# Simple Data Format – A Platform Independent Data Format that works in Fortran, C, and IDL

http://solarmuri.ssl.berkeley.edu/~fisher/public/software/SDF

George H. Fisher

Space Sciences Lab

UC Berkeley

## Abstract

Saving the binary output from large-scale numerical simulations, especially large multi-dimensional arrays, in a form that can be easily post-processed and analyzed on many different computing platforms, has been a challenge for many years.

Here I describe one possible solution for this problem, the “Simple Data Format” (SDF) file format, and an implementation of this format that works in Fortran, C, and IDL. The file format and software has been demonstrated to work for all 3 languages in Linux on x86 and x86_64 architectures, in Windows XP (x86), and on large-endian platforms such as SUN-Solaris, and the Mac G4, G5 processors running the OSX operating system. SDF binary data files can be passed from one architecture to another transparently.

SDF was designed specifically to allow a clean replacement of unformatted and direct-access I/O in Fortran 77 or Fortran 90/95 programs with simple subroutine calls, and to accommodate “large” (over 2GB) file sizes on all platforms. Specific examples of how to read, write, and edit SDF files in all 3 languages are shown.

## Motivation (or - why did I waste my valuable time as a scientist writing this software?)

This project was born from over 20 years of accumulated frustration with difficulties in analyzing results from numerical calculations. While tools such as IDL and OpenDX have come a long way in improving our ability to visualize simulation results, it is still very difficult to produce binary output from numerical calculations that these tools can use in a way that is efficient, transparent, and independent of OS platform and/or computer hardware. Excellent platform-independent data formats do exist (FITS and HDF5, for example) but their user interface is too cumbersome for rapid analysis, debugging, and code development purposes. Further, many of these tools have deficiencies in some OS environments, while they work fine in others. This lack of uniform capability across languages, hardware, and operating systems greatly diminishes the generality and usefulness of most existing platform independent data formats.

And, I’m anal.

# The Problem:

**How does one get the binary output from a large-scale numerical simulation (written e.g. in Fortran) from this Beowulf cluster so that it can be easily analyzed from any computer architecture, running any operating system?**

## Possible Solutions

| **Approach** | **Advantages** | **Disadvantages** |
|---|---|---|
| Formatted I/O | Platform independent | Loss of precision, very inefficient |
| Unformatted binary I/O | Efficient and simple | Endian, platform, and language-dependent |
| Platform independent file formats (HDF, HDF5, NetCDF, FITS) | Platform independent | Difficult to install, complicated to use, not available for all languages and platforms |

Platform independent file formats are the most logical solution, yet existing formats can be hard to use and have other deficiencies…

## Summary of several platform independent file formats:

| Format Name | Advantages | Disadvantages |
|---|---|---|
| IDL Save file | All platforms on which IDL runs are supported | Proprietary, no support for C/Fortran |
| FITS format | Heritage in solar physics, wide software base, extensions, many platforms, large file support recently added | IDL versions not large file capable, C/Fortran code not large file capable in MS Windows. |
| NetCDF | Wide use in atmospheric sciences, large file support recently added | Cumbersome user interface |
| HDF | In wide use in scientific and engineering communities | Cumbersome user interface, no support for large files |
| HDF5 | In wide use in scientific and engineering communities | Cumbersome user interface, no Fortran 77 support, IDL support only recently |

## Design a file format and I/O software that is easy to use and very portable: Simple Data Format (SDF).

1. The SDF file structure is open, simple, and easy to understand.
2. Since IDL is the dominant analysis language used in Solar Physics, SDF was implemented in C and Fortran (on the simulation side) and in IDL (on the analysis side).
3. Both the C and Fortran user-callable functions (i.e. the “interface”) are written entirely in C to simplify the compilation and installation of the library.  The code is designed to compile on a wide variety of different systems.
4. The Fortran interface works in legacy Fortran 77 codes, as well as with Fortran 95.
5. The C, Fortran, and IDL function calls for reading and writing data can be done with only 1 statement, `sdf_read` (or `sdf_read_f77`), and `sdf_write (or sdf_write_f77)`..
6. The SDF functions called from all 3 languages support the reading and writing of “large” files greater than the 2GB limit.

