sdhdfProc guide for developers

Version 1: George Hobbs

Introduction and design philosophy

The sdhdfProc software is still under intense development and is likely to change significantly.

The software package is written in C and has been designed:

- To process SDHDF data files that can contain multiple frequency bands, spectral dumps and telescope beams,
- So that the file sizes can be many GBs in size and the frequency bands may cover very large bandwidths,
- For the Parkes radio telescope, but should be usable by any single dish telescope (or beam formed output from an interferometer).

The basic philosophy is that each program loads in one or more SDHDF files, carries out a small number of processing tasks and then outputs either a final result (figure or set of values) or produces more SDHDF files. Those SDHDF files can then be read into other sdhdfProc packages for further processing. For instance a typical pipeline may include:

- sdhdf_extractBand …. (to extract the relevant parts of the observations)
- sdhdf_flag …. (to flag radio frequency interference)
- sdhdf_modify … (to average over the spectral dumps)
- sdhdf_identify … (to identify the ON and OFF source pointing positions)
- sdhdf_onoff …. (to form [ON-OFF]/OFF quotient spectra)
- sdhdf_modify … (to convert to the local standard of rest)
- sdhdf_plotSpectrum … (to plot the final output spectrum)

The intention is that most users would be able to understand the algorithm applied by looking in the relevant source code - i.e., the code is not highly abstracted.

The code requires a minimal number of external libraries to simplify installation and longevity.

The code is stored in a git repository:
https://bitbucket.csiro.au/projects/CPDA/repos/sdhdf/browse

Command line arguments

The sdhdf packages usually require command line arguments. The choice is mostly left to the developer of that package however:

- All packages should allow `-h`, which will provide help information on what the package does and how the input arguments.

- If a single input file is required then the filename is indicated by `-f`
- If a single output file is produced then the filename of that file is indicated by `-o`
- If multiple input file names are given then they are listed without any pre-argument, e.g., `sdhdf_modify *.hdf`
- If the code produces output file names that are the same as the input, but with an extra extension then the `-e` option is used, e.g., `sdhdf_modify -T -e t_average *.hdf` -- here the output files will be the `inputFile.hdf.t_average`.

The directory structure, environment variable and making the software

The git repository contains:

```
c  docs  python  README.md  sh
```

For the code described here we enter the “c” directory, which currently only has the sub-directory

```
sdhdfProc
```

Changing into this directory gives us a set of makefiles and the following sub-directories

```
bin docs runtime src ...
```

The runtime directory is used to store information required when the software is running. The environment variable SDHDF_RUNTIME needs to point at this directory. If using tcsh then:

```
setenv SDHDF_RUNTIME <base directory>/runtime/
```

Note that this directory can be moved elsewhere on the file system (as long as the environment variable is correctly set).

Configuring and making the software

To make the software the standard process is followed

```
./bootstrap
configure
make
make install
```

Depending on your login scripts, you may need to include flags for the configure step – e.g. on ATNF machines:

```
configure CFLAGS="-I/pulsar/psr/software/next/buster/include
-I/usr/include/hdf5/serial/"
LDFLAGS="-L/pulsar/psr/software/next/buster/lib
-L/usr/lib/x86_64-linux-gnu/hdf5/serial"
```

Unless there is any major change to the software or the system that it is being installed on, subsequent installation should simply be:

```
make
make install
```

The compiled software packages will be in the `bin` directory.

Documentation

Pdf copies of the document should be included in the `docs` directory in the git repository.

