baltrad toolbox
data processing APIs

Daniel Michelson, SMHI

Training workshop

9 June 2011

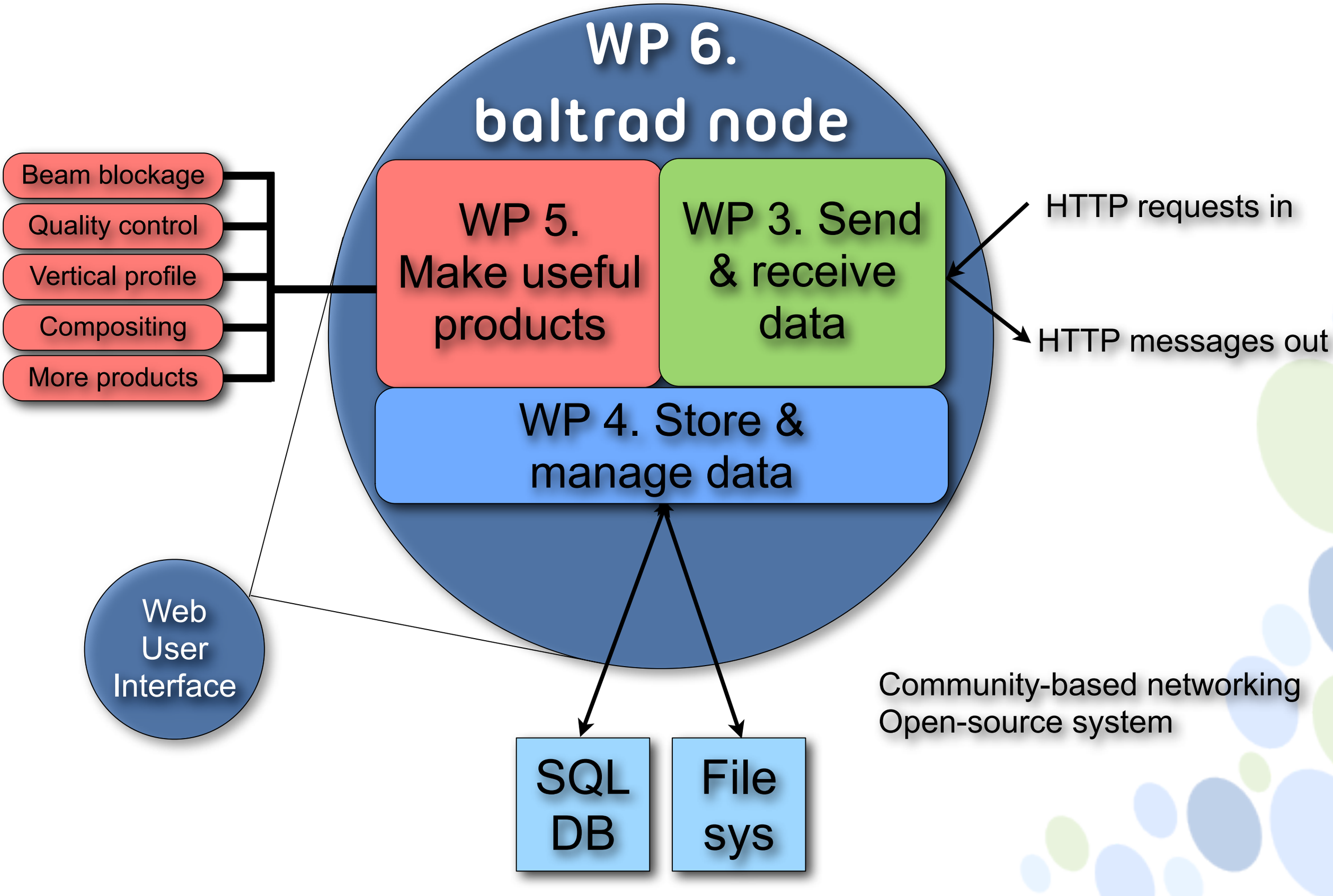
Riga, Latvia

Baltic Sea Region Programme 2007-2013

Part-financed by the European Union (European Regional Development Fund and European Neighbourhood and Partnership Instrument)

