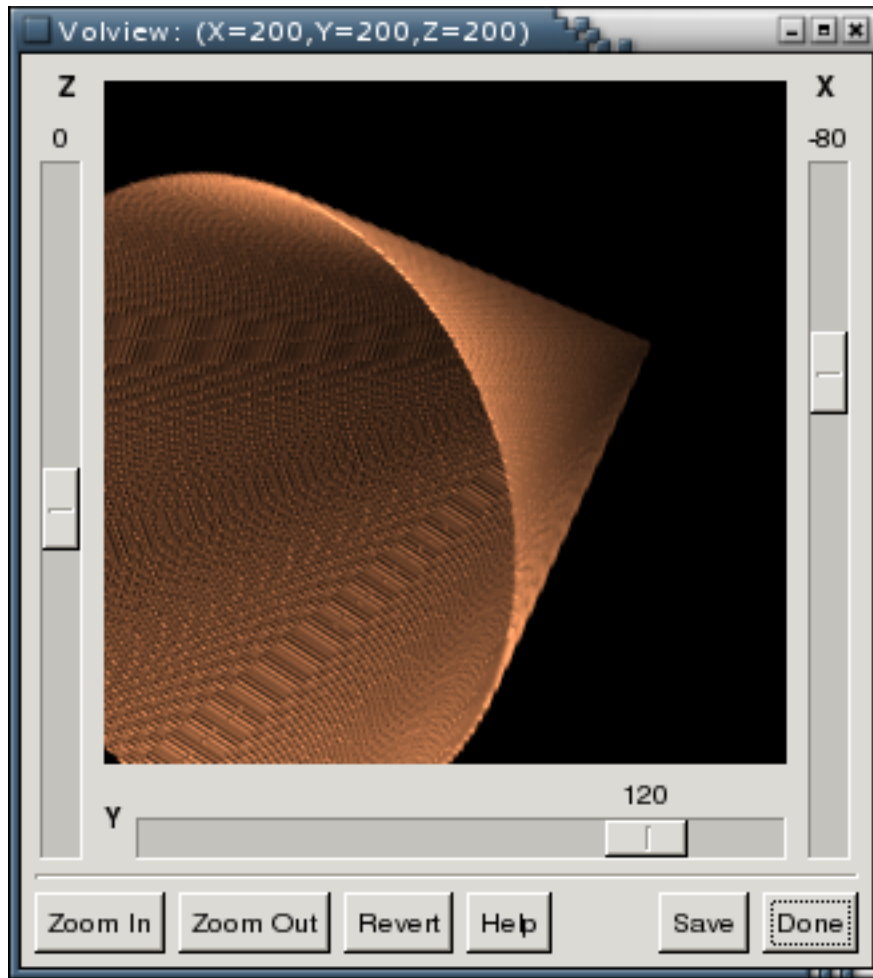
```
isis> c = cone(5, 10, 200)
```

creates a 200x200x200 cube

```
isis> c
Double_Type[200,200,200]
```

where only the elements corresponding to the cone surface have non-zero values. This can again be visualized with `volview`

```
isis> volview(c)
```



(where we've applied a copper colormap, and rotated the volume to show both the interior and exterior surfaces) and then written, restored, and verified as before:

```
isis> h5_write("cone.h5", c)
isis> cone_copy = h5_read("cone.h5")
isis> any(c != cone_copy)
0
```

Additional examples may be found in the regression test subdirectory (`./tests`) of the SLh5 distribution.

## Chapter 2

# High Level API

### 2.1 h5\_new

#### Synopsis

Create a new HDF5 file

#### Usage

```
file_id = h5_new(name, [,access_flags])
```

#### Description

This function attempts to create & open the named file and return a corresponding handle/file id. The access flags may be either `H5F_ACC_EXCL` or `H5F_ACC_TRUNC`: the first causes an error to be thrown if the file already exists, while the second causes any existing file of the same name to be erased first (the default).

#### Notes

This function is vectorized on its first parameter: if an array of file names is passed then an array of file identifiers will be returned.

Unlike with the HDF5 C library, files do not need to be explicitly closed with `SLh5`; instead, they will be closed automatically when the returned handle goes out of scope.

