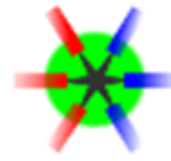


# Data Acquisition Backbone Core



## User Manual

*J.Adamczewski-Musch, S.Linev, H.G.Essel*  
GSI Darmstadt,  
Experiment Electronics Department

Produced: February 18, 2009, Revisions:  
Titel: DABC: User Manual

Document	Date	Editor	Revision	Comment
DABC-user	2009-01-05	Hans G.Essel	1.0.0	First scetch



# Contents

<b>2</b>	<b>DABC User Manual: Overview</b>	<b>5</b>
2.1	User manual overview . . . . .	5
2.1.1	Download and installation . . . . .	5
2.1.2	Application working directory . . . . .	5
2.1.3	Operation . . . . .	5
<b>3</b>	<b>DABC User Manual: Setup</b>	<b>7</b>
3.1	Installing DABC . . . . .	7
3.2	Set-up the DABC environment . . . . .	8
3.3	Setting up DABC user workspace . . . . .	8
3.3.1	DABC data simulator . . . . .	9
3.3.2	DABC MBS event server . . . . .	9
3.3.3	PCI connected front-ends . . . . .	9
3.4	Installation of additional plug-ins . . . . .	9
3.4.1	Add plug-in packages to \$DABCSYS . . . . .	9
3.4.2	Plug-in packages in user directory . . . . .	10
<b>4</b>	<b>DABC User Manual: GUI</b>	<b>13</b>
4.1	GUI Guide lines . . . . .	13
4.2	GUI Panels . . . . .	14
4.2.1	Main DABC GUI buttons . . . . .	14
4.2.2	DABC control panel . . . . .	16
4.2.2.1	DABC controller buttons . . . . .	17
4.2.3	Action in progress . . . . .	17
4.2.4	MBS control panel . . . . .	17
4.2.5	Combined DABC and MBS control panel . . . . .	17
4.2.6	Command panel . . . . .	18
4.2.7	Parameter table . . . . .	18

4.2.7.1	Parameter selection . . . . .	19
4.2.8	Monitoring panels . . . . .	19
4.2.8.1	States . . . . .	19
4.2.8.2	Rate meters . . . . .	20
4.2.8.3	Histograms . . . . .	20
4.2.8.4	Information . . . . .	20
4.2.8.5	Logging window . . . . .	20
4.3	GUI save/restore setups . . . . .	22
<b>5</b>	<b>DABC User Manual: MBS GUI</b>	<b>23</b>
5.1	MBS event building . . . . .	23
5.1.1	MBS setup . . . . .	23
5.1.2	MBS control panel . . . . .	23
5.1.2.1	MBS controller buttons . . . . .	24
5.1.3	MBS command panel . . . . .	25
5.2	MBS DIM parameters . . . . .	25
5.2.1	MBS states . . . . .	25
5.2.2	MBS rates . . . . .	25
5.2.3	MBS infos . . . . .	27
5.2.4	MBS tasks . . . . .	27
5.2.5	MBS text . . . . .	27
5.2.6	MBS numbers . . . . .	27
5.3	Working directories . . . . .	27
5.3.1	MBS configuration of DIM . . . . .	27
<b>6</b>	<b>DABC User Manual: DABC Application MBS</b>	<b>29</b>
6.1	MBS event building with DABC . . . . .	29
6.1.1	MBS setup . . . . .	29
6.1.2	DABC setup . . . . .	29
6.1.3	Combined DABC and MBS control panel . . . . .	30
6.1.3.1	Combined DABC and MBS controller buttons . . . . .	30
	<b>References</b>	<b>33</b>
	<b>Index</b>	<b>33</b>

## Chapter 2

# *DABC* User Manual: Overview

[user/user-overview.tex]

### 2.1 User manual overview

In the following sections we describe the usage on *DABC* applications.

#### 2.1.1 Download and installation

This starts with the installation described in section 3.1, page 7.

#### 2.1.2 Application working directory

We assume that all the necessary components of the application are ready. All plug-ins are implemented. Then we need mainly the setup files. These are typically in a working directory. More in section 3.3, page 8.

#### 2.1.3 Operation

Once we have the working directory setup we can begin. In the GUI part we describe the general functionality of the GUI which is common for most applications.

More specific functionality is described in the sections of the applications (as far as delivered with *DABC*).

As examples we take sometimes the *DABC* simulator plug-in which runs out of the box.

External application specific GUI add-ons cannot be described here.



## Chapter 3

# *DABC* User Manual: Setup

[user/user-setup.tex]

### 3.1 Installing *DABC*

When working at the gsi linux cluster, the *DABC* framework is already installed and will be maintained by people of the gsi EE department. Here *DABC* needs just to be activated from any gsi shell by typing `. dabclogin`. In this case, please skip this installation section and proceed with following section 3.2 describing the set-up of the user environment.

However, if working on a separate DAQ cluster outside gsi, it is mandatory to install the *DABC* software from scratch. Hence the *DABC* distribution is available for download at <http://dabc.gsi.de>. It is provided as a compressed tarball of sources `dabc1.tar.gz`. The following steps describe the recommended installation procedure:

1. **Unpack this *DABC* distribution** at an appropriate installation directory, e. g. :

```
cd /opt/dabc;  
tar zxvf dabc1.tar.gz
```

This will extract the archive into a subdirectory which is labelled with the current version number like `/opt/dabc/dabc_1_0.00`. This becomes the future *DABC* system directory.

2. **Prepare the *DABC* environment login script:** A template for this script can be found at `scripts/dabclogin.sh`

- Edit the `DABCSYS` environment according to your local installation directory. This is done in the following lines:

```
export DABCSYS=/opt/dabc/_installations/dabc_1_0.00
```

- Edit the `DIM_DNS_NODE` environment according to the machine where the DIM name server [1] for the *DABC* control system will run:

```
export DIM_DNS_NODE=hostname.domain.de
```

- Copy the script to a location in your global `$PATH` for later login, e. g. `/usr/bin`. Alternatively, you may set an *alias* to the full pathname of `dabclogin.sh` in your shell profile.

3. Execute the just modified login script in your shell to set the environment:

```
. dabclogin.sh
```

This will set the environment for the compilation.

4. Change to the **DABC** installation directory and start the build:

```
cd \${DABCSYS}
make
```

This will compile the **DABC** framework and install a suitable version of DIM in a subdirectory of `\${DABCSYS}/dim`.

After succesful compilation, the **DABC** framework installation is complete and can be used from any shell after invoking `. dabclogin.sh`. The next sections 3.2 and 3.3 will describe further steps to set-up the **DABC** working environment for each user.

## 3.2 Set-up the **DABC** environment

Once the general **DABC** framework is installed on a system, still each user must "activate" the environment and do further preparations to work with it.

1. Execute the **DABC** login script in a linux shell to set the environment. At gsi linux installation, this is done by

```
. dabclogin
```

For the user installation as described in above section 3.1, by default the script is named

```
. dabclogin.sh
```

The login script will already enable the **DABC** framework for compilation of user written components. Additionally, the general executable `dabc_run` now provides the **DABC** runtime environment and may be started directly for simple "batch mode" applications on a single node. However, further preparations are necessary if **DABC** shall be used with DIM control system and GUI.

2. Open a dedicated shell on the machine that shall provide the DIM name server, e. g.  

```
ssh nsnode.cluster.domain
```

 Then call  

```
. dabclogin.sh
```

```
dimDns
```

 to launch the DIM name server. This is done **once** at the beginning of the DAQ setup; usually the DIM name server needs not to be shut down when **DABC** applications terminate.
3. Set the DIM name server environment variable in any **DABC** working shell (e. g. the shell that will start the `dabc` gui later):  

```
. dabclogin.sh
```

```
export DIM\_DNS\_NODE=nsnode.cluster.domain
```

 Note that a user installation of **DABC** framework may set this environment variable already in the general `dabclogin.sh` script, if the DIM name server will mostly run for all users on the same default node (see section 3.1).
4. Now the **DABC** GUI can be started in such prepared shell by typing `dabc`, (or `mbs` for a plain **MBS** gui, resp.). See below in gui section.

## 3.3 Setting up **DABC** user workspace

To operate a **DABC** application one should create a dedicated working directory to keep all relevant files:



- Setup files for *DABC* (XML).
- Log files (text).

The GUI may run on a different directory with no access to the *DABC* working directory. There we place:

- Data files for startup panels (XML).
- Configuration files for GUI (XML).