System concept

baltrad



Community-based networking
Open-source system

rave product generation framework

purpose: to provide central infrastructure for processing data

- ✓ Written in C
- ✓ Python wrappers
- ✓ Uses HDF5, PROJ.4, expat, CURL, and others
- ✓ XML-RPC server and stand-alone binaries
- ✓ PGF server is loosely integrated with node
- ✓ Built-in documentation: ‘make doc’

bottom-up approach

baltrad

in C:

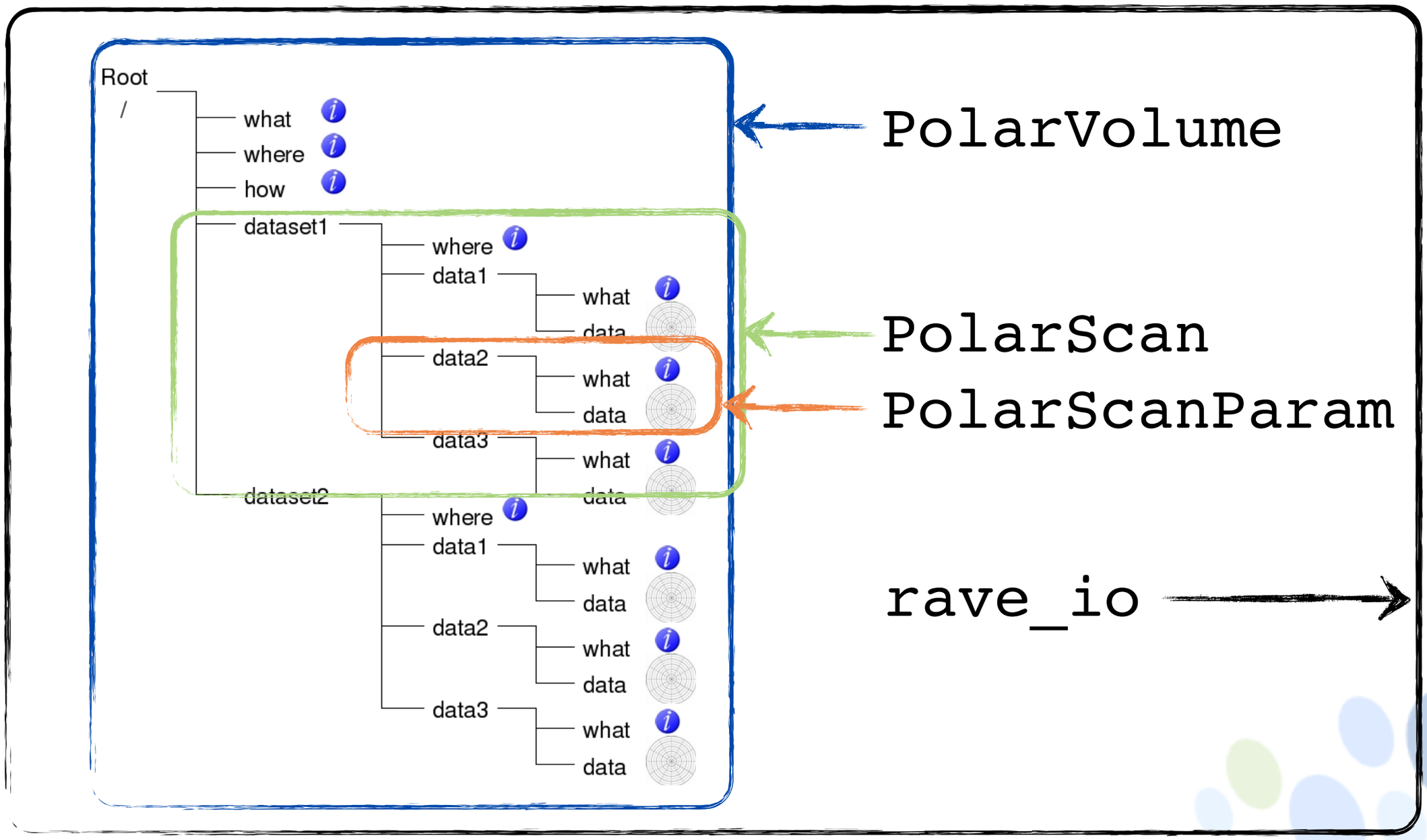
- Read ODIM_H5 file
- Access metadata and data
- Process data (e.g. navigation)
- Collect results
- Write output ODIM_H5 file