#### See Also

`h5_open`, `h5_close`, `h5_read`

### 2.2 h5\_open

#### Synopsis

Open an HDF5 file, and optionally group, dataset, or attribute

#### Usage

```
hid_t = h5_open(name, [,access_flags])
```

### Description

This function attempts to open the HDF5 file specified by name and return a corresponding handle/file id. The access flags may be either `H5F_ACC_RDWR` (read/write) or `H5F_ACC_RDONLY` (read-only, and the default if omitted).

The name parameter may optionally contain a path to an object with the file being opened. For example, `d = h5_open("test.h5:/G/D")` opens the file `test.h5`, the group `G`, and the dataset `D`, assigning an identifier for the latter to the variable `d`.

### Notes

This function is vectorized on its first parameter: if an array of file names is passed then an array of file identifiers will be returned.

Unlike with the HDF5 C library, files do not need to be explicitly closed with `SLh5`; instead, they will be closed automatically when the returned handle goes out of scope.

### See Also

`h5_read`, `h5_close`

## 2.3 `h5_close`

### Synopsis

Close an HDF5 file

### Usage

```
h5_close(file_id)
```

### Description

This function will close the HDF5 file associated with the specified file id.

### Notes

This function is vectorized: if an array of file ids is passed then the file associated with each element will be closed.

Unlike with the HDF5 C library, files do not need to be explicitly closed with `SLh5`; instead, they will be closed automatically when the returned handle goes out of scope.

### See Also

`h5_open`

## 2.4 h5\_read

### Synopsis

Read an HDF5 dataset into a S-Lang array

### Usage

```
array = h5_read(from [, dataset_or_attribute_name])
```

### Description

This function will read an HDF5 dataset or attribute into a S-Lang array of the appropriate type. The from parameter may be any string acceptable to h5\_open() or any valid HDF5 identifier (such as might be returned by h5\_open, H5Dopen, or other open calls). If no dataset or attribute name is supplied the routine will attempt to open the first dataset it can find within the root group hierarchy.

### Notes

This function is vectorized on either its first or second parameter: if an array is passed for either then an array (e.g. of arrays) of commensurate size will be returned. A dataset/attribute name of "\*" may be used as shorthand for "everything," in which case a vectored value will be returned, containing all datasets or attributes in the given container.

Partial I/O is not yet supported.

### See Also

h5\_write

## 2.5 h5\_write

### Synopsis

Create an HDF5 file, dataset, or attribute from a S-Lang object

### Usage

```
h5_write(to, data [,name [, kind ]])
```

### Description

This function attempts to create a new dataset or attribute in the HDF5 file specified by the to parameter, which may either a file name or an HDF5 identifier associated with an open file. Named files will be created if they don't already exist, or overwritten if they do. The data parameter is typically an array of values. If a name is not specified then if a dataset is created it will have the same name as the file, minus any extension. If kind is not specified a dataset object will be written, otherwise H5I\_DATASET or H5I\_ATTR may be specified to explicitly request that either a dataset or attribute be created.

### Notes

Writing compound types is currently unsupported.

### See Also

h5\_new

## 2.6 h5\_list

### Synopsis

List named members of an HDF5 file, group, or dataset

### Usage

```
StringType[] = h5_list(from [,flags])
```

### Description

This function may be used to browse the contents of an HDF5 container, and is loosely analogous to the `h5dump` utility. The `from` parameter may either be a file name or an identifier associated with an opened HDF5 object. The `flags` option may be any OR-ed combination of `H5_LIST_OBJECTS`, `H5_LIST_ATTRIBUTES`, or `H5_LIST_RECURSE`; a value of `H5_LIST_OBJECTS | H5_LIST_ATTRIBUTES` will be assigned to `flags` when it is omitted.

### Notes

Recursive listing has not been implemented yet; only one level of the container object hierarchy may be inspected at a time.

### See Also

`h5_open`

## 2.7 read\_amr

### Synopsis

Load FLASH HDF5 adaptive mesh refinement simulation

### Usage

```
(tree, params, unknowns) = read_amr(file [, variable_name])
```

### Description

This function attempts to read FLASH simulation data, in the form of an adaptive mesh, from the given file. If `variable_name` is specified only a single simulation unknown is loaded, otherwise all unknowns will be loaded. Upon success three S-Lang structures are returned.

### Notes

This is a S-Lang translation of `read_amr.pro` contained in the FLASH IDL(tm) package (FIDLR3). Consult the FLASH documentation for details.

### See Also

`SLh5`, `h5_open`, `h5_read`