Cosmic Ray Data Processing

Formats & Tools

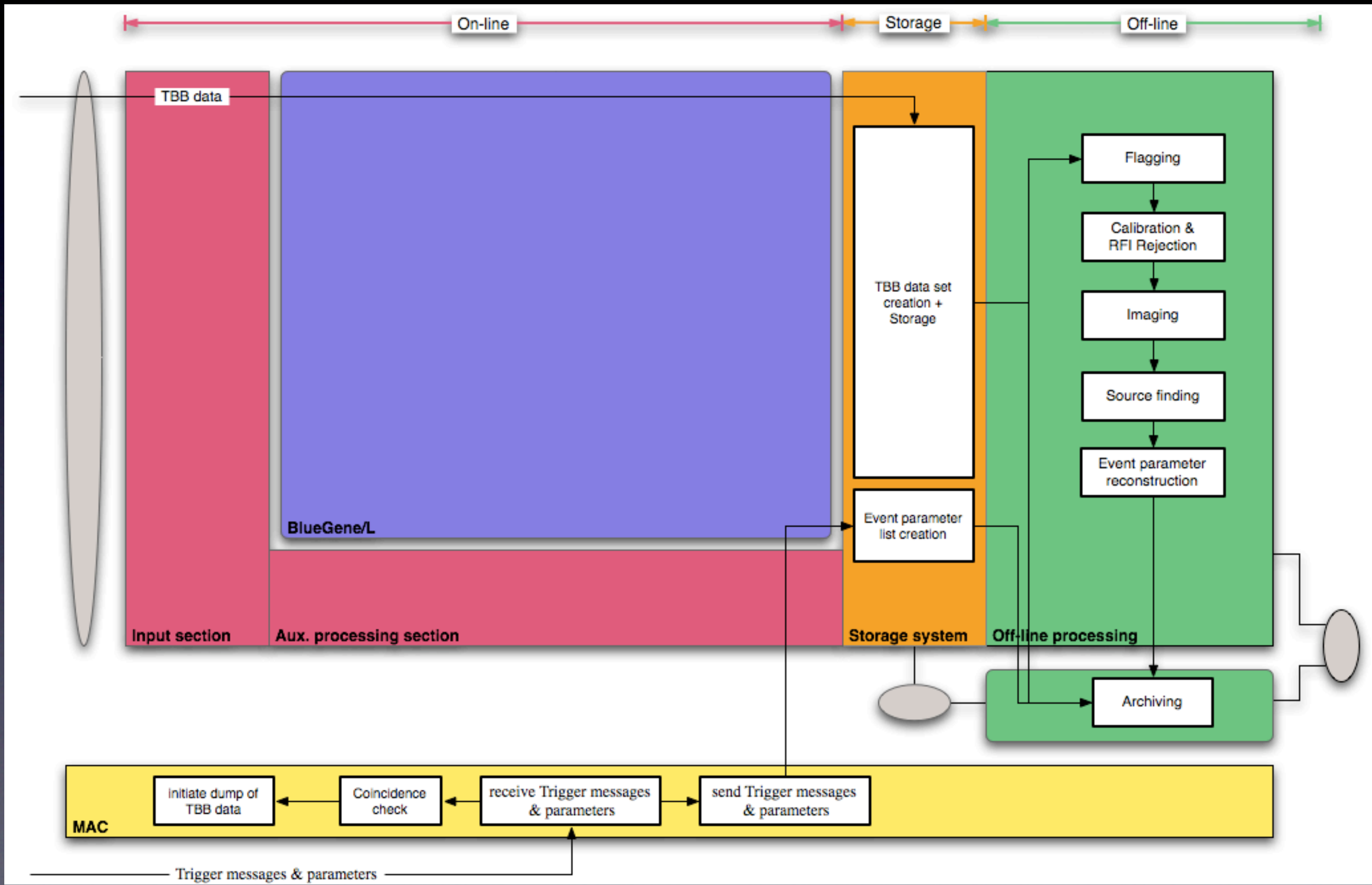
Lars Bähren

LOFAR Data Processing School, 12. Feb 2009

Outline

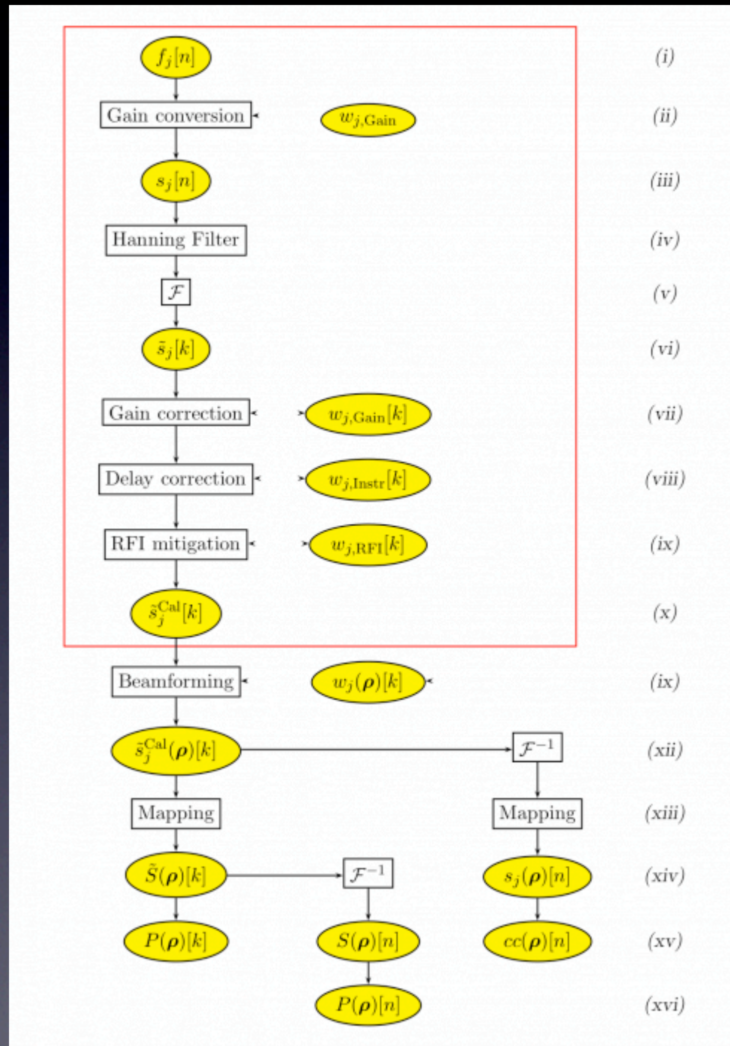
- CR pipeline: Layout & Data
- Input data
- Data products
- CR-Tools

CR pipeline: Processing locations



CR pipeline: Layout

- Fourier transforming the data to the frequency domain
- Correction of instrumental delays from the TV-transmitter
- Frequency dependent gain correction
- Suppression of narrow band RFI
- Flagging of antennas with high noise
- Correction of trigger delay
