Manuscript Number:

Title: Data Acquisition Backbone Core DABC release v1.0

Article Type: Contributed

Corresponding Author: Dr. Hans G. Essel,

Corresponding Author's Institution: GSI

First Author: Hans G Essel, Dr.

Order of Authors: Hans G Essel, Dr.; Jörn Adamczewski-Musch, Dr.; Nikolaus Kurz, Dr.; Sergey Linev, Dr.

Data Acquisition Backbone Core DABC release v1.0

J Adamczewski-Musch, H G Essel[1], N Kurz and S Linev
GSI Helmholtzzentrum für Schwerionenforschung,
Planckstr. 1, 64291 Darmstadt, Germany

E-mail: H.Essel@gsi.de

Abstract. The Data Acquisition Backbone Core (DABC) is a general purpose software framework designed for the implementation of a wide-range of data acquisition systems – from various small detector test beds to high performance systems. DABC consists of a compact data-flow kernel and a number of plug-ins for various functional components like data inputs, device drivers, user functional modules and applications. DABC provides configurable components for implementing event building over fast networks like InfiniBand or Gigabit Ethernet. A generic Java GUI provides the dynamic control and visualization of control parameters and commands, provided by DIM servers. A first set of application plug-ins has been implemented to use DABC as event builder for the front-end components of the GSI standard DAQ system MBS (Multi Branch System). Another application covers the connection to DAQ readout chains from detector front-end boards (N-XYTER) linked to read-out controller boards (ROC) over UDP into DABC for event building, archiving and data serving. This was applied for data taking in the September 2008 test beamtime for the CBM experiment at GSI. DABC version 1.0 is released and available from the website.

1. Introduction
After the design phase of data acquisition concepts for the new experiments of the FAIR facility at GSI it was necessary to develop a DAQ framework for two main use cases. First as a prototype of a high speed network event building system, second as a test bed for the full readout chain from detectors, digitizers, readout controllers, data combiners to event building, filtering, and archiving. The concept of self-triggered front-end systems includes a very precise time distribution system. The sorting of the data over the network will probably be not based on events, which are not yet defined at that point, but rather on data packets of time slices. Only after the composition of these time slice fragments at the receiver nodes of the network an event definition will be possible. After that event data can be processed for filtering, compression, on-line analysis and storage. This processing might need large processing farms able to handle high data flows. A first layout of such a system has been published in [1].
The requirements of the test bed use case turned out to be the same as for a general purpose DAQ framework, because a variety of front-ends must be integrated, some from legacy systems, others not yet ready or even known. An overall description of DABC and some performance results have been published in [2][3].

[1] Corresponding author.

2. DABC architecture

The main concept of the DABC design was to organize the data flow between various components of a distributed system in an efficient way. In addition it was necessary to provide mechanisms to connect arbitrary data sources, i.e. front-end systems, to DABC. These front-end systems are not yet known in detail. Therefore the DABC does not cover the development of front-ends. Specific front-ends must be integrated by specific software components which are plugged into the DABC system. This feature ensures that the DABC is a general purpose DAQ backbone for various DAQ applications.

2.1. Multithreading

Current development of computer architecture involves more and more CPU cores on one node. To use this new facility efficiently, application code should run in multiple threads. The main problem of any multi-threaded application is the correct locking of shared resources and synchronization of jobs, running in different threads. In DABC this is solved by event driven execution queues in each thread which can be triggered from other threads. Waiting for a resource means waiting for an event among others. The threads never block.

2.2. Memory management

Another important component of DABC design is memory management. Modern data acquisition systems should handle hundreds of Mbytes of data per second on each computing node. At the same time DAQ components usually perform little transformation of received data before delivering it further. During such data handling one should avoid memory copy as much as possible. DABC provides a memory management which allows zero-copy data buffer transfers (by reference) between threads and administrating buffers for DMA transfers.