We also have:

- This document (for developers) that can be updated at https://docs.google.com/document/d/1OyWBQ8-BayjOhaTHbh5O6oQIcWXih2ay7VXRmhFK3hI/edit?usp=sharing
- A priority listing of tasks https://docs.google.com/document/d/1Sn7ojiUiFs0BbndOCVlv-uWoyTPzbxuQjtus0Ehz8ss/edit?usp=sharing
- General user documentation: https://docs.google.com/document/d/1PhVhzOmVZ7iqA97LxRHF0gP5OHUl4Mw0Hp3KVjU9yIA/edit?usp=sharing
- Use-case tutorials: https://docs.google.com/document/d/1jA5lOn5MlltnOrchrggwLSnBtKRPJVUL2F984NmVRKk/edit?usp=sharing

Bug Tracking

Currently we are recording bugs and feature requests in https://docs.google.com/document/d/1Sn7ojiUiFs0BbndOCVlv-uWoyTPzbxuQjtus0Ehz8ss/edit?usp=sharing. In the future it is likely we will make use of Jira tickets and, when the code is more stable and used by more people, the Bitbucket bug tracker.

The runtime directory

Note that the structure of this directory is actively being updated and so this information may change.

The runtime directory (pointed to by $SDHDF_RUNTIME) contains:

```
earth  fileConversion  observatory  README
```

These are described in the sub-sections below:

Earth

This contains the international Earth rotation and reference systems service: Earth orientation parameters:

```
eopc04_IAU2000.62-now
```

fileConversion

```
README  sdHeader.fits
```

Observatory

In the observatory directory we have directories for different observatory sites:

```
parkes
```

Inside each observatory directory we have subdirectories

```
calibration rfi
```

The `calibration` directory contains:

```
README  scalAverage.dat  tcal_noflag.dat
```

Which provides system and calibrator equivalent flux density values and the temperature of the calibrator.

The `rfi` directory contains:

`persistentRFI.dat`

Which is a listing of the persistent RFI at the specific observatory.

Opening a file and reading the metadata

The sdhdf_describe code demonstrates how the metadata from an SDHDF file can be obtained. In all sdhdf codes we require:

```
...
#include "sdhdfProc.h"
…
```

The file is stored as an sdhdf_fileStruct:

```
sdhdf_fileStruct *inFile;
```

And memory needs to be allocated (and later free'd) for this structure:

```
...
 inFile = (sdhdf_fileStruct *)malloc(sizeof(sdhdf_fileStruct));
…
// CODE GOES HERE
free(inFile);
```

The file structure should be manipulated as follows:

```
    sdhdf_initialiseFile(inFile);
    if (sdhdf_openFile(fname[i],inFile,1)==-1)
            printf("Warning: unable to open file >%s<. Skipping\n",fname[i]);
    else
     {
       sdhdf_loadMetaData(inFile);
      // WORK WITH THE FILE
       ….
     sdhdf_closeFile(inFile);
     }
```

The structure containing the file information is currently set in sdhdf_v1.9.h (and in the SDHDF definition). Details are provided below, but a few key parameters are:

```
inFile->beam[0].nBand                    // The number of sub-bands for beam 0 in the file
inFile->beam[0].bandHeader[j].ndump          // The number of spectral dumps for beam 0, sub-band j
nFile->beamHeader[0].source              // The source name for beam 0
inFile->beam[0].bandHeader[j].nchan   // The number of channels for beam 0 and sub-band j
inFile->beam[0].bandHeader[j].dtime   // The spectral dump time for beam 0 and sub-band j
```

Accessing the spectra

The `sdhdf_quickdump` code demonstrates how to access the spectra (for both the astronomy signal and any calibration signal). In brief, the file is opened as above. The frequency corresponding to each channel is obtained from:

```
freq = inFile->beam[beam].bandData[band].astro_data.freq[j];  // for specified beam and sub-band and channel number j
```

Note that the value is as recorded in the file. The SDHDF attributes defines whether that frequency is topocentric, barycentric or in the local standard of rest.

```
pol1 = inFile->beam[beam].bandData[band].astro_data.pol1[j+k*nchan];
pol2 = inFile->beam[beam].bandData[band].astro_data.pol2[j+k*nchan];
pol3 = inFile->beam[beam].bandData[band].astro_data.pol3[j+k*nchan];
pol4 = inFile->beam[beam].bandData[band].astro_data.pol4[j+k*nchan];
```

This is for spectral dump, `k`, and channel number `j`. Note that the number of polarisation channels stored in the file can be obtained by:

```
npol = inFile->beam[beam].bandHeader[band].npol;
```