- Beam-forming in the direction of the air shower
- Quantification of peak parameters
- Optimizing the radius of curvature
- Identification of good events



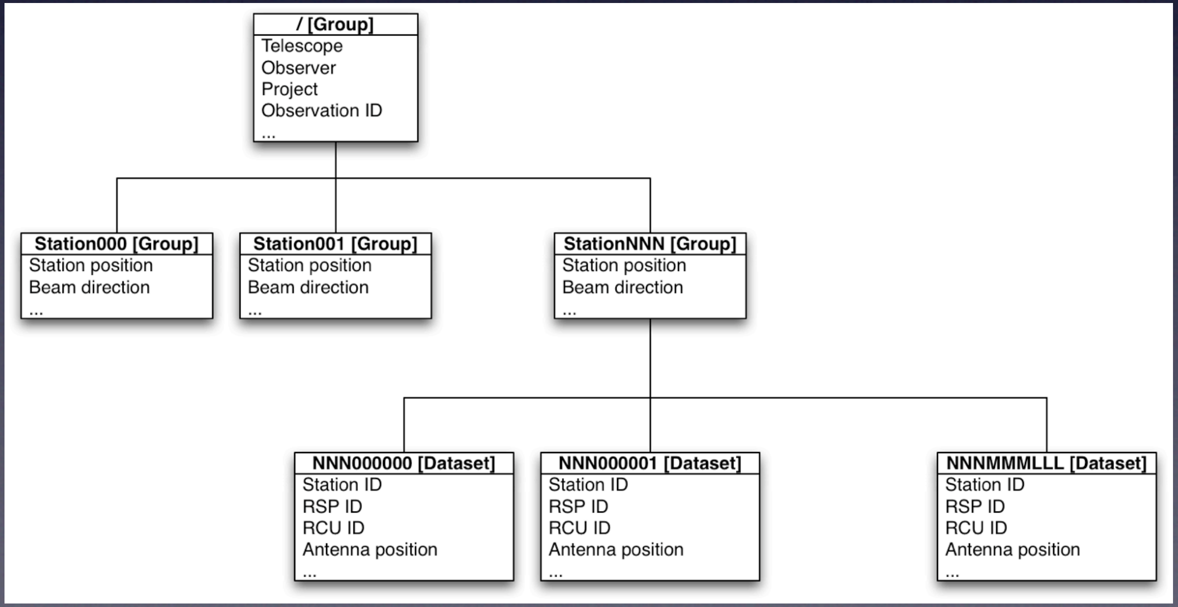
CR pipeline: Data

- Inputs:
  - Time-series data of the individual dipoles
  - Station calibration information
  - System health information
  - Reception pattern of the individual dipoles
- Outputs:
  - Multi-dimensional images (near-field, full time-resolution)
  - Event-list with reconstructed physical parameters of the shower (position, pulse height, electric field strength, etc.)