2.3. Modules

All application processing code runs in *Module* objects. There are two general types of modules: synchronous and asynchronous. In asynchronous modules code is implemented in callback functions called by events of the event queue. Several modules can "run" in one thread. In synchronous modules code is implemented as main loop. Only one can execute per thread.

2.4. Ports and transports

Buffers are entering and leaving a module through *Port* objects. Each port has a buffer queue of configurable length. A module may have several input, output, or bidirectional ports. Outside the modules the ports are connected to *Transport* objects. The transport object is responsible either to get data from its data source or deliver data to data sink. There is a special local transport subclass, which transfers data (which means references) between two modules in the same process (address space). For connection of modules on different nodes a network transport should be used. Up to now there are implementations for Ethernet (TCP, UDP) and InfiniB (OFED verbs library [4]). To instantiate and configure transports of special kinds, *Device* classes are introduced in DABC.

2.5. Commands concept

Command execution in DABC involves two sides: First the code from where a *Command* is sent (caller), and second the code where the command is executed (receiver). A command object is just a container for parameters (including the command name) and does not contain any code for its execution. Such code must be implemented in command receiver. Such approach makes it easy to let commands be executed on other nodes without the need of code transfers over the network.

2.6. Parameters
A module may register *Parameter* objects. Parameters are accessible by name; their values can be monitored and optionally changed by the control system. Initial parameter values can be set from XML configuration files.

2.7. Manager
The manager is the central object of DABC. It manages objects in a tree-like structure with subfolders. Most components of DABC have names and can be accessed by these names via manager methods. The manager also dispatches commands to their receivers.

2.8. Plug-ins
The relation of packages is shown in figure 1. Main DABC classes are collected in one library called *libDabcBase*. This library includes the complete data flow core and is independent from any external libraries. To integrate user or application specific functionality into that code, a plug-in approach is used. DABC provides factory classes, which can be used for instantiation of custom components like modules, transports, devices, and application. Several functional components, distributed together with DABC, are implemented in form of such plug-ins.

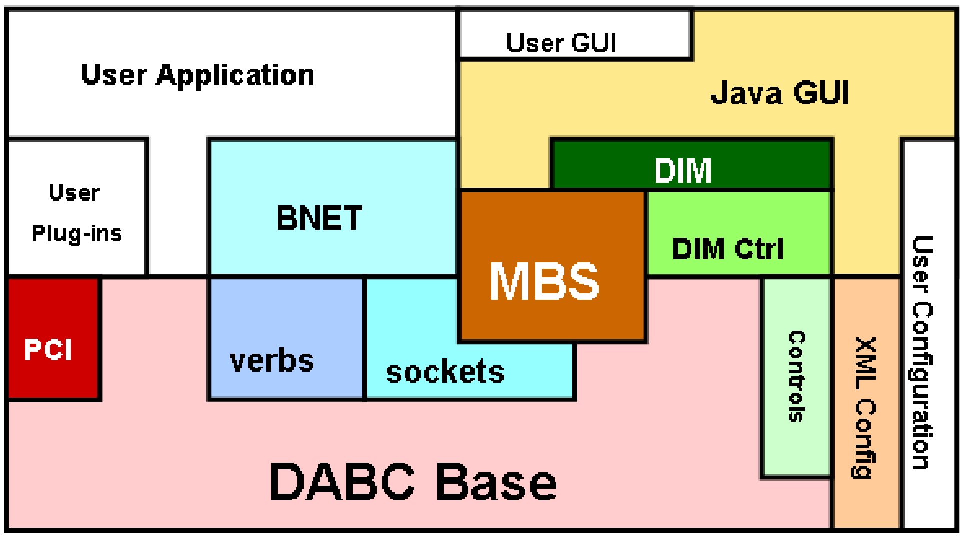


Figure 1. Structure of packages.