Golden rule 1: separate file I/O from the rest!

general api structure

baltrad

APIs for accessing/managing metadata & data follow ODIM_H5



read a file

baltrad

```c
#include "rave_io.h"
#include "polarvolume.h"

RaveIO_t* raveio = RaveIO_open(filename);
PolarVolume_t* volume = NULL;

if (RaveIO_getObjectType(raveio) == Rave_ObjectType_PVOL) {
   volume = (PolarVolume_t*)RaveIO_getObject(raveio);
}else{
   printf("Input file is not a polar volume. Giving up ...\n");
   RAVE_OBJECT_RELEASE(volume);
   RAVE_OBJECT_RELEASE(raveio);
   return 0;
}
/* Carrying on ... */
```

object types

baltrad

From rave_types.h, follows objects in ODIM_H5

```
typedef enum Rave_ObjectType {
  Rave_ObjectType_UNDEFINED = -1,
  Rave_ObjectType_PVOL = 0,      /* Polar volume */
  Rave_ObjectType_CVOL = 1,      /* Cartesian volume */
  Rave_ObjectType_SCAN,          /* Polar scan */
  Rave_ObjectType_RAY,           /* Single polar ray */
  Rave_ObjectType_AZIM,          /* Azimuthal object */
  Rave_ObjectType_IMAGE,         /* 2-D cartesian image */
  Rave_ObjectType_COMP,          /* Cartesian composite image(s) */
  Rave_ObjectType_XSEC,          /* 2-D vertical cross section(s) */
  Rave_ObjectType_VP,            /* 1-D vertical profile */
  Rave_ObjectType_PIC,           /* Embedded graphical image */
  Rave_ObjectType_ENDOFTYPES     /* Last entry */
} Rave_ObjectType;
```

PolarVolume.h

baltrad

```
int nscans = PolarVolume_getNumberOfScans(volume);

PolarVolume_sortByElevations(volume, 0); /* Assert descending */
PolarVolume_sortByElevations(volume, 1); /* Assert ascending */
if PolarVolume_isAscendingScans(volume) { /*1 = ascending */ }

const char* date = PolarVolume_getDate(volume);
const char* time = PolarVolume_getTime(volume);
double lon = PolarVolume_getLongitude(volume); /* radians! */
if PolarVolume_hasAttribute(volume, "how/bob") {/*1 = true*/}

PolarScan_t* scan = NULL;
scan = PolarVolume_getScan(volume, index);
scan = PolarVolume_getScanClosestToElevation(volume, e, inside);
```

PolarScan.h

baltrad

```
const char* startdate = PolarScan_getStartDate(scan);
const char* starttime = PolarScan_getStartTime(scan);
double elangle = PolarScan_getElangle(scan)*RAD2DEG; /*degrees*/
long nbins = PolarScan_getNbins(scan);
long nrays = PolarScan_getNrays(scan);
long a1gate = PolarScan_getA1gate(scan);
double rscale = PolarScan_getRscale(scan);

if PolarScan_hasParameter(scan, “DBZH”) {/* 1 = exists */}
if PolarScan_setDefaultParameter(scan, “DBZH”) {/* 1=success */}

RaveValueType t;
int bin, ray;
double* value;
t = PolarScan_getValue(scan, bin, ray, value);
if (t == RaveValueType_DATA) { DoSomething(value); }
```

value types

baltrad

From rave_types.h, follows objects in ODIM_H5

```
typedef enum RaveValueType {
  RaveValueType_UNDEFINED = -1,      /* unknown */
  RaveValueType_UNDETECT = 0,        /* undetect */
  RaveValueType_NODATA = 1,          /* nodata */
  RaveValueType_DATA = 2             /* data */
} RaveValueType;
```