The header information

The header information is currently included as sdhdf_v1.9.h. The primary structure is:

```
// filename/
//        obs_meta/
//              primary header
//              software_versions
//              history
//        obs_config/                (this group is currently ignored by sdhdfProc)
//              backend_config
//              cal_backend_config
//        beam_XX/
//              beam_header
//              band_header
//              band_YY/
//                    astronomy/
//                          data
//                          frequency
//                            obs_params
```

```
//                  calibrator/
//                     cal_data_on
//                     cal_data_off
//                     cal_frequency
//                     obs_meta/
//                           cal_band_header
//                           cal_obs_params
//                     cal_proc/
//                           cal_proc_tsys
//                           cal_proc_diff_gain
//                           cal_proc_diff_phase
```

Attributes

Each data set and group has attributes. These are currently not well defined and not used properly within the sdhdfProc code. This needs updating.

Writing a new sdhdf-format file

The `sdhdf_onoff.c` program demonstrates how to load in two input files, produce a new data set (in this case a subtraction or quotient of the input files) and then outputs a new SDHDF-format file. The basic commands are

```
sdhdf_fileStruct *outFile;
…
 if (!(outFile = (sdhdf_fileStruct *)malloc(sizeof(sdhdf_fileStruct))))
  {
    printf("ERROR: unable to allocate sufficient memory for >outFile<\n");
    exit(1);
  }
…
    sdhdf_initialiseFile(outFile);
    sdhdf_openFile("outfile.sdhdf",outFile,3);

….
sdhdf_writeSpectrumData(outFile,onFile->beam[b].bandHeader[i].label,b,i,out_data,freq,nchan,npol,ndump,0,dataAttributes,
nDataAttributes,freqAttributes,nFreqAttributes);
…
    sdhdf_writeHistory(outFile,onFile->history,onFile->nHistory);
    sdhdf_copyRemainder(onFile,outFile,0);
    sdhdf_closeFile(outFile);
….
 free(outFile);
```