Plug-ins distributed are:
- Verbs – thread, transport and device implementation for InfiniBand.
- BNet – components for constructing multi-node event building network.
- MBS – transport and modules for connection to standard GSI data acquisition system [5].
- Bnet/MBS – concrete implementation of Bnet for MBS.
- CBM ROC – transport and modules for connecting CBM readout controller via Ethernet.
- PCIe/mprace – device and transport classes for DMA transfer from/to PCIe board.
- DIMctrl – control via DIM interface [6].
- Slimctrl – simple, batch-like control for test purposes.

2.9. Configuration
An important part of any software system (not only DAQ) is the ability to configure programmable components. The configuration of any component in DABC is done using parameter objects.

In the constructor of module classes one can create parameters, which can be later used for configuration purposes. Values of these parameters can be defined from configuration XML files with very simple and clear syntax (see figure 2). If not found in an XML file, a specified default value of the parameter will be used during configuration. The syntax of the XML files allows to define variables and defaults values. A value for many parameters can be set at once making the layout of such file very compact. For instance, the configuration file for a four-nodes network test application is shown in figure 3.

2.10. Control and GUI

The control system of DABC communicates over defined interfaces with the core system. Several implementations of the control system could be used. Currently a simple control system for batch like operation is provided, and a DIM based for interactiv operation. A generic GUI based on a Java DIM client library visualizes all services provided by the DIM servers. The control system of DABC and/or the application specific plug-ins can define commands. These commands are encoded as DIM services including a full description of arguments. Therefore the GUI can build up at runtime a command tree and provide the proper forms for each command. Parameters can be data structures. There are some defined structures for special parameters like rate meters, states and others. These are recognized by the GUI and handled properly. The GUI itself can be configured. The visibility of parameters and attributes of their visualization can be modified interactively and stored in GUI setup files. Application plug-ins may need dedicated panels taylored for convenient operation. Such GUI elements can be easily integrated by subclassing and interface implementation. These application panels have full access to all DIM parameters. One can connect call back functions to parameter updates, get a list of available commands, use the graphical primitives like rate meters in own panels.

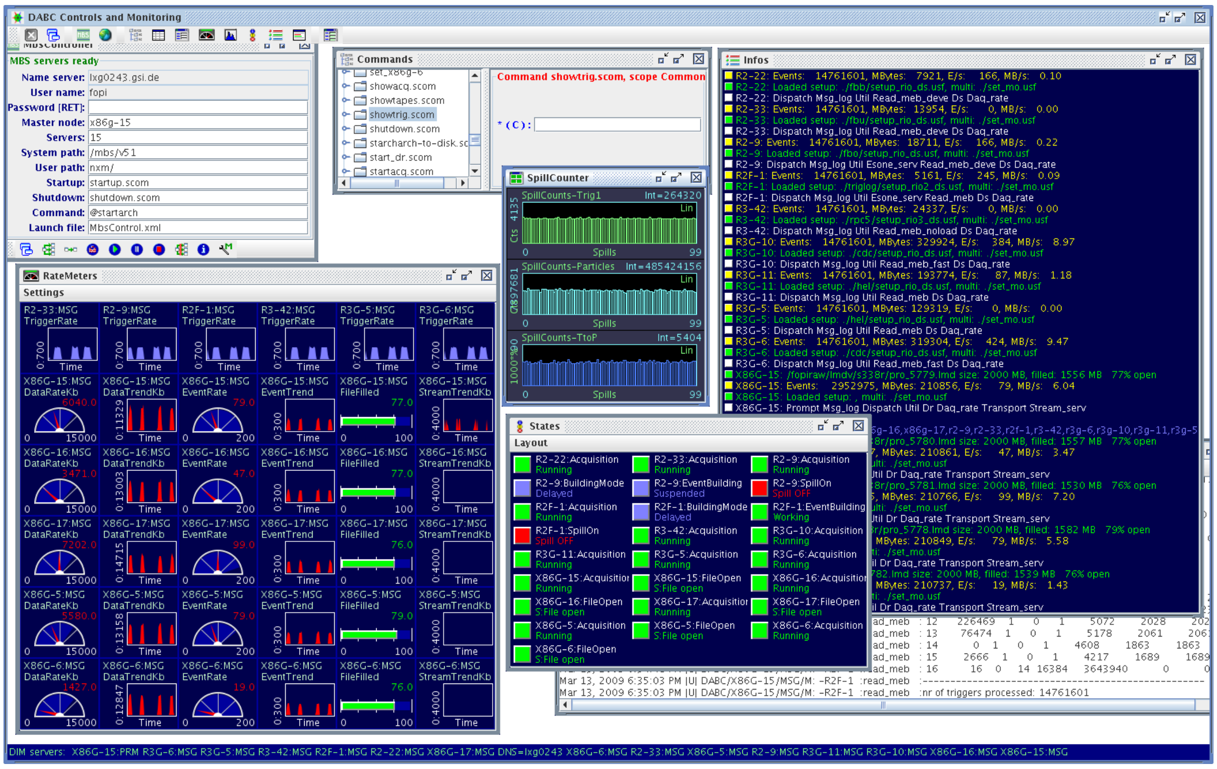
```
<?xml version="1.0"?>
<dabc version="1">
<Context name="Readout">
  <Run>
    <lib value="libDabcMbs.so"/>
    <lib value="libDabcKnut.so"/>
  </Run>
  <Application class="roc::Readout">
    <DoCalibr value="0"/>
    <NumRocs value="3"/>
    <RocIp0 value="cbmtest01"/>
    <RocIp1 value="cbmtest02"/>
    <RocIp2 value="cbmtest04"/>
    <BufferSize value="65536"/>
    <NumBuffers value="100"/>
    <TransportWindow value="30"/>
    <RawFile value="run090.lmd"/>
    <MbsServerKind value="Stream"/>
    <CalibrFile value=""/>
    <MbsFileSizeLimit value="110"/>
  </Application>
 </Context>
</dabc>
```