Of course, one can use one directory for all.

The setup files are application specific. Therefore they are described in the application sections.

### 3.3.1 *DABC* data simulator

### 3.3.2 *DABC MBS* event server

[5.1](#), page [23](#)

### 3.3.3 PCI connected front-ends

## 3.4 Installation of additional plug-ins

Apart from the *DABC* base package, there may be additional plug-in packages for specific use cases. Generally, these plug-in packages may consist of a **plugins** part and an **applications** part. The *plugins* part offers a library containing new components (like *Devices*, *Transports*, or *Modules*). The *applications* part mostly contains the XML setup files to use these new components in the *DABC* runtime environment; however, it may contain an additional library defining the *DABC Application* class.

As an example, we may consider a plug-in package for reading out data from specific PCIe hardware like the Active Buffer Board ABB [2]. This package is separately available for download at <http://dabc.gsi.de> and described in detail in chapter ?? of the *DABC* programmer's manual.

There are principally two different ways to install such separate plug-in packages: Either within the general DABCSYS directory as part of the central *DABC* installation, as described in following section [3.4.1](#). Or at an independent location in a user directory, as described in section [3.4.2](#).

### 3.4.1 Add plug-in packages to \$DABCSYS

This is the recommended way to install a plug-in package if this package should be provided for all users of the *DABC* installation. A typical scenario would be that an experimental group owns dedicated DAQ machines with system manager privileges. In this case, the plugin-package may be installed under the same account as the central *DABC* installation (probably, but not necessarily even the root account). The new plug-in package should be directly installed in the \$DABCSYS directory then, with the following steps:

1. Download the plug-in package tarball, e. g. `abb1.tar.gz`
2. Call the `dabclogin.sh` script of the *DABC* installation (see section [user-env](#))
3. Copy the downloaded tarball to the \$DABCSYS directory and unpack it there:
 

```
cp abb1.tar.gz $DABCSYS
cd $DABCSYS
tar zxvf abb1.tar.gz
```

This will extract the new components into the appropriate `plugins` and `applications` folders below `$DABCSYS`.

4. Build the new components with the top Makefile of `$DABCSYS`:  

```
make
```
5. To work with the new components, the configuration script(s) of the *applications* part should be copied to the personal workspace of each user (see section 3.3). For the *ABB* example, this is found at  
`$DABCSYS/applications/bnet-test/SetupBnetIB-ABB.xml`

### 3.4.2 Plug-in packages in user directory

This is the case when *DABC* is installed centrally at the fileserver of an institute, and several experimental groups shall use different plug-ins. It is also the recommended way if several users want to modify the source code of a plug-in library independently without affecting the general installation.

The new plug-in package should be installed in a user directory then, with the following steps:

1. Download the plug-in package tarball, e. g. `abb1.tar.gz`
2. Create a directory to contain your additional *DABC* plugin packages:  

```
mkdir $HOME/mydabcpackages
```
3. Call the `dabclogin.sh` script of the *DABC* installation (see section user-env)
4. Copy the downloaded tarball to the `$DABCSYS` directory and unpack it there:  

```
cp abb1.tar.gz $HOME/mydabcpackages
cd $HOME/mydabcpackages
tar zxvf abb1.tar.gz
```

This will extract the new components into the appropriate `plugins` and `applications` folders below the working directory.
5. To build the *plugins* part, change to the appropriate package plugin directory and invoke the local Makefile, e. g. for the *ABB* example:  

```
cd $HOME/mydabcpackages/plugins/abb
make
```

This will create the corresponding plug-in library in a subfolder denoted by the computer architecture, e. g. :  
`$HOME/mydabcpackages/plugins/abb/x86_64/lib/libDabcAbb.so`
6. For some plug-ins, there may be also small test executables with different Makefiles in subfolder `test`. These can be optionally build and executed independent of the *DABC* runtime environment.
7. The *DABC* working directory for the new plug-in will be located in subfolder `applications/plugin-name`  

For the *ABB* example, the application will set up a builder network with optional Active Buffer Board readouts, so this is at  
`$HOME/mydabcpackages/applications/bnet-test`

As in this example, there may be an additional library to be build containing the actual *Application* class. This is done by invoking the Makefile within the directory:  

```
cd $HOME/mydabcpackages/applications/bnet-test
make
```

Here the application library is produced directly on top of the working directory:  
`$HOME/mydabcpackages/applications/bnet-test/libBnetTest.so`
8. The actual locations of the newly build libraries (plugins, and optionally applications part) has to be edited in the `<lib>` tag of the corresponding *DABC* setup-file (here: `SetupBnetIB-ABB.xml`). The default set-up examples in the plug-in packages assume that the library is located at `$DABCSYS/lib`,

---

as it is in the alternative installation case as described in section [3.4.1](#).



## Chapter 4

# *DABC* User Manual: GUI

[user/user-gui.tex]

### 4.1 GUI Guide lines

The current *DABC* GUI is written in Java using the DIM software as communication layer. The standard part of the GUI described here may be extended by application specific parts. How to add such extensions is described in the programmer's manual. Typically they are started as prompter panels via buttons in the main GUI menu.

The standard part builds a set of panels (windows) according the parameters the DIM servers offer. Only services from one single DIM name server (node name specified as shell variable DIM\_DNS\_NODE) defining a name space can be processed.

The GUI needs no file access to the *DABC* working directory. However, user must have ssh (or rsh) access to the *DABC* (or *MBS*) master node. Currently the GUI must run under the same account as the *DABC*. In spectator mode (no commands) the GUI may run under different account. Master node must have remote access to all worker nodes. The user's ssh settings must enable remote access without prompts.

The layout of the GUI can be adjusted to individual needs. It is strongly recommended to save these settings to see the same layout after a restart of the GUI. The GUI can be restarted any time. *DABC* and *MBS* systems continue without GUI.

The GUI is packed in a JAR file `xgui.jar` which can be moved everywhere. The file must be in the CLASSPATH paths together with the `$DIMDIR/jdim/classes` path. The GUI is invoked by `java -Xmx200m xgui.xGui`. For standard installations short cuts are provided like:

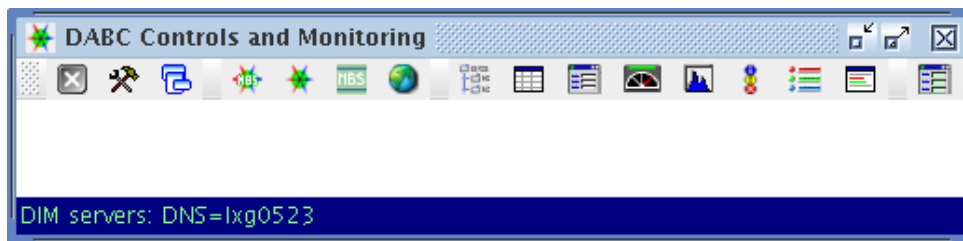
**dabc** : Start standard *DABC* GUI with *DABC* control panel.

**mbs** : Start GUI with *MBS* control panel only. This is for controlling a stand-alone running *MBS*.

**dabs** : Start GUI with combined *MBS* and *DABC* control panel. This is for using *DABC* as event builder with *MBS* front-ends.

**moni** : Start GUI without active elements. Works then as monitor.

**guru** : Starts GUI with all buttons for all functions.




**Figure 4.1:** Main toolbar buttons.


## 4.2 GUI Panels


Fig. 4.1, page 14 shows the main menu of *DABC* (minimal view). The GUI as it comes up is divided in three major parts: one sees on top a toolbar with icon buttons. Most of these open other windows. The dark line at the bottom shows a list of active DIM servers. The other windows are placed in the white middle pane. The functions of the buttons and the invoked panels is described in the next sections. Depending on the application some buttons may be not seen, additional ones may show up. If one does not work with *MBS* plug-ins the control panels for *MBS* are of course not useful.

Fig. 4.2, page 15 shows a more typical view of a running *DABC*. In general, all panels (including the GUI itself) can be closed and reopened any time.


### 4.2.1 Main *DABC* GUI buttons


 Quit GUI. Will prompt (RET will quit). The *DABC* will continue to run. The GUI may be started anywhere again. In case you saved the layout (recommended, see 4.3, page 22) and you start the GUI from the same directory it will look pretty much the same as you left it.


 Test, shell script


 Save settings: window layout, record attributes, parameter selection filters. Details see 4.3, page 22. Note that the content of the control panels must be saved by similar buttons in these panels.


