Day 1 Finals

```
[15:00:58][INFO]        "processed_processor1String"
[15:06:53][INFO]        "Captain Falcons_processor1String"
[15:08:22][INFO]        "Someones modified messageprocessor1"
[16:21:09][INFO]        "klausg_processor1String"
[16:21:26][INFO]        "Nasenbaer"
[16:22:42][INFO]        "Dennis"
[16:24:21][INFO]        “Yong_processor1String"
[16:24:56][INFO]        "Hello"
[16:25:24][INFO]        "DoS attack"
[16:27:17][INFO]        "Tobias_proceesed_processor1String"
[16:27:51][INFO]        "Parinya_processed_processor1String"
[16:29:04][INFO]        " walter modified processor1String"
[16:29:18][INFO]        ":)"
[16:31:35][INFO]        “processed_processor_aBanjongString"
[16:45:33][INFO]        "processed_TawanchatString"
```

Day 2

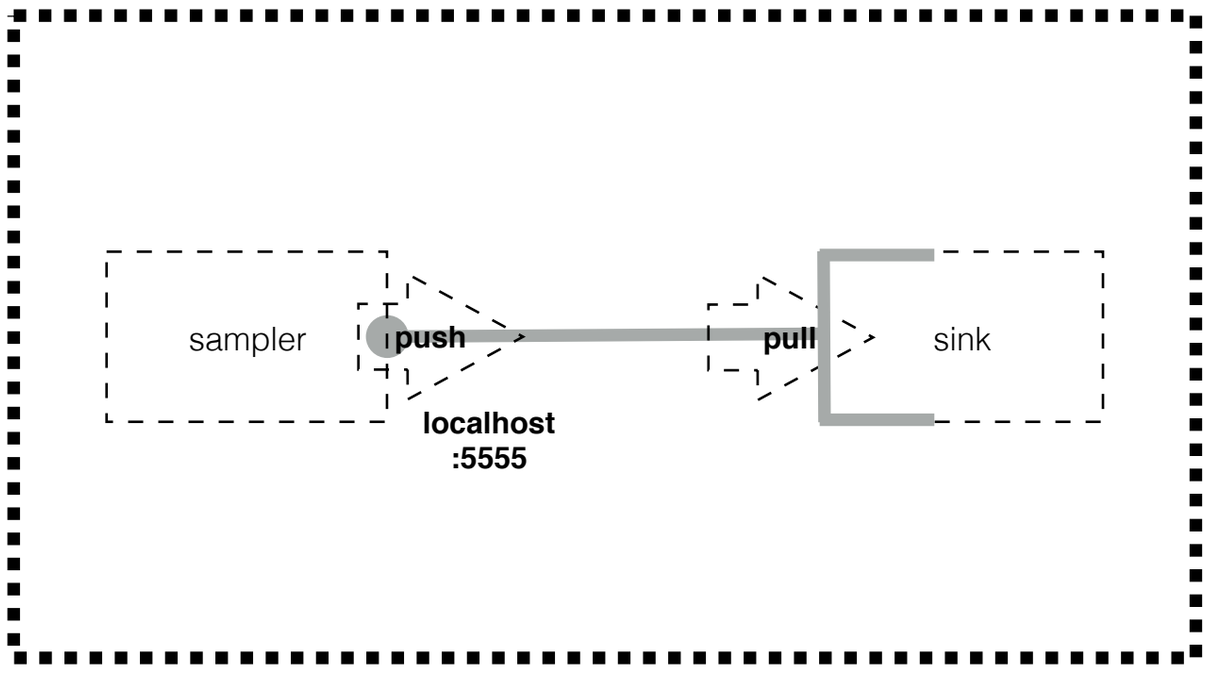
- transport root data (TMessage)
- multipart message
- parmq server

```
panda@panda-workshop:~/workshop/PandaRoot-trunk/$ mv MQ_TMessage MQ_Message
panda@panda-workshop:~/workshop/PandaRoot-trunk/$ wget http://web-docs.gsi.de/~karabowi/thailand/MQ_TMessage.tgz
panda@panda-workshop:~/workshop/PandaRoot-trunk/$ tar xvzf MQ_TMessage.tgz
```

add_subdirectory (MQ_TMessage) to the main CMakeLists.txt
do `. ./config.sh` and `make` in the pandaroot build directory;