Figure 2. Configuration file for ROC readout application.

```
<?xml version="1.0"?>
<dabc version="1">
 <Context host="node01"/>
 <Context host="node02"/>
 <Context host="node03"/>
 <Context host="node04"/>
 <Defaults>
  <Context name="*">
   <Run>
     <lib value="libDabcNetTest.so"/>
     <runfunc value="RunAllToAll"/>
     <logfile value="${Context}.log"/>
   </Run>
   <User>
     <BufferSize value="8192"/>
     <InputQueueSize value="8"/>
     <OutputQueueSize value="4"/>
   </User>
  </Context>
 </Defaults>
</dabc>
```

Figure 3. Configuration file for net test example for four nodes.

Figure 4 shows a screen shot of a running experiment. However, no DABC system is controlled, but the standard GSI DAQ system MBS which uses the DABC conventions for DIM services. Therefore it can be controlled by the DABC GUI ad hoc. Any other system implementing such DIM services can be controlled by the GUI.



MBS servers ready

Name server:	lxg0243.gsi.de
User name:	fopi
Password [RET]:	
Master node:	x86g-15
Servers:	15
System path:	/mbs/v51
User path:	nxm/
Startup:	startup.scom
Shutdown:	shutdown.scom
Command:	@startarch
Launch file:	MbsControl.xml