General use functions

Generic functions are defined in sdhdfProc.h. These are currently:

```
// File input/output
void sdhdf_initialiseFile(sdhdf_fileStruct *inFile);
int  sdhdf_openFile(char *fname,sdhdf_fileStruct *inFile,int openType);
void sdhdf_closeFile(sdhdf_fileStruct *inFile);

// Data
void sdhdf_releaseBandData(sdhdf_fileStruct *inFile,int beam,int band,int type);
void sdhdf_loadBandData(sdhdf_fileStruct *inFile,int beam,int band,int type);
void sdhdf_loadBandData2Array(sdhdf_fileStruct *inFile,int beam,int band,int type,float *arr);
void sdhdf_loadFrequency2Array(sdhdf_fileStruct *inFile,int beam,int band,float *arr);
void sdhdf_allocateBandData(sdhdf_spectralDumpsStruct *spec,int nchan,int ndump,int npol);
void sdhdf_extractPols(sdhdf_spectralDumpsStruct *spec,float *in,int nchan,int ndump,int npol);

// Command line arguments
void sdhdf_add1arg(char *args,char *add);
void sdhdf_add2arg(char *args,char *add1,char *add2);


// Loading metadata information
void sdhdf_loadMetaData(sdhdf_fileStruct *inFile);  // Include loading attributes
void sdhdf_loadPrimaryHeader(sdhdf_fileStruct *inFile);
void sdhdf_loadBeamHeader(sdhdf_fileStruct *inFile);
void sdhdf_loadBandHeader(sdhdf_fileStruct *inFile,int type);
void sdhdf_loadObsHeader(sdhdf_fileStruct *inFile,int type);
void sdhdf_loadHistory(sdhdf_fileStruct *inFile);
void sdhdf_loadSoftware(sdhdf_fileStruct *inFile);
void sdhdf_loadSpectrum(sdhdf_fileStruct *inFile,int ibeam, int iband, sdhdf_spectralDumpsStruct *spectrum);
void sdhdf_initialise_spectralDumps(sdhdf_spectralDumpsStruct *in);
void sdhdf_initialise_bandHeader(sdhdf_bandHeaderStruct *header);
void sdhdf_initialise_obsHeader(sdhdf_obsParamsStruct *obs);
void sdhdf_copyBandHeaderStruct(sdhdf_bandHeaderStruct *in,sdhdf_bandHeaderStruct *out,int n);
void sdhdf_allocateBeamMemory(sdhdf_fileStruct *inFile,int nbeam);
int sdhdf_checkGroupExists(sdhdf_fileStruct *inFile,char *groupName);
void sdhdf_copySingleObsParams(sdhdf_fileStruct *inFile,int ibeam,int iband,int idump,sdhdf_obsParamsStruct *obsParam);
void sdhdf_loadCalProc(sdhdf_fileStruct *inFile,int ibeam,int iband,char *cal_label,float *vals);

// Attributes
int sdhdf_getNattributes(sdhdf_fileStruct *inFile,char *dataName);
void sdhdf_readAttributeFromNum(sdhdf_fileStruct *inFile,char *dataName,int num,sdhdf_attributes_struct *attribute);
void sdhdf_copyAttributes(sdhdf_attributes_struct *in,int n_in,sdhdf_attributes_struct *out,int *n_out);
void sdhdf_writeAttribute(sdhdf_fileStruct *outFile,char *dataName,char *attrName,char *result);

// Loading data
void sdhdf_loadFrequencies(sdhdf_fileStruct *inFile,int ibeam,int iband,int type);
void sdhdf_loadData(sdhdf_fileStruct *inFile,int ibeam,int iband,float *in_data,int type);
int  sdhdf_loadFlagData(sdhdf_fileStruct *inFile);
int  sdhdf_loadTcal(sdhdf_tcal_struct *tcalData,char *fname);
int  sdhdf_loadTsys(sdhdf_fileStruct *inFile,int band,float *tsys);
int  sdhdf_loadPhase(sdhdf_fileStruct *inFile,int band,float *phase);
int  sdhdf_loadGain(sdhdf_fileStruct *inFile,int band,float *gain);

// SDHDF identification
```

```
int sdhdf_doWeHave(sdhdf_fileStruct *inFile,char *groupLabel);
int sdhdf_getBandID(sdhdf_fileStruct *inFile,char *input);

// Manipulation and processing
void sdhdf_convertStokes(float p1,float p2,float p3,float p4,float *stokesI,float *stokesQ,float *stokesU,float *stokesV);
void sdhdf_get_tcal(sdhdf_tcal_struct *tcalData,int n,double f0,double *tcalA,double *tcalB);
void sdhdf_addHistory(sdhdf_historyStruct *history,int n,char *procName,char *descr,char *args);

// Output information
void sdhdf_copyEntireGroup(char *groupLabel,sdhdf_fileStruct *inFile,sdhdf_fileStruct *outFile);
void sdhdf_copyEntireGroupDifferentLabels(char *bandLabelIn,sdhdf_fileStruct *inFile,char *bandLabelOut,sdhdf_fileStruct
*outFile);
