baltrad

getting your tools into the baltrad toolbox

Daniel Michelson, SMHI

BUF IV, DiY workshop

21 November 2013

Berlin, Germany

Baltic Sea Region Programme 2007-2013

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

updated from 2011

baltrad

Procedure for integrating new tools:

- ✓ Write basic C functionality
- ✓ Write Python wrappers to the tool's functionality
- ✓ Write unit tests
- ✓ Write a command-line binary
- ✓ Write a so-called Quality plugin
- ✓ Write Product Generation Framework plugin
- ✓ Integrate with your node (Groovy script)

1 .“hello” C functionality

baltrad

First, download the tutorial:

$ cd ..../node-installer/packages/rave

$ wget http://git.baltrad.eu/lab/hello.tgz

$ source /opt/baltrad/etc/bltnode.rc

hello.h and hello.c in librave/toolbox

Can be placed elsewhere, e.g. radvol

baltrad

```c
#include "hello.h"

int hello(RaveIO_t* raveio) {
   RaveCoreObject* obj = NULL;
   RaveAttribute_t* att = RAVE_OBJECT_NEW(&RaveAttribute_TYPE);
   int retcode = Rave_ObjectType_UNDEFINED;

   obj = (RaveCoreObject*)RaveIO_getObject(raveio);
   retcode = RaveIO_getObjectType(raveio);

   RaveAttribute_setName(att, "how/greeting");
   RaveAttribute_setString(att, "Howzit?");

   if (retcode != Rave_ObjectType_UNDEFINED) {

      if (retcode == Rave_ObjectType_SCAN) {
         PolarScan_addAttribute((PolarScan_t*)obj, att);
      }
      else if (retcode == Rave_ObjectType_PVOL) {
         PolarVolume_addAttribute((PolarVolume_t*)obj, att);
      }
      else if (retcode == Rave_ObjectType_IMAGE) {
         Cartesian_addAttribute((Cartesian_t*)obj, att);
      }
      else if (retcode == Rave_ObjectType_VP) {
         VerticalProfile_addAttribute((VerticalProfile_t*)obj, att);
      }

   }

   RAVE_OBJECT_RELEASE(att);
   RAVE_OBJECT_RELEASE(obj);
   return retcode;
}
```

```c
#ifndef HELLO_H
#define HELLO_H
#include <stdio.h>

#include "rave_io.h"
#include "rave_types.h"
#include "rave_attribute.h"
#include "polarscan.h"
#include "polarvolume.h"
#include "cartesian.h"
#include "vertical_profile.h"

/**
 * Queries the ODIM object type for a given file and added a greeting to the
 * top-level /how group.
 * @param[in] raveio - the input RaveIO_t object
 * @returns object type, defaulting to Rave_ObjectType_UNDEFINED (-1)
 */
int hello(RaveIO_t* raveio);

#endif
```

2. Modify build directive

baltrad

In librave/toolbox/Makefile:
add hello.c to $RAVESOURCES
and
hello.h to $INSTALL_HEADERS

3. Create Python wrapper

baltrad

Add modules/pyhello.c

```c
static PyObject* _hello_func(PyObject* self, PyObject* args) {
  PyObject* object = NULL;
  PyRaveIO* pyrio = NULL;
  int retcode = Rave_ObjectType_UNDEFINED;

  if (!PyArg_ParseTuple(args, "O", &object)) {
    return NULL;
  }

  if (PyRaveIO_Check(object)) {
    pyrio = (PyRaveIO*)object;
  } else {
    raiseException_returnNULL(PyExc_AttributeError,
                              "Hello check requires RaveIO object as input");
  }

  retcode = hello(pyrio->raveio);
  return PyInt_FromLong((long)retcode);
}
```

4. Modify build directive

baltrad

In modules/Makefile

a. add SOURCE_35, OBJECTS_35, TARGET_35

b. Above ".PHO Make! d an entry for $(TARGET_35)

c. Last line, add:

-include $(SOURCE_35:%.c=$(DEPDIR)/%.P)

5. Write unit tests

baltrad

a. Create test/pytest/PyHelloTest.py

b. Import in test/pytest/RaveTestSuite.py

```
import unittest
import _raveio
import _hello

class PyHelloTest(unittest.TestCase):
    FIXTURE = "fixtures/rix_volume.h5"
    BADINPUT = "fixtures/vp_fixture.h5"

    def setUp(self):
        pass

    def tearDown(self):
        pass

    def testHello(self):
        rio = _raveio.open(self.FIXTURE)
        self.assertEquals(_hello.hello(rio), _raveio.Rave_ObjectType_PVOL)

    def testGoodbye(self):
        rio = _raveio.open(self.BADINPUT)
        self.assertNotEquals(_hello.hello(rio), _raveio.Rave_ObjectType_VP)
```

