YARP IDL and ROS interoperability

Two ways to define types

- Thrift IDL
  - Allows defining new types (marshalling and de-marshalling)
  - Produce code for handling rpc/services
- ROS IDL
  - Allows new types for topics and services
- See: tutorials → Using IDLs
  - http://wiki.icub.org/yarpdoc/idls.html

Two ways to define types

- Thrift IDL
  - Allows defining new types (marshalling and de-marshalling)
  - Produce code for handling rpc/services
- ROS IDL
  - Allows new types for topics and services
- See: tutorials → Using IDLs
  - http://wiki.icub.org/yarpdoc/idls.html

YARP's way (manually)

```
class Target : public Portable {
public:
int x;
int y;
virtual bool write(ConnectionWriter& connection) {
        connection.appendInt(x);
        connection.appendInt(y);
        return true;
  }

virtual bool read(ConnectionReader& connection) {
        x = connection.expectInt();
        y = connection.expectInt();
        return !connection.isError();
  }
};
```

Doing that automatically

```
struct SharedData {
1: string text;
2: list<double> content;
}
```

yarpidl_thrift --gen yarp --out ./ SharedData.thrift

Produces SharedData.cpp/SharedData.h

Writing to port…

```
#include "SharedData.h"
#include <iostream>
#include <yarp/os/Network.h>
#include <yarp/os/BufferedPort.h>
#include <yarp/os/Time.h>
yarp::os::Port port;

if (!port.open("/sender"))

while(true)
        {
                SharedData d;
                // d.text is a string
                d.text="Hello from sender";
                //d.content is a vector, let's push some data
                d.content.push_back(0.0);
                d.content.push_back(0.0);
                port.write(d);
        }
return 0;
```

Reading from port…

```
#include "SharedData.h"
#include <iostream>
#include <yarp/os/Network.h>
#include <yarp/os/BufferedPort.h>
#include <yarp/os/Time.h>
yarp::os::Port port;

if (!port.open("/receiver"))

while(true)
        {
                SharedData d;
                port.read(d);

                cout<<d.text";
                //d.content is a vector, let's pop some data
                cout<< d.content.pop();
                cout<< d.content.pop();

        }
return 0;
```

Writing a “service”

```
service Memory {
  string get_answer(1: string k);
  bool push(1:string k, 2:string v);
  bool show_list();
  bool clear();
}
```

```
yarpidl_thrift --gen yarp --out . memory.thrift
```

Produce Memory.h with method interfaces and Memory.cpp

```
class Memory : public yarp::os::Wire {
public:
  Memory();
  virtual std::string get_answer(const std::string& k);
  virtual bool push(const std::string& k, const std::string& v);
  virtual bool show_list();
  virtual bool clear();
  virtual bool read(yarp::os::ConnectionReader& connection);
  virtual std::vector<std::string> help(const std::string& functionName="--all");
};
```

```
#include <yarp/os/all.h>
#include <Memory.h>

typedef std::map<std::string, std::string> DataBase;
typedef DataBase::const_iterator DataBaseConstIterator;
typedef DataBase::iterator DataBaseIterator;

class MemoryServer : public Memory
{
   // Memory interface see Memory.thrift
   std::string get_answer(const std::string& k)
   {}
   bool push(const std::string& k, const std::string& v)
   {}
   bool show_list()
   { }
   bool clear()
   { }
};
```

How to use it

```
int main(int argc, char *argv[]) {
        yarp::os::Network yarp;
        MemoryServer memoryServer;
        yarp::os::Port port;

       memoeryServer.yarp().attachAsServer(port);

        if (!port.open("/memoryServer")) { return 1; }

        while (true) {
                printf("Server running happily\n");
                yarp::os::Time::delay(10);
        }

        port.close();
return 0;
}
```

How to use it

```
int main(int argc, char * argv[])
{
   Network yarp;

   RpcClient port;

   port.open("/client");

   Network::connect("/client", "/myModule");

   Memory client;

   client.yarp().attachAsClient(port);

   //use functions defined in Memory.h
   client.push("hello", "world");
   std::string answer=client.get_answer("hello");

   cout<<"Main returning..."<<endl;
   return 0;
}
```

Using ROS

- You can:
  - Use on roscore and configure YARP to talk to it
  - Make yarpserver syncronize with roscore
- Make types available
  - Explicitly use ROS .msg and convert them using yarp's idl
  - Run a server that makes type information available on demand

Configure *yarpserver* to talk to roscore

set ROS_MASTER_URI

yarpserver --ros

yarp read /testtopic@/testnode // open a ros topic with yarp

Check what happen on the ros side:

rostopic list # /testtopic should be listed

rosnode list # /testnode should be listed

rostopic pub /testtopic std_msgs/String "Hello YARP" # yarp read should echo this

Configure YARP to talk to roscore

…stop yarpserver…
set ROS_MASTER_URI
yarp namespace /ros //change namespace (name is not important)
yarp detect --ros --write // detect roscore and write config
yarp read /testtopic@/testnode // open a ros topic with yarp

Check what happen on the ros side:
rostopic list # /testtopic should be listed
rosnode list # /testnode should be listed
rostopic pub /testtopic std_msgs/String "Hello YARP" # yarp read should echo this

Make types available

yarpidl_rosmsg --name /typ@/yarpidl

Or

yarpidl_rosmsg --web true --name /typ@/yarpidl

Subscribing

yarp read /msg@/test_node

$ rosnode list

...

/test_node

...

$ rostopic info /msg

Type: unknown type

Publishers: None

Subscribers:

* /test_node (... address ...)

rostopic pub /msg std_msgs/String "hello yarp"

rostopic pub /msg turtlesim/Pose 0 5 10 15 20

Publishing

rosrun roscpp_tutorials listener

yarp write /chatter@/yarp_writer
Hello?

[INFO] [1386605949.838711935]: I heard: [hello?]

rosrun rospy_tutorials add_two_ints_server

$ yarp rpc /add_two_ints

22 20

Response: 42

Code…

e.g. message.msg
yarpidl_rosmsg message.msg
Generate header file, i.e. message.h

```
yarp::os::Node node("/mynode");   // added a Node
yarp::os::Subscriber<message> reader;

if (!reader.topic("/read"))
  {
     cerr<<"Error opening topic, check your network\n";
     return -1;
  }

Message msg;
reader.read(msg);
```

e.g. message.msg
yarpidl_rosmsg message.msg
Generate header file, i.e. message.h

```
yarp::os::Node node("/mynode");   // added a Node
yarp::os::Publisher<message> writer;

if (!cmd.topic("/writer"))
  {
     cerr<<"Error opening topic, check your network\n";
     return -1;
  }


Message msg;
writer.write(msg);
```