void sdhdf_setMetadataDefaults(sdhdf_primaryHeaderStruct *primaryHeader,sdhdf_beamHeaderStruct *beamHeader,
                        sdhdf_bandHeaderStruct *bandHeader,sdhdf_softwareVersionsStruct
*softwareVersions,sdhdf_historyStruct *history,
                        int nbeam,int nband);
void sdhdf_writeHistory(sdhdf_fileStruct *outFile,sdhdf_historyStruct *outParams,int n);
void sdhdf_writeSoftwareVersions(sdhdf_fileStruct *outFile,sdhdf_softwareVersionsStruct *outParams);
void sdhdf_writePrimaryHeader(sdhdf_fileStruct *outFile,sdhdf_primaryHeaderStruct *primaryHeader);
void sdhdf_writeBandHeader(sdhdf_fileStruct *outFile,sdhdf_bandHeaderStruct *outBandParams,int ibeam,int outBands,int
type);
void sdhdf_writeBeamHeader(sdhdf_fileStruct *outFile,sdhdf_beamHeaderStruct *beamHeader,int nBeams);
void sdhdf_writeSpectrumData(sdhdf_fileStruct *outFile,char *blabel, int ibeam,int iband,  float *out,float *freq,long
nchan,long npol,long nsub,int type,sdhdf_at\
tributes_struct *dataAttributes,int nDataAttributes,sdhdf_attributes_struct *freqAttributes,int nFreqAttributes);
void sdhdf_copyRemainder(sdhdf_fileStruct *inFile,sdhdf_fileStruct *outFile,int type);
void sdhdf_writeObsParams(sdhdf_fileStruct *outFile,char *bandLabel,int ibeam,int iband,sdhdf_obsParamsStruct
*obsParams,int ndump,int type);
void sdhdf_writeFlags(sdhdf_fileStruct *outFile,int ibeam,int iband,int *flag,int nchan,char *bandLabel);
void sdhdf_writeDataWeights(sdhdf_fileStruct *outFile,int ibeam,int iband,float *flag,int nchan,int ndump,char *bandLabel);
void sdhdf_writeCalProc(sdhdf_fileStruct *outFile,int ibeam,int iband,char *band_label,char *cal_label,float *vals,int nchan,int
npol,int ndumps);

// HDF5 reading functions
void sdhdf_loadHeaderString(hid_t header_id,char *parameter,char *outStr);
void sdhdf_loadHeaderInt(hid_t header_id,char *parameter,int *outInt);
void sdhdf_loadColumnDouble(char *param,hid_t header_id,hid_t headerT,double *dVals);
void sdhdf_loadColumnString(char *param,hid_t header_id,hid_t headerT,char **vals,int nrows);
void sdhdf_loadColumnInt(char *param,hid_t header_id,hid_t headerT,long int *dVals);
void sdhdf_loadHeaderDouble(hid_t header_id,char *parameter,double *outDouble);

// Ephemerides
long   sdhdf_loadEOP(sdhdf_eopStruct *eop);
double sdhdf_calcVoverC(sdhdf_fileStruct *inFile,int ibeam,int iband,int idump,sdhdf_eopStruct *eop,int nEOP,int lsr);
void sdhdf_obsCoord_IAU2000B(double observatory_trs[3],
                        double zenith_trs[3],
                        long double tt_mjd, long double utc_mjd,
                        double observatory_crs[3],
                        double zenith_crs[3],
                        double observatory_velocity_crs[3],sdhdf_eopStruct *eop,int nEOP);
void sdhdf_interpolate_EOP(double mjd, double *xp, double *yp, double *dut1, double *dut1dot, sdhdf_eopStruct *eop,int
nEOP);
void sdhdf_ITRF_to_GRS80(double x,double y,double z,double *long_grs80,double *lat_grs80,double *height_grs80);

// Mathematics

void sdhdf_setIdentity_4x4(float mat[4][4]);
```

```
void sdhdf_display_vec4(float *vec);
void sdhdf_setFeed(float mf[4][4],float gamma);
void sdhdf_multMat_vec_replace(float mat[4][4],float *vec);
void sdhdf_copy_mat4(float in[4][4],float out[4][4]);
void sdhdf_mult4x4_replace(float src1[4][4], float src2[4][4]);
void sdhdf_copy_vec4(float *in,float *out);


void sdhdf_setGainPhase(float ma[4][4],float diffGain,float diffPhase);
void sdhdf_setGain2Phase(float ma[4][4],float diffGain,float diffPhase);
void sdhdf_setParallacticAngle(float msky[4][4],float pa);

int sdhdf_inv4x4(float m[4][4],float inv[4][4]);
void sdhdf_mult4x4(float src1[4][4], float src2[4][4], float dest[4][4]);
double sdhdf_dotproduct(double *v1,double *v2);
void sdhdf_para(double dxd,double ddc,double q,double *axd,double *eld);
void displayMatrix_4x4(float matrix[4][4]);
```