6. Write a command-line tool

baltrad

a. Create bin/hello

b. Add “hello” to “install:” directive in bin/Makefile

```
if __name__ == "__main__":
    from optparse import OptionParser

    description = "Hello tutorial"

    usage = "usage: %prog -i <infile> [-o <outfile>] [h]"
    parser = OptionParser(usage=usage, description=description)

    parser.add_option("-i", "--input", dest="ifile", help="Input file name")
    parser.add_option("-o", "--output", dest="ofile",
                      help="Output file name. Input file will be overwritten if
not specified", default=None)

    (options, args) = parser.parse_args()

    if not options.ifile:
        parser.print_help()
        sys.exit()

    if not options.ofile:
        options.ofile = options.ifile

    hello(options)
```

```
import sys
import _raveio
import _hello


def hello(options):
    rio = _raveio.open(options.ifile)

    myobj = _hello.hello(rio)

    response = "Hello, file %s contents: " % options.ifile

    if myobj is _raveio.Rave_ObjectType_SCAN:
        print response + "polar scan"
    elif myobj is _raveio.Rave_ObjectType_PVOL:
        print response + "polar volume"
    elif myobj is _raveio.Rave_ObjectType_IMAGE:
        print response + "Cartesian image"
    elif myobj is _raveio.Rave_ObjectType_COMP:
        print response + "composite"
    elif myobj is _raveio.Rave_ObjectType_VP:
        print response + "vertical profile"
    elif myobj is _raveio.Rave_ObjectType_CVOL:
        print response + "Cartesian volume"
    elif myobj is _raveio.Rave_ObjectType_AZIM:
        print response + "azimuthal object"
    elif myobj is _raveio.Rave_ObjectType_PIC:
        print response + "graphics picture"
    elif myobj is _raveio.Rave_ObjectType_XSEC:
        print response + "cross section"
    elif myobj is _raveio.Rave_ObjectType_RAY:
        print response + "ray"
    elif myobj is _raveio.Rave_ObjectType_UNDEFINED:
        print response + "unknown"

    rio.save(options.ofile)
```

baltrad

c. Test from test/pytest/fixtures/hello/in

```
$ hello -i bymin_pvol_20120226T1015Z.h5
-o bymin_hello.h5
```

7. Create quality plugin

baltrad

a. Write Lib/rave_hello_quality_plugin.py

b. Add entry to etc/rave_quality_registry.xml

```
<quality-plugin name="hello" module="rave_hello_quality_plugin" class="hello_plugin"/>
```

c. Verify using odc_toolbox, from test/pytest/fixtures/hello

$ odc_toolbox -i in -o out -q 'ropo,radvol-att,beamb,hello'

```python
from rave_quality_plugin import rave_quality_plugin

class hello_plugin(rave_quality_plugin):
  ##
  # Default constructor
  def __init__(self):
    super(hello_plugin, self).__init__()

  ##
  # @return a list containing the string se.smhi.detector.hello
  # noting that this "tool" doesn't create any quality field.
  def getQualityFields(self):
    return ["se.smhi.detector.hello"]

##
# @param obj: A RAVE IO object that should be processed.
# @return: The modified object if this quality plugin has performed changes
# to the object.
def process(self, rio):
  try:
    import _hello
    ret = _hello.hello(rio)
  except:
    pass
  return rio
```

8. Create PGF plugin

baltrad

a. Write Lib/rave_pgf_hello_plugin.py

b. Start the PGF server and register the plugin with:

$ pgf_registry -a -H http://localhost:8085/RAVE --name=eu.baltrad.beast.hello -m rave_pgf_hello_plugin -f generate -d 'Hello tutorial'

c. Verify:

$ pgf_registry -l -H http://localhost:8085/RAVE

```
import _raveio
import _hello


## Say hello.
# @param files list of files to scan. Keep in mind that each file can be
# from a different radar.
# @return nothing
def generate(files, arguments):
    for ifstr in files:
        rio = _raveio.open(ifstr)
        _hello.hello(rio)

    return None
```

baltrad

Test:

$ tail -f $RAVEROOT/etc/rave_pgf.log

from test/pytest/fixtures/hello/in

$ hello_pgf bymin_pvol_20120226T1015Z.h5

9. Integrate with Beast

baltrad

a. Write Groovy Script (etc/Hello.groovy)

b. Paste into “Routes->Create script” route

Groovy scripts documentation:

http://git.baltrad.eu/manual/beast/doc/doxygen/p_groovyrules.html

recommendations

baltrad

1. librave/toolbox contains mostly infrastructure; please keep it that way.
2. Add your tools in one of the following ways:
   a. As a separate directory under librave, e.g. radvol.
   b. As a separate package that uses RAVE as a dependency, e.g. bRopo.