			Rates		Space	
Mode	Type	Events	DAQ	Archive	temp.	perm.
UHEP	RAW	1/10 min	806 GB/day	806 GB/day	24 TB	$\sim$1 TB
	cubes			<10 GB/day	<10 TB	<10 TB
	event list			< 1 GB/day	<1 TB	<1 TB
VHECR	RAW	1/10 min/station	7400 GB/day	542 GB/day	386 TB	285 TB
	cubes			<1 GB/day	1 TB	1 TB
	event list			< 1 GB/day	<1 TB	<1 TB
HECR	RAW	1/10 min/station	542 GB/day	542 GB/day	386 TB	9 TB
	cubes			< 1 GB/day	1 TB	1 TB
	event list			< 1 GB/day	<1 TB	<1 TB
TS mode	RAW	> 20/day	118 TB/day	10 TB/day	600 TB	600 TB
TOTAL			118 TB/day	10 TB/day	1 PB	1 PB

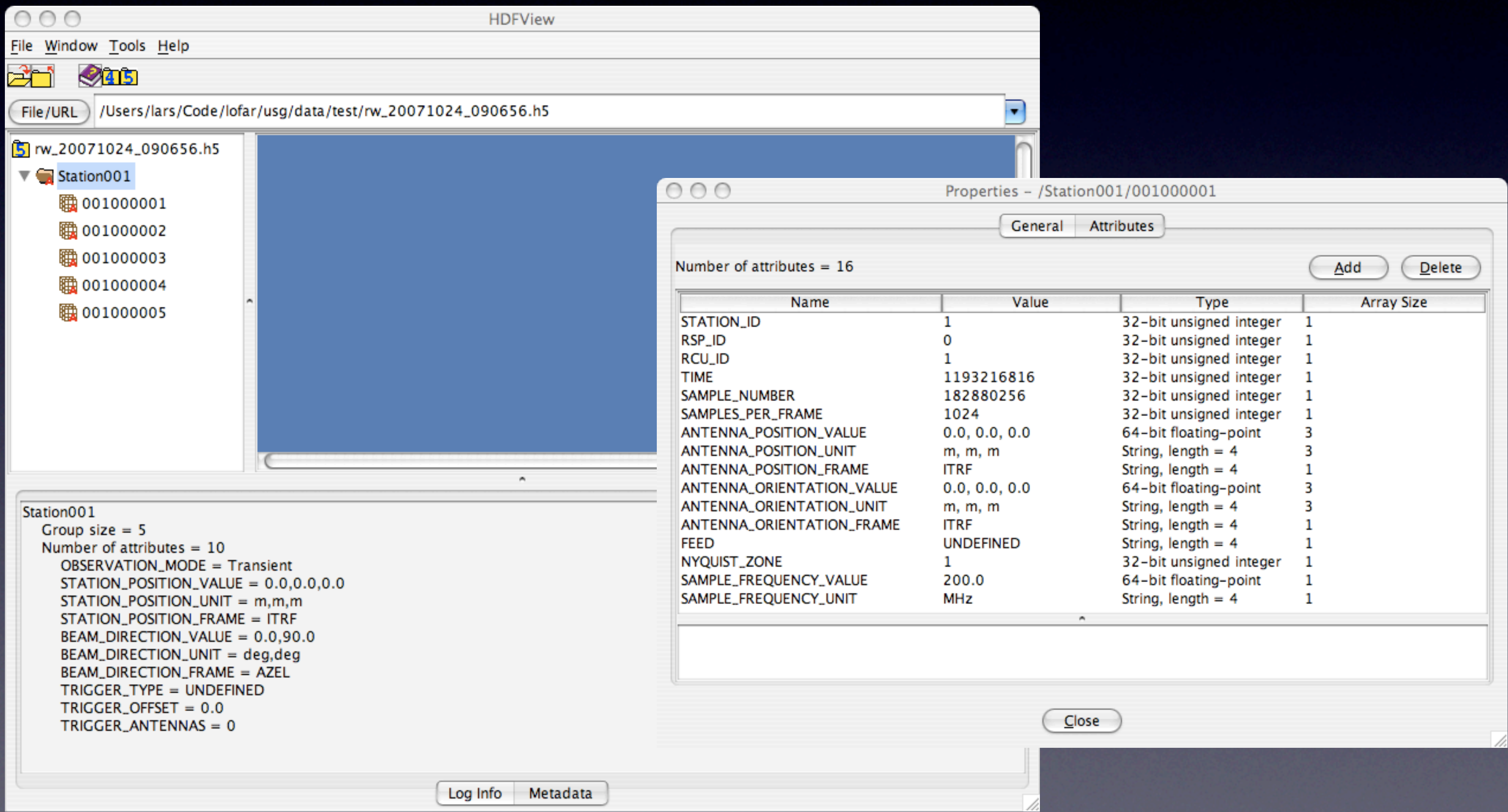
Input data

- Time-series = digitized waveform received by an individual dipole
- System-internal data-format converted through DAL to HDF5 dataset
- Representation of system-hierarchy within structure of dataset
  - Format definition available (ICD)
  - Higher-level interface through set of C++ classes
  - Python bindings under development (pydal)