## Design a file format and I/O software that is easy to use and very portable: Simple Data Format (SDF) -- (continued).

7. The C/Fortran callable version supports a multi-dimensional transpose and index reversal function for large arrays, `sdf_transpose`, which does the transpose *in-place* to minimize the impact on memory.
8. The C/Fortran and IDL interfaces allow one to edit “datasets” (arrays or variables) anywhere in an SDF File with a single function call, including deleting, inserting, or replacing existing datasets.
9. The IDL interface contains functions to write and read datasets from an IDL session directly into an SDF file, `sdf_write_all`, and `sdf_read_all`, very similar to the ‘SAVE’ and ‘RESTORE’ capabilites in IDL.
10. The IDL interface has a function, `sdf_read_arr`, that can retrieve a series of saved datasets and re-construct a “time-series” describing the evolution of a given scalar or multi-dimensional array.

# How does one get and install SDF?

- Download the most recent tarball from http://solarmuri.ssl.berkeley.edu/~fisher/public/software/SDF

- Unpack the tarball (e.g. tar zxvf sdf-0.74.tgz ), get into the top level directory created when unpacking the tarball.

- To create the Fortran/C callable library, type “make”. If you want to install the library and include file into /usr/local/lib and /usr/local/include, become root and then type “make install”. Typing “make all” compiles all of the test programs. You will need to edit the Makefile to make sure that the choice of C and Fortran compilers matches what you have on your system before you type “make all”.

- To install the IDL version of the SDF procedures, copy the contents of the idl folder from the tarball into some location that is in your IDL path. That should be all that is necessary.

- Much more detail on installation procedure details, and how to link to the library is in the file INSTALL.txt in the distribution; details on usage of all the SDF functions are given in the file SDF_USAGE_NOTES.txt, and the motivation/synopsis for SDF is given in the file SDF_MANIFESTO.txt . And when I have time, I’m happy to answer emails and phone calls.

# On what platforms has SDF been tested?

| | Linux x86 (32-bit) | Linux x86_64 (64-bit) | Windows XP (32-bit) | OSX G4 (32-bit MAC) | OSX G5 (64-bit MAC) | SUN Solaris (64-bit) | SGI Altix 64-bit Itanium2 | SGI Origin 64-bit IPxx series | IBM SP4,SP5 | Cray XD1 64-bit Opteron 275 |
|---|---|---|---|---|---|---|---|---|---|---|
| C | gcc, icc | gcc, pgcc, icc | gcc/mingw, msvc | gcc | gcc | gcc, cc | icc | cc (SGI) | xlC_r | pgcc |
| Fortran | g77, g95, gfortran, ifort, lf95 | g77, g95, gfortran, ifort, pgf77, pgf90 | g77, g95, gfortran | g77, g95 | g77, g95 | f77, f95 | ifort | f77(SGI) f90(SGI) | xlf_r xlff95_r | pgf77 pgf90 |
| IDL | yes | yes | yes | yes | yes | yes | ? | ? | ? | ? |

Note that the tests performed included the ability to read and write large (>2GB) files in all 3 languages. The SDF distribution includes a set of test programs for Fortran and C.

# What is the Structure of an SDF file?

| |
|---|
| Bytes 0-10: the string “SDF format” including null terminator |
| Bytes 11-18: 64-bit integer, “hdrpos” – location of next available byte in header |
| Bytes 19-26: 64-bit integer, “datapos” – location of next available byte in data area.  This is also equal to the file size. |
| Bytes 27-30: 32-bit integer, “ndatasets” – number of datasets currently in file |
| Bytes 31-38: 64-bit integer, “hdrsize” – The current size of the header (initially set to HINITSZ, and incremented in blocks of HINITSZ as necessary.  Default value of HINITSZ is 2000, but can be set by user.) |
| Bytes 39 – 19+hdrsize-1:  The header area, containing identifier strings for all the datasets on the file. |
| Bytes 19+hdrsize -- datapos:  The data area, where all the datasets are stored. |

All the above integers, and all the data in the file, are stored in large-endian byte order.  The SDF software converts to and from small-endian when necessary.

