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

What is the Concept behind SDF (Simple Data Format)?

- Complex numerical models must be able to write data in binary form to disk for later analysis.
- My opinion was that existing I/O software for binary output for large arrays is too cumbersome for rapid code development and debugging.
- Overarching goals:
  - Writing out a large array should be as simple as adding a single “print” statement to a code.
  - Reading in large arrays, or series of variables and arrays in an analysis package like IDL should as simple as entering a single, simple command that makes all of those arrays and variables available for detailed analysis
  - The file format, and the software to read it, should work transparently on all platforms that a scientific user might conceivably use (“platform independence”).
  - File size should not be limited to 2GB.
  - Software should be fast and efficient compared to other data formats
- The current version of SDF meets these goals. The software is publicly available over the web with an LGPL license.
- SDF data output is available as an option in ANMHD and RADMHD. The software is also being used in NRL’s ARMS code for Dump/Restart files because of its platform independence. Recently, it was also adopted into Jim McTiernan’s NLFFF code.

Example of writing an array in a Fortran program, and reading in the data from IDL:

To write out the 3-d array `rho` to the file `junk.sdf` :

```
call sdf_write_f77('junk.sdf','rho','f',8,3,dims3,rho)
```

That's all there is to it! Generally, one includes one sdf_write statement for each variable or array to be written to the file. Each write statement appends a 'dataset' to the file. There are analogous statements that are used from C programs.

From IDL, to read all datasets from an sdf file, just enter this:

```
sdf_read_varlist,'junk.sdf'
```

That's it! If you only want to read one array, say the `rho` array, one can enter this:

```
sdf_read_varlist,'junk.sdf',vars = 'rho'
```

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.

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_varlist,'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.