Input data

- Off-the-shelf tools can be used for basic inspection of datasets



Input data: Access through DAL

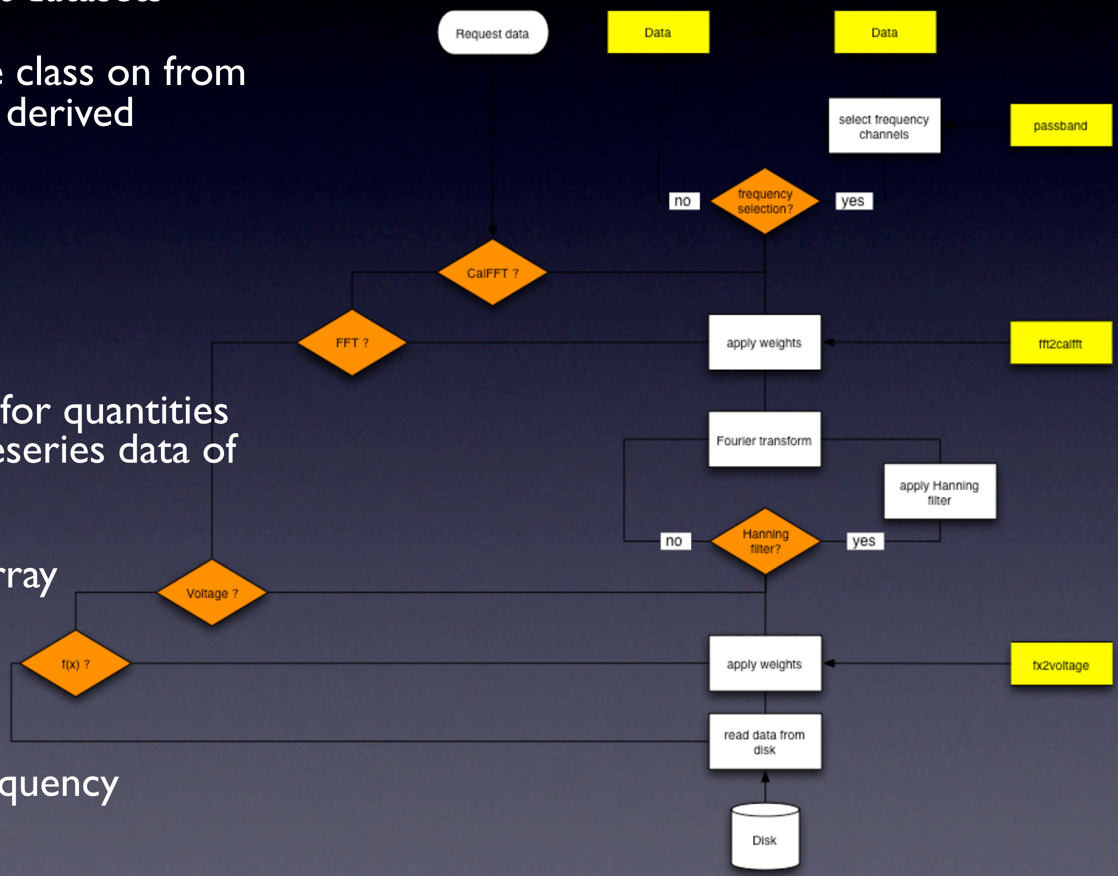
- For C++ programmers: set of classes encapsulating the hierarchical organization of the data
  - TBB_Timeseries
  - TBB_StationGroup
  - TBB_DipoleDataset
- Access to:
  - Metadata
    - Observation info
    - Station/antenna positions
  - Raw antenna data
- For Python programmers:
  - access methods part of pydal

```
TBB_Timeseries ts (filename);

assert (nofFailedTests==0);

cout << "[1] Retrieve attributes attached to the root group ..." << endl;
try {
  std::string telescope        = ts.telescope();
  std::string observer         = ts.observer();
  std::string project          = ts.project();
  std::string observation_id   = ts.observation_id();
  std::string observation_mode = ts.observation_mode();
  //
  cout << "-- TELESCOPE ...... = " << telescope        << endl;
  cout << "-- OBSERVER ....... = " << observer         << endl;
  cout << "-- PROJECT ........ = " << project          << endl;
  cout << "-- OBSERVATION_ID   = " << observation_id   << endl;
  cout << "-- OBSERVATION_MODE = " << observation_mode << endl;
} catch (std::string message) {
  cerr << message << endl;
  nofFailedTests++;
}
```

```
int start          = 0;
int nofSamples     = 1024;
TBB_Timeseries timeseries (filename);

try {
  casa::Matrix<double> data = timeseries.fx (start,
                                             nofSamples);
  // feedback
  cout << "-- Data start     = " << start        << endl;
  cout << "-- Data blocksize = " << nofSamples   << endl;
  cout << "-- Data array     = " << data.shape() << endl;
  cout << "-- Data [0,]      = " << data.row(0)  << endl;
  cout << "-- Data [1,]      = " << data.row(1)  << endl;
} catch (std::string message) {
  std::cerr << message << endl;
  nofFailedTests++;
}
```