Command showtrig.scom, scope Common

```
R2-22: Events:   14761601, MBytes:   7921, E/s:    166, MB/s:   0.10
R2-22: Loaded setup: ./fbb/setup_rio_ds.usf, multi: ./set_mo.usf
R2-22: Dispatch Msg_log Util Read_meb_deve Ds Daq_rate
R2-33: Events:   14761601, MBytes:  13954, E/s:      0, MB/s:   0.00
R2-33: Loaded setup: ./fbu/setup_rio_ds.usf, multi: ./set_mo.usf
R2-33: Dispatch Msg_log Util Read_meb_deve Ds Daq_rate
R2-9: Events:   14761601, MBytes:  18711, E/s:    166, MB/s:   0.22
R2-9: Loaded setup: ./fbo/setup_rio_ds.usf, multi: ./set_mo.usf
R2-9: Dispatch Msg_log Util Esone_serv Read_meb_deve Ds Daq_rate
R2F-1: Events:   14761601, MBytes:   5161, E/s:    245, MB/s:   0.09
R2F-1: Loaded setup: ./triglog/setup_rio2_ds.usf, multi: ./set_mo.usf
R2F-1: Dispatch Msg_log Util Esone_serv Read_meb Ds Daq_rate
R3-42: Events:   14761601, MBytes:  24337, E/s:      0, MB/s:   0.00
R3-42: Loaded setup: ./rpc5/setup_rio3_ds.usf, multi: ./set_mo.usf
R3-42: Dispatch Msg_log Util Read_meb_noload Ds Daq_rate
R3G-10: Events:   14761601, MBytes: 329924, E/s:    384, MB/s:   8.97
R3G-10: Loaded setup: ./cdc/setup_rio_ds.usf, multi: ./set_mo.usf
R3G-10: Dispatch Msg_log Util Read_meb_fast Ds Daq_rate
R3G-11: Events:   14761601, MBytes: 193774, E/s:     87, MB/s:   1.18
R3G-11: Loaded setup: ./hel/setup_rio_ds.usf, multi: ./set_mo.usf
R3G-11: Dispatch Msg_log Util Read_meb Ds Daq_rate
R3G-5: Events:   14761601, MBytes: 129319, E/s:      0, MB/s:   0.00
R3G-5: Loaded setup: ./hel/setup_rio_ds.usf, multi: ./set_mo.usf
R3G-5: Dispatch Msg_log Util Read_meb Ds Daq_rate
R3G-6: Events:   14761601, MBytes: 319304, E/s:    424, MB/s:   9.47
R3G-6: Loaded setup: ./cdc/setup_rio_ds.usf, multi: ./set_mo.usf
R3G-6: Dispatch Msg_log Util Read_meb_fast Ds Daq_rate
X86G-15: /fopiraw/lmdv/s338r/pro_5779.lmd size: 2000 MB, filled: 1556 MB   77% open
X86G-15: Events:   2952975, MBytes: 210856, E/s:     79, MB/s:   6.04
X86G-15: Loaded setup: , multi: ./set_mo.usf
X86G-15: Prompt Msg_log Dispatch Util Dr Daq_rate Transport Stream_serv
6g-16,x86g-17,r2-9,r2-33,r2f-1,r3-42,r3g-6,r3g-10,r3g-11,r3g-5
8r/pro_5780.lmd size: 2000 MB, filled: 1557 MB   77% open
7, MBytes: 210861, E/s:     47, MB/s:   3.47
ulti: ./set_mo.usf
Util Dr Daq_rate Transport Stream_serv
8r/pro_5781.lmd size: 2000 MB, filled: 1530 MB   76% open
5, MBytes: 210766, E/s:     99, MB/s:   7.20
ulti: ./set_mo.usf
Util Dr Daq_rate Transport Stream_serv
r/pro_5778.lmd size: 2000 MB, filled: 1582 MB   79% open
 MBytes: 210849, E/s:     79, MB/s:   5.58
ti: ./set_mo.usf
il Dr Daq_rate Transport Stream_serv
782.lmd size: 2000 MB, filled: 1539 MB   76% open
 MBytes: 210737, E/s:     19, MB/s:   1.43
ti: ./set_mo.usf
il Dr Daq_rate Transport Stream_serv
```