 Open *DABC MBS* control panel, see 6.1.3, page 30.

 Open *DABC* control panel, see 4.2.2, page 16.


 Open *MBS* control panel, see 5.1.2, page 23.

 Refresh. Update DIM service list from DIM name server. New parameters and commands are added. Existing ones are not removed, but only deactivated. Parameters and Commands are sorted alphabetically by name. All panels are updated. This update function is called automatically whenever the frontend systems might have added or deleted DIM services. In normal operation there is no need to update manually.

 Open command panel (4.2.6, page 18).

 Open parameter table (4.2.7, page 18).

 Open parameter selection panel (4.2.7.1, page 19).

 Open rate meter panel (4.2.8, page 19).

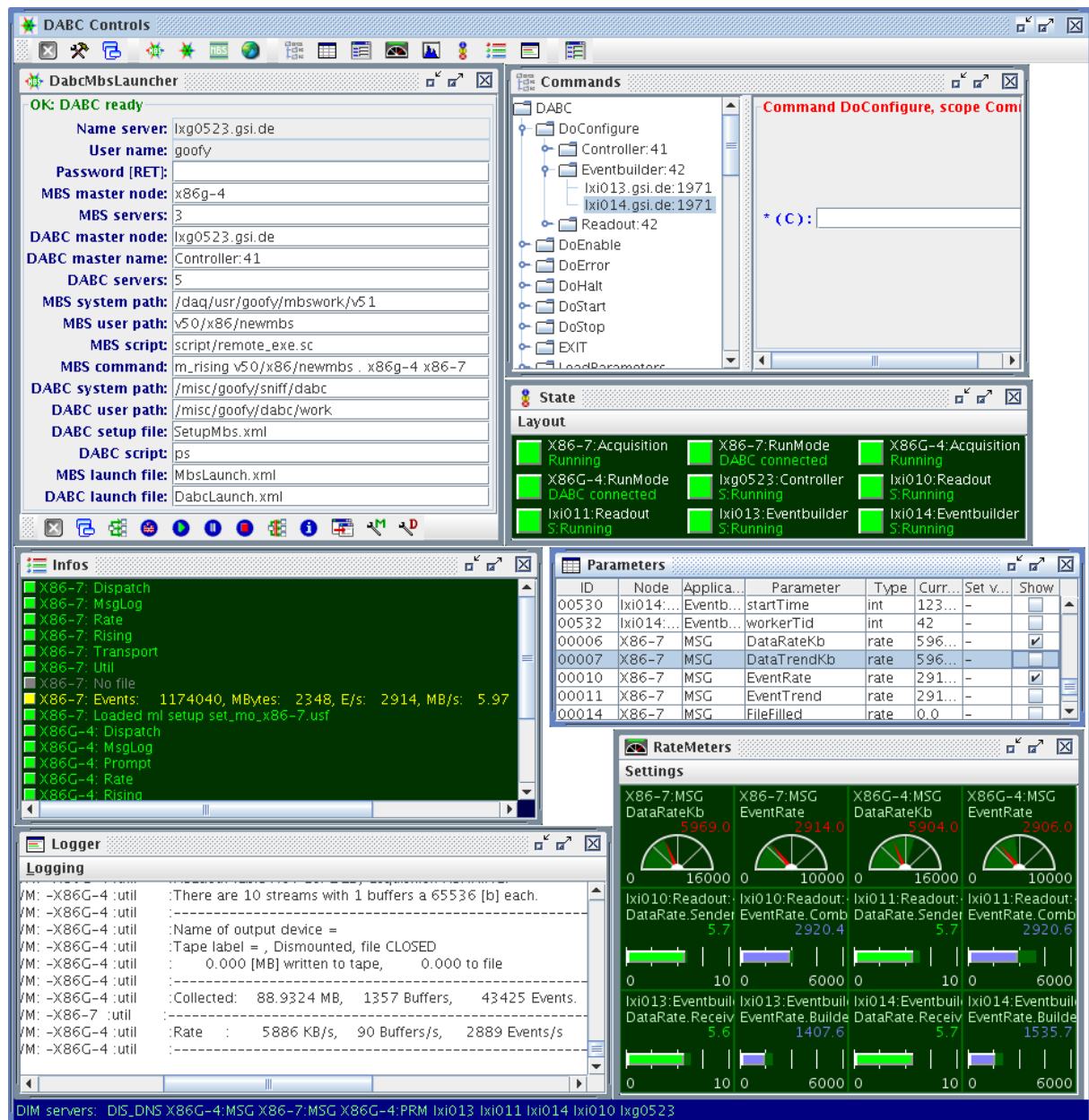


Figure 4.2: More typical full screen view.



Open histogram panel (4.2.8, page 19).



Open state panel (4.2.8, page 19).



Open info panel (4.2.8, page 19).



Open log panel (4.2.8, page 19).



Eventually one might see additional icons from application panels (this one is only an example).

The three control panels (*DABC*, *MBS*, combined *DABC* and *MBS*) are used depending on the application to be controlled. Eventually an application provides additional specific control panels.

### 4.2.2 DABC control panel

The standard DABC control panel is shown in 4.3, page 16. As mentioned already some applications may provide their own control panels like the MBS applications (see section 5.1.2, page 23). But most of the buttons are very common. From left to right they startup a system, configure it, start data taking, pause data taking, stop tasks, shut down. At the very left we see a save button, at the right a shell execution button. Values are read from file `DabcControl.xml` (default, may be saved/restored to/from other

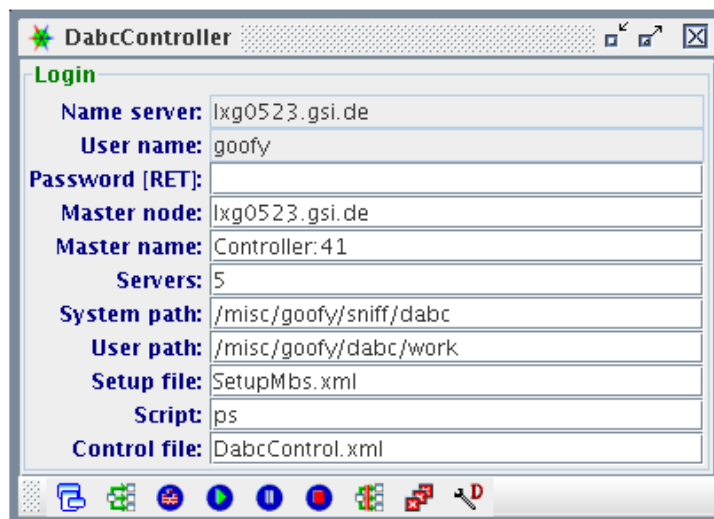


Figure 4.3: DABC controller panel.

file, see 4.3, page 22).

```
<?xml version="1.0" encoding="utf-8"?>
<DabcLaunch>
<DabcMaster prompt="DABC Master" value="node.xxx.de" />
<DabcName prompt="DABC Name" value="Controller:41" />
<DabcUserPath prompt="DABC user path" value="myWorkDir" />
<DabcSystemPath prompt="DABC system path" value="/dabc" />
<DabcSetup prompt="DABC setup file" value="SetupDabc.xml" />
<DabcScript prompt="DABC Script" value="ps" />
<DabcServers prompt="%Number of needed DIM servers%" value="5" />
</DabcLaunch>
```

**DabcMaster:** Node where the master controller shall be started. Can be one of the worker nodes.

**DabcName:** A unique name inside DABC of the system.

**DabcUserPath:** User working directory. The GUI does not need to have access to the filesystem.

**DabcSystemPath:** Path where the DABC is installed.

**DabcSetup:** Setup file name.

**DabcScript:** Command to be executed in an ssh at the master node.

**DabcServers:** Number of workers and controllers. This information is minimum for the GUI to know when all DABC nodes are up. The GUI waits until this number of DIM servers is up and running.

The name server name is translated from shell environment variable `DIM_DNS_NODE`, the user name from shell environment variable `USER`. Password can be chosen when the first remote shell script is executed (which itself is protected by user password). All following commands then need this password.



#### 4.2.2.1 *DABC* controller buttons



Save panel settings to the file Control file. If you choose a name different from the default you must set a shell variable to it to get the values from that file (see 4.3, page 22).



Startup all tasks. Executes a *DABC* script `dabcstartup.sc` via `ssh` on the master node under user name. Then it waits until the number of DIM servers expected are announced. A progress panel pops up during that time (see 4.2.3, page 17). When the servers are up the main GUI Update is triggered building all panels from scratch according the parameters offered by the servers.