DataReader

- Abstraction layer to hide details for dealing with different input datasets
- C++ implementation: base class on from which all special cases are derived
  - LOPES
  - ITS
  - LOFAR
- Hard-coded mini-pipeline for quantities derived from the raw timeseries data of individual antennas
- Conversion/Calibration: array multiplication
  - value per antenna
  - value per antenna & frequency

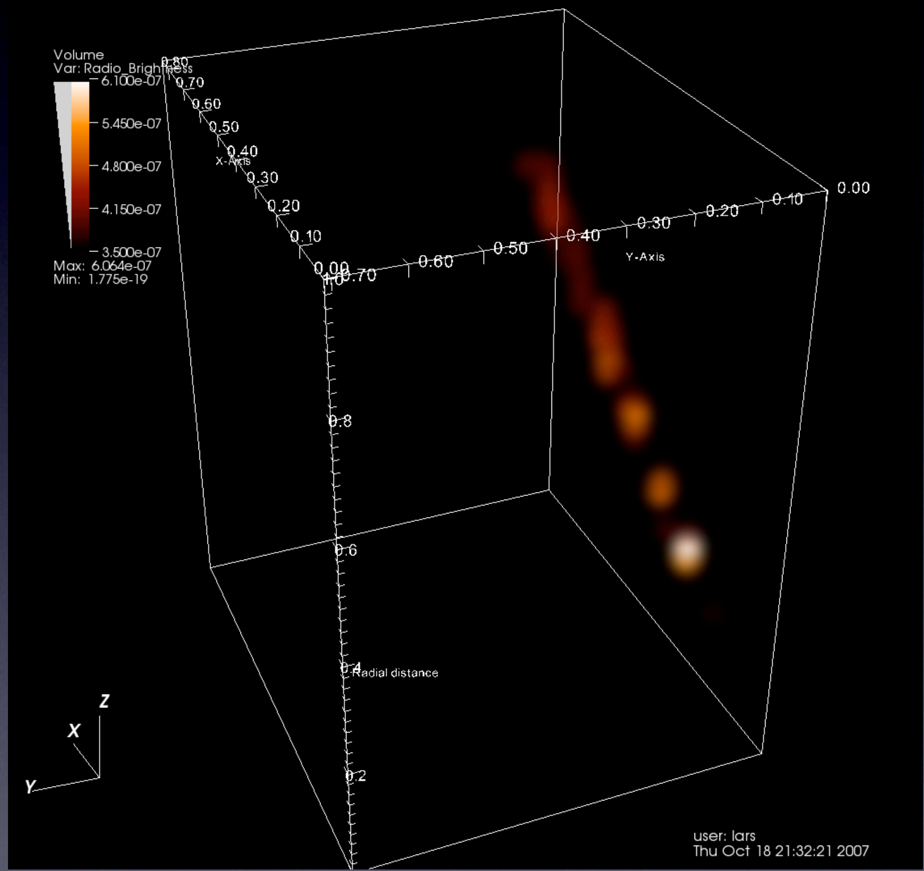


CR near-field imager

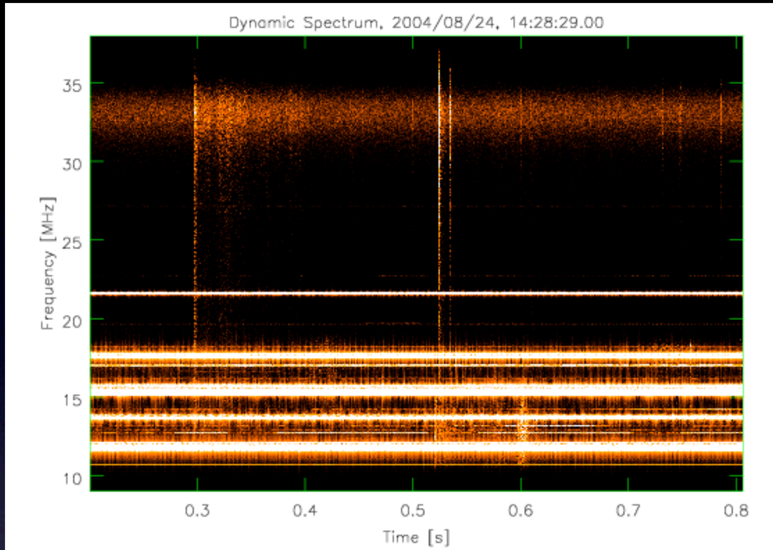
- Imaging not based on visibility data
  - CR pulse to short to allow integration of antenna signal
  - Required calibration/corrections per individual dipole
- Near-field imaging at full time-resolution not part of standard imaging pipeline
- Implementation of various types of beam
  - adding beam
  - CC beam
  - X Beam
- Image hyper-cube of >= 5 dimensions