Day 2. Excercise1

- create sampler and sink to send/receive a TMessage with TClonesArray
- create CMakeLists.txt (library, executables)
- create topology consisting of two devices:
  - sampler1 (binding transport channel to push on this channel)
  - sink1 (connecting to said channel to pull from this channel)



Implement sampler

- get the TClonesArray from an input root file
- wrap in the TMessage and send:

```
TMessage* message = new TMessage(kMESS_OBJECT);

message->WriteObject(fObject);
```

create a TMessage

write TObject (f.e. TClonesArray) to it

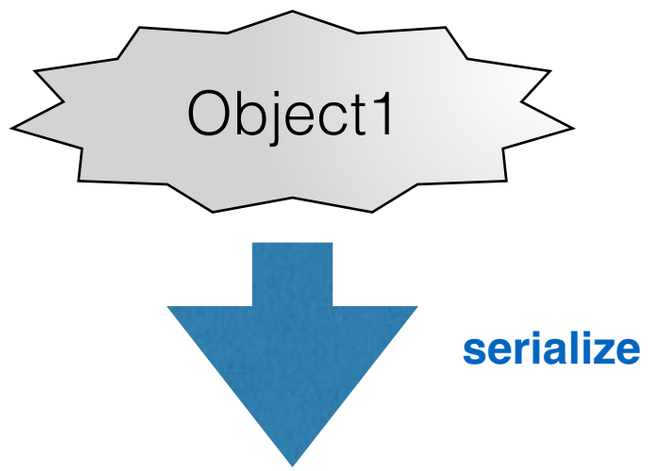
```
FairMQMessagePtr msg(NewMessage(message->Buffer(),
                                message->BufferSize(),
                                [](void* /*data*/, void* object){delete (TMessage*)(object);},
                                message));

Send(msg, "data");
```

create a FairMQMessage

send it

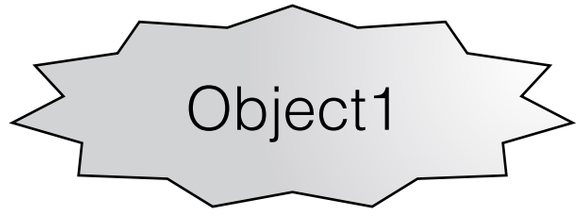
Serialization

- Object in memory



- Byte stream

- Object in memory



Check*

- on the receiver side (sink), you can just convert the message to string;
- —> use the existing sink (for string) to print it out;

```
build$ ../bin/samplerTMessage --id sampler1 --mq-config ~/workshop/PandaRoot_trunk/MQ_samp_sink/options/MQ1_sampler_sink.json --file-name ~/workshop/PandaRoot_trunk/trunk/macro/run/sim_complete.root --branch-name STTPoint
```

```
build$ ./bin/mq1-sink --id sink1 --mq-config ~/workshop/PandaRoot_trunk/MQ_samp_sink/options/MQ1_sampler_sink.json
```

Implement sink

- receive TMessage…

```
TMessage recMessage(msg->GetData(), msg->GetSize());
TObject*  recObject = (TObject*)recMessage.ReadObject(recMessage.GetClass());

LOG(INFO) << "message class is \"" << recMessage.GetClass() << "\" "
          << "(\"" << recMessage.GetClass()->GetName() << "\" : “
          << "\”" << recMessage.GetClass()->GetTitle() << "\")";
LOG(INFO) << "object name is \"" << recObject->GetName() << "\"";

if ( strcmp( recMessage.GetClass()->GetName(),"TClonesArray" ) == 0 ) {
  LOG(INFO) << " the TClonesArray has " << ((TClonesArray*)recObject)->GetEntries() << " entries.";
}
else {
  LOG(INFO) << "\"" << recMessage.GetClass()->GetName() << "\" != \"TClonesArray\"";
```

PndTMessage

- workaround for the lack of TMessage's public constructor:

```
panda@panda-workshop:~/workshop/PandaRoot_trunk/MQ_TMessage/devices$ cat PndTMessage.h
#ifndef PNDTMESSAGE_H_
#define PNDTMESSAGE_H_

#include "TMessage.h"

class PndTMessage : public TMessage
{
 public:
  PndTMessage(void* buf, Int_t len);
};

#endif /* PNDTMESSAGE_H_ */
panda@panda-workshop:~/workshop/PandaRoot_trunk/MQ_TMessage/devices$ cat PndTMessage.cxx
#include "PndTMessage.h"

PndTMessage::PndTMessage(void* buf, Int_t len)
  : TMessage(buf, len)
{
  ResetBit(kIsOwner);
}
panda@panda-workshop:~/workshop/PandaRoot_trunk/MQ_TMessage/devices$ grep PndTMessage
PndMQSinkTMessage.cxx
devices/PndMQSinkTMessage.cxx:#include "PndTMessage.h"
devices/PndMQSinkTMessage.cxx:  PndTMessage recMessage(msg->GetData(), msg->GetSize());
```