# What kind of data can be written into an SDF file?

**Floating point** datasets, which includes both single precision and double precision.  These data can be of any length, ranging from a single variable to large arrays with an arbitrary number of dimensions (`datatype = ‘f’`).

**Integer** datasets, ranging from 2-byte or short integers to 8-byte or long long integers.  These data can range from a single variable to large arrays with an arbitrary number of dimensions (`datatype = ‘i’`).

**Complex** number datasets, single or double precision, and of arbitary length and dimensionality (`datatype = ‘c’`).

**Byte** array datasets, ranging from a single byte to large arrays with an arbitrary number of dimensions.  These can include strings (but null terminators are treated the same as any another byte) (`datatype = ‘b’`).

The reading and writing of structures is currently not supported, though individual structure members which are arrays of the above types can be used.

# How is the “metadata” for each dataset stored in the header of an SDF file?

The amount of metadata for each dataset is kept to an absolute minimum. Each dataset in the SDF file is described by a single linefeed terminated string stored in the header part of the file. Each string consists of a series of tokens, separated by blanks. The tokens correspond to the following quantities:

`iorder` – the order of this dataset in the file (starts from 0)
`label` – a short string (no blanks) denoting the dataset – (e.g. a variable name)
`datatype` – a single character denoting the type of data, i.e. ‘f’, ‘i’, ‘c’, or ‘b’
`nbpw` – the number of bytes per word
`ndim` – the number of dimensions of the dataset or array
`dims` – the ndim values of the array dimensions

For example, the 3-dimensional (100 x 200 x 300) double precision array “rho” that is the 4th dataset in the SDF file would have an identifier string in the header that looks like this: `3 rho f 8 3 100 200 300`

The identifier strings are stored sequentially starting at byte 39.

## How does one use SDF to write an array from a Fortran or C program and then read it into IDL?

**Fortran:**

```
call sdf_write_f77(fname,lb,dt,nb,nd,dims,data)
```

fname: string containing sdf file name
lb: short string identifier
dt: datatype (single character, ‘f’, ‘i’, ‘c’ or ‘b’)
nb: no. bytes per word
nd: no. of dimensions
dims: 1-d array of dimensions
data: the array of data
ord: order of dataset in file fname
id: structure containing ord,lb,dt,nb,nd,dims

**IDL:**

```
sdf_read,fname,ord,lb,data
```

**C:**

```
id=sdf_create_id(0,lb,dt,nb,nd,dims);
sdf_write(fname,id,data);
```

## How does one write an array from IDL and then read that array into a C or Fortran program?

**Fortran:**

```
call sdf_read_f77(fname,ord,lb,dt,nb,nd,dims,data)
```

**IDL:**

```
sdf_write,fname,lb,data
```

fname: string containing sdf file name
ord: order of dataset in file fname
lb: short string with identifier
dt: datatype (single character, ‘f’, ‘i’, ‘c’ or ‘b’)
nb: no. bytes per word
nd: no. of dimensions
dims: 1-d array of dimensions
data: the array of data
id: structure containing lb, dt, nb, nd, dims

**C:**

```
id=sdf_read(fname,ord,(void **)&data);
```

# Navigating the datasets in an SDF file

- The SDF software uses the order of datasets in the file as the primary means to identify them.
- The SDF software contains several functions to aid in navigating the contents of an SDF file. `sdf_query` will provide a brief summary of all the datasets in a file. `sdf_details` will provide all of the details for a single selected dataset. `sdf_labmatch` returns the dataset orders for all datasets whose "label" matches a user-specified string.
- The C/Fortran version of SDF includes a simple command-line program, `sdf_browse`, which allows the user to interactively examine the various datasets in a file, and a primitive capability to print out a limited range of values.