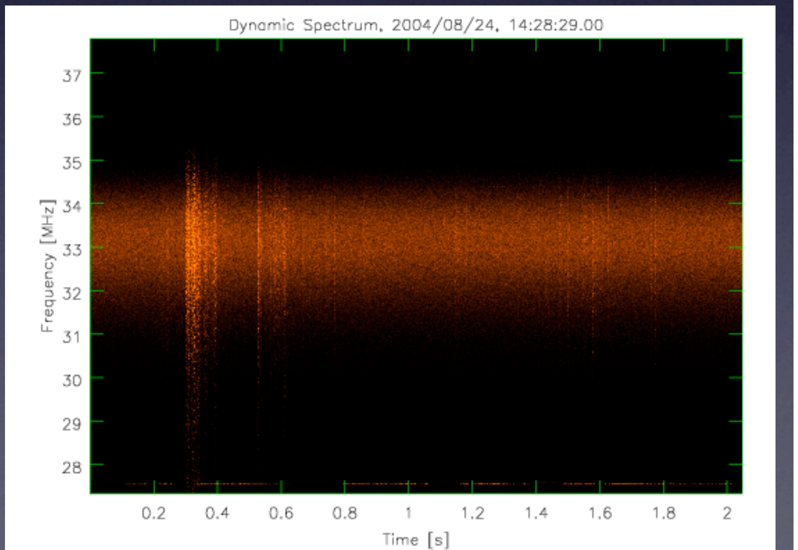
$$I = I(\rho, t, \nu, \mathbf{p}) = I\Big(\rho(\xi_1, \xi_2, \xi_3), t, \nu, \mathbf{p}\Big)$$

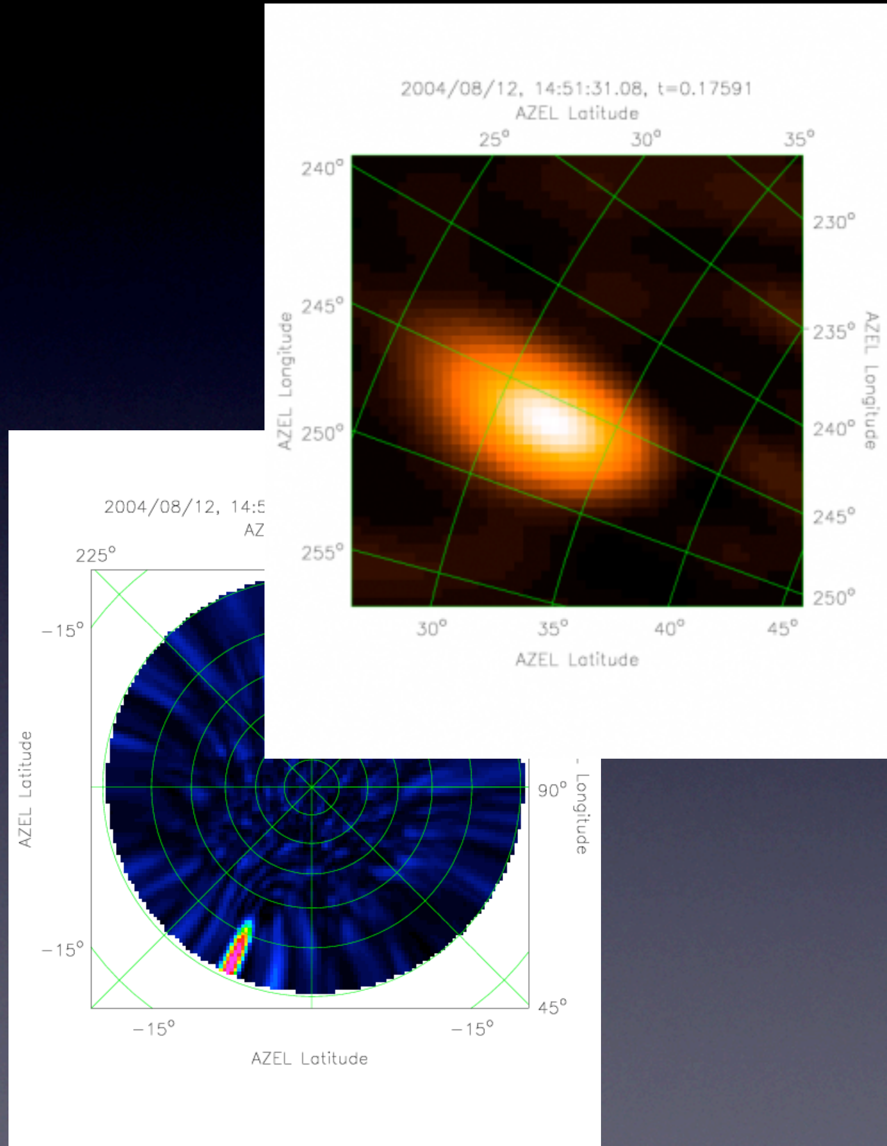
- Distributed version required to handle dumbs of all TBBs within the LOFAR array



Near-field imaging







CR-Tools: Overview

- LOPES-Tools (I) [earlier than mid-2004]
  - written for the LOPES (LOFAR Prototype Station) experiment located at the FZ Karlsruhe
  - based on AIPS++, mostly written in Glish scripting language - little C code
  - hand-written Makefiles
- LOPES-Tools (II) [2004]
  - introduction of C++
- LOPES-Tools (III) [2004-2007]
  - embedding into AIPS++ build environment as site package
- CR-Tools [fall 2007]
  - reorganization of modules with C/C++ code
  - transition to configuration/build system employed by the USG: CMake
  - source code moved to USG repository
  - start replacing Glish by Python
  - new graphical user interface under development (Qt, pyQt)