Configure. Executes state transition command `Configure` on master node and waits for the transition. All plug-in components are created. Then execute `Enable`. Waits until all workers go into Ready state. Now the *DABC* is ready to run. Triggers the main GUI Update.



Start acquisition. Executes `Start` command. All components go into running state `Running`.



Pause acquisition. Executes `Stop` command. All components go into standby state `Ready`.



Halt acquisition. Executes `Halt` command. This closes all plug-ins. States go into `Halted`. Next must be shut down or configure.



Exit all processes by `EXIT` commands. After 2 seconds trigger the main GUI Update.



Shut down all processes on all nodes by script. This is the hard shut down.



`ssh` shell script execution on master node.

#### 4.2.3 Action in progress

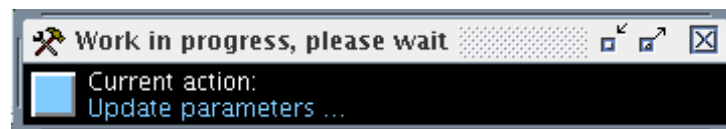


Figure 4.4: Launching progress.

When starting up, configure or shut down the GUI has to wait until the front-ends have completed the action. During that time a progress window similar to the one shown in Fig. 4.4, page 17 pops up. Please wait until the popup disappears.

#### 4.2.4 *MBS* control panel

To control and monitor a stand-alone *MBS* system a dedicated control panel is provided by the *MBS* application. This panel is described in the *MBS* section 5.1.2, page 23.

#### 4.2.5 Combined *DABC* and *MBS* control panel

To control and monitor *MBS* front-ends with *DABC* event builders a dedicated control panel is provided by the *MBS* application. This panel is described in the *MBS* section 6.1.3, page 30.

## 4.2.6 Command panel

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. Commands are executed in all components of *DABC*.

The *DABC* naming convention for commands and parameters defines four main name fields separated by slashes:

1. DIM server name space (example DABC)
2. Node (example dabcWorker1)
3. Application (example Combiner)
4. Name (example doConfigure)

Example: DABC/dabcWorker1/Combiner/doConfigure. Fig. 4.5, page 18 shows on the left side the command

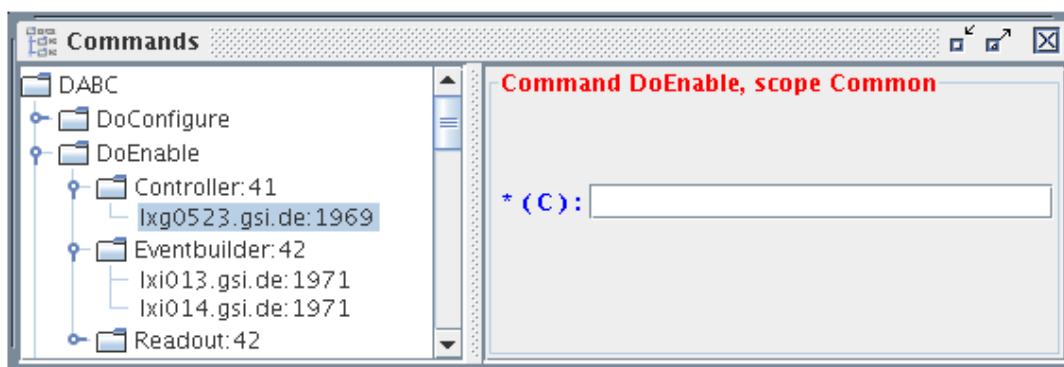


Figure 4.5: Command panel.

tree. The tree is built from name, application, nodes. Double click (or RETURN) on a treenode executes the command on all treenodes below. A click on a command opens at the right side the argument panel. Entering argument values and RETURN executes the command. In the example shown in the figure double click on doEnable would execute that command on three nodes. Double click on Eventbuilder would execute only on two nodes.

## 4.2.7 Parameter table

*DABC* parameters are DIM services as the commands. The naming convention is the same. The server providing parameters can make them (no)visible and (un)changable. *DABC* defines some special parameter types having a data structure and a specific interpretation like a rate parameter having a value, limits, a color, and a graphic presentation. A rate parameter is assumed to be changed and updated regularly. The GUI displays these special parameters in dedicated panels. Parameters are used in all components of *DABC*. The central place for all param-

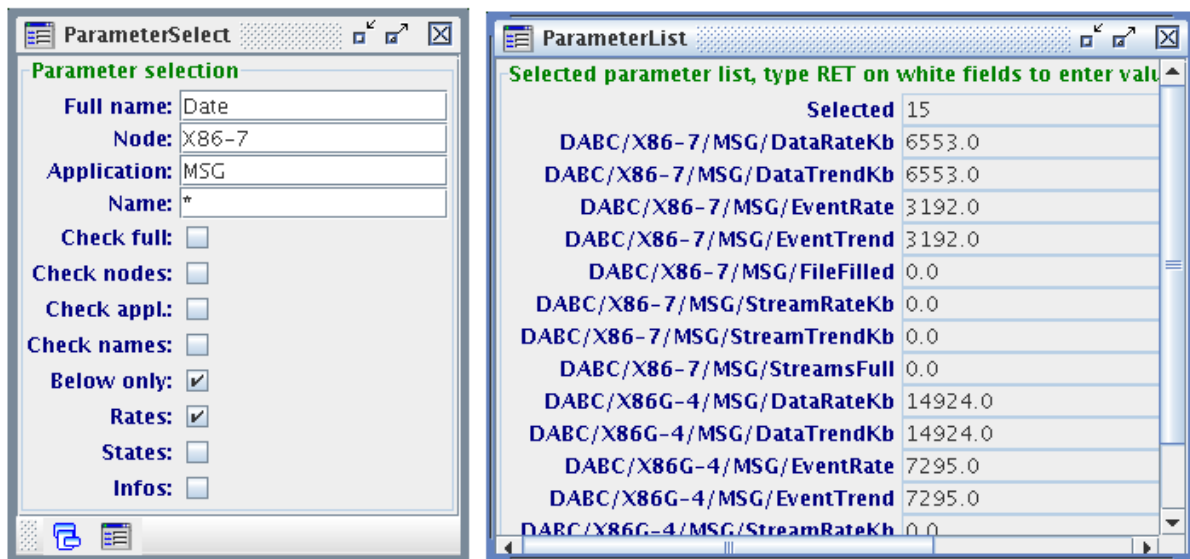
Parameters							
ID	Node	Application	Parameter	Type	Current	Set value	Show
00308	lxi010:1970	Readout: 42	CtrlPoolSize.BnetPlugin	int	2097152		<input type="checkbox"/>
00309	lxi010:1970	Readout: 42	DABCVersion	char	DABC C...	-	<input type="checkbox"/>
00310	lxi010:1970	Readout: 42	DataRate.Sender	rate	0.0	-	<input checked="" type="checkbox"/>
00311	lxi010:1970	Readout: 42	EventBuffer.BnetPlugin	int	524288		<input type="checkbox"/>
00312	lxi010:1970	Readout: 42	EventPoolSize.BnetPlugin	int	4194304		<input type="checkbox"/>
00313	lxi010:1970	Readout: 42	EventRate.Combiner	rate	0.0	-	<input checked="" type="checkbox"/>
00314	lxi010:1970	Readout: 42	InfoMessage	info	State ma...	-	<input checked="" type="checkbox"/>

Figure 4.6: Parameter table.

eters in the GUI is the parameter table as shown in Fig. 4.6, page 18. The parameter table holds all parameters

which are marked by the provider to be visible. The parameter values can be changed in the Set value column if no minus sign is there in which case the provider does not grant modification. The buttons in the Show column indicate if the parameter is shown in some graphics panel. It can be removed from or added to this panel by the buttons. The table can be ordered by columns (click on column header). The column width can be adjusted and is saved/restored by main save button (see 4.3, page 22).

#### 4.2.7.1 Parameter selection



**Figure 4.7:** Parameter selection panel and selected parameter list.

To get a more selective view on the parameters one can specify filters in the panel shown at the left side of Fig. 4.7, page 19. Text substrings for each of the four name fields can be specified as well as a selection of record types. Values can be saved (see 4.3, page 22). With the check boxes the filter function for each of these can (de)activated. The parameter list at the right window in Fig. 4.7, page 19 shows only the parameters matching all filters.