PolarScanParam.h

baltrad

```
PolarScanParam_t* th = NULL;
PolarScanParam_t* dbzh = NULL;
int ray, bin;
RaveValueType th_vt, dbzh_vt;
double *th_val, *dbzh_val;

th = PolarScan_getParameter(scan, "TH");
dbzh = PolarScan_getParameter(scan, "DBZH");

/* reads raw value, e.g. 0...255 */
th_vt = PolarScanParam_getValue(th,ray,bin,th_val);

/* reads physical variable's value, e.g. -32...95 dBZ */
dbzh_vt = PolarScanParam_getConvertedValue(dbzh,ray,bin,dbzh_val);
```

PolarScanParam.h

baltrad

```
double gain = PolarScanParam_getGain(dbzh);
double offset = PolarScanParam_getOffset(dbzh);
double nodata = PolarScanParam_getNodata(dbzh);
double undetect = PolarScanParam_getUndetect(dbzh);

etc.
```

“how” attributes

baltrad

```
double tmpd;
double rpm, bob, doug;

if (!getDoubleAttribute((RaveCoreObject*)volume, "how/rpm",
                                                 &tmpd)) rpm = 2.0;
else rpm = tmpd;

if (!getDoubleAttribute((RaveCoreObject*)scan, "how/bob",
                                                 &tmpd)) bob = 0.0;
else bob = tmpd;


if (!getDoubleAttribute((RaveCoreObject*)dbzh, "how/doug",
                                                 &tmpd)) doug = 0.5;
else doug = tmpd;
```

datasets

baltrad

Loop using RAVE's functionality:

```
for (ir=0 ; ir<nrays ; ir++) {
  for (ib=0 ; ib<nbins ; ib++) {
    dbzh_vt = PolarScanParam_getConvertedValue(dbzh, ir,
                                               ib, dbzh_val);
    YourCode_OneAtATime(dbzh_val);
  }
}
```

Or roll your own with the data buffer:

```
void* dbzh_data; /* Don't release! */
dbzh_data = PolarScanParam_getData(dbzh);
YourCode_AllAtOnce(dbzh_data);
```

memory management

baltrad

rave_alloc.h

```
PolarVolume_t* volume = NULL;
...
volume = RAVE_OBJECT_NEW(&PolarVolume_TYPE);
...
RAVE_OBJECT_RELEASE(volume);
```

```
MyObject* obj = RAVE_MALLOC(sizeof(MyObject));
...
RAVE_FREE(obj);
```

cartesian navigation

baltrad

projectionregistry.h and **arearegistry.h**

projection.h and **area.h**

Remember Golden Rule 1:
separate file I/O from the rest!

Why? Because doing so enables
chaining tools in memory!

main

baltrad

```c
#include "reproj.h"

int main(int argc,char *argv[]) {
  RaveIO_t* raveio = RaveIO_open(argv[1]);
  Cartesian_t* inobj = NULL;
  Cartesian_t* result = NULL;

  if (argc<4) {
     printf("Usage: %s <input ODIM_H5 image>
                       <output ODIM_H5 image> <area_id>\n",argv[0]);
     exit(1);
  }

  inobj = (Cartesian_t*)RaveIO_getObject(raveio);
  RaveIO_close(raveio);

  result = reproj(inobj, argv[3]);  /* Re-projecting */

  RaveIO_setObject(raveio, (RaveCoreObject*)result);
  RaveIO_save(raveio, argv[2]);

  RAVE_OBJECT_RELEASE(raveio);
  RAVE_OBJECT_RELEASE(inobj);
  RAVE_OBJECT_RELEASE(result);

  exit(0);
}
```

declarations

baltrad

```
Cartesian_t* reproj(Cartesian_t* inobj, const char* areaid) {
  Cartesian_t* result = NULL;
  ProjectionRegistry_t* preg = NULL;
  AreaRegistry_t* areg = NULL;
  Projection_t* iproj = NULL;
  Projection_t* oproj = NULL;
  Area_t* oarea = NULL;
  RaveDataType dt;
  RaveValueType rvt;

  long ix=0, iy=0, ixsize=0, iysize=0;  /* input */
  long ox=0, oy=0, oxsize=0, oysize=0;  /* output */
  double herex=0.0, herey=0.0, therex=0.0, therey=0.0;
  double llX=0.0, llY=0.0, urX=0.0, urY=0.0;
  double result_val;

...
```

