Raw OLAP data formats

OLAP produces several data formats, which are intended to be replaced by their final format, such as HDF5. The formats below are not officially supported and subject to change without notice.

Beamformed Data

Beamformed data can be recorded as either complex voltages (yielding X and Y polarisations) or one or more stokes. In either case, a sequence of blocks will be stored, each of which consists of a header and data. The header is defined as:

```
struct header {
  uint32 sequence_number; /* big endian */
  char padding[508];
};
```

in which sequence_number starts at 0, and is increased by 1 for every block. Missing sequence numbers implies missing data. The padding can have any value and is to be ignored.

Complex Voltages

Each (pencil) beam produces two files: one containing the X polarisation, and one containing the Y polarisation. The names of these files adhere to the following scheme:

Lxxxxx_Byyy_S0-bf.raw	X polarisations of beam yyy of observation xxxxx
Lxxxxx_Byyy_S1-bf.raw	Y polarisations of beam yyy of observation xxxxx

Each file is a sequence of blocks of the following structure:

```
struct block {
  struct header header;

  /* big endian */
  fcomplex voltages[SAMPLES|2][SUBBANDS][CHANNELS];
}
```

Stokes

Each (pencil) beam produces one or four files: one containing the Stokes I (power) values, and optionally three files for Stokes Q, U, and V, respectively. The names of these files adhere to the following scheme:

Lxxxxx_Byyy_S0-bf.raw	Stokes I of beam yyy of observation xxxxx
Lxxxxx_Byyy_S1-bf.raw	Stokes Q of beam yyy of observation xxxxx
Lxxxxx_Byyy_S2-bf.raw	Stokes U of beam yyy of observation xxxxx
Lxxxxx_Byyy_S3-bf.raw	Stokes V of beam yyy of observation xxxxx

Currently (release 2010-09-20), the Stokes U and V are multiplied by a factor of 1/2, but that will be changed in a subsequent release.

Each file is a sequence of blocks of the following structure:

```
struct block {
  struct header header;

  /* big endian */
  float stokes[SAMPLES|2][SUBBANDS][CHANNELS];
}
```

Types and constants

Types

A 'float' is a 32-bit IEEE floating point number. An 'fcomplex' is a complex number defined as

```
struct fcomplex {
  float real;
  float imag;
};
```

Constants

Constants can be computed using the parset file. Below is a translation between the C constants used above and their respective parset keys:

SAMPLES	The number of time samples in a block	OLAP.CNProc.integrationSteps
SUBBANDS	The number of subbands (beamlets) specified	len(Oberservation.subbandList)
CHANNELS	The number of channels per subband	Observation.channelsPerSubband

Useful routines

The following routines might be useful when reading raw OLAP data.

Byte swapping

Needed if you read data on a machine which used a different endianness. Typically, x86 machines (intel, amd) are little-endian, while the rest (sparc, powerpc, including the BlueGene/P) is big-endian.

```
uint32 swap_uint32( uint32 x )
{
  union {
```

```
    char c[4];
    uint32 i;
  } src,dst;

  src.i = x;
  dst.c[0] = src.c[3];
  dst.c[1] = src.c[2];
  dst.c[2] = src.c[1];
  dst.c[3] = src.c[0];

  return dst.i;
}

/* Do NOT take a float as an argument. An incorrectly read float
   (because it has the wrong endianness) is subject to modification
   by the platform/compiler (normalisation etc). */
float swap_float( char *x )
{
  union {
    char c[4];
    float f;
  } dst;return

  dst.c[0] = x[3];
  dst.c[1] = x[2];
  dst.c[2] = x[1];
  dst.c[3] = x[0];

  return dst.f;
}
```

Variable-sized arrays

Since the dimensions of the arrays produced by OLAP depend on the parset, it's handy to have access to arrays with variable size. The easiest way is to use C++ and the boost library (which is often installed by default):

```
#include "boost/multi_array.hpp"

int main() {
  /* create an array of floats with 2 dimensions, and initialise it to have
dimensions [2][3] */
  boost::multi_array<float,2> myarray(boost::extents[2][3]);

  /* getting and setting is the same as with regular C arrays */
  myarray[1][2] = 1.0;

  /* note: &myarray[0][0] is the address of the first element, which can be
used if the full
     array needs to be read from disk. */
```

```
  return 0;
}
```

See also http://www.boost.org/doc/libs/1_43_0/libs/multi_array/doc/user.html

If you need to use C, things become a bit more cumbersome. You need to roll out your own multi-dimensional array, although you'll have to customise your code for each number of dimensions in order to keep your code readable. For example:

```
struct myarray {
  float *data;
  unsigned max1,max2;
};

/* return myarray[one][two] */
float get( struct myarray *array, int one, int two )
{
  return *(myarray.data + one * myarray.max2 + two);
}

/* set myarray[one][two] to value */
void set( struct myarray *array, int one, int two, float value )
{
  *(myarray.data + one * myarray.max2 + two) = value;
}

int main() {
  /* create an array of floats */
  struct array myarray;

  /* allocate the array with dimensions [2][3] */
  myarray.max1 = 2;
  myarray.max2 = 3;
  myarray.data = malloc( myarray.max1 * myarray.max2 * sizeof *myarray );

  /* emulate myarray[1][2] = 1.0 */
  set(&myarray,1,2,1.0);

  /* note: myarray.data is the address of the first element, which can be
used if the full
     array needs to be read from disk. */

  /* free the array */
  free( myarray.data );

  return 0;
}
```

Keep in mind that if you need to switch endianness as well, you first need to read into a char array, and convert it to a float array after reading from disk.

From:
https://www.astron.nl/lofarwiki/ - **LOFAR *Wiki***

Permanent link:
https://www.astron.nl/lofarwiki/doku.php?id=public:documents:raw_olap_data_formats&rev=1287669968

Last update: **2010-10-21 14:06**