## The SDF software contains the ability to edit (insert, delete, & replace) datasets anywhere in the file

- To delete datasets anywhere in a file, one can use the functions `sdf_delete` (IDL and C) and `sdf_delete_f77` (Fortran).
- To insert a new dataset anywhere in an SDF file, the functions `sdf_insert` (IDL and C) and `sdf_insert_f77` (Fortran) will do the job. The function call includes a reference to the new variable or array to be written, plus the dimensions and other details.
- To replace an existing dataset anywhere in an SDF file, one uses the functions `sdf_replace` (IDL and C) and `sdf_replace_f77` (Fortran). These function calls also include the new data and dimensions as arguments.

# Transposing arrays and changing index directions in SDF

Frequently, one needs to change the indexing order of large, multi-dimensional arrays, and/or to reverse the direction of one or more of the array indices. The SDF library for C/Fortran has functions `sdf_transpose` (C) and `sdf_transpose_f77` (Fortran) which perform these operations **in place**, which means the operation is done in memory without creating a temporary copy of the array. This is important when one is near the maximum memory limit. Here is how `sdf_transpose` is used from a C program:

```
sdf_transpose(ind,rev,id,data);
```

Here, ind and rev are integer arrays that describe the new index order and directions for the transposed array in terms of the original order. The structure “id” contains the information about the array dimension, etc. and “data” is a pointer to the array. On output, both id and data are changed to reflect the changed dimensions and reshuffled array values.

The vacancy-cycle tracking method of C. H. Q. Ding, "An Optimal Index Reshuffle Algorithm for Multidimensional Arrays and its Applications for Parallel Architectures", IEEE Transactions on Parallel and Distributed Systems, vol. 12, No. 3, pp 306-315, 2001 is used to perform the in-place transpose.

# The IDL version of SDF allows one to “save” and “restore” variables in an IDL session to, and from, an SDF file:

To write all of the variables and arrays in your current IDL session into an SDF file,  enter the command
**`sdf_write_all,’fname.sdf’`**
where it is assumed in this example that the output file is fname.sdf .

There are some restrictions – IDL strings must be converted to byte arrays with the byte () function before they will be written, and IDL structures will not be written out.

To read in all of the IDL variables from the SDF file in the above example, enter the command
**`sdf_read_all,’fname.sdf’`**
To convert byte arrays back into IDL strings, just use the IDL string() function.

In contrast to IDL save files, SDF files are not proprietary and can be used in any other application, in addition to being platform independent.

# The SDF Library includes other useful low-level I/O functionality

The `sdf_wb` and `sdf_rb` (`sdf_wb_f77` and `sdf_rb_f77` in Fortran) functions will perform large endian binary writes and reads without metadata, which for some applications is more useful than a formal file format.  These functions are also large-file capable and platform-independent.

Functions for detecting platform endian-ness (`is_big_endian` and `ibe_f77` in Fortran), and for byte-swapping (`byteswap` and `byteswap_f77` in Fortran) are included in the SDF library, and callable from both C and Fortran.

The Fortran-callable part of the library includes wrappers for most of the low-level C disk I/O functions, such as `fopen, fclose, fread, fwrite, fseek`, and `ftell`.  These functions allow one to develop low-level I/O capability within Fortran programs without resorting to the pitfalls of Fortran unformatted disk I/O.

Details can be found in the `SDF_USAGE_NOTES.txt` file in the SDF distribution.

## Summary

The Simple Data Format platform-independent data format, and associated software, now exist as a working prototype. I welcome comments, criticisms, and especially, people willing to try it out. If you find bugs, I will try to fix them.

This work was supported by the DoD MURI grant, “Understanding Magnetic Eruptions on the Sun and their Interplanetary Consequences”, and by a grant from NASA’s Heliophysics Theory Program.