Check

```
build$ ../bin/samplerTMessage --id sampler1 --mq-config ~/workshop/PandaRoot_trunk/MQ_samp_sink/options/MQ1_sampler_sink.json --file-name ~/workshop/PandaRoot_trunk/trunk/macro/run/sim_complete.root --branch-name STTPoint
```

```
build$ ./bin/sinkTMessage --id sink1 --mq-config ~/workshop/PandaRoot_trunk/MQ_samp_sink/options/MQ1_sampler_sink.json
```

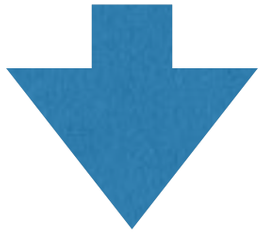
Processor

- receives TMessage with input TClonesArray;
- sends TMessage with output TClonesArray;

```
build$ ../bin/samplerTMessage --id sampler1 --mq-config ~/workshop/PandaRoot_trunk/MQ_samp_sink/options/MQ1_samp_proc_sink.json --file-name ~/workshop/PandaRoot_trunk/trunk/macro/run/sim_complete.root --branch-name STTPoint
```

```
build$ ./bin/processorTMessage --id processor1 --mq-config ~/workshop/PandaRoot_trunk/MQ_samp_sink/options/MQ1_samp_proc_sink.json
```

```
build$ ./bin/sinkTMessage --id sink1 --mq-config ~/workshop/PandaRoot_trunk/MQ_samp_sink/options/MQ1_samp_proc_sink.json
```



```
build$ ../bin/samplerTMessage --id sampler1 --mq-config ~/workshop/PandaRoot_trunk/MQ_samp_sink/options/MQ1_samp_proc_sink.json --file-name ~/workshop/PandaRoot_trunk/trunk/macro/run/sim_complete.root --branch-name MVDPoint
```

What about the params?

- the processor needs access to the parameters -> parmq-server
- access to parmq-server device -> request - reply pattern
- which parameters to request -> info available in the event header
- send event header to processor -> use multi-part message

```
TMessage* messageEH  = new TMessage(kMESS_OBJECT);
TMessage* messageTCA = new TMessage(kMESS_OBJECT);

messageEH ->WriteObject(fEventHeader);
messageTCA->WriteObject(fObject);

FairMQParts parts;
parts.AddPart(NewMessage(messageEH ->Buffer(),
                         messageEH ->BufferSize(),
                         [](void* /*data*/, void* object){delete (TMessage*)(object);},
                         messageEH ));
parts.AddPart(NewMessage(messageTCA->Buffer(),
                         messageTCA->BufferSize(),
                         [](void* /*data*/, void* object){delete (TMessage*)(object);},
                         messageTCA));

Send(parts, "data");
```

What about the params?

- receive parts:

```
bool PndMQProcessorTMessage::ProcessData(FairMQParts& parts, int /*index*/)
{
  TObject* tempObjects[10];
  for ( int ipart = 0 ; ipart < parts.Size() ; ipart++ )
    {
      PndTMessage tm(parts.At(ipart)->GetData(), parts.At(ipart)->GetSize());
      tempObjects[ipart] = (TObject*)tm.ReadObject(tm.GetClass());
      if ( strcmp(tempObjects[ipart]->GetName(),"EventHeader.") == 0 )
        {
          fEventHeader = (FairEventHeader*)tempObjects[ipart];
          fNewRunId = fEventHeader->GetRunId();

         if(fNewRunId!=fCurrentRunId)
            {
              fCurrentRunId=fNewRunId;
              UpdateParameters();
            }
        }
      else
        {
          // do proper things
        }
  }
}
```

What about the params?