If the data field is white the parameter can be changed. This cannot be done in place because the parameter might be updated in the mean time. Instead press RETURN in the field. A prompter will pop up to enter the value.

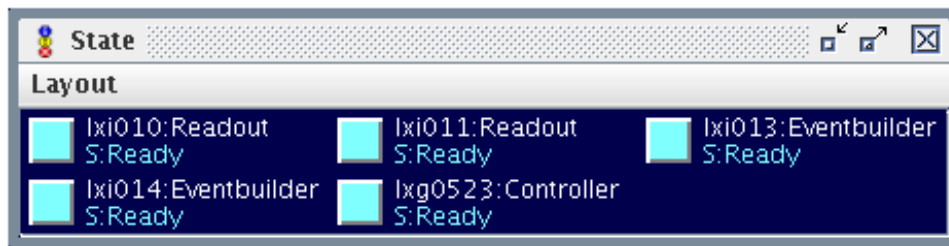
### 4.2.8 Monitoring panels

As already mentioned the *DABC* provides definitions of special purpose DIM parameters. These Records can be recognized by the GUI and are handled in appropriate way. Currently there are

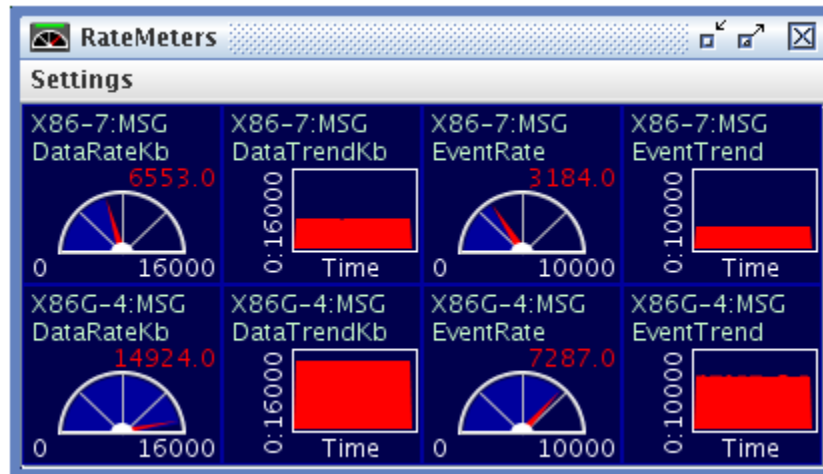
- States
- Rates
- Histograms
- Infos

#### 4.2.8.1 States

States are records having a number for severity (0 to 4), a color, and a brief state description (see Fig. 4.8, page 20). Of course the states of the *DABC* state machine are shown as states. Application plug-ins may use this kind of records also for other information.



**Figure 4.8:** States.



**Figure 4.9:** Rates.

#### 4.2.8.2 Rate meters

All rate meters are displayed in the meter panel, Fig.4.9, page 20. Meters can be removed in the parameter table (See Fig. 4.6, page 18) with the Show buttons like the other graphical parameters. Saving the setup, the visibility will be preserved.

On the left side in Fig. 4.10, page 21 the Settings menu is shown. It affects all items in the panel. One can Zoom (toggle between large and normal view), change the number of columns, change the display mode, toggle Autoscale, and set limits (applied to all meters).

Besides that each individual item can be adjusted by right mouse button. The context menu is shown on the right. All changes done individually are changing the defaults! The global changes can be overwritten by these defaults. All settings are saved with the setup and restored on GUI startup (see 4.3, page 22).

#### 4.2.8.3 Histograms

Histogram panels are handled in pretty much the same way as the rate meters. All histograms are displayed in the histogram panel, Fig.4.11, page 21. Histograms can have arbitrary size set in Layout menu.

#### 4.2.8.4 Information

Information records mainly display one line of text with a color (see Fig. 4.12, page 21).

#### 4.2.8.5 Logging window

Fig. 4.13, page 22 show the logging window.

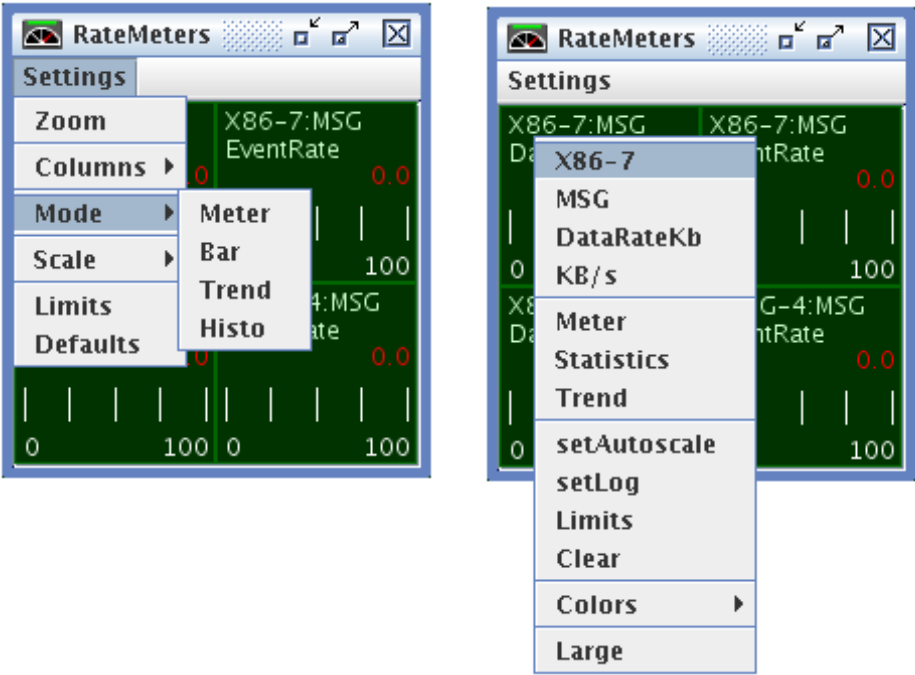


Figure 4.10: Steering menus.

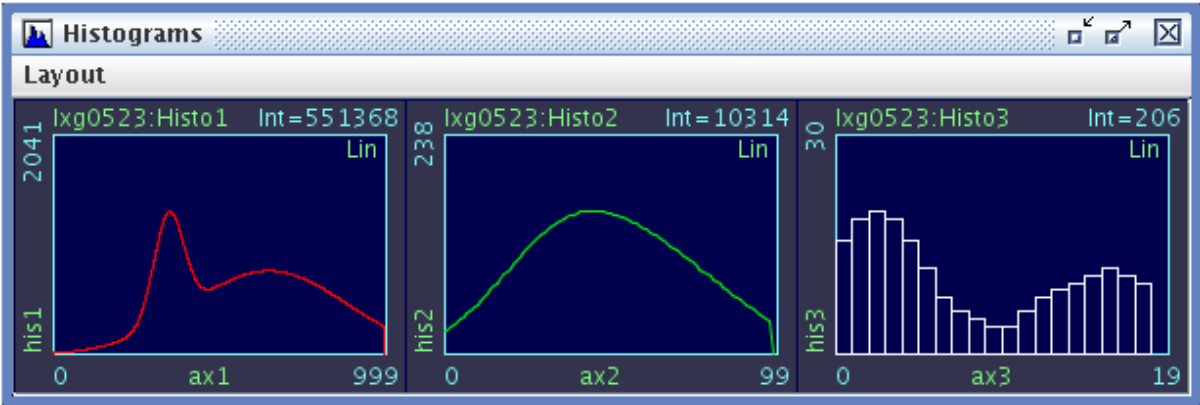


Figure 4.11: Histograms.