🟩 R2-22:Acquisition Running	🟩 R2-33:Acquisition Running	🟩 R2-9:Acquisition Running
▨ R2-9:BuildingMode Delayed	▨ R2-9:EventBuilding Suspended	🟥 R2-9:SpillOn Spill OFF
🟩 R2F-1:Acquisition Running	▨ R2F-1:BuildingMode Delayed	🟩 R2F-1:EventBuilding Working
🟥 R2F-1:SpillOn Spill OFF	🟩 R3-42:Acquisition Running	🟩 R3G-10:Acquisition Running
🟩 R3G-11:Acquisition Running	🟩 R3G-5:Acquisition Running	🟩 R3G-6:Acquisition Running
🟩 X86G-15:Acquisition Running	🟩 X86G-15:FileOpen S:File open	🟩 X86G-16:Acquisition Running
🟩 X86G-16:FileOpen S:File open	🟩 X86G-17:Acquisition Running	🟩 X86G-17:FileOpen S:File open
🟩 X86G-5:Acquisition Running	🟩 X86G-5:FileOpen S:File open	🟩 X86G-6:Acquisition Running
🟩 X86G-6:FileOpen S:File open		

```
ad_meb  : 12    226469   1    0    1     5072     2028     202
ad_meb  : 13     76474   1    0    1     5178     2061     206
ad_meb  : 14         0   1    0    1     4608     1863     1863
ad_meb  : 15      2666   1    0    1     4217     1689     1689
ad_meb  : 16        16   0   14  16384  3643940      0        0
Mar 13, 2009 6:35:03 PM |U| DABC/X86G-15/MSG/M: -R2F-1  :read_meb  :-----------------------------------------------
Mar 13, 2009 6:35:03 PM |U| DABC/X86G-15/MSG/M: -R2F-1  :read_meb  :nr of triggers processed: 14761601
```

DIM servers: X86G-15:PRM R3G-6:MSG R3G-5:MSG R3-42:MSG R2F-1:MSG R2-22:MSG X86G-17:MSG DNS=lxg0243 X86G-6:MSG R2-33:MSG X86G-5:MSG R2-9:MSG R3G-11:MSG R3G-10:MSG X86G-16:MSG X86G-15:MSG

Figure 4. Screen shot of a running experiment.

3. Use cases

3.1. CBM readout controller

The n-XYTER is a 128 channel asynchronous, self-triggered, self-sparsifying readout ASIC developed as a front-end for neutron scattering detector applications. In FAIR experiments like CBM it is used as prototype and test-bed of front-end electronics for several kinds of detectors prototypes. The CBM readout controller (ROC), developed in *Kirchhoff Institut für Physik, Heidelberg*, is an FPGA-based board, designed to read time-stamped data from n-XYTERs and transfer that data via optical link to combiner boards or, as alternative option, via Ethernet directly to PCs.

A C++ library has been written (ROClib), which implements a protocol for control and data transfer from ROC to PC. Because the data transport is based on UDP/IP, the protocol cares about data retransmission in case of packets lost. There is a ROOT-based GUI, used for configuration and testing of front-end boards equipped with n-XYTER chips.

A specific *Transport* subclass was implemented to integrate the ROClib functionality into DABC. It mainly implements several methods of the *DataTransport* class:

- *Read_Size* defines size of next buffer
- *Read_Start* request data from ROClib
- *Read_Complete* completes reading

In addition, a *Device* subclass provides configuration and command execution capabilities. Two *Module* subclasses were implemented to combine and sort data from several ROCs. As output, MBS events format was used. Application class *Readout* instantiates and configures all necessary components to perform readout from several ROCs. The configuration for readout from three ROCs is shown in figure 2.

In a first experiment this setup was used to test detectors, front-end and readout boards, and data integrity. The next test will utilize some full readout chains as shown in figure 5. Several front-end boards, readout boards, new combiner boards and PCIexpress cards merge the data streams into PCs where then the time sliced data is combined via InfiniBand.