- receive parts:

```
PndMQProcessorTMessage
 std::string paramName = thisPar->GetName();
  //  boost::this_thread::sleep(boost::posix_time::milliseconds(1000));
  std::string* reqStr = new std::string(paramName + "," + std::to_string(fCurrentRunId));
  LOG(WARN) << "Requesting parameter \"" << paramName << "\" for Run ID " << fCurrentRunId
            << " (" << thisPar << ")";
std::unique_ptr<FairMQMessage> req(NewMessage(const_cast<char*>(reqStr->c_str()),
                                       reqStr->length(),
                                       [](void* /*data*/, void *hint) {delete (std::string*)hint},
                                       reqStr));
std::unique_ptr<FairMQMessage> rep(NewMessage());

if (Send(req,fParamChannelName) > 0)
  {
    if (Receive(rep,fParamChannelName) > 0)
      {
        PndTMessage tm(rep->GetData(), rep->GetSize());
        thisPar = (FairParGenericSet*)tm.ReadObject(tm.GetClass());
        LOG(WARN) << "Received parameter"<< paramName <<" from the server (" << thisPar << ")";
        return thisPar;
      }
  }
return NULL;
```

more examples in https://github.com/FairRootGroup/FairRoot/tree/dev/examples/advanced/Tutorial3

Binary transport

```
int numEntries = fInput->GetEntriesFast();
```

check how many FairTestDetectorDigi in fInput TClonesArray

```
size_t digisSize = numEntries * sizeof(TestDetectorPayload::Digi);
```

struct TestDetectorPayload::Digi

```
fPayload = FairMQMessagePtr(fTransportFactory->CreateMessage(digisSize));
```

create empty message with a given size

```
TestDetectorPayload::Digi* digiPayload = static_cast<TestDetectorPayload::Digi*>(fPayload->GetData());
```

declare an array of TestDetectorPayload::Digi in the place of your message

```
for (int i = 0; i < numEntries; ++i)
{
    FairTestDetectorDigi* digi = static_cast<FairTestDetectorDigi*>(fInput->At(i));
    if (!digi)
    {
        continue;
    }
    new (&digiPayload[i]) TestDetectorPayload::Digi();
    digiPayload[i] = TestDetectorPayload::Digi();
    digiPayload[i].fX = digi->GetX();
    digiPayload[i].fY = digi->GetY();
    digiPayload[i].fZ = digi->GetZ();
    digiPayload[i].fTimeStamp = digi->GetTimeStamp();
    digiPayload[i].fTimeStampError = digi->GetTimeStampError();
}
```

fill the array

boost transport

```
std::ostringstream oss;
TPayloadOut OutputArchive(oss);
for (int i = 0; i < fInput->GetEntriesFast(); ++i)
{
    TOut* digi = static_cast<TOut*>(fInput->At(i));
    if (!digi)
    {
        continue;
    }
    fDigiVector.push_back(*digi);
}
```

```
OutputArchive << fDigiVector;
std::string* strMsg = new std::string(oss.str());
fPayload=FairMQMessagePtr(NewMessage(const_cast<char*>(strMsg->c_str()), // data
                                     strMsg->length(), // size
                                     [](void* /*data*/,void* obj){delete static_cast<std::string*>(obj)
                                     strMsg)); // object that manages the data

fDigiVector.clear();
```

- auto boost serialization, offered thanks to an addition in data classes:

```
template <class Archive>
void serialize(Archive& ar, const unsigned int /*version*/)
{
    ar& boost::serialization::base_object<FairTimeStamp>(*this);
    ar& fX;
    ar& fY;
    ar& fZ;
}
```

Day 3

- brainstorming

- how to have one task that could be used both in FairRoot and in FairMQ?

Day 3. MQ/example9

- one idea (MQ/example9): implement three functions in a given FairTask:
  - `virtual void GetParList(TList* tempList);` to allow device to obtain list of parameters needed by a task
- `virtual void InitMQ (TList* tempList);` to supply the parameters received by device to the task
- `virtual void ExecMQ (TList* inputList,TList* outputList);` to run the Task::Exec on the input data (inside inputList) and create output data (transferred via outputList)

Day 3. ***FairDataManager*** ***FairParameterManager***

- currently we have hundrends of tasks that are running in FairRoot.
- they communicate with the framework scheleton via FairRootManager and FairRuntimeDb
- maybe replace them with:
- FairDataManager - it should simply be doing job of FairRootManager in FairRoot. In FairMQ, the data received by a device would be put in FairDataManager, so that it can be accessed in task
- FairParameterManager - it should simply be doing job of FairRuntimeDb in FairRoot. In FairMQ, the task will obtain the parameters via this FairParameterManager….