input geometry

baltrad

```
Cartesian_getAreaExtent(inobj, &llX, &llY, &urX, &urY);

dt = Cartesian_getDataType(inobj);

iproj = Cartesian_getProjection(inobj);

ixsize = Cartesian_getXSize(inobj);

iysize = Cartesian_getYSize(inobj);
```

output geometry

baltrad

```
preg = ProjectionRegistry_load("/path/projection_registry.xml");

areg = AreaRegistry_load("/path/area_registry.xml", preg);

oarea = AreaRegistry_getByName(areg, areaid);
```

```
result = RAVE_OBJECT_NEW(&Cartesian_TYPE);

Cartesian_init(result, oarea, dt);

oproj = Area_getProjection(oarea);

oxsize = Cartesian_getXSize(result);
oysize = Cartesian_getYSize(result);

CopyMetaData(inobj, result); /* convenience function */
```

housekeeping

baltrad

```
void CopyMetaData(Cartesian_t* source, Cartesian_t* dest) {
  Cartesian_setDate(dest, Cartesian_getDate(source));
  Cartesian_setTime(dest, Cartesian_getTime(source));
  Cartesian_setStartDate(dest, Cartesian_getStartDate(source));
  Cartesian_setStartTime(dest, Cartesian_getStartTime(source));
  Cartesian_setEndDate(dest, Cartesian_getEndDate(source));
  Cartesian_setEndTime(dest, Cartesian_getEndTime(source));
  Cartesian_setSource(dest, Cartesian_getSource(source));
  Cartesian_setObjectType(dest, Cartesian_getObjectType(source));
  Cartesian_setProduct(dest, Cartesian_getProduct(source));
  Cartesian_setQuantity(dest, Cartesian_getQuantity(source));

  Cartesian_setNodata(dest, Cartesian_getNodata(source));
  Cartesian_setUndetect(dest, Cartesian_getUndetect(source));
  Cartesian_setOffset(dest, Cartesian_getOffset(source));
  Cartesian_setGain(dest, Cartesian_getGain(source));
}
```

main loop

baltrad

```
for (oy=0;oy<oysize;oy++) {
  herey = Cartesian_getLocationY(result, oy);

  for (ox=0;ox<oxsize;ox++) {
    herex = Cartesian_getLocationX(result, ox);

    Projection_transformx(iproj, oproj, herex, herey, 0.0,
                          &therex, &therey, NULL);

    ix = Cartesian_getIndexX(inobj, therex);
    iy = Cartesian_getIndexY(inobj, therey);

    if ((ix>=0) && (iy>=0) && (ix<ixsize) && (iy<iysize)) {
      rvt = Cartesian_getValue(inobj, ix, iy, &result_val);

      Cartesian_setValue(result, ox, oy, result_val);
    }
  }
}
```

housekeeping

baltrad

```
  RAVE_OBJECT_RELEASE(preg);
  RAVE_OBJECT_RELEASE(areg);
  RAVE_OBJECT_RELEASE(iproj);
  RAVE_OBJECT_RELEASE(oproj);
  RAVE_OBJECT_RELEASE(oarea);
  return result;
}
```

finishing

```
  result = reproj(inobj, argv[3]);  /* Re-projecting */

  RaveIO_setObject(raveio, (RaveCoreObject*)result);
  RaveIO_save(raveio, argv[2]);

  RAVE_OBJECT_RELEASE(raveio);
  RAVE_OBJECT_RELEASE(inobj);
  RAVE_OBJECT_RELEASE(result);

  exit(0);
}
```

These concepts apply to:

- ✓ Polar-to-Cartesian navigation
- ✓ Polar-to-polar navigation
- ✓ Cartesian-to-polar navigation
- ✓ Cartesian-to-Cartesian navigation

Procedure for integrating new tools

✓ Write a command-line binary

✓ Write Python wrappers to the tool's functionality

✓ Document code using Doxygen

✓ Write unit tests

✓ Write Product Generation Framework plugin (Python)