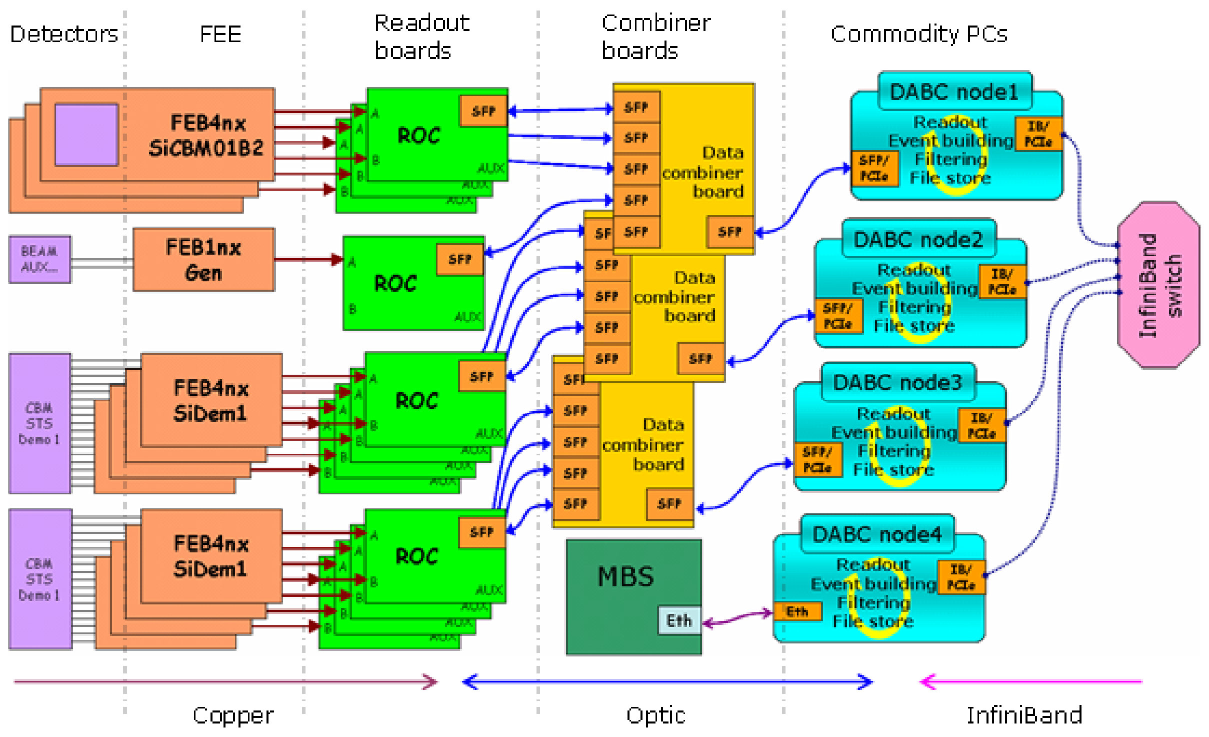


Figure 5. Setup of test beam time autum 2009.

4. Distribution

The DABC package is provided under GPL conditions. Download from the DABC website.

5. Conclusion

The DABC framework has been successfully used in a first experiment and is released.The package is available from the DABC website dabc.gsi.de.

6. Acknowledgements

We acknowledge the support of the European Community-Research Infrastructure Activity under the FP6 "Structuring the European Research Area" programme (HadronPhysics, contract number RII3-CT-2004-506078).

7. References

[1] H.G.Essel, FutureDAQ for CBM: On-line event selection,
IEEE TNS Vol.**53**, No.**3**, June 2006, pp 677-681

[2] Data Acquisition Backbone Core DABC
J Adamczewski, H G Essel, N Kurz and S Linev 2008 *J. Phys.: Conf. Ser.* **119** 022002 (8pp)

[3] H.G.Essel, Data Acquisition Backbone Core DABC,
IEEE TNS Vol.**55**, No.**1**, February 2008, pp 251-255

[4] OpenFabric Alliance OFED, - http://www.openfabrics.org

[5] H.G. Essel et al., The general purpose data acquisition system MBS,
IEEE TNS Vol.**47**, No.**2**, April 2000, pp 337-339

[6] C. Gaspar, Distribution Information Management system DIM, - http://dim.web.cern.ch/dim