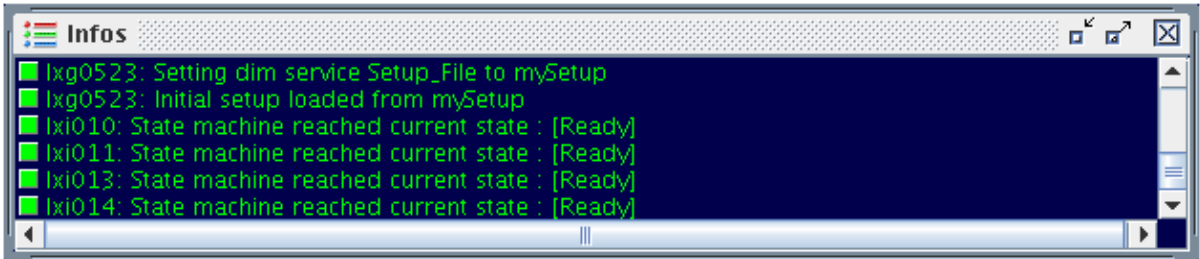
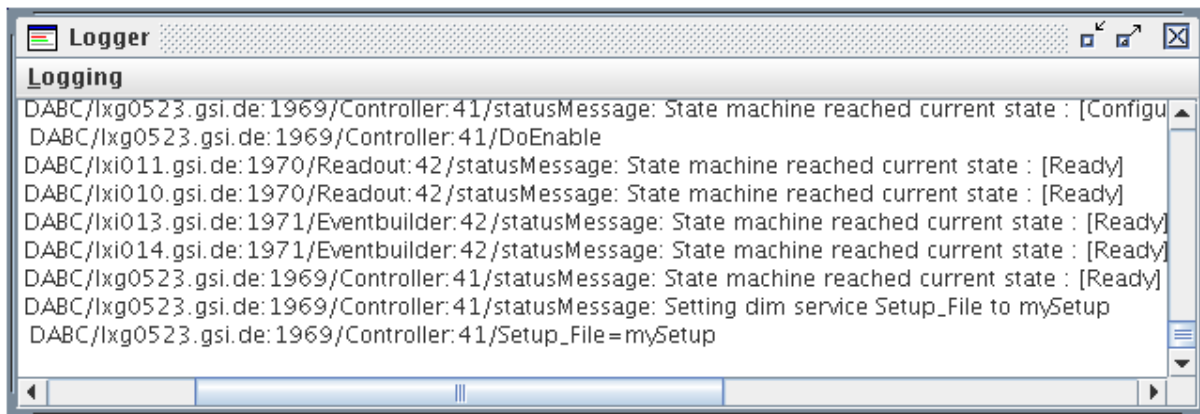


Figure 4.12: Info.



**Figure 4.13:** Logging.

### 4.3 GUI save/restore setups

There are several setups which can be stored in XML files and are retrieved when the *xGUI* is started again. The file names can be specified by shell variables.

- DABC\_CONTROL\_DABC : Values of *DABC* control panel. Saved by button in panel.  
Default `DabcControl.xml`. Filename in panel itself.
- DABC\_CONTROL\_MBS : Values of *MBS* control panel. Saved by button in panel.  
Default `MbsControl.xml`. Filename in panel itself.
- DABC\_RECORD\_ATTRIBUTES : Attributes of records. Saved by main save button.  
Default `Records.xml`.
- DABC\_PARAMETER\_FILTER : Values of parameter filter panel. Saved by main save button.  
Default `Selection.xml`.
- DABC\_GUI\_LAYOUT : Layout of frames. Saved by main save button.  
Default `Layout.xml`.

## Chapter 5

# DABC User Manual: MBS GUI

[user/user-gui-mbs.tex]

### 5.1 MBS event building

#### 5.1.1 MBS setup

Any MBS system can be controlled by the DABC GUI. It can run in two operation modes: with MBS event builder or DABC event builder (see 6.1, page 29). The first case means a standard MBS system.

To control a standard MBS nothing has to be done by the user on the MBS side. The node running the GUI must get granted rsh access at least to the MBS node where the prompter shall run. **Note**, however that in the user's MBS startup file (typically `startup.scom`) the `m_daq_rate` task must be started as last task (this is probably the case already). This task calculates the rates. The GUI waits for this task after execution of the startup file. Because MBS has no states there is no other way to know when the startup has finished. Of course, the MBS itself must have been build with the DIM option (since version v5.1). Central log file is written as usual. Optionally one can provide a text file with specifications which parameters shall appear on screen (see 5.3.1, page 27).

For the standard MBS control one needs no DABC installation. The GUI jar file is sufficient. DIM must be installed.

#### 5.1.2 MBS control panel

Fig. 5.1, page 24 shows the panel to be used to control a standard MBS. The values are restored from file `MbsControl.xml` (default, may be saved to other file, see 4.3, page 22).

```
<?xml version="1.0" encoding="utf-8"?>
<MbsLaunch>
  <MbsMaster prompt="MBS Master" value="node-xx" />
  <MbsUserPath prompt="MBS User path" value="myMbsDir" />
  <MbsSystemPath prompt="MBS system path" value="/mbs/v51" />
  <MbsStartup prompt="MBS startup" value="startup.scom"/>
  <MbsShutdown prompt="MBS shutdown" value="shutdown.scom"/>
  <MbsCommand prompt="Script command" value="whatever command" />
  <MbsServers prompt="%Number of needed DIM servers%" value="3" />
</MbsLaunch>
```

**MbsMaster** : Lynx node where the MBS prompter is started.

**MbsUserPath** : MBS user working directory. The GUI need not to have access to that filesystem.

**MbsSystemPath** : Path on Lynx where the MBS is installed. GUI needs no access to this path.



**Figure 5.1:** MBS controller.

**MbsStartup** : The user specific *MBS* startup command procedure, typically `startup.scom`, located on user path.

**MbsShutdown** : The user specific *MBS* shutdown command procedure, typically `shutdown.scom`, located on user path.

**MbsCommand** : With RET an *MBS* command is executed (on current node). The shell script button executes this string as `rsh` command on master node.

**MbsServers** : Number of nodes plus prompter. This information is minimum for the GUI to know when all *MBS* nodes are up. The GUI waits until this number of DIM servers is up and running.

That file can be created from within the GUI in the *MBS* controller panel. Enter all values necessary, and store them.

#### 5.1.2.1 *MBS* controller buttons



Save panel settings, see 4.3, page 22.



Execute script `prmstartup.sc` at master node. Starts prompter, dispatchers and message loggers and waits until they are up. Trigger the main Update. A progress panel pops up during that time (see 4.2.3, page 17).



Execute script `dimstartup.sc` at master node. Starts dispatcher and message logger for single node *MBS*. Trigger the main Update.



Configure. Execute user's *MBS* startup procedure in prompter (dispatcher). Trigger the main Update.



Start acquisition. Execute Start acquisition.



Pause acquisition. Execute Stop acquisition.



Halt acquisition. Execute user's *MBS* shutdown procedure in prompter. Prompter, dispatcher and message loggers should still be running.



Shut down all. Execute script `prmshutdown.sc` at master node. After 2 seconds trigger the main Update.



Show acquisition. Output in log panel.



Shell script executes command on master node.



### 5.1.3 MBS command panel

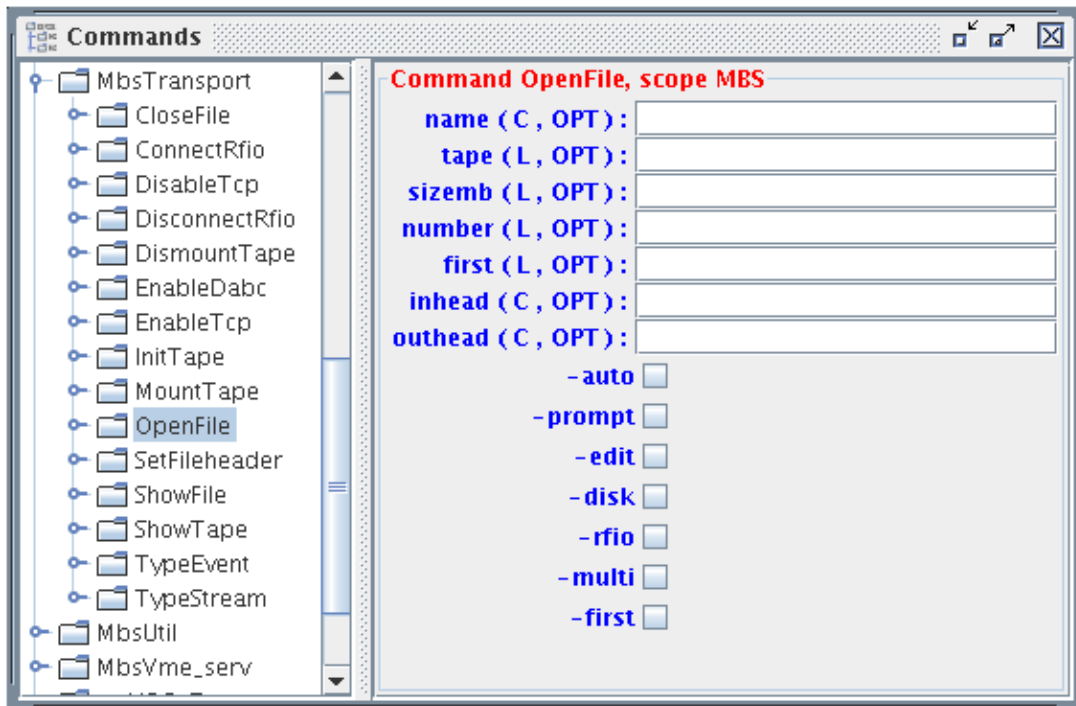


Figure 5.2: Command panel.