CR-Tools: Installation

- All the source code available through the USG repository

svn co http://usg.lofar.org/svn/code/trunk lofarsoft

- Configuration and build through build script using CMake

cd build; ./build.sh cr

```
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- [USG CMake configuration]
-- Hostname                   = yavanna
-- USG_ROOT                   = /Users/lars/Code/lofar/usg
-- USG_INSTALL_PREFIX         = /Users/lars/Code/lofar/usg/release
-- CMAKE_INSTALL_PREFIX       = /Users/lars/Code/lofar/usg/release
-- CMAKE_FIND_LIBRARY_PREFIXES = lib
-- CMAKE_FIND_LIBRARY_SUFFIXES = .dylib;.so;.a
```

```
lofarsoft
|-- data
|-- doc
|-- release
|-- build
|-- devel_common
|   |-- cmake
|   |-- scripts
|   `-- templates
|-- external
|   |-- casacore
|   |-- hdf5
|   `-- wcslib
`-- src
    |-- contrib
    |-- CR-Tools
    |-- DAL
    `-- pybdsm
```

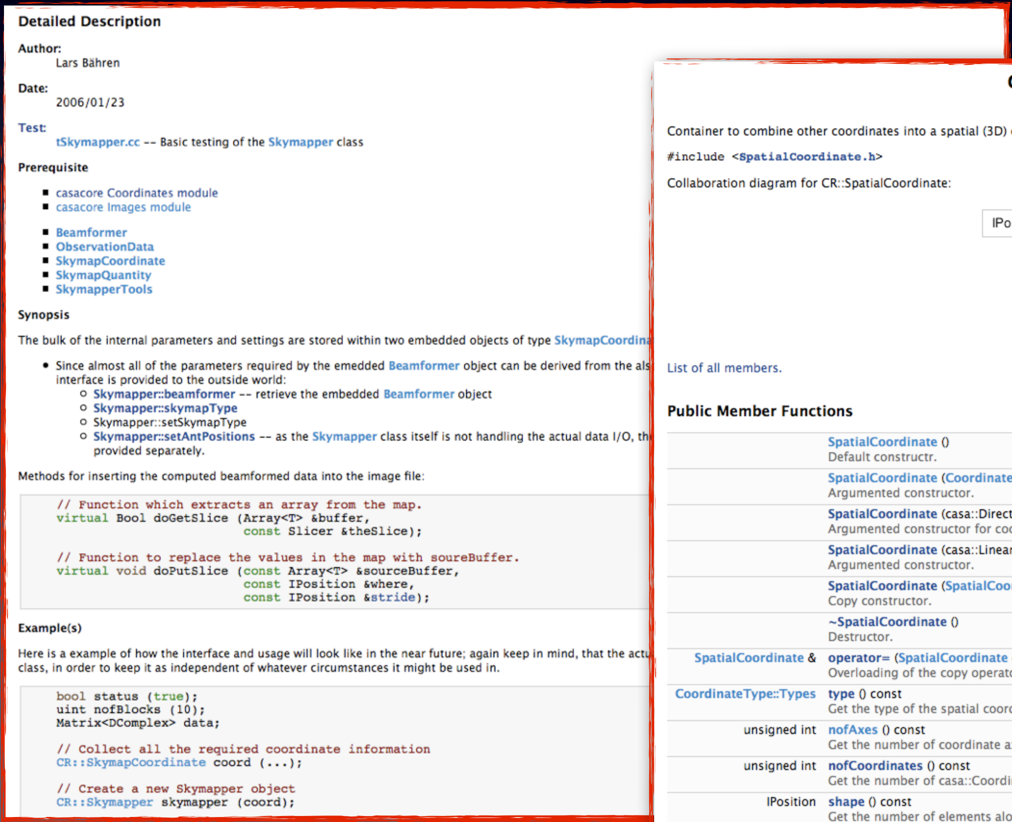
- Built components: Library, test programs, application executables
- Tested platforms: Mac OS X 10.4/10.5, Debian GNU Linux, SuSE Linux, Kubuntu

CR-Tools: Documentation

- C/C++ API Doxygen documentation:

  http://usg.lofar.org/doxygen

- Instructions/Examples on the USG Wiki



Detailed Description

Author:
Lars Bähren

Date:
2006/01/23

Test:
tSkymapper.cc -- Basic testing of the Skymapper class

Prerequisite

- casacore Coordinates module
- casacore Images module

- Beamformer
- ObservationData
- SkymapCoordinate
- SkymapQuantity
- SkymapperTools

Synopsis

The bulk of the internal parameters and settings are stored within two embedded objects of type SkymapCoordina

- Since almost all of the parameters required by the emedded Beamformer object can be derived from the als
  interface is provided to the outside world:
  - Skymapper::beamformer -- retrieve the embedded Beamformer object
  - Skymapper::skymapType
  - Skymapper::setSkymapType
  - Skymapper::setAntPositions -- as the Skymapper class itself is not handling the actual data I/O, th
    provided separately.