Fig. 5.2, page 25 shows on the left side the command tree. Double click (or RETURN) on a command executes the command. The top tree level is the executing **MBS** task, below that are the commands, and the master node (prompter node) is the only node below each command. However, command is sent to the prompter node, but executed on the current node which is displayed in the info panel (see Fig. 5.4, page 26). Click on a command opens at the right side the argument panel. Entering argument values and RETURN executes the command. Only the **MBS** commands of the running tasks are shown. Fig. 5.3, page 26 shows that only dispatcher and prompter are up and therefore only their commands are seen. Fig. 5.4, page 26 shows in addition the commands of util and transport after configuration.

## 5.2 MBS DIM parameters

### 5.2.1 MBS states

**Acquisition/State** Running | Stopped  
**BuildingMode/State** Delayed | Immediate  
**EventBuilding/State** Working | Suspended  
**FileOpen/State** File open | File closed  
**RunMode/State** DABC connected | MBS to DABC | Transport client | MBS standalone  
**SpillOn/State** Spill ON | Spill OFF  
**Trigger/State** Master | Slave

### 5.2.2 MBS rates

**MSG/DataRateKb** KByte/s  
**MSG/DataTrendKb** KBytes/s as trend  
**MSG/EventRate** Events/s  
**MSG/EventTrend** Events/s as trend

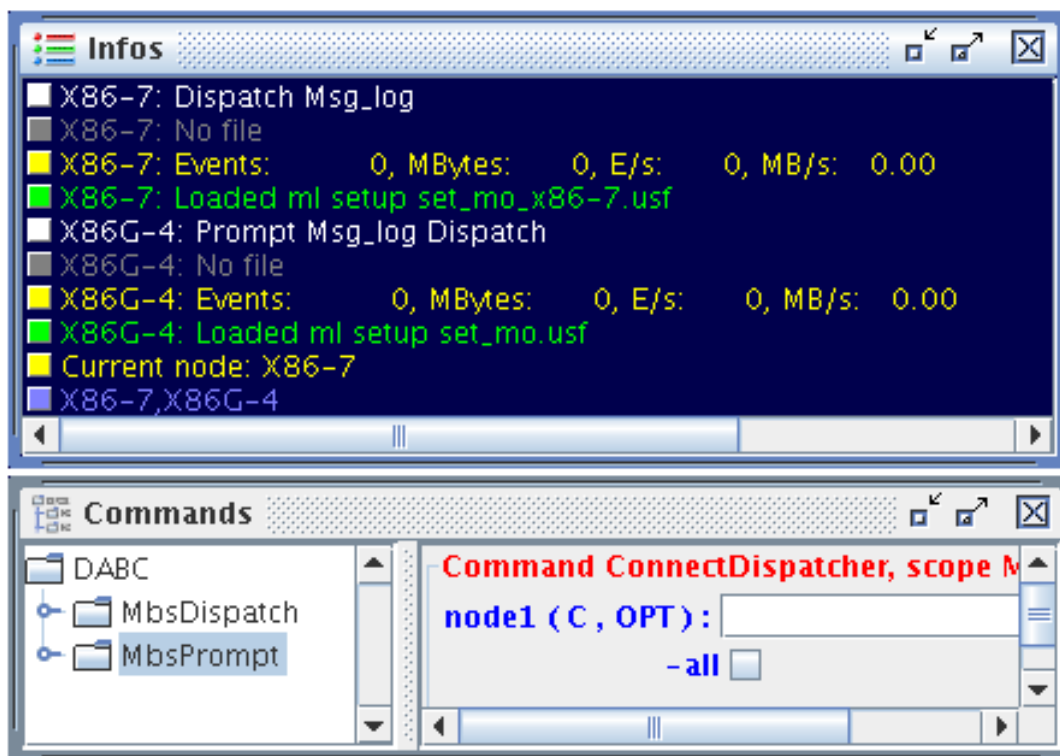


Figure 5.3: Info and command panel.

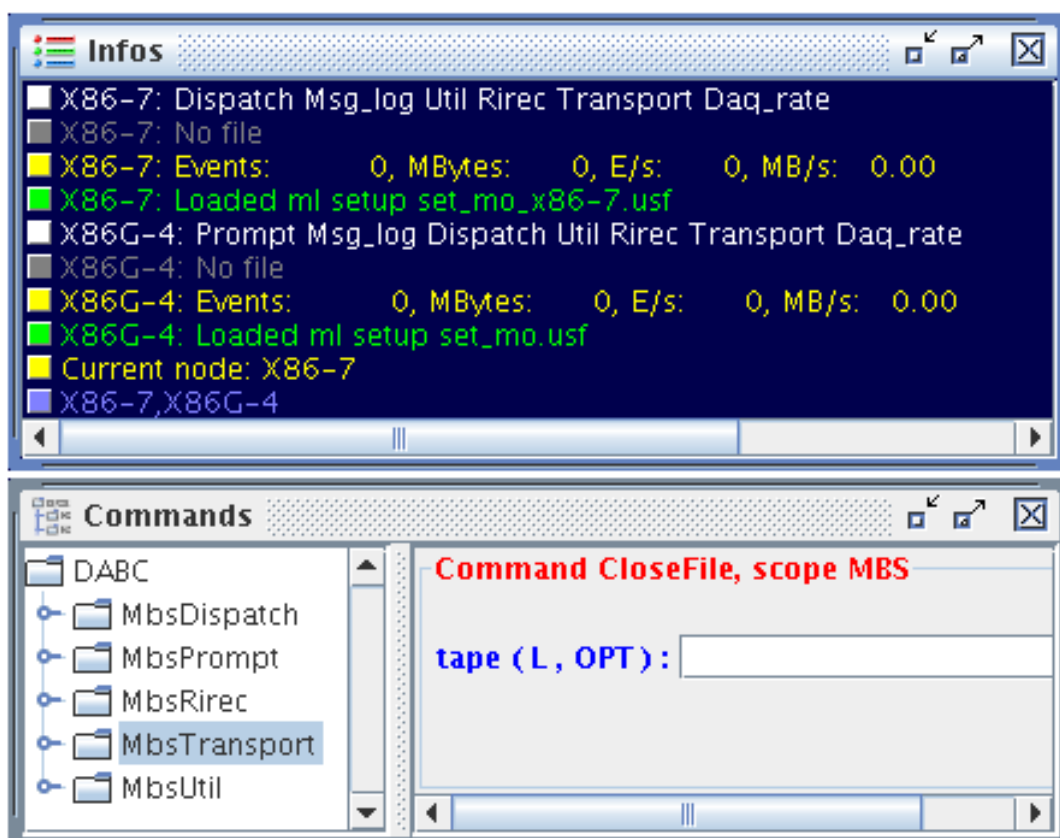


Figure 5.4: Info and command panel.

**MSG/StreamRateKb** Stream server Kbyte/s  
**MSG/StreamTrendKb** Stream server Kbyte/s as trend  
**MSG/FileFilled** File filled in percent  
**MSG/StreamsFull** Number of full streams in percent

### 5.2.3 MBS infos

Shown in info window.

**MSG/eFile** Name of file.  
**MSG/ePerform** Events, MBytes, Events/s and MBytes/s.  
**MSG/eSetup** Name of setup file loaded.  
**PRM/Current** Current command execution node (master node only).  
**PRM/NodeList** List of nodes (master node only).

### 5.2.4 MBS tasks

Task list is shown in info window (name slightly different).