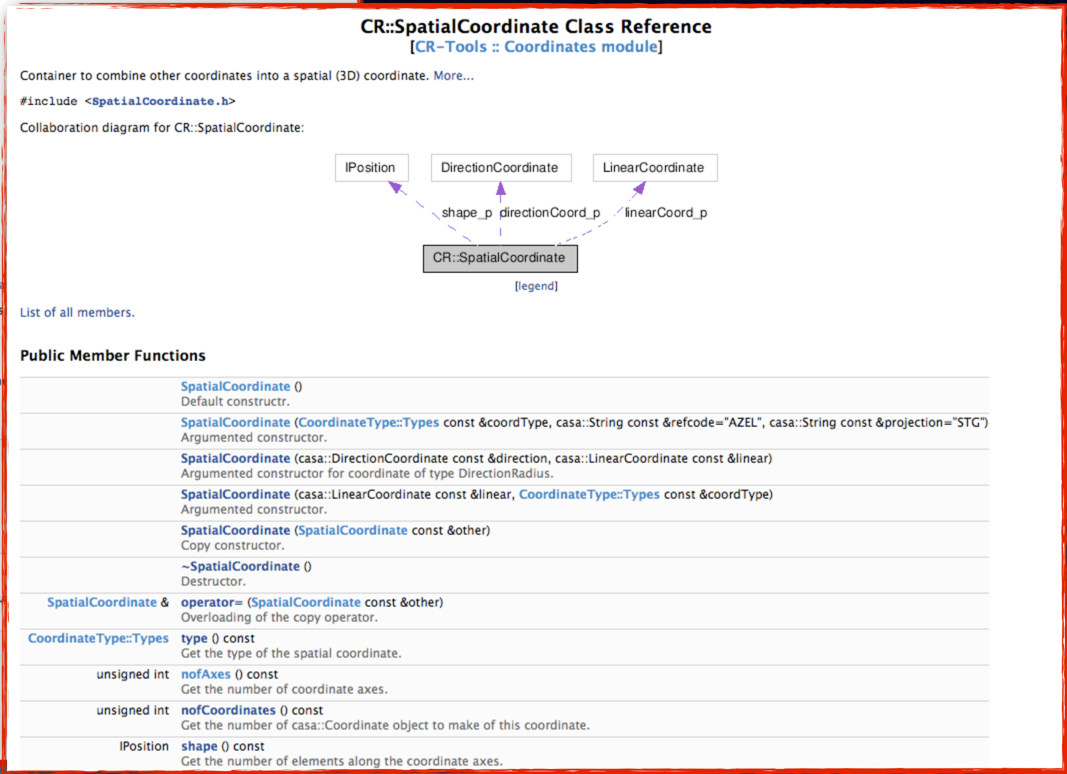
Methods for inserting the computed beamformed data into the image file:

```
// Function which extracts an array from the map.
virtual Bool doGetSlice (Array<T> &buffer,
                         const Slicer &theSlice);

// Function to replace the values in the map with soureBuffer.
virtual void doPutSlice (const Array<T> &sourceBuffer,
                         const IPosition &where,
                         const IPosition &stride);
```

Example(s)

Here is a example of how the interface and usage will look like in the near future; again keep in mind, that the actu
class, in order to keep it as independent of whatever circumstances it might be used in.

```
bool status (true);
uint nofBlocks (10);
Matrix<DComplex> data;

// Collect all the required coordinate information
CR::SkymapCoordinate coord (...);

// Create a new Skymapper object
CR::Skymapper skymapper (coord);
```



CR::SpatialCoordinate Class Reference
[CR-Tools :: Coordinates module]

Container to combine other coordinates into a spatial (3D) coordinate. More...

#include <SpatialCoordinate.h>

Collaboration diagram for CR::SpatialCoordinate:

List of all members.

Public Member Functions

	SpatialCoordinate () Default constructr.
	SpatialCoordinate (CoordinateType::Types const &coordType, casa::String const &refcode="AZEL", casa::String const &projection="STG") Argumented constructor.
	SpatialCoordinate (casa::DirectionCoordinate const &direction, casa::LinearCoordinate const &linear) Argumented constructor for coordinate of type DirectionRadius.
	SpatialCoordinate (casa::LinearCoordinate const &linear, CoordinateType::Types const &coordType) Argumented constructor.
	SpatialCoordinate (SpatialCoordinate const &other) Copy constructor.
	~SpatialCoordinate () Destructor.
SpatialCoordinate &	operator= (SpatialCoordinate const &other) Overloading of the copy operator.
CoordinateType::Types	type () const Get the type of the spatial coordinate.
unsigned int	nofAxes () const Get the number of coordinate axes.
unsigned int	nofCoordinates () const Get the number of casa::Coordinate object to make of this coordinate.
IPosition	shape () const Get the number of elements along the coordinate axes.