Dispatch MsgLog ReadMeb Collector Transport EventServ Util ReadCam EsoneServ StreamServ Histogram  
 Prompt Rate SMI Sender Receiver AsynchReceiver Rising TimeOrder VmeServ

### 5.2.5 MBS text

**MSG/GuiNode** Node where GUI runs  
**MSG/Date** Date as written in file header  
**MSG/Run** Run ID as written in file header  
**MSG/Experiment** Experiment as written in file header  
**MSG/User** Lynx user name as written in file header  
**MSG/Platform** CPU platform

### 5.2.6 MBS numbers

**MSG/BufferSize**  
**MSG/Buffers** collected so far.  
**MSG/Events** collected so far.  
**MSG/FileMbytes** written in file.  
**MSG/FlushTime**  
**MSG/MBytes** collected so far.  
**MSG/StreamKeep**  
**MSG/StreamMbytes**  
**MSG/StreamScale**  
**MSG/StreamSync**

## 5.3 Working directories

### 5.3.1 MBS configuration of DIM

Optional file `.guirc` in the *MBS* working directory specifies which rate meters and states shall appear in the GUI. Upper limits of the rate meters can be specified. This file can be copied from `$MBSROOT/set`. Only the parameters which are in this file are optional.

```
## This file controls the rate meter and state appearance.
```

```
## File name must be .guirc and in the MBS working directory.
## The value numbers are the maximum values for rate meters
## Colons only if value is specified!
## Node names must be uppercase, * wildcards all
## RunMode shows if MBS is set into DABC event builder mode
## and client is connected.

##===== All nodes:
##---- Rates:
* EventRate      : 10000.
* EventTrend     : 10000.
* DataRateKb     : 16000.
* DataTrendKb    : 16000.
* StreamRateKb   : 16000.
* StreamTrendKb  : 16000.
# ++ File filling status in percent, typically only on one node (transport)
#* FileFilled    : 100.
* StreamsFull    : 100.

##---- States:
# ++ Delayed or immediate event building:
* BuildingMode
# ++ Current eventbuilding running or suspended:
* EventBuilding
# ++ Running mode, stand alone or DABC:
* RunMode
# ++ Shows spill signal:
#* SpillOn
# ++ Shows if file is open, typically only on one node (transport)
#* FileOpen
#* Trigger

##===== Node XXX
#XXX EventRate   : 10000.
#XXX DataRateKb  : 16000.
#XXX RunMode
#XXX FileOpen
#XXX FileFilled  : 100.
#XXX SpillOn
#XXX EventTrend  : 10000.
#XXX DataTrendKb : 16000.
```

## Chapter 6

# *DABC* User Manual: *DABC* Application *MBS*

[user/user-app-mbs.tex]

### 6.1 *MBS* event building with *DABC*

To run *MBS* front-ends with *DABC* nodes as event builders some modifications of the *MBS* setup files must be done. For the *DABC* side setup files must be provided.

#### 6.1.1 *MBS* setup

When we want to use *DABC* nodes as event builders, we need a different setup on the *MBS* side. We assume that we have more than one *MBS* node. Such multi-node system is controlled by an *MBS* prompter running on one node. The setup has to be changed such that all nodes run as if they are stand alone. That means that each node runs the Readout - Collector - Transport chain. The *DABC* event builders connect to these transports. The *MBS* buffer size should be set to the stream size and the number of buffers per stream must be set to one.

#### 6.1.2 *DABC* setup

On the *DABC* user working directory we need configuration files.

Summary of parameters:

**MbsFileName** File name for list mode data file (LMD). Overwritten by command.

**MbsFileSizeLimit** File closes when size is reached, and new file opens.

**BufferSize** Should match *MBS* buffer size.

**MbsServerKind** Transport | Stream.

**MbsServerPort** Port number transport (6000).

**MbsServerName** *MBS* node of transport.

**NumInputs** Number of *MBS* channels for one combiner.

**DoFile** Provide output file.

**DoServer** Provide server.

These parameters are used to configure an event generator:

**NumSubevents**

**FirstProcId**

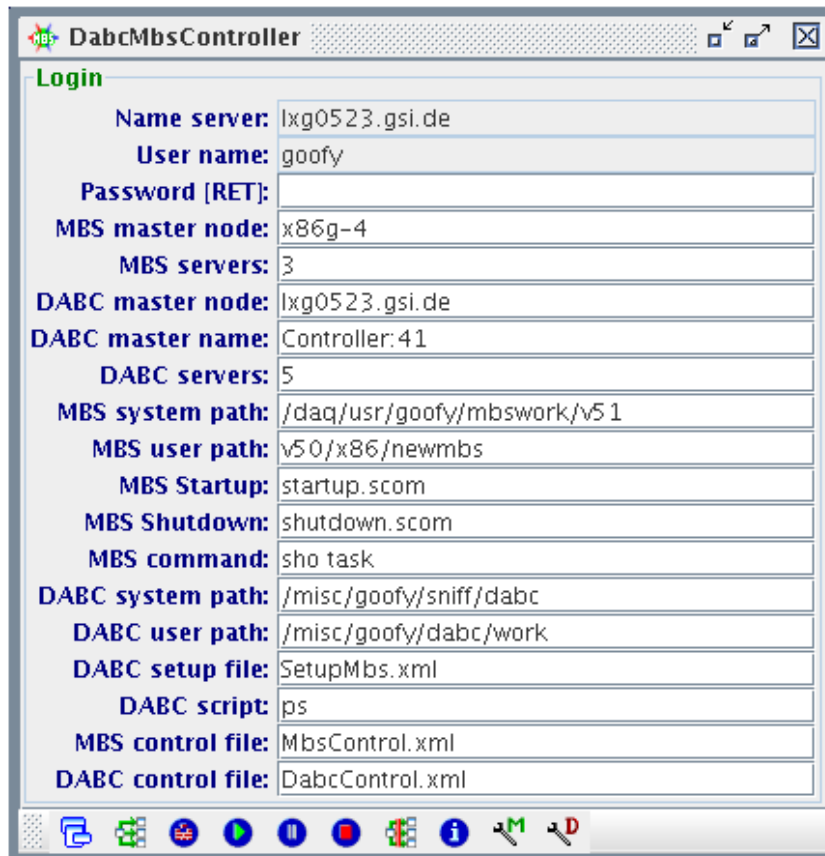
**SubeventSize**

**Go4Random**

The following example configuration file `$DABCSYS/applications/mbs/Combiner.xml` shows how to configure one combiner module reading from three *MBS* transport servers. Any running program is described by a Context, in this case named Combiner. The Run specifies the library, start function and log file name. We have one module Combiner (could be different name from context) with three input ports and two output ports.

Now we can use the combined controller panel to startup *MBS* and *DABC*.

### 6.1.3 Combined *DABC* and *MBS* control panel



**Figure 6.1:** Combined DABC and MBS controller.

This panel shown in Fig. 6.1, page 30 is simply a superposition of the single ones.

#### 6.1.3.1 Combined *DABC* and *MBS* controller buttons



Save panel settings, see 4.3, page 22.









Execute script `dabcstartup.sc` at *DABC* master node. Starts DIM servers. Execute script `prmstartup.sc` at *MBS* master node. Starts prompter, dispatchers and message loggers. Waits for all components. A progress panel pops up during that time (see 4.2.3, page 17). If all components are up trigger the main Update.



Configure. Execute user's *MBS* startup procedure in prompter. Executes state transition command Configure on *DABC* master node and wait for the transition. All plug-in components are created. Then execute Enable. If all components are up trigger the main Update.



Start *MBS* acquisition, then executes *DABC* Start command. All components go into running state Running.

-  Pause acquisition. Execute *MBS* stop acquisition. Execute *DABC* Stop command. All components go into standby state Ready.
-  Halt acquisition. Executes *DABC* Halt command. This closes all plug-ins. States go into Halted. Execute user's *MBS* shutdown procedure in prompter. Prompter, dispatcher and message loggers should still be running. Next must be shut down or configure. After two seconds trigger the main Update.
-  Shut down all. Execute EXIT command on all *DABC* nodes. Execute script `prmshtutdown.sc` at *MBS* master node. After two seconds trigger the main Update.
-  *MBS* Show acquisition. Output in log panel.
-  Shell script for *MBS* master node.
-  Shell script for *DABC* master node.





# References

- [1] Clara Gaspar. Dim - distributed information management system <http://dim.web.cern.ch/dim/>, 2008.
- [2] Andreas Kugel, Wenxue Gao, and Guillermo Marcus. The active buffer board online documentation, <http://cbm-wiki.gsi.de/cgi-bin/view/DAQ/ActiveBufferBoardV1>, 2008.

# Index

## DABC

Environment set-up, [8](#)

Installation, [7](#)

Plug-in installation, [9](#)

## TODO

dabcsetupfiles, [30](#)

Setup, [8](#)